

インテル[®] マス・カーネル・ライブラリー

リファレンス・マニュアル

資料番号 : 630813-023J

Web サイト :

<http://developer.intel.com> (英語)

<http://www.intel.com/jp/developer/software/products/> (日本語)

バージョン	バージョン情報	日付
-001	初版。	1994年 11月
-002	crotg と zrotg の両関数を追加。従来の版では説明されていなかった関数 ?her2k、?symm、?syrk、?syr2k のドキュメント・バージョン。また、ページの割り当てを変更。	1995年 5月
-003	タイトルを『Intel BLAS Library for the Pentium® Processor Reference Manual』から現在のものに変更。?rotm と ?rotmg の両関数を追加するとともに付録 C を更新。	1996年 1月
-004	並列化機能を持つインテル® マス・カーネル・ライブラリーのリリース 2.0 について説明。並列化に関する情報を第 1 章と、第 2 章の「BLAS レベル 3 ルーチン」のセクションに追加。	1996年 11月
-005	2 次元の FFT を追加。1 次元 FFT と 2 次元 FFT の両方に C インターフェイスを追加。	1997年 8月
-006	インテル・マス・カーネル・ライブラリーのリリース 2.1 について説明。第 2 章にスパース BLAS のセクションを追加。	1998年 1月
-007	インテル・マス・カーネル・ライブラリーのリリース 3.0 について説明。LAPACK ルーチン (第 4 章と第 5 章) ならびに CBLAS インターフェイス (付録 C) の説明を追加。マニュアルから「クイック・リファレンス」を削除。現時点では、MKL 3.0 の「クイック・リファレンス」が HTML 形式で利用可能。	1999年 1月
-008	インテル・マス・カーネル・ライブラリーのリリース 3.2 について説明。FFT ルーチンの説明を変更。第 4 章と第 5 章で LAPACK ルーチンの NAG 名を削除。	1999年 6月
-009	固有値問題用の新しい LAPACK ルーチンを第 5 章に追加。	1999年 11月
-010	インテル・マス・カーネル・ライブラリーのリリース 4.0 について説明。VML 関数について説明する第 6 章を追加。	2000年 6月
-011	インテル・マス・カーネル・ライブラリーのリリース 5.1 について説明。LAPACK のセクションを拡張し、すべての計算ルーチンとドライバルーチンの一覧を追加。	2001年 4月
-6001	インテル・マス・カーネル・ライブラリーのリリース 6.0 beta について説明。DFT インターフェイスとベクトル統計ライブラリー関数を新規に追加。	2002年 7月
-6002	インテル・マス・カーネル・ライブラリーのリリース 6.0 beta update について説明。DFT 関数の説明を更新。CBLAS インターフェイスの説明を拡張。	2002年 12月
-6003	インテル・マス・カーネル・ライブラリーのリリース 6.0 gold について説明。DFT 関数をアップデート。補助 LAPACK ルーチンの説明を本マニュアルに追加。	2003年 3月
-6004	インテル・マス・カーネル・ライブラリーのリリース 6.1 について説明。	2003年 7月
-6005	インテル・マス・カーネル・ライブラリーのリリース 7.0 beta について説明。ScaLAPACK およびスパースソルバーの説明を含む。	2003年 11月
-017	インテル MKL およびインテル® MKL クラスター・エディションのリリース 7.0 gold について説明。補助 ScaLAPACK および代替スパース・ソルバー・インターフェイスの追加。	2004年 4月
-018	インテル MKL およびインテル MKL クラスター・エディションのリリース 8.0 beta について説明。スパース BLAS および DFTI のセクションを拡張。新機能の追加: スパース BLAS、クラスター DFTI、反復法スパースソルバー、多倍長演算、区間連立ソルバー、および畳み込み / 相関。LAPACK 関数に Fortran95 インターフェイスを追加。	2005年 3月
-019	インテル MKL およびインテル MKL クラスター・エディションのリリース 8.0 gold について説明。BLAS およびスパース BLAS 関数に Fortran95 インターフェイスを追加。	2005年 08月
-020	インテル MKL およびインテル MKL クラスター・エディションのリリース 8.0.2. について説明。PARDISO の説明に、不定値対称行列のピボット演算の説明を追加。	2003年 06月
-021	インテル MKL およびインテル MKL クラスター・エディションのリリース 8.1 gold について説明。三角変換関数について説明する第 13 章を追加。LAPACK ルーチン用 Fortran-95 インターフェイス特有の機能に関する情報を付録 E と第 3 章に収録。	2003年 06月
-022	インテル MKL およびインテル MKL クラスター・エディションのリリース 9.0 beta について説明。統計関数、フーリエ変換関数 (クラスター DFT) に関する記述を更新。新しい VML 関数、RCI スパースソルバー、ポワソソルバーを追加。第 13 章のタイトルを「偏微分方程式のサポート」に変更。コードサンプルを拡張。	2006年 5月
-023	インテル MKL およびインテル MKL クラスター・エディションのリリース 9.0 gold について説明。複素区間ソルバーを追加。古い FFT 関数の説明を削除。	2006年 9月

本資料に掲載されている情報は、インテル製品の概要説明を目的としたものです。本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスを許諾するためのものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証（特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的所有権を侵害していないことへの保証を含む）にも一切応じないものとします。インテル製品は、医療、救命、延命措置、重要な制御または安全システム、核施設などの目的に使用することを前提としたものではありません。インテル製品は、予告なく仕様や説明が変更される場合があります。

本資料で説明されているソフトウェアには、不具合が含まれている可能性があり、公開されている仕様とは異なる動作をする場合があります。現在までに判明している不具合の情報については、インテルのサポートサイトをご覧ください。

本資料およびこれに記載されているソフトウェアはライセンス契約に基づいて提供されるものであり、その使用および複製はライセンス契約で定められた条件下でのみ許可されます。本資料で提供される情報は、情報供与のみを目的としたものであり、予告なく変更されることがあります。また、本資料で提供される情報は、インテルによる確約と解釈されるべきものではありません。インテルは本資料の内容およびこれに関連して提供されるソフトウェアにエラー、誤り、不正確な点が含まれていたとしても一切責任を負わないものとします。

ライセンス契約で許可されている場合を除き、インテルからの文書による承諾なく、本書のいかなる部分も複製したり、検索システムに保持したり、他の形式や媒体によって転送したりすることは禁じられています。

機能または命令の一覧で「留保」または「未定義」と記されているものがありますが、その「機能が存在しない」あるいは「性質が留保付である」という状態を開発の前提にしないでください。留保または未定義の機能を不適當な方法で使用すると、開発したソフトウェア・コードをインテル・プロセッサ上で実行する際に、予測不可能な動作や障害が発生するおそれがあります。これらの機能や命令は、インテルが将来のために留保しているものです。不正な使用により、衝突が生じたり互換性が失われたりしても、インテルは一切責任を負いません。

Intel、インテル、Intel ロゴ、Intel Core、Itanium、MMX、Pentium、Xeon は、米国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

© 1994-2006 Intel Corporation. 無断での引用、転載を禁じます。

Portions © Copyright 2001 Hewlett-Packard Development Company, L.P.

Chapters 4 and 5 include derivative work portions that have been copyrighted:
© 1991, 1992, and 1998 by The Numerical Algorithms Group, Ltd.

目次

第 1 章	概要	
	本ソフトウェアについて.....	1-1
	技術サポート.....	1-2
	BLAS ルーチン.....	1-2
	スパース BLAS ルーチン.....	1-2
	LAPACK ルーチン.....	1-3
	ScaLAPACK ルーチン.....	1-3
	スパース・ソルバー・ルーチン.....	1-3
	VML 関数.....	1-4
	統計関数.....	1-4
	フーリエ変換関数.....	1-4
	区間ソルバールーチン.....	1-4
	偏微分方程式のサポート.....	1-4
	GMP 数学関数.....	1-5
	性能の改善.....	1-5
	並列化.....	1-5
	対応プラットフォーム.....	1-6
	本書について.....	1-6
	本書の対象読者.....	1-6
	本書の構成.....	1-6
	表記の規則.....	1-8
	ルーチン名の省略表記.....	1-8
	字体の規則.....	1-8
第 2 章	BLAS ルーチンとスパース BLAS ルーチン	
	BLAS のルーチンと関数.....	2-2
	ルーチン命名規則.....	2-2
	Fortran-95 インターフェイス規則.....	2-3
	行列の格納形式.....	2-4
	BLAS レベル 1 のルーチンと関数.....	2-4
	?asum.....	2-5
	?axpy.....	2-6
	?copy.....	2-7

?dot.....	2-9
?sdot	2-10
?dotc	2-11
?dotu	2-12
?nrm2.....	2-13
?rot	2-14
?rotg.....	2-16
?rotm.....	2-17
?rotmg.....	2-19
?scal	2-20
?swap	2-22
i?amax.....	2-23
i?amin.....	2-24
dcabs1	2-25
BLAS レベル 2 のルーチン.....	2-26
?gbmv.....	2-27
?gemv.....	2-30
?ger	2-32
?gerc	2-34
?geru	2-35
?hbmV	2-37
?hemv	2-39
?her.....	2-41
?her2	2-43
?hpmv	2-45
?hpr.....	2-47
?hpr2	2-49
?sbmv.....	2-51
?spmV.....	2-53
?spr	2-55
?spr2	2-57
?symv.....	2-59
?syr.....	2-61
?syr2.....	2-62
?tbmv	2-64
?tbsv.....	2-67
?tpmV	2-69
?tpsv.....	2-71
?trmV	2-73
?trsv	2-75
BLAS レベル 3 のルーチン.....	2-78
インテル® MKL の対称型マルチプロセッシング・バージョン.....	2-78

?gemm.....	2-78
?hemm.....	2-81
?herk.....	2-84
?her2k.....	2-86
?symm.....	2-89
?syrk.....	2-92
?syr2k.....	2-94
?trmm.....	2-97
?trsm.....	2-100
スパース BLAS レベル 1 のルーチンと関数.....	2-103
ベクトルの引数.....	2-103
命名規則.....	2-103
ルーチンとデータ型.....	2-104
スパースベクトルで使用できる BLAS レベル 1 のルーチン.....	2-104
?axpyi.....	2-105
?doti.....	2-106
?dotci.....	2-107
?dotui.....	2-108
?gthr.....	2-110
?gthrz.....	2-111
?roti.....	2-112
?sctr.....	2-113
スパース BLAS レベル 2 およびレベル 3.....	2-115
スパース BLAS レベル 2 およびレベル 3 における命名規則.....	2-115
スパース行列のデータ構造.....	2-116
ルーチンおよびサポートされる演算.....	2-116
標準インターフェイスを備えたルーチン.....	2-116
簡易インターフェイスを備えたルーチン.....	2-117
インターフェイス考慮事項.....	2-117
インテル MKL インターフェイスと NIST インターフェイスの相違点....	2-117
簡易インターフェイス.....	2-119
部分行列の演算.....	2-119
三角ソルバールーチンの制限.....	2-120
スパース BLAS レベル 2 およびレベル 3 のルーチン.....	2-121
mkl_dcsmv.....	2-122
mkl_dcsgemv.....	2-124
mkl_dcsrcsymv.....	2-126
mkl_dcscmv.....	2-127
mkl_dcoomv.....	2-129
mkl_dcoogemv.....	2-131
mkl_dcoosymv.....	2-133

mkl_ddiamv	2-134
mkl_ddiagemv	2-136
mkl_ddiasymv	2-138
mkl_dskymv	2-139
mkl_dcsrsv	2-141
mkl_dcsrtrsv	2-143
mkl_dcscsv	2-145
mkl_dcoosv	2-147
mkl_dcootrsv	2-149
mkl_ddiasv	2-150
mkl_ddiatrsv	2-152
mkl_dskysv	2-154
mkl_dcsrmm	2-156
mkl_dcscmm	2-158
mkl_dcoomm	2-160
mkl_ddiamm	2-162
mkl_dskymm	2-165
mkl_dcsrsm	2-167
dcscsm	2-169
mkl_dcoosm	2-171
mkl_ddiasm	2-173
mkl_dskysm	2-175

第 3 章 LAPACK ルーチン : 線形方程式

ルーチン命名規則	3-2
Fortran-95 インターフェイス規則	3-3
MKL LAPACK ルーチン用 Fortran-95 インターフェイスと Netlib 実装	3-4
行列の格納形式	3-5
数学的表記	3-6
誤差の分析	3-6
計算ルーチン	3-8
行列の因子分解用のルーチン	3-9
?getrf	3-10
?gbtrf	3-11
?gttrf	3-13
?potrf	3-15
?pptrf	3-17
?pbtrf	3-18
?pttrf	3-20
?sytrf	3-21
?hetrf	3-24
?sptrf	3-27
?hptrf	3-29

連立線形方程式を解くためのルーチン	3-32
?getrs	3-32
?gbtrs	3-34
?gttrs	3-36
?potrs	3-38
?pptrs	3-40
?pbtrs	3-42
?pttrs	3-44
?sytrs	3-46
?hetrs	3-48
?sptrs	3-50
?hptrs	3-52
?trtrs	3-54
?tptrs	3-56
?tbtrs	3-58
条件数を推定するためのルーチン	3-61
?gecon	3-61
?gbcon	3-63
?gtcon	3-65
?pocon	3-67
?ppcon	3-69
?pbcon	3-71
?ptcon	3-73
?sycon	3-74
?hecon	3-76
?spcon	3-78
?hpcon	3-80
?trcon	3-81
?tpcon	3-83
?tbcon	3-85
解の精度の改善と誤差の推定	3-87
?gerfs	3-88
?gbrfs	3-90
?gtrfs	3-93
?porfs	3-96
?pprfs	3-98
?pbrfs	3-101
?ptrfs	3-103
?syrfs	3-106
?herfs	3-108
?sprfs	3-111
?hprfs	3-113

?trrfs	3-116
?tprfs.....	3-118
?tbrfs.....	3-121
行列の反転用のルーチン.....	3-124
?getri.....	3-124
?potri.....	3-126
?pptri.....	3-127
?sytri.....	3-129
?hetri.....	3-131
?sptri.....	3-132
?hptri.....	3-134
?trtri.....	3-136
?tptri.....	3-137
行列の平衡化.....	3-140
?geequ.....	3-140
?gbequ.....	3-142
?poequ.....	3-144
?ppequ.....	3-145
?pbequ.....	3-147
ドライバールーチン.....	3-150
?gesv.....	3-150
?gesvx.....	3-152
?gbsv.....	3-157
?gbsvx.....	3-159
?gtsv.....	3-164
?gtsvx.....	3-166
?posv.....	3-170
?posvx.....	3-171
?ppsv.....	3-176
?ppsvx.....	3-177
?pbsv.....	3-182
?pbsvx.....	3-183
?ptsv.....	3-188
?ptsvx.....	3-189
?sysv.....	3-192
?sysvx.....	3-195
?hesv.....	3-199
?hesvx.....	3-201
?spsv.....	3-205
?spsvx.....	3-207
?hpsv.....	3-210
?hpsvx.....	3-212

第 4 章	LAPACK ルーチン : 最小二乗問題および固有値問題	
	ルーチン命名規則	4-2
	行列の格納形式	4-3
	数学的表記	4-4
	計算ルーチン	4-5
	直交因子分解	4-5
	?geqrf	4-6
	?geqpf	4-9
	?geqp3	4-11
	?orgqr	4-13
	?ormqr	4-15
	?ungqr	4-18
	?unmqr	4-20
	?gelqf	4-22
	?orglq	4-24
	?ormlq	4-26
	?unglq	4-28
	?unmlq	4-30
	?geqlf	4-32
	?orgql	4-34
	?ungql	4-36
	?ormql	4-37
	?unmql	4-39
	?gerqf	4-42
	?orgrq	4-44
	?ungrq	4-45
	?ormrq	4-47
	?unmrq	4-49
	?tzzrf	4-51
	?ormrz	4-53
	?unmrz	4-56
	?ggqrf	4-58
	?ggrqf	4-61
	特異値分解	4-64
	?gebrd	4-66
	?gbbrd	4-69
	?orgbr	4-72
	?ormbr	4-74
	?ungbr	4-77
	?unmbr	4-80
	?bdsqr	4-83
	?bdscd	4-86

対称固有値問題	4-89
?sytrd.....	4-93
?orgtr	4-95
?ormtr.....	4-96
?hetrd.....	4-99
?ungtr.....	4-101
?unmtr	4-103
?sptrd.....	4-105
?opgtr.....	4-106
?opmtr	4-108
?hptrd.....	4-110
?upgtr.....	4-111
?upmtr	4-113
?sbtrd.....	4-115
?hbtrd.....	4-117
?sterf.....	4-119
?steqr.....	4-120
?stedc.....	4-123
?stegr.....	4-127
?pteqr.....	4-131
?stebz	4-134
?stein	4-137
?disna.....	4-139
汎用対称固有値問題	4-141
?sygst	4-141
?hegst.....	4-143
?spgst	4-145
?hpgst.....	4-147
?sbgst	4-149
?hbgst.....	4-151
?pbstf.....	4-153
非対称固有値問題	4-155
?gehrd.....	4-157
?orghr.....	4-159
?ormhr	4-161
?unghr	4-164
?unmhr.....	4-166
?gebal.....	4-168
?gebak.....	4-171
?hseqr	4-173
?hsein.....	4-177
?trevc.....	4-182

?trsna	4-186
?trexc	4-190
?trsen	4-192
?trsyl.....	4-196
汎用非対称固有値問題.....	4-199
?gghrd	4-200
?ggbal	4-203
?ggbak.....	4-205
?hgeqz.....	4-207
?tgevc	4-213
?tgexc	4-217
?tgsen	4-220
?tgsyl.....	4-225
?tgsna.....	4-228
汎用特異値分解.....	4-233
?ggsvp	4-233
?tgsja.....	4-237
ドライバルーチン	4-243
線形最小二乗 (LLS) 問題	4-243
?gels.....	4-243
?gelsy	4-246
?gelss	4-249
?gelsd.....	4-252
汎用 LLS 問題.....	4-256
?gglse	4-256
?ggglm	4-258
対称固有値問題.....	4-261
?syev	4-262
?heev	4-264
?syevd.....	4-266
?heevd	4-268
?syevx.....	4-271
?heevx.....	4-274
?syevr.....	4-278
?heevr	4-282
?spev	4-287
?hpev.....	4-289
?spevd.....	4-291
?hpevd	4-293
?spevx.....	4-296
?hpevx.....	4-299
?sbev	4-303

?hbev.....	4-305
?sbevd	4-307
?hbevd.....	4-309
?sbevz.....	4-312
?hbevz	4-316
?stev.....	4-320
?stevd.....	4-321
?stevz	4-324
?stevr.....	4-327
非対称固有値問題	4-332
?gees.....	4-332
?geesx.....	4-336
?geev.....	4-341
?geevz.....	4-344
特異値分解.....	4-350
?gesvd.....	4-350
?gesdd	4-354
?ggsvd.....	4-357
汎用対称固有値問題	4-363
?sygv	4-363
?hegv.....	4-366
?sygvd.....	4-368
?hegvd	4-371
?sygvz.....	4-374
?hegvz.....	4-378
?spgv	4-382
?hpgv.....	4-385
?spgvd.....	4-387
?hpgvd	4-390
?spgvz.....	4-393
?hpgvz.....	4-396
?sbgv	4-400
?hbgv.....	4-402
?sbgvd.....	4-404
?hbgvd	4-407
?sbgvz.....	4-410
?hbgvz.....	4-413
汎用非対称固有値問題	4-418
?gges	4-418
?ggesx.....	4-423
?ggev	4-429
?ggevz.....	4-433

第 5 章	LAPACK 補助ルーチンとユーティリティー・ルーチン	
	補助ルーチン	5-1
	?lacgv	5-8
	?lacrm	5-8
	?lacrt	5-9
	?laesy	5-10
	?rot.....	5-11
	?spmv	5-12
	?spr	5-14
	?symv	5-15
	?syr	5-17
	i?max1	5-18
	?sum1	5-19
	?gbtf2	5-19
	?gebd2.....	5-21
	?gehd2.....	5-22
	?gelq2.....	5-24
	?geql2.....	5-25
	?geqr2	5-27
	?gerq2	5-28
	?gesc2.....	5-29
	?getc2	5-30
	?getf2	5-32
	?gtts2	5-33
	?labrd.....	5-34
	?lacon.....	5-36
	?lacpy	5-37
	?ladiv	5-38
	?lae2	5-39
	?laebz	5-40
	?laed0.....	5-44
	?laed1	5-46
	?laed2.....	5-48
	?laed3.....	5-50
	?laed4.....	5-51
	?laed5.....	5-53
	?laed6.....	5-54
	?laed7	5-55
	?laed8.....	5-58
	?laed9.....	5-60
	?laeda	5-62
	?laein	5-63

?laev2.....	5-66
?laexc.....	5-67
?lag2.....	5-68
?lags2.....	5-70
?lagtf.....	5-72
?lagtm.....	5-73
?lagts.....	5-75
?lagv2.....	5-76
?lahqr.....	5-78
?lahrd.....	5-80
?laic1.....	5-82
?lain2.....	5-84
?lals0.....	5-86
?lalsa.....	5-89
?lalsd.....	5-92
?lamrg.....	5-94
?langb.....	5-95
?lange.....	5-96
?langt.....	5-97
?lanhs.....	5-98
?lansb.....	5-100
?lanhb.....	5-101
?lansp.....	5-103
?lanhp.....	5-104
?lanst/?lanht.....	5-105
?lansy.....	5-106
?lanhe.....	5-108
?lantb.....	5-109
?lantp.....	5-110
?lantr.....	5-112
?lanv2.....	5-113
?lapll.....	5-114
?lapmt.....	5-115
?lapy2.....	5-116
?lapy3.....	5-117
?laqgb.....	5-117
?laqge.....	5-119
?laqp2.....	5-120
?laqps.....	5-122
?laqsb.....	5-123
?laqsp.....	5-125
?laqsy.....	5-126

?laqtr	5-128
?lar1v	5-130
?lar2v	5-132
?larf	5-133
?larfb	5-134
?larfg	5-136
?larft	5-137
?larfx	5-140
?largv	5-141
?larv	5-142
?larrb	5-143
?larre	5-145
?larrf	5-146
?larrv	5-148
?lartg	5-150
?lartv	5-151
?laruv	5-152
?larz	5-153
?larzb	5-154
?larzt	5-156
?las2	5-159
?lascl	5-160
?lasd0	5-161
?lasd1	5-162
?lasd2	5-165
?lasd3	5-168
?lasd4	5-170
?lasd5	5-171
?lasd6	5-172
?lasd7	5-176
?lasd8	5-179
?lasd9	5-181
?lasda	5-182
?lasdq	5-186
?lasdt	5-188
?laset	5-189
?lasq1	5-190
?lasq2	5-191
?lasq3	5-192
?lasq4	5-193
?lasq5	5-194
?lasq6	5-195

?lasr.....	5-196
?lasrt.....	5-198
?lassq.....	5-198
?lasv2.....	5-200
?laswp.....	5-201
?lasy2.....	5-202
?lasyf.....	5-204
?lahef.....	5-206
?latbs.....	5-208
?latdf.....	5-210
?latps.....	5-212
?latrd.....	5-214
?latrs.....	5-216
?latrz.....	5-219
?lauu2.....	5-221
?lauum.....	5-222
?org2l/?ung2l.....	5-223
?org2r/?ung2r.....	5-225
?orgl2/?ungl2.....	5-226
?orgr2/?ungr2.....	5-227
?orm2l/?unm2l.....	5-229
?orm2r/?unm2r.....	5-231
?orml2/?unml2.....	5-233
?ormr2/?unmr2.....	5-235
?ormr3/?unmr3.....	5-237
?pbt2.....	5-239
?potf2.....	5-240
?ptts2.....	5-241
?rscl.....	5-243
?sygs2/?hegs2.....	5-244
?sytd2/?hetd2.....	5-245
?sytf2.....	5-247
?hetf2.....	5-248
?tgex2.....	5-250
?tgsy2.....	5-252
?trti2.....	5-255
ユーティリティー関数とルーチン.....	5-256
ilaenv.....	5-257
ieeck.....	5-259
lsame.....	5-260
lsamen.....	5-260
?labad.....	5-261

?lamch.....	5-261
?lamc1.....	5-262
?lamc2.....	5-263
?lamc3.....	5-264
?lamc4.....	5-264
?lamc5.....	5-265
second/dsecnd.....	5-266
xerbla.....	5-266

第 6 章

ScaLAPACK ルーチン

概要.....	6-1
ルーチン命名規則.....	6-2
計算ルーチン.....	6-3
1 次方程式.....	6-3
行列の因子分解用のルーチン.....	6-5
p?getrf.....	6-5
p?gbtrf.....	6-6
p?dbtrf.....	6-8
p?potrf.....	6-10
p?pbtrf.....	6-12
p?pttrf.....	6-14
p?dttrf.....	6-16
連立 1 次方程式を解くためのルーチン.....	6-18
p?getrs.....	6-18
p?gbtrs.....	6-20
p?potrs.....	6-22
p?pbtrs.....	6-23
p?pttrs.....	6-25
p?dttrs.....	6-27
p?dbtrs.....	6-30
p?trtrs.....	6-32
条件数を推定するためのルーチン.....	6-34
p?gecon.....	6-34
p?pocon.....	6-36
p?trcon.....	6-39
解の精度の改善と誤差の推定.....	6-41
p?gerfs.....	6-41
p?porfs.....	6-44
p?trfs.....	6-47
行列の反転用のルーチン.....	6-51
p?getri.....	6-51
p?potri.....	6-53
p?trtri.....	6-54

行列の平衡化	6-56
p?geequ.....	6-56
p?poequ.....	6-58
直交因子分解	6-60
p?geqrf.....	6-60
p?geqpf.....	6-62
p?orgqr	6-65
p?ungqr.....	6-66
p?ormqr.....	6-68
p?unmqr	6-71
p?gelqf.....	6-74
p?orglq.....	6-76
p?unglq.....	6-77
p?ormlq.....	6-79
p?unmlq.....	6-82
p?geqlf	6-85
p?orgql.....	6-87
p?ungql	6-88
p?ormql.....	6-90
p?unmql.....	6-93
p?gerqf.....	6-96
p?orgrq	6-98
p?ungrq.....	6-99
p?ormrq.....	6-101
p?unmrq.....	6-104
p?tzzrf	6-107
p?ormrz	6-109
p?unmrz	6-112
p?ggqrf.....	6-114
p?ggrqf.....	6-118
対称固有値問題	6-123
p?sytrd	6-123
p?ormtr.....	6-126
p?hetrd.....	6-129
p?unmtr.....	6-132
p?stebz.....	6-135
p?stein.....	6-139
非対称固有値問題	6-143
p?gehrd.....	6-143
p?ormhr.....	6-146
p?unmhr	6-149
p?lahqr.....	6-151

特異値分解	6-154
p?gebrd	6-154
p?ormbr	6-158
p?unmbr	6-161
汎用対称固有値問題	6-166
p?sygst	6-166
p?hegst	6-168
ドライバルーチン	6-170
p?gesv	6-170
p?gesvx	6-172
p?gbsv	6-177
p?dbsv	6-179
p?dtsv	6-182
p?posv	6-184
p?posvx	6-186
p?pbsv	6-191
p?ptsv	6-193
p?gels	6-195
p?syev	6-198
p?syevx	6-201
p?heevx	6-206
p?gesvd	6-213
p?sygvx	6-216
p?hegvx	6-223

第 7 章

ScaLAPACK 補助ルーチンとユーティリティー・ルーチン

補助ルーチン	7-1
p?lacgv	7-5
p?max1	7-6
?combamax1	7-7
p?sum1	7-7
p?dbtrsv	7-8
p?dttrsv	7-11
p?gebd2	7-14
p?gehd2	7-17
p?gelq2	7-20
p?geql2	7-22
p?geqr2	7-24
p?gerq2	7-26
p?getf2	7-28
p?labrd	7-29
p?lacon	7-33
p?laconsb	7-35

p?lacp2	7-36
p?lacp3	7-37
p?lacpy	7-39
p?laevswp	7-40
p?lahrd	7-42
p?laiect	7-44
p?lange	7-45
p?lanhs	7-47
p?lansy, p?lanhe	7-49
p?lantr	7-51
p?lapiv	7-53
p?laqge	7-55
p?laqsy	7-57
p?lared1d	7-59
p?lared2d	7-60
p?larf	7-61
p?larfb	7-64
p?larfc	7-67
p?larfg	7-70
p?larft	7-71
p?larz	7-74
p?larzb	7-77
p?larzc	7-80
p?larzt	7-83
p?lascl	7-86
p?laset	7-88
p?lasmsub	7-89
p?lassq	7-90
p?laswp	7-92
p?latra	7-93
p?latrd	7-94
p?latrs	7-98
p?latrz	7-100
p?lauu2	7-102
p?lauum	7-103
p?lawil	7-104
p?org2l/p?ung2l	7-105
p?org2r/p?ung2r	7-107
p?orgl2/p?ungl2	7-109
p?orgr2/p?ungr2	7-111
p?orm2l/p?unm2l	7-113
p?orm2r/p?unm2r	7-116

p?orml2/p?unml2.....	7-119
p?ormr2/p?unmr2	7-122
p?pbtrsv	7-126
p?pttrsv	7-129
p?potf2	7-132
p?rscl	7-133
p?sygs2/p?hegs2.....	7-134
p?sytd2/p?hetd2	7-137
p?trti2	7-140
?lamsh	7-141
?laref	7-143
?lasorte.....	7-145
?lasrt2	7-146
?stein2	7-147
?dbtf2.....	7-149
?dbtrf.....	7-150
?dttrf.....	7-152
?dttrsv	7-153
?pttrsv	7-154
?steqr2.....	7-155
ユーティリティー関数とルーチン.....	7-158
p?labad.....	7-158
p?lachkieee	7-159
p?lamch	7-160
p?lasnbt.....	7-161
pxerbla	7-161

第 8 章

スパース・ソルバー・ルーチン

PARDISO - 並列化対応直接法スパース・ソルバー・インターフェイス...	8-1
pardiso	8-2
直接法スパースソルバー (DSS) インターフェイス・ルーチン.....	8-13
インターフェイスの説明.....	8-15
ルーチンオプション	8-15
ユーザーデータ配列	8-15
DSS ルーチン	8-16
dss_create	8-16
dss_define_structure.....	8-16
dss_reorder	8-18
dss_factor_real、dss_factor_complex	8-18
dss_solve_real、dss_solve_complex	8-19
dss_delete.....	8-20
dss_statistics	8-21
mkl_cvt_to_null_terminated_str.....	8-23

実装の詳細	8-25
メモリー割り当てとハンドル	8-25
リバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復 法スパースソルバー	8-27
インターフェイスの説明	8-31
ルーチンオプション	8-31
ユーザーデータ配列	8-31
CG の共通パラメーター	8-31
FGMRES の共通パラメーター	8-34
RCI CG ルーチン	8-38
dcg_init	8-38
dcg_check	8-39
dcg	8-40
dcg_get	8-42
RCI FGMRES ルーチン	8-42
dfgmres_init	8-42
dfgmres_check	8-44
dfgmres	8-45
dfgmres_get	8-47
実装の詳細	8-48
C/C++ からのスパース・ソルバー・ルーチンの呼び出し	8-48
C ユーザーが気を付けるべきこと	8-49

第 9 章

ベクトル数学関数	
データ型と精度モード	9-1
関数命名規則	9-2
関数インターフェイス	9-3
VML 数学関数	9-3
Pack 関数	9-3
Pack 関数	9-3
サービス関数 :	9-4
入力パラメーター	9-4
出力パラメーター	9-4
ベクトル・インデックス方式	9-5
エラー診断	9-5
ベクトル数学関数のスレッド化	9-6
VML 数学関数	9-6
累乗関数と累乗根関数	9-8
Inv	9-8
Div	9-9
Sqrt	9-10
InvSqrt	9-11
Cbrt	9-12

InvCbrt.....	9-13
Pow.....	9-14
Powx.....	9-15
Hypot.....	9-17
指数関数と対数関数.....	9-18
Exp.....	9-18
Ln.....	9-20
Log10.....	9-21
三角関数.....	9-22
Cos.....	9-22
Sin.....	9-23
SinCos.....	9-24
Tan.....	9-25
Acos.....	9-26
Asin.....	9-27
Atan.....	9-28
Atan2.....	9-29
双曲線関数.....	9-30
Cosh.....	9-30
Sinh.....	9-32
Tanh.....	9-33
Acosh.....	9-34
Asinh.....	9-35
Atanh.....	9-36
特殊関数.....	9-38
Erf.....	9-38
Erfc.....	9-39
ErfInv.....	9-40
丸め関数.....	9-41
Floor.....	9-41
Ceil.....	9-42
Trunc.....	9-42
Round.....	9-43
NearbyInt.....	9-44
Rint.....	9-45
Modf.....	9-46
VML Pack/Unpack 関数.....	9-48
Pack.....	9-48
Unpack.....	9-50
VML サービス関数.....	9-52
SetMode.....	9-53
GetMode.....	9-55

	SetErrStatus.....	9-56
	GetErrStatus	9-57
	ClearErrStatus.....	9-58
	SetErrorCallBack.....	9-58
	GetErrorCallBack.....	9-60
	ClearErrorCallBack	9-61
第 10 章	統計関数	
	乱数生成器.....	10-1
	規則	10-2
	数学的表記.....	10-3
	命名規則.....	10-4
	基本生成器.....	10-7
	BRNG パラメーターの定義.....	10-8
	ランダムストリーム	10-8
	データ型.....	10-9
	エラー報告.....	10-9
	サービスルーチン	10-10
	NewStream.....	10-12
	NewStreamEx.....	10-13
	iNewAbstractStream.....	10-14
	dNewAbstractStream.....	10-16
	sNewAbstractStream.....	10-18
	DeleteStream	10-20
	CopyStream	10-21
	CopyStreamState.....	10-22
	SaveStreamF.....	10-23
	LoadStreamF	10-24
	LeapfrogStream.....	10-26
	SkipAheadStream	10-28
	GetStreamStateBrng	10-31
	GetNumRegBrngs	10-32
	分布生成器.....	10-33
	連続分布	10-34
	Uniform.....	10-34
	Gaussian.....	10-36
	GaussianMV	10-38
	Exponential.....	10-42
	Laplace.....	10-44
	Weibull	10-46
	Cauchy	10-48
	Rayleigh.....	10-50
	Lognormal	10-52

Gumbel.....	10-54
Gamma.....	10-56
Beta.....	10-58
離散分布.....	10-60
Uniform.....	10-60
UniformBits.....	10-62
Bernoulli.....	10-63
Geometric.....	10-65
Binomial.....	10-66
Hypergeometric.....	10-68
Poisson.....	10-70
PoissonV.....	10-71
Negbinomial.....	10-73
アドバンスド・サービス・ルーチン.....	10-75
データ型.....	10-75
RegisterBrng.....	10-76
GetBrngProperties.....	10-77
ユーザー定義生成器のフォーマット.....	10-78
iBRng.....	10-80
sBRng.....	10-81
dBRng.....	10-81
畳み込みと相関.....	10-82
概要.....	10-82
命名規則.....	10-83
データ型.....	10-84
パラメーター.....	10-84
タスクステータスとエラー報告.....	10-86
タスク・コンストラクター.....	10-87
NewTask.....	10-88
NewTask1D.....	10-90
NewTaskX.....	10-91
NewTaskX1D.....	10-94
タスクエディター.....	10-96
SetMode.....	10-97
SetInternalPrecision.....	10-98
SetStart.....	10-99
SetDecimation.....	10-100
タスク実行ルーチン.....	10-101
Exec.....	10-102
Exec1D.....	10-104
ExecX.....	10-106
ExecX1D.....	10-107

タスク・ディストラクター	10-109
DeleteTask	10-109
タスクコピー	10-110
CopyTask	10-110
使用法の例	10-112
マルチスレッドの使用	10-113
数学表記と定義	10-115
線形畳み込み	10-115
線形相関	10-116
データの割り当て	10-116
有限関数とデータベクトル	10-116
データベクトルの割り当て	10-117

第 11 章 フーリエ変換関数

DFT 関数	11-1
DFT の計算	11-2
DFT インターフェイス	11-3
ステータス確認関数	11-4
ErrorClass	11-4
ErrorMessage	11-5
ディスクリプター操作	11-6
CreateDescriptor	11-6
CommitDescriptor	11-8
CopyDescriptor	11-9
FreeDescriptor	11-10
DFT 計算	11-10
ComputeForward	11-11
ComputeBackward	11-13
ディスクリプター構成	11-14
SetValue	11-15
GetValue	11-16
構成設定	11-18
変換精度	11-21
順方向変換	11-21
変換の次元と長さ	11-22
変換回数	11-22
倍率	11-22
結果の配置	11-22
圧縮形式	11-23
格納体系	11-25
ユーザースレッド数	11-33
入力と出力との距離	11-34
ストライド	11-34

順序指定	11-35
転置	11-36
クラスター DFT 関数	11-37
クラスター DFT の計算	11-38
プロセス間でのデータの分配	11-39
多次元変換	11-39
1次元変換	11-40
ローカルデータのメモリーサイズ	11-40
利用可能な補助関数	11-41
変換の長さに対する制限	11-41
クラスター DFT インターフェイス	11-41
ディスクリプター操作	11-42
CreateDescriptorDM	11-42
CommitDescriptorDM	11-44
FreeDescriptorDM	11-45
DFT 計算	11-46
ComputeForwardDM	11-46
ComputeBackwardDM	11-48
ディスクリプター構成	11-50
SetValueDM	11-51
GetValueDM	11-52
エラーコード	11-54

第 12 章

区間線形ソルバー

ルーチン命名規則	12-2
区間方程式を高速に解くためのルーチン	12-3
?trtrs	12-3
?gegas	12-4
?gehss	12-6
?gekws	12-7
?gegss	12-9
?gehbs	12-10
区間方程式の精密な解を求めるためのルーチン	12-12
?gepps	12-12
?gepss	12-13
区間行列逆転用のルーチン	12-16
?trtri	12-16
?geszi	12-17
区間行列の特性確認用のルーチン	12-18
?gerbr	12-18
?gesvr	12-19
補助およびユーティリティー・ルーチン	12-21
?gemip	12-21

第 13 章	偏微分方程式のサポート	
	三角変換ルーチン	13-1
	実装されている変換	13-2
	TT ルーチン起動の順序	13-2
	インターフェイスの説明	13-4
	ルーチンオプション	13-4
	ユーザーデータ配列	13-4
	TT ルーチン	13-5
	?_init_trig_transform	13-5
	?_commit_trig_transform	13-6
	?_forward_trig_transform	13-8
	?_backward_trig_transform	13-9
	free_trig_transform	13-11
	共通パラメーター	13-11
	パラメーター修正に関する警告	13-14
	実装の詳細	13-15
	C 固有のヘッダーファイル	13-15
	Fortran 固有のヘッダーファイル	13-15
	ポアソン・ライブラリー・ルーチン	13-19
	実装されているポアソン・ライブラリー	13-19
	2 次元問題	13-19
	3 次元問題	13-21
	PL ルーチン起動の順序	13-22
	インターフェイスの説明	13-24
	ルーチンオプション	13-25
	ユーザーデータ配列	13-25
	PL ルーチン	13-25
	?_init_Helmholtz_2D/?_init_Helmholtz_3D	13-26
	?_commit_Helmholtz_2D/?_commit_Helmholtz_3D	13-28
	?_Helmholtz_2D/?_Helmholtz_3D	13-32
	free_Helmholtz_2D/free_Helmholtz_3D	13-36
	共通パラメーター	13-37
	パラメーター修正に関する警告	13-41
	実装の詳細	13-41
	C 固有のヘッダーファイル	13-42
	Fortran 固有のヘッダーファイル	13-43
	Fortran-90 からの PDE サポートルーチンの呼び出し	13-47
付録 A	線形ソルバーの基礎	
	スパース連立線形方程式	A-1
	行列の基礎事項	A-2
	直接法	A-3
	スパース行列のフィルインとリオーダーリング	A-3

スパース行列の格納形式	A-6
PARDISO* ソルバーの格納形式	A-6
スパース BLAS レベル 2-3 のスパース格納形式	A-9
CSR 形式	A-9
CSC 形式	A-10
座標形式	A-11
対角格納方式	A-11
スカイライン格納方式	A-12
区間線形方程式	A-14
区間	A-14
区間ベクトルと行列	A-15
区間線形方程式	A-16
前処理	A-18
区間行列の逆転	A-19
付録 B	
ルーチンおよび関数の引数	
BLAS のベクトル引数	B-1
VML のベクトル引数	B-3
正増分インデックス	B-3
インデックス・ベクトル・インデックス	B-3
マスク・ベクトル・インデックス	B-3
行列引数	B-3
付録 C	
コード例	
BLAS のコード例	C-1
PARDISO のコード例	C-6
スパース対称連立線形方程式の例	C-6
対称連立線形方程式の計算結果の例	C-6
スパース非対称連立線形方程式の例	C-14
非対称連立線形方程式の計算結果の例	C-14
直接法スパースソルバーのコード例	C-20
対称連立線形方程式の計算結果の例	C-20
反復法スパースソルバーのコード例	C-27
RCI (前処理付き) 共役勾配ソルバーの使用例	C-27
RCI (前処理付き) フレキシブル汎用最小残差ソルバーの使用例	C-31
Fortran の例	C-31
C の例	C-36
フーリエ変換関数のコード例	C-42
DFT 関数の例	C-42
DFT 計算でのマルチスレッディングの使用例	C-52
クラスター DFT 関数の例	C-56
区間連立線形方程式ソルバーのコード例	C-58
PDE サポートのコード例	C-65

	三角変換のコード例	C-65
	ポアソン・ライブラリーのコード例	C-81
付録 D	BLAS に対する CBLAS インターフェイス	
	CBLAS の引数	D-1
	列挙型	D-1
	レベル 1 の CBLAS	D-3
	レベル 2 の CBLAS	D-5
	レベル 3 の CBLAS	D-10
	スパース CBLAS	D-13
付録 E	LAPACK ルーチン用 Fortran-95 インターフェイス特有の機能	
	Netlib と同一のインターフェイス	E-2
	引数名が置換されるインターフェイス	E-4
	修正された Netlib インターフェイス	E-5
	Netlib にはないインターフェイス	E-6
	新機能のインターフェイス	E-10
用語集		
参考文献		
索引		

インテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) では、ベクトルおよびスパース行列や区間行列などの行列に対して広範囲にわたる演算を実行する Fortran のルーチンおよび関数を提供します。また、Fortran および C のインターフェイスを使用するベクトル数値演算関数とベクトル統計関数だけでなく、離散フーリエ変換関数も含まれています。

インテル[®] MKL クラスター・エディションと呼ばれるライブラリーのバージョンはインテル MKL のスーパーセットで、分散メモリの並列処理コンピューター上での個々の計算問題を解決するための ScaLAPACK ソフトウェアと Cluster DFT ソフトウェアが含まれます。

インテル MKL は最新のインテル[®] プロセッサ用に最適化されているため、MKL を使用することにより、アプリケーション・プログラムの性能が向上します。本章では、マス・カーネル・ライブラリーを紹介するとともに、本マニュアルの構成について説明します。

本ソフトウェアについて

インテル・マス・カーネル・ライブラリーには、以下に示すグループのルーチンが含まれます。

- **Basic Linear Algebra Subprograms (BLAS):**
 - ベクトル演算
 - 行列 - ベクトル演算
 - 行列 - 行列演算
- スパース BLAS レベル 1、2、および 3 (スパースベクトルと行列に対する基本演算)
- 連立線形方程式を解くための LAPACK ルーチン
- 最小二乗問題、固有値ならびに特異値問題、およびシルベスター式を解くための LAPACK ルーチン
- 補助 LAPACK ルーチン
- ScaLAPACK の計算、ドライバー、および補助ルーチン (インテル MKL クラスター・エディションのみ)
- 直接法および反復法スパース・ソルバー・ルーチン
- ベクトル引数に対する基本的な数値演算関数を計算するための VML (Vector Mathematical Library) 関数 (Fortran および C のインターフェイスを使用)
- さまざまな種類の統計的分布に従った擬似乱数ベクトルを生成および畳み込みや相関計算を実行するベクトル統計ライブラリー (VSL) 関数
- Fortran および C のインターフェイスを備え、高速フーリエ変換 (FFT) アルゴリズムによって高速に DFT 計算を実行する一般的な離散フーリエ変換関数 (DFT)
- クラスター DFT 関数 (インテル MKL クラスター・エディションのみ)

- 実離散三角変換ルーチン
- 連立区間線形方程式を解くための区間ソルバールーチン
- 偏微分方程式を解くためのツール - 三角変換ルーチンとポワソンソルバー
- GMP 数学関数

ライブラリーの使用に関する詳細は、*MKL* リリースノートを参照のこと。

技術サポート

インテル MKL には、製品の特徴、ホワイトペーパー、技術記事など、製品に関する総合的な情報が適宜掲載される製品 Web サイトが用意されている。最新情報については、以下のサイトを参照のこと。

<http://www.intel.co.jp/jp/developer/software/products/>

インテルでは、基本操作のヒント、製品に関する確認済みの問題点、製品のエラッタ、ライセンス情報、ユーザーフォーラムなど、大量のセルフヘルプ情報を利用できるサポート Web サイトも提供している (<http://support.intel.co.jp/> を参照)。

ユーザー登録を行うと、インテル® プレミアサポートによる 1 年間の技術サポートと製品アップデートのサービスが受けられる。インテル・プレミア・サポートは、以下のサービスを提供する双方向型の問題管理 / コミュニケーション Web サイトである。

- 問題点の送信とその状態の検討
- 製品のアップデートのダウンロード (1 日 24 時間)

ユーザー登録、インテルへの問い合わせ、製品サポートについては、以下のサイトを参照のこと。

<http://www.intel.com/software/products/support> (英語)

BLAS ルーチン

BLAS のルーチンと関数は、実行する演算によって以下のグループに分類される。

- 「[BLAS レベル 1 のルーチンと関数](#)」は、ベクトルデータに対して加算と減算を実行する。典型的な演算として、スケーリングやドット積などがある。
- 「[BLAS レベル 2 のルーチン](#)」は、行列 - ベクトルの乗算、階数 1 と階数 2 の行列の更新、三角法の計算などの行列 - ベクトル演算を実行する。
- 「[BLAS レベル 3 のルーチン](#)」は、行列 - 行列の乗算、階数 k の更新、三角法の計算などの行列 - 行列演算を実行する。

インテル MKL 8.0 以降では、BLAS ルーチンに対する Fortran 95 インターフェイスもサポートしている。

スパース BLAS ルーチン

「[スパース BLAS レベル 1 のルーチンと関数](#)」と「[スパース BLAS レベル 2 およびレベル 3](#)」ルーチンおよび関数は、スパースベクトルと行列での演算を行う。これらのルーチンは、BLAS レベル 1、2、および 3 ルーチンと同じようなベクトル演算を実行する。スパース BLAS ルーチンは、ベクトルが疎であることを利用し、ベクトルの非零成分だけを格納できる。また、インテル MKL では、スパース BLAS ルーチンに対する Fortran 95 インターフェイスをサポートしている。

LAPACK ルーチン

インテル・マス・カーネル・ライブラリーは、LAPACK の計算、ドライバールーチン、補助ルーチン、およびユーティリティ・ルーチンをすべて含んでいる。

インテル MKL の一部の基となった LAPACK の原版は <http://www.netlib.org/lapack/index.html> から入手できる。LAPACK の開発は、E. Anderson、Z. Bai、C. Bischof、S. Blackford、J. Demmel、J. Dongarra、J. Du Croz、A. Greenbaum、S. Hammarling、A. McKenney、D. Sorensen らによって行われた。

LAPACK ルーチンは、実行する演算によって以下のグループに分類できる。

- 連立線形方程式を解くためのルーチンや、行列の因子分解ならびに逆行列の計算、条件数の推定などを実行するためのルーチン ([第 3 章](#) を参照)。
- 最小二乗問題、固有値ならびに特異値問題、およびシルベスター式を解くためのルーチン ([第 4 章](#) を参照)。
- 特定のサブタスク、共通のローレベル計算、または関連するタスクに使用される補助ルーチンおよびユーティリティ ([第 5 章](#) を参照)。

インテル MKL 8.0 以降では、LAPACK の計算およびドライバールーチンに対する Fortran 95 インターフェイスもサポートしている。このインターフェイスにより、必要な引数を少なくして LAPACK ルーチンの呼び出しを簡略化できる。

ScaLAPACK ルーチン

ScaLAPACK パッケージ (インテル MKL クラスター・エディションのみ。 [第 6 章](#) および [第 7 章](#) を参照) は、分散メモリー・アーキテクチャー上で動作し、連立線形方程式の解の算出、線形最小二乗問題、固有値、特異値問題、およびそれらに関連する各種の計算タスクを実行するためのルーチンを提供する。

インテル MKL クラスター・エディションの一部の基となった ScaLAPACK の原版は <http://www.netlib.org/scalapack/index.html> から入手できる。ScaLAPACK の開発は、L. Blackford、J. Choi、A. Cleary、E. D'Azevedo、J. Demmel、I. Dhillon、J. Dongarra、S. Hammarling、G. Henry、A. Petitet、K. Stanley、D. Walker、R. Whaley らによって行われた。

ScaLAPACK のインテル MKL クラスター・エディションのバージョンは、インテル・プロセッサ向けに最適化されており、MPICH とインテル[®] MPI を使用する。

スパース・ソルバー・ルーチン

インテル MKL の直接法スパース・ソルバー・ルーチン ([第 8 章](#) を参照) は、実数または複素数係数の対称および対称構造スパース行列を解く。対称行列において、これらのインテル MKL サブルーチンは、正定値式と不定値式の両方を計算できる。インテル MKL には、ユーザーからの呼び出しが可能な直接法スパース・ソルバー・ルーチンの代替セットだけでなく、PARDISO* スパース・ソルバー・インターフェイスも含まれる。

インテル MKL より PARDISO 直接法スパースソルバーを使用する場合、次を参照のこと。

O.Schenk、K.Gartner 著。『Solving unsymmetric sparse systems of linear equations with PARDISO』。J.of Future Generation Computer Systems、20(3):475- 487、2004 年。

また、スパース BLAS レベル 2 および 3 ルーチンを使用し、異なるスパースデータ形式で動作する反復法スパースソルバー ([第 8 章](#) を参照) も含まれる。

VML 関数

VML (Vector Mathematical Library) 関数 ([第 9 章](#) を参照) には、実数 / 複素数ベクトルを操作する、大量の計算を必要とする一連の基本的な数値演算関数 (累乗、三角関数、指数関数、双曲線関数など) の高度に最適化されたコードが含まれる。

VML により、非線形プログラミング・ソフトウェアや積分計算などの多くのプログラムの性能が大幅に向上すると期待できる。VML は FORTRAN と C 言語のどちらのインターフェイスも備えている。

統計関数

ベクトル統計ライブラリー (VSL) には、2 つの関数群のセットが含まれる ([第 10 章](#) を参照)。

- 1 つ目のセットは、基本的な連続分布または離散分布に対応した擬似乱数生成サブルーチン群で構成されている。VSL サブルーチンは高度に最適化された基本乱数生成器とベクトル数学関数ライブラリーを呼び出すことで最高レベルの性能を実現している。
- 2 つ目のセットは、広範囲にわたる畳み込みと相関演算に対応したサブルーチン群で構成されている。

フーリエ変換関数

インテル MKL 多次元離散フーリエ変換関数の混合基数サポート ([第 11 章](#) を参照) は、DFT 計算機能の統一性を提供するとともに、多機能性と使いやすさの両立を実現する。Fortran と C のインターフェイス仕様が含まれる。また、分散メモリー・アーキテクチャーで動作する DFT 関数のクラスターバージョンは、インテル MKL クラスタ・エディションで提供される。

DFT 関数は、2 の累乗だけでなく、基数 3、5、7、11 に対しても高速フーリエ変換 (FFT) アルゴリズムによって高速に計算を実行する。

区間ソルバールーチン

インテル MKL ([第 12 章](#) を参照) に含まれる区間ソルバールーチンは、連立区間線形方程式および関連する問題を解くために使用できる。

偏微分方程式のサポート

インテル MKL には、偏微分方程式 (PDE) を解くためのツール ([第 13 章](#) を参照) が用意されている。これらのツールは、三角変換インターフェイス・ルーチンとポワソン・ライブラリーである。

三角変換ルーチンは、ポワソン・ライブラリーのソルバーに似た独自のソルバーを実装するのに役立つ。三角変換インターフェイスに実装されている高速正弦変換、高速余弦変換、および高速スタガード余弦変換を使用して、独自ソルバーのパフォーマンスを向上させることができる。

ポアソン・ライブラリーは、単純なヘルムホルツ、ポアソン、およびラプラス問題を素早く解くように設計されている。ソルバーの基本となる三角変換インターフェイスは、インテル[®] プロセッサ向けに最適化されたインテル MKL DFT インターフェイス ([第 11 章](#) を参照) に基づいている。

GMP 数学関数

インテル MKL に実装されている GMP 数学関数には、任意精度の整数演算が含まれている。これらの関数のインターフェイスは、GMP (GNU Multiple Precision) 演算ライブラリーと互換性がある。これらの関数の詳細については、<http://www.swox.com/gmp/manual/Integer-Functions.html> (英語) を参照のこと。

性能の改善

インテル・マス・カーネル・ライブラリーは、プロセッサおよびシステム両方の機能と能力を利用することによって最適化される。これらのルーチンでは、その大半がキャッシュ管理テクノロジーのメリットを活かせるよう特に注意が払われている。そのメリットを最大限に活かせるのが、`dgemm()` などの行列 - 行列演算である。

また、整数演算ユニットと浮動小数点演算ユニットのスケジューリングがプロセッサ内の結果に依存するのを最小限に抑えるため、コード最適化技法が適用されている。

ライブラリーで使用される主な最適化技法には、次のものがある。

- ループの繰り返しを避けて、ループ管理コストを低減。
- データをブロック化して、データ再利用の可能性を改善。
- コピーにより、データがキャッシュから追い出される可能性を低減。
- データをプリフェッチして、メモリー・レイテンシーをカバー。
- `dgemm` でのドット積などの複数の演算を同時に実行して、演算ユニットのパイプラインが原因で生じるストールを排除。
- 必要に応じて、SIMD 算術演算ユニットなどのハードウェア機能を使用。

これらの技法により、演算コードにおいてメリットを最大限に活用できる。

並列化

すでに説明した性能改善に加え、インテル MKL では対称型マルチプロセッシング (SMP) 機能の採用により可能になった並列化で、さらに性能を向上させている。以下に示す方法で性能改善を図れる。

- プログラム内のユーザー管理スレッドを基本とし、さらにデータ分解、領域分解、制御分解、あるいはその他の並列技法に基づいて複数のスレッドに処理を分配する。ライブラリーはスレッドセーフとなるよう設計されているため、それぞれのスレッドで任意のインテル MKL 関数を使用できる。
- FFT ルーチンや BLAS レベル 3 ルーチンを使用する。これらのルーチンは並列化されているため、アプリケーションを変更しなくても多重処理による性能改善が得られる。BLAS レベル 3 で複数のプロセッサを使用した場合の性能は、プロセッサ数に比例して改善される。スレッドはライブラリー内で呼び出されて管理されるため、アプリケーションをスレッドセーフに再コンパイルする必要がない (第 2 章の「[Fortran-95 インターフェイス規則](#)」を参照) 。

- 性能改善に役立つもう 1 つの方法は、機能調整した *LAPACK* ルーチンの使用である。現時点では、これらのルーチンには、一般行列の *QR* 因子分解、一般行列と対称正定値行列の三角因子分解、このような行列を使用する連立方程式の解、対称固有値問題の解を得るための単精度と倍精度のルーチンが含まれている。

BLAS レベル 3 ルーチンや *LAPACK* ルーチンに対して使用可能なプロセッサ数を設定する方法については、*インテル MKL テクニカル・ユーザー・ノート* を参照のこと。

対応プラットフォーム

インテル・マス・カーネル・ライブラリー (*MKL*) には、多重処理をサポートするオペレーティング・システム上で稼働するインテル・プロセッサ・ベースのコンピュータ用に最適化された *Fortran* のルーチンと関数を含んでいる。また、*インテル MKL* には、*Fortran* インターフェイスに加え、ベクトル数値演算ライブラリー関数とベクトル統計ライブラリー関数だけでなく、離散フーリエ変換関数に対応する C 言語インターフェイスも含まれている。

インテル MKL を使用する際のハードウェアおよびソフトウェアの動作環境に関する詳細は、*MKL リリースノート* を参照のこと。

本書について

本書では、*インテル MKL* と *インテル MKL クラスタ・エディション* のルーチンおよび関数について説明する。

リファレンスの各セクションでは、一般的に 4 つの基本的なデータ型 (単精度実数、倍精度実数、単精度複素数、倍精度複素数) で使用するルーチンを 1 つのルーチングループにまとめて説明する。

それぞれのルーチングループの説明では、名前と機能の簡単な説明とともに、グループ内の各ルーチンで使用するデータ型それぞれについての呼び出しシーケンスまたは構文を紹介する。また、以下のセクションも含まれる。

説明	1 つ以上の式を例に挙げて、グループ内の各ルーチンが実行する機能を説明する。引数のデータ型は、グループ全体に共通する一般項で定義する。
入力パラメーター	入力パラメーターそれぞれについてデータ型を定義する。例えば、 a REAL (saxpy の場合) DOUBLE PRECISION (daxpy の場合)
出力パラメーター	終了時に結果として得られるパラメーターを列挙する。

本書の対象読者

本書では、数値演算に精通し、線形代数、数理統計学、およびフーリエ変換の原理や用語についての知識があるプログラマーを対象にしている。

本書の構成

本書は、以下の各章と付録で構成される。

- 第 1 章 「[概要](#)」。インテル・マス・カーネル・ライブラリー・ソフトウェアを紹介するとともに、マニュアルの構成や表記上の規則について説明する。
- 第 2 章 「[BLAS ルーチンと スパース BLAS ルーチン](#)」。BLAS ならびにスパース BLAS の関数とルーチンを説明する。
- 第 3 章 「[LAPACK ルーチン：線形方程式](#)」。連立線形方程式の解の算出と、これに関連する多数の計算タスク（三角因子分解、逆行列の計算、行列の条件数の推定など）を実行するための LAPACK ルーチンについて説明する。
- 第 4 章 「[LAPACK ルーチン：最小二乗問題および 固有値問題](#)」。最小二乗問題、標準ならびに一般化された固有値問題、特異値問題、およびシルベスター式を解くための LAPACK ルーチンを説明する。
- 第 5 章 「[LAPACK 補助ルーチンと ユーティリティー・ルーチン](#)」。特定のサブタスク、共通のローレベル計算を実行する、補助およびユーティリティー LAPACK ルーチンについて説明する。
- 第 6 章 「[ScaLAPACK ルーチン](#)」。ScaLAPACK の計算、ドライバールーチン（インテル MKL クラスタ・エディションのみ）について説明する。
- 第 7 章 「[ScaLAPACK 補助ルーチンと ユーティリティー・ルーチン](#)」。ScaLAPACK の補助ルーチン（インテル MKL クラスタ・エディションのみ）について説明する。
- 第 8 章 「[スパース・ソルバー・ルーチン](#)」。対称および対称構造スパース行列を計算する直接法スパース・ソルバー・ルーチンについて説明する。また、反復法スパース・ソルバー・ルーチンについても説明する。
- 第 9 章 「[ベクトル数学関数](#)」。ベクトル引数に対する基本的な数値演算関数を計算するための VML 関数について説明する。
- 第 10 章 「[統計関数](#)」。擬似乱数ベクトルを生成し、畳み込みや相関計算を実行する VSL 関数について説明する。
- 第 11 章 「[フーリエ変換関数](#)」。離散フーリエ変換を計算する多次元関数およびクラスタ DFT 関数（インテル MKL クラスタ・エディションのみ）について説明する。
- 第 12 章 「[区間線形ソルバー](#)」。連立区間線形方程式および関連する問題を解くために使用できるルーチンについて説明する。
- 第 13 章 「[偏微分方程式のサポート](#)」。偏微分方程式 (PDE) を解くための三角変換インターフェイス・ルーチンおよびポワソン・ライブラリーについて説明する。
- 付録 A 「[線形ソルバーの基礎](#)」。連立線形方程式を解くための線形代数での基本定義とアプローチについて簡単に説明する。また、区間演算の基本概念およびスパースデータ格納形式についても説明する。
- 付録 B 「[ルーチンおよび関数の 引数](#)」。BLAS ルーチンと VML 関数の主要な引数（ベクトル引数と行列引数）について説明する。
- 付録 C 「[コード例](#)」。さまざまな MKL 関数やルーチンを呼び出す場合のコード例を掲載する (BLAS、PARDISO、直接法および反復法スパースソルバー、DFT、クラスタ DFT、区間線形ソルバー、三角変換、ポワソン・ライブラリー)。

- 付録 D 「[BLAS に対する CBLAS インターフェイス](#)」。BLAS に対する C インターフェイスを掲載する。
- 付録 E 「[LAPACK ルーチン用 Fortran-95 インターフェイス特有の機能](#)」。インテル MKL に含まれている LAPACK 計算ルーチンと Netlib の LAPACK 計算ルーチンを比較する。
- 本書には、「[参考文献](#)」、「[用語集](#)」、および「[索引](#)」も含まれる。

表記の規則

本書では、以下の表記上の規則を使用する。

- ルーチン名の省略表記 (cungqr/zungqr の代わりに ?ungqr と表記)。
- 本文とコードを区別するためのフォント表記規則。

ルーチン名の省略表記

省略表記を行うため、特定のルーチングループの名前のキャラクター・コードは疑問符 (?) で表している。この疑問符は、特定の関数についての各データ型用の別形を表している。次に例を示す。

?swap ベクトル - ベクトルの ?swap ルーチンの 4 つのデータ型すべて、つまり sswap、dswap、cswap、zswap を表す。

字体の規則

次の字体の表記規則を使用する。

UPPERCASE COURIER	Fortran インターフェイスの入力パラメーターと出力パラメーターの説明で使用されるデータ型。 例えば、CHARACTER*1
lowercase courier	コード例： a(k+i,j) = matrix(i,j) および C インターフェイスのデータ型。 例えば、const float*。
lowercase courier mixed with UpperCase courier	C インターフェイスの関数名。 例えば、vmlSetMode。
lowercase courier italic	引数やパラメーターの説明における変数。 例えば、incx。
*	コード例や式で乗算記号として使用。また、Fortran の構文で必要な箇所に使用。

BLAS ルーチンと スパース BLAS ルーチン

2

本章では、インテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) の BLAS ルーチンとスパース BLAS ルーチンについて説明する。ルーチンの説明は、BLAS の演算レベルによって以下のセクションに分かれている。

- 「[BLAS レベル 1 のルーチンと関数](#)」 (ベクトル - ベクトル演算)
- 「[BLAS レベル 2 のルーチン](#)」 (行列 - ベクトル演算)
- 「[BLAS レベル 3 のルーチン](#)」 (行列 - 行列演算)
- 「[スパース BLAS レベル 1 のルーチンと関数](#)」 (ベクトル - ベクトル演算)
- 「[スパース BLAS レベル 2 およびレベル 3](#)」 (行列 - ベクトル演算および行列 - 行列演算)。

各セクションでは、ルーチンと関数のグループを、例えば、`?asum` グループや `?axpy` グループなどのように、アルファベット順に説明する。グループ名にある疑問符は、データ型を示す各種のキャラクター・コード (s、d、c、z、あるいはそれらの組み合わせ) に対応している。次のページの「[ルーチン命名規則](#)」を参照のこと。

BLAS ルーチンまたは Sparse BLAS ルーチンでエラーが発生した場合は、エラー報告ルーチンの [xerbla](#) が呼び出される。エラー報告を見るためには、コード内に `xerbla` を組み込む必要がある。`xerbla` 用のソースコードのコピーが、ライブラリーに含まれている。

BLAS レベル 1 のグループ `i?amax` と `i?amin` には、キャラクター・コードの前に「i」が付いており、ベクトル成分のインデックスに対応している。これらのグループは、BLAS レベル 1 のセクションの最後に収録している。

BLAS のルーチンと関数

ルーチン命名規則

BLAS ルーチンの名前は、次の構造になっている。

`<character code> <name> <mod> ()`

`<character code>` は、データ型を示すキャラクター・コードである。

s	実数、単精度
c	複素数、単精度
d	実数、倍精度
z	複素数、倍精度

一部のルーチンや関数では、`sc` や `dz` などの組み合わせられたキャラクター・コードを持つことができる。例えば、関数 `scasum` は、入力として複素配列を使用し、出力として実数値を返す。

`<name>` フィールドは、BLAS レベル 1 では、演算のタイプを示す。例えば、BLAS レベル 1 ルーチンの `?dot`、`?rot`、`?swap` は、それぞれベクトルのドット積、ベクトルの回転、ベクトルの交換を計算する。

BLAS レベル 2 と 3 では、`<name>` は行列の引数のタイプを表す。

ge	一般行列
gb	一般帯行列
sy	対称行列
sp	対称行列 (圧縮格納形式)
sb	対称帯行列
he	エルミート行列
hp	エルミート行列 (圧縮格納形式)
hb	エルミート帯行列
tr	三角行列
tp	三角行列 (圧縮格納形式)
tb	三角帯行列

`<mod>` フィールド (存在する場合) では、演算をさらに詳しく指示する。

BLAS レベル 1 のルーチン名の `<mod>` フィールドには、次のキャラクターが入る。

c	共役ベクトル
u	非共役ベクトル
g	Givens 回転

BLAS レベル 2 のルーチン名の `<mod>` フィールドには、次のキャラクターが入る。

mv	行列 - ベクトルの積
sv	行列 - ベクトル演算による連立 1 次方程式の解
r	階数 1 の行列の更新
r2	階数 2 の行列の更新

BLAS レベル 3 のルーチン名の `<mod>` フィールドには、次のキャラクターが入る。

mm	行列 - 行列の積
sm	行列 - 行列演算による連立 1 次方程式の解
rk	階数 k の行列の更新
r2k	階数 $2k$ の行列の更新

以下に、BLAS ルーチン名を解釈する方法の例を示す。

ddot	<d> <dot>: 倍精度実数のベクトル - ベクトルのドット積
cdotc	<c> <dot> <c>: 単精度複素共役のベクトル - ベクトルのドット積
scasum	<sc> <asum>: 単精度複素数を入力とし、単精度実数を出力とするベクトルの成分の大きさの合計
cdotu	<c> <dot> <u>: 単精度複素非共役のベクトル - ベクトルのドット積
sgemv	<s> <ge> <mv>: 単精度一般行列の行列 - ベクトルの積
ztrmm	<z> <tr> <mm>: 倍精度複素三角行列の行列 - 行列の積

スパース BLAS における命名規則は、BLAS レベル 1 の規則と似ている。詳細は、「[命名規則](#)」を参照。

Fortran-95 インターフェイス規則

BLAS および Sparse BLAS レベル 1 ルーチンに対する Fortran-95 インターフェイスは、それぞれの Fortran-77 ルーチン呼び出しラッパーを通して実装される。このインターフェイスは、形状引継ぎ配列や、より少ない引数で簡略化した BLAS および Sparse BLAS レベル 1 ルーチンの呼び出しを提供するオプション引数などの Fortran-95 の機能を使用する。

Fortran-95 インターフェイスで使用される主な規則を以下に示す。

- Fortran-95 呼び出しで使用される引数名は、一般的にそれぞれの標準 (Fortran-77) インターフェイスと同じである。ただし、ライブラリーで使用される引数名の数を減らすための、簡略化された引数名を以下に示す。

標準引数名	Fortran-95 引数名
<i>ap</i>	<i>a</i>

これらの正式引数名の変更はプログラムの動作には影響を与えず、また統一命名規則に従うものである。

- 配列次元などの入力引数は Fortran-95 では不要であり、呼び出しシーケンスからスキップされる。配列次元は、必要な配列形状に正確に従うユーザーデータから再構築される。また、引数は、呼び出しシーケンス内の他の引数の有無により、その値が完全に定義される場合にスキップされる。復元値はスキップされた引数にとってのみ意味のある値である。
- 引数 *incx* および *incy* はスキップされる。あらゆる場合において、これらの引数の値は 1 であるとみなされる。*incx* および *incy* に 1 以外の値を設定するには、対応する Fortran-95 の機能を使用して、インデックスの増分を実引数で直接指定する。Fortran-77 の呼び出しを使用した場合も、同様の効果が得られる。
- いくつかの標準引数は Fortran-95 インターフェイスではオプションとして宣言され、呼び出しシーケンスから省略される場合がある。以下の条件のいずれかを満たす場合は、引数をオプションとして宣言できる。

1. 入力引数が、ごくわずかの設定可能な値を持つ場合、オプションとして宣言できる。これらの引数のデフォルト値は、通常、リストの最初の値として設定される。この規則における例外のすべてはルーチンの説明で明示的に述べる。
 2. 入力引数が自然デフォルト値を持つ場合は、オプションとして宣言できる。これらのオプション引数のデフォルト値は、自然デフォルト値に設定される。
- Fortran-95 呼び出し構文では、オプション引数を大括弧 ([]) で表現する。

オプション・パラメーターの値の再構築に使用される具体的な規定は、各ルーチンごとに特有で、ルーチン仕様の最後の各「Fortran-95 ノート」で詳細を説明する。各ルーチンに特有な規定が省略されると、指定されたルーチンの Fortran-95 インターフェイスは、対応する Fortran-77 インターフェイスと変わらない。

このインターフェイスは、現バージョンのスパース BLAS レベル 2 およびレベル 3 のルーチンでは実装されていない点に注意する。これらのルーチンに対する Fortran-95 インターフェイスは、ルーチン仕様の最後の各「インターフェイス - Fortran-95」に収録されている。

行列の格納形式

BLAS ルーチンの行列引数では、次の格納形式を使用できる。

- フル格納では、行列 A は 2 次元配列 a に格納され、行列成分 a_{ij} は配列成分 $a(i, j)$ に格納される。
- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が 1 次元配列の各列に圧縮される。
- 帯格納では、帯行列が 2 次元配列にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納する。

行列の格納形式の詳細は、付録 B の「[行列引数](#)」を参照。

BLAS レベル 1 のルーチンと関数

BLAS レベル 1 には、ベクトル - ベクトル演算を実行するルーチンと関数が含まれる。表 2-1 に、BLAS レベル 1 のルーチンと関数のグループと、それらに関連するデータ型を示す。

表 2-1 BLAS レベル 1 のルーチングループおよびそのデータ型

ルーチンまたは関数のグループ	データ型	説明
?asum	s, d, sc, dz	ベクトルの大きさの合計 (関数)
?axpy	s, d, c, z	スカラー - ベクトルの積 (ルーチン)
?copy	s, d, c, z	ベクトルのコピー (ルーチン)
?dot	s, d	ドット積 (関数)
?sdot	sd, d	精度を拡張したドット積 (関数)
?dotc	c, z	共役のドット積 (関数)
?dotu	c, z	非共役のドット積 (関数)
?nrm2	s, d, sc, dz	正規ベクトルまたはヌルベクトルのベクトル 2-ノルム (ユークリッド・ノルム) (関数)
?rot	s, d, cs, zd	点の面回転 (ルーチン)

表 2-1 BLAS レベル 1 のルーチングループおよびそのデータ型

ルーチンまたは関数のグループ	データ型	説明
?rotg	s, d, c, z	点の Givens 回転 (ルーチン)
?rotm	s, d	点の変形面回転
?rotmg	s, d	点の Givens 変形面回転
?scal	s, d, c, z, cs, zd	ベクトルのスケーリング (ルーチン)
?swap	s, d, c, z	ベクトル - ベクトルの交換 (ルーチン)
i?amax	s, d, c, z	ベクトルの最大値、すなわちベクトルの絶対最大成分 (関数)。 <i>i</i> は、ベクトル配列内のこの値に対するインデックス
i?amin	s, d, c, z	ベクトルの最小値、すなわちベクトルの絶対最小成分 (関数)。 <i>i</i> は、ベクトル配列内のこの値に対するインデックス
dcabs1	d	倍精度複素数 <i>z</i> の絶対値

?asum

ベクトル成分の大きさの合計を計算する。

構文

Fortran 77:

```
res = sasum( n, x, incx )
res = scasum( n, x, incx )
res = dasum( n, x, incx )
res = dzasum( n, x, incx )
```

Fortran 95:

```
res = asum(x)
```

説明

?asum 関数は、ベクトル *x* に対してその成分の大きさの合計を計算する。複素ベクトルの場合は、成分の実数部の大きさに虚数部の大きさを加算した合計を計算する。

$$res = |Re(x(1))| + |Im(x(1))| + |Re(x(2))| + |Im(x(2))| + \dots + |Re(x(n))| + |Im(x(n))|$$

x は次数 *n* のベクトルである。

入力パラメーター

n INTEGER。ベクトル *x* の次数を指定する。

x REAL (sasum の場合)
 DOUBLE PRECISION (dasum の場合)
 COMPLEX (scasum の場合)
 DOUBLE COMPLEX (dzasum の場合)

配列、次元は $(1 + (n-1)*abs(incx))$ 以上。

incx INTEGER。 *x* の成分に対する増分を指定する。

出力パラメーター

res REAL (*sasum* の場合)
 DOUBLE PRECISION (*dasum* の場合)
 REAL (*scasum* の場合)
 DOUBLE PRECISION (*dzasum* の場合)

すべての成分の実数部の大きさに虚数部の大きさを加算した合計が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *asum* のインターフェイスの詳細を以下に示す。

x サイズ (*n*) の配列を格納する。

?axpy

ベクトル・スカラー積を計算し、その結果をベクトルに加える。

構文

Fortran 77:

```
call saxpy( n, a, x, incx, y, incy )
call daxpy( n, a, x, incx, y, incy )
call caxpy( n, a, x, incx, y, incy )
call zaxpy( n, a, x, incx, y, incy )
```

Fortran 95:

```
call axpy(x, y [,a])
```

説明

?axpy ルーチンは、次のように定義されるベクトル・ベクトル演算を実行する。

$$y := a*x + y$$

a はスカラーである。

x と *y* は次数 *n* のベクトルである。

入力パラメーター

n INTEGER。ベクトル *x* と *y* の次数を指定する。

<i>a</i>	REAL (<i>saxpy</i> の場合) DOUBLE PRECISION (<i>daxpy</i> の場合) COMPLEX (<i>caxpy</i> の場合) DOUBLE COMPLEX (<i>zaxpy</i> の場合) スカラー <i>a</i> を指定する。
<i>x</i>	REAL (<i>saxpy</i> の場合) DOUBLE PRECISION (<i>daxpy</i> の場合) COMPLEX (<i>caxpy</i> の場合) DOUBLE COMPLEX (<i>zaxpy</i> の場合) 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。
<i>y</i>	REAL (<i>saxpy</i> の場合) DOUBLE PRECISION (<i>daxpy</i> の場合) COMPLEX (<i>caxpy</i> の場合) DOUBLE COMPLEX (<i>zaxpy</i> の場合) 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。

出力パラメーター

y 更新されたベクトル *y* が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *axpy* のインターフェイスの詳細を以下に示す。

<i>x</i>	サイズ (<i>n</i>) の配列を格納する。
<i>y</i>	サイズ (<i>n</i>) の配列を格納する。
<i>a</i>	デフォルト値は '1' である。

?copy

ベクトルを別のベクトルにコピーする。

構文

Fortran 77:

```
call scopy( n, x, incx, y, incy )
call dcopy( n, x, incx, y, incy )
call ccopy( n, x, incx, y, incy )
call zcopy( n, x, incx, y, incy )
```

Fortran 95:

```
call copy(x, y)
```

説明

?copy ルーチンは、次のように定義されるベクトル-ベクトル演算を実行する。

$$y = x$$

x と y はベクトルである。

入力パラメーター

n	INTEGER。ベクトル x と y の次数を指定する。
x	REAL (scopy の場合) DOUBLE PRECISION (dcopy の場合) COMPLEX (ccopy の場合) DOUBLE COMPLEX (zcopy の場合) 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 x の成分に対する増分を指定する。
y	REAL (scopy の場合) DOUBLE PRECISION (dcopy の場合) COMPLEX (ccopy の場合) DOUBLE COMPLEX (zcopy の場合) 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。
$incy$	INTEGER。 y の成分に対する増分を指定する。

出力パラメーター

y	n が正の場合は、ベクトル x のコピーが格納される。そうでない場合には、パラメーターは変更されない。
-----	---

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン copy のインターフェイスの詳細を以下に示す。

x	長さ (n) のベクトルを格納する。
y	長さ (n) のベクトルを格納する。

?dot

ベクトル-ベクトルのドット積を計算する。

構文

Fortran 77:

```
res = sdot( n, x, incx, y, incy )
```

```
res = ddot( n, x, incx, y, incy )
```

Fortran 95:

```
res = dot(x, y)
```

説明

?dot 関数は、次のように定義されるベクトル-ベクトルの縮退演算を実行する。

$$res = \sum (x*y)$$

x と y はベクトルである。

入力パラメーター

n	INTEGER。ベクトル x と y の次数を指定する。
x	REAL (sdot の場合) DOUBLE PRECISION (ddot の場合) 配列、次元は $(1+(n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 x の成分に対する増分を指定する。
y	REAL (sdot の場合) DOUBLE PRECISION (ddot の場合) 配列、次元は $(1+(n-1)*abs(incy))$ 以上。
$incy$	INTEGER。 y の成分に対する増分を指定する。

出力パラメーター

res	REAL (sdot の場合) DOUBLE PRECISION (ddot の場合) n が正の場合は、 x と y のドット積の結果が格納される。そうでない場合には、 res には 0 が格納される。
-------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン dot のインターフェイスの詳細を以下に示す。

x 長さ (n) のベクトルを格納する。

y 長さ (n) のベクトルを格納する。

?sdot

ベクトル-ベクトルのドット積を拡張精度で計算する。

構文

Fortran 77:

```
res = sdsdot( n, sb, sx, incx, sy, incy )
res = dsdot( n, sx, incx, sy, incy )
```

Fortran 95:

```
res = sdot(sx, sy)
res = sdot(sx, sy, sb)
```

説明

?sdot 関数は 2 つのベクトルの内積を拡張精度で計算する。どちらの関数も中間結果は拡張精度で蓄積されるが、sdsdot 関数は最終結果を単精度で出力し、dsdot は倍精度で出力する。また、sdsdot 関数では内積にスカラー値 sb が加算される。

入力パラメーター

N INTEGER。入力ベクトル sx と sy 内の成分の数を指定する。
 sb REAL。内積に加算する単精度スカラー (sdsdot 関数のみ)。
 sx, sy REAL。配列、次元はそれぞれ $(1+(n-1)*abs(incx))$ 、 $(1+(n-1)*abs(incy))$ 以上。入力単精度ベクトルが格納される。
 $incx$ INTEGER。 sx の成分に対する増分を指定する。
 $incy$ INTEGER。 sy の成分に対する増分を指定する。

出力パラメーター

res REAL (sdsdot の場合)
 DOUBLE PRECISION (dsdot の場合)
 n が正の場合は sx と sy の内積の結果が格納される (sdsdot では sb が加算されている)。N が正でない場合は、sdsdot では sb 、dsdot では 0 が res に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sdot のインターフェイスの詳細を以下に示す。

sx 長さ (*n*) のベクトルを格納する。
sy 長さ (*n*) のベクトルを格納する。



注: スカラー・パラメーター *sb* は、異なる精度の最終結果を出力する関数を区別するため、関数 *sdot* に対する Fortran-95 インターフェイスで、必須パラメーターとして定義される。

?dotc

共役ベクトルと別のベクトルのドット積を計算する。

構文

Fortran 77:

```
res = cdotc( n, x, incx, y, incy )
res = zdotc( n, x, incx, y, incy )
```

Fortran 95:

```
res = dotc(x, y)
```

説明

?dotc 関数は、次のように定義されるベクトル - ベクトル演算を実行する。

$$res = \sum (\text{conjg}(x) * y)$$

x と *y* は *n* 個の成分を持つベクトルである。

入力パラメーター

n INTEGER。ベクトル *x* と *y* の次数を指定する。
x COMPLEX (cdotc の場合)
 DOUBLE COMPLEX (zdotc の場合)
 配列、次元は $(1 + (n-1) * \text{abs}(incx))$ 以上。
incx INTEGER。 *x* の成分に対する増分を指定する。
y COMPLEX (cdotc の場合)
 DOUBLE COMPLEX (zdotc の場合)
 配列、次元は $(1 + (n-1) * \text{abs}(incy))$ 以上。
incy INTEGER。 *y* の成分に対する増分を指定する。

出力パラメーター

res COMPLEX (cdotc の場合)
 DOUBLE COMPLEX (zdotc の場合)

n が正の場合は、共役の x と非共役の y のドット積の結果が格納される。そうでない場合には、 res には 0 が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `dotc` のインターフェイスの詳細を以下に示す。

x 長さ (n) のベクトルを格納する。
 y 長さ (n) のベクトルを格納する。

?dotu

ベクトル-ベクトルのドット積を計算する。

構文

Fortran 77:

```
res = cdotu( n, x, incx, y, incy )
res = zdotu( n, x, incx, y, incy )
```

Fortran 95:

```
res = dotu(x, y)
```

説明

?DOTU 関数は、次のように定義されるベクトル-ベクトルの縮退演算を実行する。

$$res = \sum (x*y)$$

x と y は n 個の成分を持つベクトルである。

入力パラメーター

n	INTEGER。ベクトル x と y の次数を指定する。
x	COMPLEX (cdotu の場合) DOUBLE COMPLEX (zdotu の場合) 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 x の成分に対する増分を指定する。
y	COMPLEX (cdotu の場合) DOUBLE COMPLEX (zdotu の場合) 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。
$incy$	INTEGER。 y の成分に対する増分を指定する。

出力パラメーター

res COMPLEX (cdotu の場合)
 DOUBLE COMPLEX (zdotu の場合)

n が正の場合は、*x* と *y* のドット積の結果が格納される。そうでない場合には、*res* には 0 が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン dotu のインターフェイスの詳細を以下に示す。

x 長さ (*n*) のベクトルを格納する。
y 長さ (*n*) のベクトルを格納する。

?nrm2

ベクトルのユークリッド・ノルムを計算する。

構文

Fortran 77:

```
res = snrm2( n, x, incx )
res = dnrnm2( n, x, incx )
res = scnrm2( n, x, incx )
res = dznrm2( n, x, incx )
```

Fortran 95:

```
res = nrm2(x)
```

説明

?nrm2 関数は、次のように定義されるベクトルの縮退演算を実行する。

```
res = ||x||
```

x はベクトルである。

res は *x* の成分のユークリッド・ノルムが格納される値である。

入力パラメーター

n INTEGER。ベクトル *x* の次数を指定する。
x REAL (snrm2 の場合)
 DOUBLE PRECISION (dnrm2 の場合)
 COMPLEX (scnrm2 の場合)
 DOUBLE COMPLEX (dznrm2 の場合)

配列、次元は $(1 + (n-1) * \text{abs}(incx))$ 以上。

incx INTEGER。 *x* の成分に対する増分を指定する。

出力パラメーター

res REAL (snrm2 の場合)
 DOUBLE PRECISION (dnrm2 の場合)
 REAL (scnrm2 の場合)
 DOUBLE PRECISION (dznrm2 の場合)
 ベクトル *x* のユークリッド・ノルムが格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン nrm2 のインターフェイスの詳細を以下に示す。

x 長さ (*n*) のベクトルを格納する。

?rot

面における点の回転を実行する。

構文

Fortran 77:

```
call srot( n, x, incx, y, incy, c, s )
call drot( n, x, incx, y, incy, c, s )
call csrot( n, x, incx, y, incy, c, s )
call zdrot( n, x, incx, y, incy, c, s )
```

Fortran 95:

```
call rot(x, y [,c] [,s])
```

説明

2つの複素ベクトル *x* と *y* が与えられたときに、これらのベクトルの各ベクトル成分が次のように置き換えられる。

$$x(i) = c * x(i) + s * y(i)$$

$$y(i) = c * y(i) - s * x(i)$$

入力パラメーター

n INTEGER。ベクトル *x* と *y* の次数を指定する。

x	REAL (srot の場合) DOUBLE PRECISION (drot の場合) COMPLEX (csrot の場合) DOUBLE COMPLEX (zdrot の場合) 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 x の成分に対する増分を指定する。
y	REAL (srot の場合) DOUBLE PRECISION (drot の場合) COMPLEX (csrot の場合) DOUBLE COMPLEX (zdrot の場合) 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。
$incy$	INTEGER。 y の成分に対する増分を指定する。
c	REAL (srot の場合) DOUBLE PRECISION (drot の場合) REAL (csrot の場合) DOUBLE PRECISION (zdrot の場合) スカラー。
s	REAL (srot の場合) DOUBLE PRECISION (drot の場合) REAL (csrot の場合) DOUBLE PRECISION (zdrot の場合) スカラー。

出力パラメーター

x	各成分は、 $c*x + s*y$ で置き換えられる。
y	各成分は、 $c*y - s*x$ で置き換えられる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン rot のインターフェイスの詳細を以下に示す。

x	長さ (n) のベクトルを格納する。
y	長さ (n) のベクトルを格納する。
c	デフォルト値は '1' である。
s	デフォルト値は '1' である。

?rotg

Givens 回転に対するパラメーターを計算する。

構文

Fortran 77:

```
call srotg( a, b, c, s )
call drotg( a, b, c, s )
call crotg( a, b, c, s )
call zrotg( a, b, c, s )
```

Fortran 95:

```
call rotg(a, b, c, s)
```

説明

点 p の直交座標 (a, b) が与えられたときに、これらのルーチンはその点の y 座標をゼロにする *Givens* 回転に関連付けられるパラメーター a, b, c, s を返す。

より精度の高い LAPACK バージョンの [?lartg](#) を参照。

入力パラメーター

- a REAL (srotg の場合)
DOUBLE PRECISION (drotg の場合)
COMPLEX (crotg の場合)
DOUBLE COMPLEX (zrotg の場合)
点 p の x 座標を与える。
- b REAL (srotg の場合)
DOUBLE PRECISION (drotg の場合)
COMPLEX (crotg の場合)
DOUBLE COMPLEX (zrotg の場合)
点 p の y 座標を与える。

出力パラメーター

- a *Givens* 回転に関連付けられたパラメーター r が格納される。
- b *Givens* 回転に関連付けられたパラメーター z が格納される。
- c REAL (srotg の場合)
DOUBLE PRECISION (drotg の場合)
REAL (crotg の場合)
DOUBLE PRECISION (zrotg の場合)
Givens 回転に関連付けられたパラメーター c が格納される。

s REAL (srotg の場合)
 DOUBLE PRECISION (drotg の場合)
 COMPLEX (crotg の場合)
 DOUBLE COMPLEX (zrotg の場合)

Givens 回転に関連付けられたパラメーター *s* が格納される。

?rotm

変形面における点の回転を実行する。

構文

Fortran 77:

```
call srotm( n, x, incx, y, incy, param )
call drotm( n, x, incx, y, incy, param )
```

Fortran 95:

```
call rotm(x, y [,param])
```

説明

2つの複素ベクトル *x* と *y* が与えられたときに、これらのベクトルの各ベクトル成分が次のように置き換えられる。

$$x(i) = H*x(i) + H*y(i)$$

$$y(i) = H*y(i) - H*x(i)$$

H は変形 Givens 変換行列で、その値は *param*(2) から *param*(5) の配列に格納される。
param 引数の説明を参照のこと。

入力パラメーター

n INTEGER。ベクトル *x* と *y* の次数を指定する。

x REAL (srotm の場合)
 DOUBLE PRECISION (drotm の場合)
 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。

incx INTEGER。 *x* の成分に対する増分を指定する。

y REAL (srotm の場合)
 DOUBLE PRECISION (drotm の場合)
 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。

incy INTEGER。 *y* の成分に対する増分を指定する。

param REAL (srotm の場合)
 DOUBLE PRECISION (drotm の場合)
 配列、次元は 5。
param 配列の成分は次のようになる。

$param(1)$ にはスイッチ、 $flag$ が格納される。

$param(2-5)$ にはそれぞれ、配列 H の成分である $h11$ 、 $h21$ 、 $h12$ 、 $h22$ が格納される。

$flag$ の値により、 H の成分は次のとおりに設定される。

$$flag = -1.: H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0.: H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1.: H = \begin{bmatrix} h11 & 1. \\ -1. & h22 \end{bmatrix}$$

$$flag = -2.: H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

上記で行列の成分が 1.、-1.、0. の場合には、 $flag$ の最後の 3 つの値に基づくものとみなされ、実際には $param$ ベクトルにロードされない。

出力パラメーター

x	各成分は、 $h11*x + h12*y$ で置き換えられる。
y	各成分は、 $h21*x + h22*y$ で置き換えられる。
H	更新された Givens 変換行列。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `rotm` のインターフェイスの詳細を以下に示す。

x	長さ (n) のベクトルを格納する。
y	長さ (n) のベクトルを格納する。
$param$	$param(1)$ のデフォルト値は -2 である。

?rotmg

Givens 回転に対する変形パラメータを計算する。

構文

Fortran 77:

```
call srotmg( d1, d2, x1, y1, param )
call drotmg( d1, d2, x1, y1, param )
```

Fortran 95:

```
call rotmg(x1, y1, param [,d1] [d2])
```

説明

入力ベクトルの直交座標 (x1, y1) が与えられたときに、これらのルーチンは、結果として得られるベクトルの y 成分をゼロにする変形 Givens 変換行列 H の成分を計算する。

$$\begin{bmatrix} x \\ 0 \end{bmatrix} = H \begin{bmatrix} x1 \\ y1 \end{bmatrix}$$

入力パラメーター

d1	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトル (sqrt(d1)x1) の x 座標に対する更新されたスケール係数を与える。
d2	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトル (sqrt(d2)y1) の y 座標に対する更新されたスケール係数を与える。
x1	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトルの回転された x 座標を与える。
y1	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 入力ベクトルの y 座標を与える。

出力パラメーター

param	REAL (srotmg の場合) DOUBLE PRECISION (drotmg の場合) 配列、次元は 5。 param 配列の成分は次のようになる。
-------	---

$param(1)$ にはスイッチ、 $flag$ が格納される。
 $param(2-5)$ にはそれぞれ、配列 H の成分である $h11$ 、 $h21$ 、 $h12$ 、 $h22$ が格納される。

$flag$ の値により、 H の成分は次のとおりに設定される。

$$flag = -1.: H = \begin{bmatrix} h11 & h12 \\ h21 & h22 \end{bmatrix}$$

$$flag = 0.: H = \begin{bmatrix} 1. & h12 \\ h21 & 1. \end{bmatrix}$$

$$flag = 1.: H = \begin{bmatrix} h11 & 1. \\ -1. & h22 \end{bmatrix}$$

$$flag = -2.: H = \begin{bmatrix} 1. & 0. \\ 0. & 1. \end{bmatrix}$$

上記で行列の成分が 1、-1、0 の場合には、 $flag$ の最後の 3 つの値に基づくものとみなされ、実際には $param$ ベクトルにロードされない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `rotmg` のインターフェイスの詳細を以下に示す。

$d1$ デフォルト値は '1' である。

$d2$ デフォルト値は '1' である。

?scal

ベクトルとスカラーの積を計算する。

構文

Fortran 77:

```
call sscal( n, a, x, incx )
call dscal( n, a, x, incx )
call cscal( n, a, x, incx )
call zscal( n, a, x, incx )
call csscal( n, a, x, incx )
call zdscal( n, a, x, incx )
```

Fortran 95:

```
call scal(x, a)
```

説明

?scal ルーチンは、次のように定義されるベクトル-ベクトル演算を実行する。

$$x = a * x$$

a はスカラー、 x は n 個の成分を持つベクトルである。

入力パラメーター

n	INTEGER。ベクトル x の次数を指定する。
a	REAL (sscal、csscal の場合) DOUBLE PRECISION (dscal、zdscal の場合) COMPLEX (cscal の場合) DOUBLE COMPLEX (zscal の場合) スカラー a を指定する。
x	REAL (sscal の場合) DOUBLE PRECISION (dscal の場合) COMPLEX (cscal、csscal の場合) DOUBLE COMPLEX (zscal、zdscal の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incx}))$ 以上。
incx	INTEGER。 x の成分に対する増分を指定する。

出力パラメーター

x 更新されたベクトル x によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン scal のインターフェイスの詳細を以下に示す。

x 長さ (n) のベクトルを格納する。



注: スカラー・パラメーター a は、異なるデータ型の演算を行う関数を区別するため、関数 scal に対する Fortran-95 インターフェイスで、必須パラメーターとして定義される。

?swap

ベクトルを別のベクトルと交換する。

構文

Fortran 77:

```
call sswap( n, x, incx, y, incy )
call dswap( n, x, incx, y, incy )
call cswap( n, x, incx, y, incy )
call zswap( n, x, incx, y, incy )
```

Fortran 95:

```
call swap(x, y)
```

説明

2つの複素ベクトル x と y が与えられたときに、?swap ルーチンは、相互に置き換えられたベクトル y と x を返す。

入力パラメーター

n	INTEGER。ベクトル x と y の次数を指定する。
x	REAL (sswap の場合) DOUBLE PRECISION (dswap の場合) COMPLEX (cswap の場合) DOUBLE COMPLEX (zswap の場合) 配列、次元は $(1 + (n-1)*abs(incx))$ 以上。
$incx$	INTEGER。 x の成分に対する増分を指定する。
y	REAL (sswap の場合) DOUBLE PRECISION (dswap の場合) COMPLEX (cswap の場合) DOUBLE COMPLEX (zswap の場合) 配列、次元は $(1 + (n-1)*abs(incy))$ 以上。
$incy$	INTEGER。 y の成分に対する増分を指定する。

出力パラメーター

x	結果として得られるベクトル x が格納される。
y	結果として得られるベクトル y が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `swap` のインターフェイスの詳細を以下に示す。

`x` 長さ (n) のベクトルを格納する。
`y` 長さ (n) のベクトルを格納する。

i?amax

最大絶対値を持つベクトルの成分を検出する。

構文

Fortran 77:

```
index = isamax( n, x, incx )  
index = idamax( n, x, incx )  
index = icamax( n, x, incx )  
index = izamax( n, x, incx )
```

Fortran 95:

```
index = iamax(x)
```

説明

ベクトル x が与えられたときに、`i?amax` 関数は最大絶対値を持つベクトル成分 $x(i)$ の位置を返す。複素ベクトルの場合は、この関数は最大合計 $|\operatorname{Re} x(i)| + |\operatorname{Im} x(i)|$ を持つ成分の位置を返す。

n が正でない場合は、0 が返される。

同じ最大絶対値を持つベクトル成分が 2 つ以上検出された場合は、最初に検出された成分のインデックスが返される。

入力パラメーター

`n` INTEGER。ベクトル x の次数を指定する。
`x` REAL (`isamax` の場合)
 DOUBLE PRECISION (`idamax` の場合)
 COMPLEX (`icamax` の場合)
 DOUBLE COMPLEX (`izamax` の場合)
 配列、次元は $(1+(n-1)*\operatorname{abs}(\operatorname{incx}))$ 以上。
`incx` INTEGER。 x の成分に対する増分を指定する。

出力パラメーター

`index` INTEGER。最大絶対値を持つベクトル成分 x の位置が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `amax` のインターフェイスの詳細を以下に示す。

`x` 長さ (n) のベクトルを格納する。

i?amin

最小絶対値を持つベクトル成分を検出する。

構文

Fortran 77:

```
index = isamin( n, x, incx )
index = idamin( n, x, incx )
index = icamin( n, x, incx )
index = izamin( n, x, incx )
```

Fortran 95:

```
index = iamin(x)
```

説明

ベクトル x が与えられたときに、`i?amin` 関数は最小絶対値を持つベクトル成分 $x(i)$ の位置を返す。複素ベクトルの場合は、この関数は最小合計 $|Re(x(i))| + |Im(x(i))|$ を持つ成分の位置を返す。

n が正でない場合は、0 が返される。

同じ最小絶対値を持つベクトル成分が 2 つ以上検出された場合は、最初に検出した成分のインデックスが返される。

入力パラメーター

`n` INTEGER。 n にはベクトル x の次数を指定する。

`x` REAL (`isamin` の場合)
 DOUBLE PRECISION (`idamin` の場合)
 COMPLEX (`icamin` の場合)
 DOUBLE COMPLEX (`izamin` の場合)

配列、次元は $(1+(n-1)*abs(incx))$ 以上。

`incx` INTEGER。 x の成分に対する増分を指定する。

出力パラメーター

`index` INTEGER。最小絶対値を持つベクトル成分 x の位置が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `amin` のインターフェイスの詳細を以下に示す。

`x` 長さ (n) のベクトルを格納する。

dcabs1

倍精度複素数の絶対値を計算する。

構文

Fortran 77:

```
res = dcabs1(z)
```

Fortran 95:

```
res = dcabs1(z)
```

説明

`dcabs1` は、BLAS レベル 1 ルーチンの補助ルーチンである。この関数は、次のように定義される演算を実行する。

$$res = |\operatorname{Re}(z)| + |\operatorname{Im}(z)|$$

z はスカラー、 res は倍精度複素数 z の絶対値が格納される値である。

入力パラメーター

z DOUBLE COMPLEX スカラー。

出力パラメーター

res DOUBLE PRECISION。倍精度複素数 z の絶対値が格納される。

BLAS レベル 2 のルーチン

このセクションでは、行列 - ベクトル演算を実行する BLAS レベル 2 のルーチンについて説明する。表 2-2 に、BLAS レベル 2 のルーチンのグループと、それらに関連するデータ型を示す。

表 2-2 BLAS レベル 2 のルーチングループおよびそのデータ型

ルーチン グループ	データ型	説明
?qbmw	s, d, c, z	一般帯行列での行列 - ベクトルの積
?gemv	s, d, c, z	一般行列での行列 - ベクトルの積
?ger	s, d	一般行列の階数 1 の更新
?gerc	c, z	共役一般行列の階数 1 の更新
?geru	c, z	非共役一般行列の階数 1 の更新
?hbmw	c, z	エルミート帯行列での行列 - ベクトルの積
?hemv	c, z	エルミート行列での行列 - ベクトルの積
?her	c, z	エルミート行列の階数 1 の更新
?her2	c, z	エルミート行列の階数 2 の更新
?hpmw	c, z	圧縮形式のエルミート行列での行列 - ベクトルの積
?hpr	c, z	圧縮形式のエルミート行列の階数 1 の更新
?hpr2	c, z	圧縮形式のエルミート行列の階数 2 の更新
?sbmv	s, d	対称帯行列での行列 - ベクトルの積
?spmv	s, d	圧縮形式の対称行列での行列 - ベクトルの積
?spr	s, d	圧縮形式の対称行列の階数 1 の更新
?spr2	s, d	圧縮形式の対称行列の階数 2 の更新
?symv	s, d	対称行列での行列 - ベクトルの積
?syr	s, d	対称行列の階数 1 の更新
?syr2	s, d	対称行列の階数 2 の更新
?tbmv	s, d, c, z	三角帯行列での行列 - ベクトルの積
?tbsv	s, d, c, z	三角帯行列の 1 次方程式の解
?tpmv	s, d, c, z	圧縮形式の三角行列での行列 - ベクトルの積
?tpsv	s, d, c, z	圧縮形式の三角行列の 1 次方程式の解
?trmv	s, d, c, z	三角行列での行列 - ベクトルの積
?trsv	s, d, c, z	三角行列の 1 次方程式の解

?gbmv

一般帯行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```

call sgbmv( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy )
call dgbmv( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy )
call cgbmv( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy )
call zgbmv( trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy )

```

Fortran 95:

```

call gbmv(a, x, y [,kl] [,m] [,alpha] [,beta] [,trans])

```

説明

?gbmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := \alpha * a * x + \beta * y$

または

$y := \alpha * a' * x + \beta * y$

または

$y := \alpha * \text{conjg}(a') * x + \beta * y$

α と β は、スカラーである。

x と y は、ベクトルである。

a は、 kl 個の劣対角成分と ku 個の優対角成分を持つ $m \times n$ の帯行列である。

入力パラメーター

$trans$ CHARACTER*1。次に示すように、実行する演算を指定する。

$trans$ の値	実行する演算
N または n	$y := \alpha * a * x + \beta * y$
T または t	$y := \alpha * a' * x + \beta * y$
C または c	$y := \alpha * \text{conjg}(a') * x + \beta * y$

m INTEGER。行列 a の行数を指定する。 m の値は、ゼロ以上でなければならない。

n INTEGER。行列 a の列数を指定する。 n の値は、ゼロ以上でなければならない。

kl INTEGER。行列 a の劣対角成分の数を指定する。 kl の値は、 $0 \leq kl$ を満たしていなければならない。

<i>ku</i>	INTEGER。行列 <i>a</i> の優対角成分の数を指定する。 <i>ku</i> の値は、 $0 \leq ku$ を満たしていなければならない。
<i>alpha</i>	REAL (<i>sgbmv</i> の場合) DOUBLE PRECISION (<i>dgbmv</i> の場合) COMPLEX (<i>cgbmv</i> の場合) DOUBLE COMPLEX (<i>zgbmv</i> の場合) スカラー <i>alpha</i> を指定する。
<i>a</i>	REAL (<i>sgbmv</i> の場合) DOUBLE PRECISION (<i>dgbmv</i> の場合) COMPLEX (<i>cgbmv</i> の場合) DOUBLE COMPLEX (<i>zgbmv</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>a</i> の先頭の $(k1 + ku + 1) \times n$ の部分に係数の行列を格納しなければならない。この行列は、行列の主対角成分が配列の行 (<i>ku</i> + 1) にくるように、また、最初の優対角成分が行 <i>ku</i> の位置 2 から始まり、最初の劣対角成分が行 (<i>ku</i> + 2) の位置 1 から始まるように、列ごとに与えなければならない。帯行列の成分に対応しない配列 <i>a</i> の成分 (左上の $ku \times ku$ の三角など) は参照されない。 次に示すプログラムの一部は、帯行列を通常のフル格納形式から帯格納形式に転送するためのものである。 <pre> do 20, j = 1, n k = ku + 1 - j do 10, i = max(1, j-ku), min(m, j+k1) a(k+i, j) = matrix(i,j) 10 continue 20 continue </pre>
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $(k1 + ku + 1)$ 以上でなければならない。
<i>x</i>	REAL (<i>sgbmv</i> の場合) DOUBLE PRECISION (<i>dgbmv</i> の場合) COMPLEX (<i>cgbmv</i> の場合) DOUBLE COMPLEX (<i>zgbmv</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上 (<i>trans</i> = 'N' または 'n' の場合) または $(1 + (m - 1) * \text{abs}(\text{incx}))$ 以上 (それ以外の場合)。このルーチンに入る前に、増分された配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (<i>sgbmv</i> の場合) DOUBLE PRECISION (<i>dgbmv</i> の場合) COMPLEX (<i>cgbmv</i> の場合) DOUBLE COMPLEX (<i>zgbmv</i> の場合) スカラー <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。

y	REAL (sgbmv の場合) DOUBLE PRECISION (dgbmv の場合) COMPLEX (cgbmv の場合) DOUBLE COMPLEX (zgbmv の場合) 配列、次元は $(1 + (m - 1) * \text{abs}(\text{incy}))$ 以上 ($\text{trans} = 'N'$ または ' n ' の場合) または $(1 + (n - 1) * \text{abs}(\text{incy}))$ (それ以外の場合)。このルーチンに入る前に、増分された配列 y にベクトル y を格納しなければならない。
incy	INTEGER。 y の成分に対する増分を指定する。 incy の値は、ゼロであってはならない。

出力パラメーター

y	更新されたベクトル y によって上書きされる。
-----	---------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gmv` のインターフェイスの詳細を以下に示す。

a	サイズ $(k1+ku+1, n)$ の配列 A を格納する。
x	長さ (rx) のベクトルを格納する。 $rx = n$ ($\text{trans} = 'N'$ の場合) $rx = m$ (それ以外の場合)
y	長さ (ry) のベクトルを格納する。 $ry = m$ ($\text{trans} = 'N'$ の場合) $ry = n$ (それ以外の場合)
trans	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
$k1$	省略した場合、 $k1 = ku$ とみなす。
ku	$ku = lda - k1 - 1$ として復元する。
m	省略した場合、 $m = n$ とみなす。
α	デフォルト値は '1' である。
β	デフォルト値は '1' である。

?gemv

一般行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call sgemv( trans, m, n, alpha, a, lda, x, incx, beta, y, incy )
call dgemv( trans, m, n, alpha, a, lda, x, incx, beta, y, incy )
call cgemv( trans, m, n, alpha, a, lda, x, incx, beta, y, incy )
call zgemv( trans, m, n, alpha, a, lda, x, incx, beta, y, incy )
```

Fortran 95:

```
call gemv(a, x, y [,alpha] [,beta] [,trans])
```

説明

?gemv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y$$

または

$$y := \alpha * a' * x + \beta * y$$

または

$$y := \alpha * \text{conjg}(a') * x + \beta * y$$

α と β は、スカラーである。

x と y は、ベクトルである。

a は、 $m \times n$ の行列である。

入力パラメーター

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$y := \alpha * a * x + \beta * y$
T または t	$y := \alpha * a' * x + \beta * y$
C または c	$y := \alpha * \text{conjg}(a') * x + \beta * y$

m INTEGER。行列 a の行数を指定する。 m の値は、ゼロ以上でなければならない。

n INTEGER。行列 a の列数を指定する。 n の値は、ゼロ以上でなければならない。

<i>alpha</i>	REAL (sgemv の場合) DOUBLE PRECISION (dgemv の場合) COMPLEX (cgemv の場合) DOUBLE COMPLEX (zgemv の場合) スカラー <i>alpha</i> を指定する。
<i>a</i>	REAL (sgemv の場合) DOUBLE PRECISION (dgemv の場合) COMPLEX (cgemv の場合) DOUBLE COMPLEX (zgemv の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>a</i> の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, m)$ 以上でなければならない。
<i>x</i>	REAL (sgemv の場合) DOUBLE PRECISION (dgemv の場合) COMPLEX (cgemv の場合) DOUBLE COMPLEX (zgemv の場合) 配列、次元は <i>trans</i> = 'N' または 'n' の場合 $(1+(n-1)*\text{abs}(\text{incx}))$ 以上。それ以外の場合は $(1+(m-1)*\text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (sgemv の場合) DOUBLE PRECISION (dgemv の場合) COMPLEX (cgemv の場合) DOUBLE COMPLEX (zgemv の場合) スカラー <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	REAL (sgemv の場合) DOUBLE PRECISION (dgemv の場合) COMPLEX (cgemv の場合) DOUBLE COMPLEX (zgemv の場合) 配列、次元は <i>trans</i> = 'N' または 'n' の場合 $(1+(m-1)*\text{abs}(\text{incy}))$ 以上。それ以外の場合は $(1+(n-1)*\text{abs}(\text{incy}))$ 以上。 <i>beta</i> が非ゼロでこのルーチンに入る前に、増分された配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

出力パラメーター

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gemv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (m,n) の行列 A を格納する。
<code>x</code>	長さ (rx) のベクトルを格納する。 $rx = n$ ($trans = 'N'$ の場合) $rx = m$ (それ以外の場合)
<code>y</code>	長さ (ry) のベクトルを格納する。 $ry = m$ ($trans = 'N'$ の場合) $ry = n$ (それ以外の場合)
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

?ger

一般行列の階数 l の更新を実行する。

構文

Fortran 77:

```
call sger( m, n, alpha, x, incx, y, incy, a, lda )
call dger( m, n, alpha, x, incx, y, incy, a, lda )
```

Fortran 95:

```
call ger(a, x, y [,alpha])
```

説明

?ger ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + a$$

`alpha` は、スカラーである。

`x` は、 m 個の成分を持つベクトルである。

`y` は、 n 個の成分を持つベクトルである。

`a` は、 $m \times n$ の行列である。

入力パラメーター

`m` INTEGER。行列 `a` の行数を指定する。`m` の値は、ゼロ以上でなければならない。

<i>n</i>	INTEGER。行列 <i>a</i> の列数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha</i>	REAL (<i>sger</i> の場合) DOUBLE PRECISION (<i>dger</i> の場合) スカラー <i>alpha</i> を指定する。
<i>x</i>	REAL (<i>sger</i> の場合) DOUBLE PRECISION (<i>dger</i> の場合) 配列、次元は $(1 + (m - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>m</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>y</i>	REAL (<i>sger</i> の場合) DOUBLE PRECISION (<i>dger</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
<i>a</i>	REAL (<i>sger</i> の場合) DOUBLE PRECISION (<i>dger</i> の場合) 配列、次元は (lda, n) 。このルーチンに入る前に、配列 <i>a</i> の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

a 更新された行列によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ger* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 <i>A</i> を格納する。
<i>x</i>	長さ (m) のベクトルを格納する。
<i>y</i>	長さ (n) のベクトルを格納する。
<i>alpha</i>	デフォルト値は '1' である。

?gerc

一般行列の階数1の更新(共役)を実行する。

構文

Fortran 77:

```
call cgerc( m, n, alpha, x, incx, y, incy, a, lda )
call zgerc( m, n, alpha, x, incx, y, incy, a, lda )
```

Fortran 95:

```
call gerc(a, x, y [,alpha])
```

説明

?gerc ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + a$$

α は、スカラーである。

x は、 m 個の成分を持つベクトルである。

y は、 n 個の成分を持つベクトルである。

a は、 $m \times n$ の行列である。

入力パラメーター

m	INTEGER。行列 a の行数を指定する。 m の値は、ゼロ以上でなければならない。
n	INTEGER。行列 a の列数を指定する。 n の値は、ゼロ以上でなければならない。
α	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合) スカラー α を指定する。
x	SINGLE PRECISION COMPLEX (cgerc の場合) DOUBLE PRECISION COMPLEX (zgerc の場合) 配列、次元は $(1 + (m - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に m 個の成分を持つベクトル x を格納しなければならない。
incx	INTEGER。 x の成分に対する増分を指定する。 incx の値は、ゼロであってはならない。
y	COMPLEX (cgerc の場合) DOUBLE COMPLEX (zgerc の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。

<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
<i>a</i>	COMPLEX (<i>cgerc</i> の場合) DOUBLE COMPLEX (<i>zgerc</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>a</i> の先頭の <i>m</i> × <i>n</i> の部分に係数の行列を格納しなければならない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

a 更新された行列によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gerc* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>m</i>) のベクトルを格納する。
<i>y</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>alpha</i>	デフォルト値は '1' である。

?geru

一般行列の階数 1 の更新 (非共役) を実行する。

構文

Fortran 77:

```
call cgeru( m, n, alpha, x, incx, y, incy, a, lda )
call zgeru( m, n, alpha, x, incx, y, incy, a, lda )
```

Fortran 95:

```
call geru( a, x, y [,alpha])
```

説明

?geru ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + a$$

alpha は、スカラーである。

x は、*m* 個の成分を持つベクトルである。

y は、 n 個の成分を持つベクトルである。

a は、 $m \times n$ の行列である。

入力パラメーター

m	INTEGER。行列 a の行数を指定する。 m の値は、ゼロ以上でなければならない。
n	INTEGER。行列 a の列数を指定する。 n の値は、ゼロ以上でなければならない。
$alpha$	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合) スカラー $alpha$ を指定する。
x	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合) 配列、次元は $(1 + (m - 1) * abs(incx))$ 以上。このルーチンに入る前に、増分された配列 x に m 個の成分を持つベクトル x を格納しなければならない。
$incx$	INTEGER。 x の成分に対する増分を指定する。 $incx$ の値は、ゼロであってはならない。
y	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合) 配列、次元は $(1 + (n - 1) * abs(incy))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。
$incy$	INTEGER。 y の成分に対する増分を指定する。 $incy$ の値は、ゼロであってはならない。
a	COMPLEX (cgeru の場合) DOUBLE COMPLEX (zgeru の場合) 配列、次元は (lda, n) 。このルーチンに入る前に、配列 a の先頭の $m \times n$ の部分に係数の行列を格納しなければならない。
lda	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 lda の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

a 更新された行列によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geru` のインターフェイスの詳細を以下に示す。

a サイズ (m, n) の行列 A を格納する。

<i>x</i>	長さ (<i>m</i>) のベクトルを格納する。
<i>y</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>alpha</i>	デフォルト値は '1' である。

?hbmV

エルミート帯行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call chbmV( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
call zhbmv( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
```

Fortran 95:

```
call hbmV(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?hbmV ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、*k* 個の優対角成分を持つ $n \times n$ のエルミート帯行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、帯行列 *a* の上三角部分と下三角部分のどちらを与えるかを指定する。

<i>uplo</i> の値	与える行列 <i>a</i> の部分
U または u	行列 <i>a</i> の上三角部分を与える。
L または l	行列 <i>a</i> の下三角部分を与える。

n INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

k INTEGER。行列 *a* の優対角成分の数を指定する。*k* の値は、 $0 \leq k$ を満たさなければならない。

alpha COMPLEX (chbmV の場合)
DOUBLE COMPLEX (zhbmV の場合)
スカラー *alpha* を指定する。

a COMPLEX (chbmV の場合)
DOUBLE COMPLEX (zhbmV の場合)

配列、次元は (lda, n) 。

$uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $(k+1) \times n$ の部分に、エルミート行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 $(k+1)$ にくるように、また最初の優対角成分が行 k の位置 2 から始まるように、列ごとに与えなければならない。配列 a の左上の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、エルミート帯行列の上三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix(i, j)
  10 continue
20 continue
```

$uplo = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $(k+1) \times n$ の部分に、エルミート行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 a の右下の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、エルミート帯行列の下三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min( n, j + k )
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 lda の値は、 $(k+1)$ 以上でなければならない。
<i>x</i>	COMPLEX (chbmv の場合) DOUBLE COMPLEX (zhebmv の場合) 配列、次元は $(1 + (n-1) * \text{abs}(incx))$ 以上。このルーチンに入る前に、増分された配列 x にベクトル x を格納しなければならない。
<i>incx</i>	INTEGER。 x の成分に対する増分を指定する。 $incx$ の値は、ゼロであってはならない。
<i>beta</i>	COMPLEX (chbmv の場合) DOUBLE COMPLEX (zhebmv の場合) スカラー $beta$ を指定する。
<i>y</i>	COMPLEX (chbmv の場合) DOUBLE COMPLEX (zhebmv の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(incy))$ 以上。このルーチンに入る前に、増分された配列 y にベクトル y を格納しなければならない。

incy INTEGER。 y の成分に対する増分を指定する。 *incy* の値は、ゼロであってはならない。

出力パラメーター

y 更新されたベクトル y によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbmV` のインターフェイスの詳細を以下に示す。

a サイズ $(k+1, n)$ の配列 A を格納する。
x 長さ (n) のベクトルを格納する。
y 長さ (n) のベクトルを格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。
alpha デフォルト値は '1' である。
beta デフォルト値は '1' である。

?hemv

エルミート行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call chemv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call zhemv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

Fortran 95:

```
call hemv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?hemv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

alpha と *beta* は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ のエルミート行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 <i>a</i> の部分
U または u	配列 <i>a</i> の上三角部分が参照される。
L または l	配列 <i>a</i> の下三角部分が参照される。

n INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

alpha COMPLEX (*chemv* の場合)
DOUBLE COMPLEX (*zhemv* の場合)
スカラー *alpha* を指定する。

a COMPLEX (*chemv* の場合)
DOUBLE COMPLEX (*zhemv* の場合)

配列、次元は (*lda*, *n*)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならない。また *a* の厳密な下三角部分は参照されない。
また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times n$ の下三角部分に、エルミート行列の下三角部分を格納しなければならない。また *a* の厳密な上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $\max(1, n)$ 以上でなければならない。

x COMPLEX (*chemv* の場合)
DOUBLE COMPLEX (*zhemv* の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

incx INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

beta COMPLEX (*chemv* の場合)
DOUBLE COMPLEX (*zhemv* の場合)

スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y COMPLEX (*chemv* の場合)
DOUBLE COMPLEX (*zhemv* の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(incy))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。

incy INTEGER。 y の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。

出力パラメーター

y 更新されたベクトル y によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hemv` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>x</i>	長さ (n) のベクトルを格納する。
<i>y</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?her

エルミート行列の階数 1 の更新を実行する。

構文

Fortran 77:

```
call cher( uplo, n, alpha, x, incx, a, lda )
call zher( uplo, n, alpha, x, incx, a, lda )
```

Fortran 95:

```
call her(a, x [,uplo] [,alpha])
```

説明

?her ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \text{alpha} * x * \text{conjg}(x') + a$$

alpha は、実数のスカラーである。

x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ のエルミート行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 <i>a</i> の部分
U または u	配列 <i>a</i> の上三角部分が参照される。
L または l	配列 <i>a</i> の下三角部分が参照される。

n INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

alpha REAL (*cher* の場合)
DOUBLE PRECISION (*zher* の場合)
スカラー *alpha* を指定する。

x COMPLEX (*cher* の場合)
DOUBLE COMPLEX (*zher* の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

incx INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

a COMPLEX (*cher* の場合)
DOUBLE COMPLEX (*zher* の場合)

配列、次元は (lda, n) 。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならない。また *a* の厳密な下三角部分は参照されない。

また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times n$ の下三角部分に、エルミート行列の下三角部分を格納しなければならない。また *a* の厳密な上三角部分は参照されない。

対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

a *uplo* = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新された行列の上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新された行列の下三角部分によって上書きされる。

対角成分の虚数部は、ゼロに設定される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `her` のインターフェイスの詳細を以下に示す。

`a` サイズ (n, n) の行列 A を格納する。
`x` 長さ (n) のベクトルを格納する。
`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。
`alpha` デフォルト値は '1' である。

?her2

エルミート行列の階数2の更新を実行する。

構文

Fortran 77:

```
call cher2( uplo, n, alpha, x, incx, y, incy, a, lda )
call zher2( uplo, n, alpha, x, incx, y, incy, a, lda )
```

Fortran 95:

```
call her2(a, x, y [,uplo] [,alpha])
```

説明

?her2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjg}(x') + a,$$

`alpha` は、スカラーである。

`x` と `y` は、 n 個の成分を持つベクトルである。

`a` は、 $n \times n$ のエルミート行列である。

入力パラメーター

`uplo` CHARACTER*1。次に示すように、配列 `a` の上三角部分と下三角部分のどちらが参照されるかを指定する。

<code>uplo</code> の値	参照される配列 <code>a</code> の部分
U または u	配列 <code>a</code> の上三角部分が参照される。
L または l	配列 <code>a</code> の下三角部分が参照される。

`n` INTEGER。行列 `a` の次数を指定する。`n` の値は、ゼロ以上でなければならない。

<i>alpha</i>	COMPLEX (cher2 の場合) DOUBLE COMPLEX (zher2 の場合) スカラー <i>alpha</i> を指定する。
<i>x</i>	COMPLEX (cher2 の場合) DOUBLE COMPLEX (zher2 の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>y</i>	COMPLEX (cher2 の場合) DOUBLE COMPLEX (zher2 の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
<i>a</i>	COMPLEX (cher2 の場合) DOUBLE COMPLEX (zher2 の場合) 配列、次元は (lda, n) 。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならない。また <i>a</i> の厳密な下三角部分は参照されない。 また、 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の下三角部分に、エルミート行列の下三角部分を格納しなければならない。また <i>a</i> の厳密な上三角部分は参照されない。 対角成分の虚数部は設定する必要はなく、ゼロとみなされる。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

<i>a</i>	<i>uplo</i> = 'U' または 'u' と指定した場合は、配列 <i>a</i> の上三角部分が、更新された行列の上三角部分によって上書きされる。 <i>uplo</i> = 'L' または 'l' と指定した場合は、配列 <i>a</i> の下三角部分が、更新された行列の下三角部分によって上書きされる。 対角成分の虚数部は、ゼロに設定される。
----------	---

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン her2 のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>y</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。

?hpmv

圧縮形式のエルミート行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call chpmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
call zhpmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

Fortran 95:

```
call hpmv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?hpmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

alpha と *beta* は、スカラーである。

x と *y* は、*n* 個の成分を持つベクトルである。

a は、圧縮形式で与えられる $n \times n$ のエルミート行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 *a* の上三角部分と下三角部分のどちらを圧縮形式の配列 *ap* において与えるかを指定する。

<i>uplo</i> の値	与える行列 <i>a</i> の部分
U または u	行列 <i>a</i> の上三角部分を <i>ap</i> において与える。
L または l	行列 <i>a</i> の下三角部分を <i>ap</i> において与える。

<i>n</i>	INTEGER。行列 <i>a</i> の次数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha</i>	COMPLEX (chpmv の場合) DOUBLE COMPLEX (zhpmv の場合) スカラー <i>alpha</i> を指定する。
<i>ap</i>	COMPLEX (chpmv の場合) DOUBLE COMPLEX (zhpmv の場合) 配列、次元は $((n*(n+1))/2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。 対角成分の虚数部は設定する必要はなく、ゼロとみなされる。
<i>x</i>	COMPLEX (chpmv の場合) DOUBLE PRECISION COMPLEX (zhpmv の場合) 配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	COMPLEX (chpmv の場合) DOUBLE COMPLEX (zhpmv の場合) スカラー <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>y</i>	COMPLEX (chpmv の場合) DOUBLE COMPLEX (zhpmv の場合) 配列、次元は $(1 + (n - 1)*abs(incy))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

出力パラメーター

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpmv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>x</code>	長さ (n) のベクトルを格納する。
<code>y</code>	長さ (n) のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

?hpr

圧縮形式のエルミート行列の階数 1 の更新を実行する。

構文

Fortran 77:

```
call chpr( uplo, n, alpha, x, incx, ap )
call zhpr( uplo, n, alpha, x, incx, ap )
```

Fortran 95:

```
call hpr(a, x [,uplo] [,alpha])
```

説明

?hpr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(x') + a$$

`alpha` は、実数のスカラーである。

`x` は、 n 個の成分を持つベクトルである。

`a` は、圧縮形式で与えられる $n \times n$ のエルミート行列である。

入力パラメーター

`uplo` CHARACTER*1。次に示すように、行列 `a` の上三角部分と下三角部分のどちらを圧縮形式の配列 `ap` において与えるかを指定する。

<code>uplo</code> の値	与える行列 <code>a</code> の部分
U または u	行列 <code>a</code> の上三角部分を <code>ap</code> において与える。
L または l	行列 <code>a</code> の下三角部分を <code>ap</code> において与える。

`n` INTEGER。行列 `a` の次数を指定する。`n` の値は、ゼロ以上でなければならない。

`alpha` REAL (`chpr` の場合)
DOUBLE PRECISION (`zhpr` の場合)

	スカラー <i>alpha</i> を指定する。
<i>x</i>	COMPLEX (chpr の場合) DOUBLE COMPLEX (zhpr の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>ap</i>	COMPLEX (chpr の場合) DOUBLE COMPLEX (zhpr の場合) 配列、次元は $((n * (n + 1)) / 2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。 対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

出力パラメーター

<i>ap</i>	<i>uplo</i> = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。 <i>uplo</i> = 'L' または 'l' と指定した場合は、更新された行列の下三角部分によって上書きされる。 対角成分の虚数部は、ゼロに設定される。
-----------	---

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n * (n + 1) / 2)$ の配列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。

?hpr2

圧縮形式のエルミート行列の階数2の更新を実行する。

構文

Fortran 77:

```
call chpr2( uplo, n, alpha, x, incx, y, incy, ap )
call zhpr2( uplo, n, alpha, x, incx, y, incy, ap )
```

Fortran 95:

```
call hpr2(a, x, y [,uplo] [,alpha])
```

説明

?hpr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(y') + \text{conjg}(\alpha) * y * \text{conjg}(x') + a$$

α は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、圧縮形式で与えられる $n \times n$ のエルミート行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 a の上三角部分と下三角部分のどちらを圧縮形式の配列 ap において与えるかを指定する。

<i>uplo</i> の値	与える行列 a の部分
U または u	行列 a の上三角部分を ap において与える。
L または l	行列 a の下三角部分を ap において与える。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

alpha COMPLEX (chpr2 の場合)
DOUBLE COMPLEX (zhpr2 の場合)
スカラー α を指定する。

x COMPLEX (chpr2 の場合)
DOUBLE COMPLEX (zhpr2 の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。

incx INTEGER。 x の成分に対する増分を指定する。 incx の値は、ゼロであってはならない。

<i>y</i>	COMPLEX (chpr2 の場合) DOUBLE COMPLEX (zhpr2 の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
<i>ap</i>	COMPLEX (chpr2 の場合) DOUBLE COMPLEX (zhpr2 の場合) 配列、次元は $((n * (n + 1)) / 2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の上三角部分を列ごとに格納しなければならない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。 対角成分の虚数部は設定する必要はなく、ゼロとみなされる。

出力パラメーター

<i>ap</i>	<i>uplo</i> = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。 <i>uplo</i> = 'L' または 'l' と指定した場合は、更新された行列の下三角部分によって上書きされる。 対角成分の虚数部はゼロに設定される。
-----------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpr2* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n * (n + 1) / 2)$ の配列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>y</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。

?sbmv

対称帯行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call ssbmv( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
call dsbmv( uplo, n, k, alpha, a, lda, x, incx, beta, y, incy )
```

Fortran 95:

```
call sbmv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?sbmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

α と β は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、 k 個の優対角成分を持つ $n \times n$ の対称帯行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、帯行列 a の上三角部分と下三角部分のどちらを与えるかを指定する。

<i>uplo</i> の値	与える行列 a の部分
U または u	行列 a の上三角部分を与える。
L または l	行列 a の下三角部分を与える。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

k INTEGER。行列 a の優対角成分の数を指定する。 k の値は、 $0 \leq k$ を満たさなければならない。

α REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
スカラー α を指定する。

a REAL (ssbmv の場合)
DOUBLE PRECISION (dsbmv の場合)
配列、次元は (lda, n) 。
 $uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $(k+1) \times n$ の部分には、対称行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行

($k+1$) にくるように、また最初の優対角成分が行 k の位置 2 から始まるように、列ごとに与えなければならない。配列 a の左上の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、対称帯行列の上三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max( 1, j - k ), j
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、配列 a の先頭の $(k+1) \times n$ の部分に、対称行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 a の右下の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、対称帯行列の下三角部分を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min( n, j + k )
    a( m + i, j ) = matrix( i, j )
  10 continue
20 continue
```

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 <i>lda</i> の値は、 $(k+1)$ 以上でなければならない。
<i>x</i>	REAL (ssbmv の場合) DOUBLE PRECISION (dsbmv の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x にベクトル x を格納しなければならない。
<i>incx</i>	INTEGER。 x の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (ssbmv の場合) DOUBLE PRECISION (dsbmv の場合) スカラー <i>beta</i> を指定する。
<i>y</i>	REAL (ssbmv の場合) DOUBLE PRECISION (dsbmv の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 y にベクトル y を格納しなければならない。
<i>incy</i>	INTEGER。 y の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

出力パラメーター

y 更新されたベクトル y によって書き換えられる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbmv` のインターフェイスの詳細を以下に示す。

a	サイズ $(k+1, n)$ の配列 A を格納する。
x	長さ (n) のベクトルを格納する。
y	長さ (n) のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

?spmv

圧縮形式の対称行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call sspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
call dspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

Fortran 95:

```
call spmv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?spmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

α と β は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、圧縮形式で与えられる $n \times n$ の対称行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 *a* の上三角部分と下三角部分のどちらを圧縮形式の配列 *ap* において与えるかを指定する。

<i>uplo</i> の値	与える行列 <i>a</i> の部分
U または u	行列 <i>a</i> の上三角部分を <i>ap</i> において与える。
L または l	行列 <i>a</i> の下三角部分を <i>ap</i> において与える。

n INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

alpha REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
スカラー *alpha* を指定する。

ap REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列、次元は $((n*(n+1))/2)$ 以上。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(1, 2) が、*ap*(3) に *a*(2, 2) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、*ap*(1) に *a*(1, 1) が、*ap*(2) に *a*(2, 1) が、*ap*(3) に *a*(3, 1) がそれぞれ格納されるように、配列 *ap* には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

x REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分を持つベクトル *x* を格納しなければならない。

incx INTEGER。*x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

beta REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
スカラー *beta* を指定する。*beta* をゼロに設定した場合は、*y* を設定する必要はない。

y REAL (sspmv の場合)
DOUBLE PRECISION (dspmv の場合)
配列、次元は $(1 + (n - 1)*abs(incy))$ 以上。このルーチンに入る前に、増分された配列 *y* に *n* 個の成分を持つベクトル *y* を格納しなければならない。

incy INTEGER。*y* の成分に対する増分を指定する。*incy* の値は、ゼロであってはならない。

出力パラメーター

y 更新されたベクトル *y* によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spmv` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>x</code>	長さ (n) のベクトルを格納する。
<code>y</code>	長さ (n) のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

?spr

圧縮形式の対称行列の階数1の更新を実行する。

構文

Fortran 77:

```
call sspr( uplo, n, alpha, x, incx, ap )
call dspr( uplo, n, alpha, x, incx, ap )
```

Fortran 95:

```
call spr(a, x [,uplo] [,alpha])
```

説明

?spr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * x' + a$$

`alpha` は、実数のスカラーである。

`x` は、 n 個の成分を持つベクトルである。

`a` は、圧縮形式で与えられる $n \times n$ の対称行列である。

入力パラメーター

`uplo` CHARACTER*1。次に示すように、行列 `a` の上三角部分と下三角部分のどちらを圧縮形式の配列 `ap` において与えるかを指定する。

<code>uplo</code> の値	与える行列 <code>a</code> の部分
U または u	行列 <code>a</code> の上三角部分を <code>ap</code> において与える。
L または l	行列 <code>a</code> の下三角部分を <code>ap</code> において与える。

<i>n</i>	INTEGER。行列 <i>a</i> の次数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha</i>	REAL (<i>sspr</i> の場合) DOUBLE PRECISION (<i>dspr</i> の場合) スカラー <i>alpha</i> を指定する。
<i>x</i>	REAL (<i>sspr</i> の場合) DOUBLE PRECISION (<i>dspr</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>ap</i>	REAL (<i>sspr</i> の場合) DOUBLE PRECISION (<i>dspr</i> の場合) 配列、次元は $((n * (n + 1)) / 2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように配列 <i>ap</i> には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

出力パラメーター

<i>ap</i>	<i>uplo</i> = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。 <i>uplo</i> = 'L' または 'l' と指定した場合は、更新された行列の下三角部分によって上書きされる。
-----------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(n * (n + 1) / 2)$ の配列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。

?spr2

圧縮形式の対称行列の階数2の更新を実行する。

構文

Fortran 77:

```
call sspr2( uplo, n, alpha, x, incx, y, incy, ap )
call dspr2( uplo, n, alpha, x, incx, y, incy, ap )
```

Fortran 95:

```
call spr2(a, x, y [,uplo] [,alpha])
```

説明

?spr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * y' + \alpha * y * x' + a$$

α は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、圧縮形式で与えられる $n \times n$ の対称行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 a の上三角部分と下三角部分のどちらを圧縮形式の配列 ap において与えるかを指定する。

<i>uplo</i> の値	与える行列 a の部分
U または u	行列 a の上三角部分を ap において与える。
L または l	行列 a の下三角部分を ap において与える。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

α REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)
スカラー α を指定する。

x REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。

incx INTEGER。 x の成分に対する増分を指定する。 incx の値は、ゼロであってはならない。

y REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。

incy INTEGER。 y の成分に対する増分を指定する。 *incy* の値は、ゼロであってはならない。

ap REAL (sspr2 の場合)
DOUBLE PRECISION (dspr2 の場合)

配列、次元は $((n * (n + 1)) / 2)$ 以上。 *uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、 *ap*(1) に $a(1, 1)$ が、 *ap*(2) に $a(1, 2)$ が、 *ap*(3) に $a(2, 2)$ がそれぞれ格納されるように、配列 *ap* には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、 *ap*(1) に $a(1, 1)$ が、 *ap*(2) に $a(2, 1)$ が、 *ap*(3) に $a(3, 1)$ がそれぞれ格納されるように、配列 *ap* には、順番に圧縮されたエルミート行列の下三角部分を列ごとに格納しなければならない。

出力パラメーター

ap *uplo* = 'U' または 'u' と指定した場合は、更新された行列の上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、更新された行列の下三角部分によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spr2* のインターフェイスの詳細を以下に示す。

a サイズ $(n * (n + 1) / 2)$ の配列 A を格納する。

x 長さ (n) のベクトルを格納する。

y 長さ (n) のベクトルを格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

alpha デフォルト値は '1' である。

?symv

対称行列に対して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call ssymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call dsymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

Fortran 95:

```
call symv(a, x, y [,uplo] [,alpha] [,beta])
```

説明

?symv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

α と β は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ の対称行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 a の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 a の部分
U または u	配列 a の上三角部分が参照される。
L または l	配列 a の下三角部分が参照される。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

alpha REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)
スカラー α を指定する。

a REAL (ssymv の場合)
DOUBLE PRECISION (dsymv の場合)

配列、次元は (lda, n)。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の上三角部分に、対称行列の上三角部分を格納しなければならない。また a の厳密な下三角部分は参照されない。

また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る

前に、配列 a の先頭の $n \times n$ の下三角部分に、対称行列の下三角部分を格納しなければならない、また a の厳密な上三角部分は参照されない。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	REAL (ssymv の場合) DOUBLE PRECISION (dsymv の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。
<i>incx</i>	INTEGER。 x の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>beta</i>	REAL (ssymv の場合) DOUBLE PRECISION (dsymv の場合) スカラー <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 y を設定する必要はない。
<i>y</i>	REAL (ssymv の場合) DOUBLE PRECISION (dsymv の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。
<i>incy</i>	INTEGER。 y の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

出力パラメーター

y 更新されたベクトル y によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `symv` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>x</i>	長さ (n) のベクトルを格納する。
<i>y</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?syr

対称行列の階数1の更新を実行する。

構文

Fortran 77:

```
call ssyr( uplo, n, alpha, x, incx, a, lda )
call dsyr( uplo, n, alpha, x, incx, a, lda )
```

Fortran 95:

```
call syr(a, x [,uplo] [,alpha])
```

説明

?syr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * x' + a$$

α は、実数のスカラーである。

x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ の対称行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 a の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される配列 a の部分
U または u	配列 a の上三角部分が参照される。
L または l	配列 a の下三角部分が参照される。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

alpha REAL (ssyr の場合)
DOUBLE PRECISION (dsyr の場合)
スカラー α を指定する。

x REAL (ssyr の場合)
DOUBLE PRECISION (dsyr の場合)
配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。

incx INTEGER。 x の成分に対する増分を指定する。 incx の値は、ゼロであってはならない。

a REAL (ssyr の場合)
DOUBLE PRECISION (dsyr の場合)

配列、次元は (lda, n) 。

$uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の上三角部分に、対称行列の上三角部分を格納しなければならない。また a の厳密な下三角部分は参照されない。

$uplo = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならない。また a の厳密な上三角部分は参照されない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 lda の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

a $uplo = 'U'$ または $'u'$ と指定した場合は、配列 a の上三角部分が、更新された行列の上三角部分によって上書きされる。

$uplo = 'L'$ または $'l'$ と指定した場合は、配列 a の下三角部分が、更新された行列の下三角部分によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン syr のインターフェイスの詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。
 x 長さ (n) のベクトルを格納する。
 $uplo$ $'U'$ または $'L'$ でなければならない。デフォルト値は $'U'$ 。
 $alpha$ デフォルト値は $'1'$ である。

?syr2

対称行列の階数 2 の更新を実行する。

構文

Fortran 77:

```
call ssyr2( uplo, n, alpha, x, incx, y, incy, a, lda )
call dsyr2( uplo, n, alpha, x, incx, y, incy, a, lda )
```

Fortran 95:

```
call syr2(a, x, y [,uplo] [,alpha])
```

説明

?syr2 ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$a := \alpha * x * y' + \alpha * y * x' + a$

α は、スカラーである。

x と y は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ の対称行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 a の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 a の部分
U または u	配列 a の上三角部分が参照される。
L または l	配列 a の下三角部分が参照される。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

α REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
スカラー α を指定する。

x REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。

incx INTEGER。 x の成分に対する増分を指定する。 incx の値は、ゼロであってはならない。

y REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 y に n 個の成分を持つベクトル y を格納しなければならない。

incy INTEGER。 y の成分に対する増分を指定する。 incy の値は、ゼロであってはならない。

a REAL (ssyr2 の場合)
DOUBLE PRECISION (dsyr2 の場合)
配列、次元は (lda, n) 。
 $\text{uplo} = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の上三角部分に、対称行列の上三角部分を格納しなければならない。また a の厳密な下三角部分は参照されない。
 $\text{UPLO} = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならない。また a の厳密な上三角部分は参照されない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

a *uplo* = 'U' または 'u' と指定した場合は、配列 *a* の上三角部分が、更新された行列の上三角部分によって上書きされる。

uplo = 'L' または 'l' と指定した場合は、配列 *a* の下三角部分が、更新された行列の下三角部分によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syrr2` のインターフェイスの詳細を以下に示す。

a サイズ (*n*, *n*) の行列 *A* を格納する。
x 長さ (*n*) のベクトルを格納する。
y 長さ (*n*) のベクトルを格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。
alpha デフォルト値は '1' である。

?tbmv

三角帯行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call stbmv( uplo, trans, diag, n, k, a, lda, x, incx )
call dtbmv( uplo, trans, diag, n, k, a, lda, x, incx )
call ctbmv( uplo, trans, diag, n, k, a, lda, x, incx )
call ztbmv( uplo, trans, diag, n, k, a, lda, x, incx )
```

Fortran 95:

```
call tbmv(a, x [,uplo] [,trans] [,diag])
```

説明

?tbmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$x := a*x$ 、 $x := a'*x$ 、または $x := \text{conjg}(a')*x$

x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の、 $(k + 1)$ 個の対角成分を持つ、上位または下位の三角帯行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$x := a * x$
T または t	$x := a' * x$
C または c	$x := \text{conjg}(a') * x$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

k INTEGER。*uplo* = 'U' または 'u' と指定した場合のエントリーでは、 k は行列 a の優対角成分の数を指定する。*uplo* = 'L' または 'l' と指定した場合のエントリーでは、 k は a の劣対角成分の数を指定する。 k の値は、 $0 \leq k$ を満たさなければならない。

a REAL (*stbmv* の場合)
 DOUBLE PRECISION (*dtbmv* の場合)
 COMPLEX (*ctbmv* の場合)
 DOUBLE COMPLEX (*ztbmv* の場合)

配列、次元は (lda, n) 。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 a の先頭の $(k + 1) \times n$ の部分には、係数行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 $(k + 1)$ にくるように、また最初の優対角成分が行 k の位置 2 から始まるように、列ごとに与えなければならない。配列 a の左上の $k \times k$ の三角は、参照されない。次に示すプログラムの一部は、上三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```

do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix(i, j)
  10 continue
  20 continue

```

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $(k+1) \times n$ の部分に、係数行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の $k \times k$ の三角は、参照されない。次に示すプログラムの一部は、下三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```

do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min(n, j + k)
    a(m + i, j) = matrix(i, j)
  10 continue
  20 continue

```

diag = 'U' または 'u' と指定した場合は、行列の対角成分に対応する配列 *a* の成分は参照されないが、1 とみなされる。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $(k+1)$ 以上でなければならない。
<i>x</i>	REAL (<i>stbmv</i> の場合) DOUBLE PRECISION (<i>dtbmv</i> の場合) COMPLEX (<i>ctbmv</i> の場合) DOUBLE COMPLEX (<i>ztbmv</i> の場合) 配列、次元は $(1 + (n-1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。

出力パラメーター

<i>x</i>	変換されたベクトル <i>x</i> によって上書きされる。
----------	--------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tbmv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(k+1, n)$ の配列 <i>A</i> を格納する。
<i>x</i>	長さ (n) のベクトルを格納する。

<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<code>diag</code>	'N' または 'U' でなければならない。デフォルト値は 'N'。

?tbsv

係数が三角帯行列に格納されている連立1次方程式を解く。

構文

Fortran 77:

```
call stbsv( uplo, trans, diag, n, k, a, lda, x, incx )
call dtbsv( uplo, trans, diag, n, k, a, lda, x, incx )
call ctbsv( uplo, trans, diag, n, k, a, lda, x, incx )
call ztbsv( uplo, trans, diag, n, k, a, lda, x, incx )
```

Fortran 95:

```
call tbsv(a, x [,uplo] [,trans] [,diag])
```

説明

?tbsv ルーチンは、次に示す連立方程式のいずれかを解く。

$a*x = b$ 、 $a'*x = b$ 、または $\text{conjg}(a')*x = b$

b と x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の、 $(k+1)$ 個の対角成分を持つ、上位または下位の三角帯行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンを呼び出す前に実行しなければならない。

入力パラメーター

`uplo` CHARACTER*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<code>uplo</code> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

`trans` CHARACTER*1。次に示すように、実行する演算を指定する。

<code>trans</code> の値	実行する演算
N または n	$a*x = b$
T または t	$a'*x = b$

<i>trans</i> の値	実行する演算
C または c	$\text{conjg}(a') * x = b$

diag CHARACTER*1。次に示すように、*a* が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 <i>a</i>
U または u	行列 <i>a</i> は、単位三角行列とみなされる。
N または n	行列 <i>a</i> は、単位三角行列とはみなされない。

n INTEGER。行列 *a* の次数を指定する。*n* の値は、ゼロ以上でなければならない。

k INTEGER。*uplo* = 'U' または 'u' と指定した場合のエントリーでは、*k* は行列 *a* の優対角成分の数を指定する。*uplo* = 'L' または 'l' と指定した場合のエントリーでは、*k* は *a* の劣対角成分の数を指定する。*k* の値は、 $0 \leq k$ を満たさなければならない。

a REAL (*stbsv* の場合)
DOUBLE PRECISION (*dtbsv* の場合)
COMPLEX (*ctbsv* の場合)
DOUBLE COMPLEX (*ztbsv* の場合)

配列、次元は (*lda*, *n*)。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $(k+1) \times n$ の部分には、係数行列の上三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 $(k+1)$ にくるように、また最初の優対角成分が行 *k* の位置 2 から始まるように、列ごとに与えなければならない。配列 *a* の左上の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、上三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = k + 1 - j
  do 10, i = max(1, j - k), j
    a(m + i, j) = matrix(i, j)
  10 continue
20 continue
```

uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $(k+1) \times n$ の部分に、係数行列の下三角帯部分を格納しなければならない。この行列には、行列の主対角成分が配列の行 1 にくるように、また最初の劣対角成分が行 2 の位置 1 から始まるように、列ごとに与えなければならない。配列 *a* の右下の $k \times k$ の三角は、参照されない。

次に示すプログラムの一部は、下三角帯行列を通常のフル格納形式の行列から帯格納形式に転送するためのものである。

```
do 20, j = 1, n
  m = 1 - j
  do 10, i = j, min(n, j + k)
```

```

      a(m + i, j) = matrix (i, j)
      10 continue
      20 continue

```

diag = 'U' または 'u' と指定した場合、行列の対角成分に対応する配列 *a* の成分は参照されないが、1 とみなされる。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。*lda* の値は、 $(k+1)$ 以上でなければならない。

x REAL (*stbsv* の場合)
 DOUBLE PRECISION (*dtbsv* の場合)
 COMPLEX (*ctbsv* の場合)
 DOUBLE COMPLEX (*ztbsv* の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 *x* に *n* 個の成分からなる右辺のベクトル *b* を格納しなければならない。

incx INTEGER。 *x* の成分に対する増分を指定する。*incx* の値は、ゼロであってはならない。

出力パラメーター

x 解として得られたベクトル *x* によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tbsv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ $(k+1, n)$ の配列 <i>A</i> を格納する。
<i>x</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

?tpmv

圧縮形式の三角行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```

call stpmv( uplo, trans, diag, n, ap, x, incx )
call dtpmv( uplo, trans, diag, n, ap, x, incx )

```

```
call ctpmv( uplo, trans, diag, n, ap, x, incx )
call ztpmv( uplo, trans, diag, n, ap, x, incx )
```

Fortran 95:

```
call tpmv(a, x [,uplo] [,trans] [,diag])
```

説明

?tpmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$x := a*x$ 、 $x := a'*x$ 、または $x := \text{conjg}(a')*x$

x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の、圧縮格納形式の上三角行列または下三角行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 a が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$x := a*x$
T または t	$x := a'*x$
C または c	$x := \text{conjg}(a')*x$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

ap REAL (stpmv の場合)
DOUBLE PRECISION (dtpmv の場合)
COMPLEX (ctpmv の場合)
DOUBLE COMPLEX (ztpmv の場合)

配列、次元は $((n*(n+1))/2)$ 以上。*uplo* = 'U' または 'u' と指定した場合は、このルーチンに入る前に、*ap*(1) に $a(1, 1)$ が、*ap*(2) に $a(1, 2)$ が、*ap*(3) に $a(2, 2)$ がそれぞれ格納されるように、配列 *ap* には、順番に圧縮された上三角行列を列ごとに格納しなければならない

い。また、`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、`ap(1)` に $a(1, 1)$ が、`ap(2)` に $a(2, 1)$ が、`ap(3)` に $a(3, 1)$ がそれぞれ格納されるように、配列 `ap` には、順番に圧縮された下三角行列を列ごとに格納しなければならない。 `ap = 'U'` または `'u'` と指定した場合、 a の対角成分は参照されないが、1 とみなされる。

`x` REAL (`stpmv` の場合)
 DOUBLE PRECISION (`dtpmv` の場合)
 COMPLEX (`ctpmv` の場合)
 DOUBLE COMPLEX (`ztpmv` の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 `x` に n 個の成分を持つベクトル x を格納しなければならない。

`incx` INTEGER。 `x` の成分に対する増分を指定する。 `incx` の値は、ゼロであってはならない。

出力パラメーター

`x` 変換されたベクトル x によって書き換えられる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tpmv` のインターフェイスの詳細を以下に示す。

`a` サイズ $(n * (n + 1) / 2)$ の配列 A を格納する。

`x` 長さ (n) のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

`trans` 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

`diag` 'N' または 'U' でなければならない。デフォルト値は 'N'。

?tpsv

係数が圧縮形式の三角行列に格納されている連立 1 次方程式を解く。

構文

Fortran 77:

```
call stpsv( uplo, trans, diag, n, ap, x, incx )
call dtpsv( uplo, trans, diag, n, ap, x, incx )
call ctpsv( uplo, trans, diag, n, ap, x, incx )
call ztpsv( uplo, trans, diag, n, ap, x, incx )
```

Fortran 95:

```
call tpsv(a, x [,uplo] [,trans] [,diag])
```

説明

?tpsv ルーチンは、次に示す連立方程式のいずれかを解く。

$$a*x = b, a'*x = b, \text{ または } \text{conjg}(a')*x = b$$

b と x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の、圧縮格納形式の上三角行列または下三角行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンと呼び出す前に実行しなければならない。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 a が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$a*x = b$
T または t	$a'*x = b$
C または c	$\text{conjg}(a')*x = b$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

ap REAL (stpsv の場合)
DOUBLE PRECISION (dtpsv の場合)
COMPLEX (ctpsv の場合)
DOUBLE COMPLEX (ztpsv の場合)

配列、次元は $((n*(n+1))/2)$ 以上。 $uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、 $ap(1)$ に $a(1, 1)$ が、 $ap(2)$ に $a(1, 2)$ が、 $ap(3)$ に $a(2, 2)$ がそれぞれ格納されるように、配列 ap には、順番に圧縮された上三角行列を列ごとに格納しなければならない

い。また、`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、`ap(1)` に $a(1, 1)$ が、`ap(2)` に $a(2, 1)$ が、`ap(3)` に $a(3, 1)$ がそれぞれ格納されるように、配列 `ap` には、順番に圧縮された下三角行列を列ごとに格納しなければならない。 `ap = 'U'` または `'u'` と指定した場合、 a の対角成分は参照されないが、1 とみなされる。

`x` REAL (`stpsv` の場合)
 DOUBLE PRECISION (`dtpsv` の場合)
 COMPLEX (`ctpsv` の場合)
 DOUBLE COMPLEX (`ztpsv` の場合)

配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 `x` に n 個の成分からなる右辺のベクトル `b` を格納しなければならない。

`incx` INTEGER。 `x` の成分に対する増分を指定する。 `incx` の値は、ゼロであってはならない。

出力パラメーター

`x` 解として得られたベクトル `x` によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tpsv` のインターフェイスの詳細を以下に示す。

`a` サイズ $(n * (n + 1) / 2)$ の配列 `A` を格納する。

`x` 長さ (n) のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

`trans` 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

`diag` 'N' または 'U' でなければならない。デフォルト値は 'N'。

?trmv

三角行列を使用して行列-ベクトルの積を計算する。

構文

Fortran 77:

```
call strmv( uplo, trans, diag, n, a, lda, x, incx )
call dtrmv( uplo, trans, diag, n, a, lda, x, incx )
call ctrmv( uplo, trans, diag, n, a, lda, x, incx )
call ztrmv( uplo, trans, diag, n, a, lda, x, incx )
```

Fortran 95:

```
call trmv(a, x [,uplo] [,trans] [,diag])
```

説明

?trmv ルーチンは、次のように定義される行列 - ベクトル演算のいずれかを実行する。

$x := a*x$ 、 $x := a'*x$ 、または $x := \text{conjg}(a')*x$

x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の上三角行列または下三角行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 a が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$x := a*x$
T または t	$x := a'*x$
C または c	$x := \text{conjg}(a')*x$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

a REAL (strmv の場合)
DOUBLE PRECISION (dtrmv の場合)
COMPLEX (ctrmv の場合)
DOUBLE COMPLEX (ztrmv の場合)

配列、次元は (lda, n) 以上。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の上三角部分に上三角行列を格納しなければならず、 a の厳密な下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の下三角部分に下三角行列を格納しなければならず、 a の厳密な

上三角部分は参照されない。

diag = 'U' または 'u' と指定した場合は、*a* の対角成分も参照されず、1 とみなされる。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	REAL (<i>strmv</i> の場合) DOUBLE PRECISION (<i>dtrmv</i> の場合) COMPLEX (<i>ctrmv</i> の場合) DOUBLE COMPLEX (<i>ztrmv</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。

出力パラメーター

<i>x</i>	変換されたベクトル <i>x</i> によって上書きされる。
----------	--------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *trmv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>x</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

?trsv

係数が三角行列に格納されている連立 1 次方程式を解く。

構文

Fortran 77:

```
call strsv( uplo, trans, diag, n, a, lda, x, incx )
call dtrsv( uplo, trans, diag, n, a, lda, x, incx )
call ctrsv( uplo, trans, diag, n, a, lda, x, incx )
call ztrsv( uplo, trans, diag, n, a, lda, x, incx )
```

Fortran 95:

```
call trsv(a, x [,uplo] [,trans] [,diag])
```

説明

?trsv ルーチンは、次に示す連立方程式のいずれかを解く。

$$a*x = b, a'*x = b, \text{ または } \text{conjg}(a')*x = b$$

b と x は、 n 個の成分を持つベクトルである。

a は、 $n \times n$ 単位あるいは非単位の上三角行列または下三角行列である。

このルーチンは、特異点あるいは近似特異点はテストしない。この種のテストは、このルーチンを呼び出す前に実行しなければならない。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	上三角行列。
L または l	下三角行列。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$a*x = b$
T または t	$a'*x = b$
C または c	$\text{conjg}(a')*x = b$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

n INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。

a REAL (strsv の場合)
DOUBLE PRECISION (dtrsv の場合)
COMPLEX (ctrsv の場合)
DOUBLE COMPLEX (ztrsv の場合)

配列、次元は (lda, n)。

uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の上三角部分に上三角行列を格納しなければならず、 a の厳密な下三角部分は参照されない。また、*uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 a の先頭の

$n \times n$ の下三角部分に下三角行列を格納しなければならない、 a の厳密な上三角部分は参照されない。 $diag = 'U'$ または $'u'$ と指定した場合は、 a の対角成分も参照されず、1 とみなされる。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	REAL (<i>strsv</i> の場合) DOUBLE PRECISION (<i>dtrsv</i> の場合) COMPLEX (<i>ctrsv</i> の場合) DOUBLE COMPLEX (<i>ztrsv</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分からなる右辺のベクトル b を格納しなければならない。
<i>incx</i>	INTEGER。 x の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。

出力パラメーター

<i>x</i>	解として得られたベクトル x によって上書きされる。
----------	------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *trsv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>x</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

BLAS レベル 3 のルーチン

BLAS レベル 3 のルーチンは、行列 - 行列演算を実行する。表 2-3 に、BLAS レベル 3 のルーチンのグループと、それらに関連するデータ型を示す。

表 2-3 BLAS レベル 3 のルーチングループおよびそのデータ型

ルーチングループ	データ型	説明
?gemm	s, d, c, z	一般行列での行列 - 行列の積
?hemm	c, z	エルミート行列での行列 - 行列の積
?herk	c, z	エルミート行列の階数 k の更新
?her2k	c, z	エルミート行列の階数 2k の更新
?symm	s, d, c, z	対称行列での行列 - 行列の積
?syrk	s, d, c, z	対称行列の階数 k の更新
?syr2k	s, d, c, z	対称行列の階数 2k の更新
?trmm	s, d, c, z	三角行列での行列 - 行列の積
?trsm	s, d, c, z	三角行列に対する行列 - 行列による 1 次方程式の解

インテル® MKL の対称型マルチプロセッシング・バージョン

多くのアプリケーションでは、BLAS レベル 3 のルーチンを実行するのに多くの時間を費やしている。インテル MKL に組み込まれている対称型マルチプロセッシング (SMP) 機能を使用すると、システム上で利用可能なプロセッサの数に応じて、この時間を短縮できる。ユーザー側でプログラミングしなくても、プロセッサの並行使用に基づいて性能を向上できる。

MKL は、次の手法を使用して、性能の向上を実現している。

- BLAS レベル 3 の行列 - 行列関数による演算により、データの参照をローカルで実行し、キャッシュ・メモリーの利用度を改善し、メモリーバスへの依存度を低減する方法でコードを再構築できる。
- 上に説明した方法でコードが効率よくブロック化された後、行列の 1 つが複数のプロセッサに分配され、2 番目の行列で乗算される。このような分配により、キャッシュ管理の効率が改善され、メモリーバスへの依存度が低減され、プロセッサ数に応じた優れた結果が得られる。

?gemm

スカラー - 行列 - 行列の積を計算し、その結果をスカラー - 行列の積に加える。

構文

Fortran 77:

```
call sgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
call dgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
```

```
call cgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
call zgemm(transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc)
```

Fortran 95:

```
call gemm(a, b, c [,transa] [,transb] [,alpha] [,beta])
```

説明

?gemm ルーチンは、一般行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * \text{op}(a) * \text{op}(b) + \beta * c,$$

op(x) は、op(x) = x または op(x) = x' または op(x) = conjg(x') のいずれかである。

alpha と beta は、スカラーである。

a、b、および c は、行列である。

op(a) は、 $m \times k$ の行列である。

op(b) は、 $n \times k$ の行列である。

c は、 $m \times n$ の行列である。

入力パラメーター

transa CHARACTER*1。次に示すように、行列の乗算で使用する op(a) の形式を指定する。

<i>transa</i> の値	op(a) の形式
N または n	op(a) = a
T または t	op(a) = a'
C または c	op(a) = conjg(a')

transb CHARACTER*1。次に示すように、行列の乗算で使用する op(b) の形式を指定する。

<i>transb</i> の値	op(b) の形式
N または n	op(b) = b
T または t	op(b) = b'
C または c	op(b) = conjg(b')

m INTEGER。行列 op(a) の行数と行列 c の行数を指定する。*m* の値は、ゼロ以上でなければならない。

n INTEGER。行列 op(b) の列数と行列 c の列数を指定する。*n* の値は、ゼロ以上でなければならない。

k INTEGER。行列 op(a) の列数と行列 op(b) の行数を指定する。*k* の値はゼロ以上でなければならない。

<i>alpha</i>	<p>REAL (sgemm の場合) DOUBLE PRECISION (dgemm の場合) COMPLEX (cgemm の場合) DOUBLE COMPLEX (zgemm の場合)</p> <p>スカラー <i>alpha</i> を指定する。</p>
<i>a</i>	<p>REAL (sgemm の場合) DOUBLE PRECISION (dgemm の場合) COMPLEX (cgemm の場合) DOUBLE COMPLEX (zgemm の場合)</p> <p>配列、次元は (<i>lda</i>, <i>ka</i>)。 <i>ka</i> は <i>transa</i> = 'N' または 'n' の場合は <i>k</i> に、そうでない場合は <i>m</i> になる。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $m \times k$ の部分に行列 <i>a</i> を格納しなければならない。そうでない場合は、配列 <i>a</i> の先頭の $k \times m$ の部分に行列 <i>a</i> を格納しなければならない。</p>
<i>lda</i>	<p>INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、<i>a</i> の第 1 次元を指定する。 <i>transa</i> = 'N' または 'n' の場合、<i>lda</i> は $\max(1, m)$ 以上でなければならない。そうでない場合、<i>lda</i> は $\max(1, k)$ 以上でなければならない。</p>
<i>b</i>	<p>REAL (sgemm の場合) DOUBLE PRECISION (dgemm の場合) COMPLEX (cgemm の場合) DOUBLE COMPLEX (zgemm の場合)</p> <p>配列、次元は (<i>ldb</i>, <i>kb</i>)。 <i>kb</i> は <i>transb</i> = 'N' または 'n' の場合は <i>n</i> に、そうでない場合は <i>k</i> になる。 <i>transb</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $n \times k$ の部分に行列 <i>b</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $n \times k$ の部分に行列 <i>b</i> を格納しなければならない。</p>
<i>ldb</i>	<p>INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、<i>b</i> の第 1 次元を指定する。 <i>transb</i> = 'N' または 'n' の場合、<i>ldb</i> は $\max(1, k)$ 以上でなければならない。そうでない場合、<i>ldb</i> は $\max(1, n)$ 以上でなければならない。</p>
<i>beta</i>	<p>REAL (sgemm の場合) DOUBLE PRECISION (dgemm の場合) COMPLEX (cgemm の場合) DOUBLE COMPLEX (zgemm の場合)</p> <p>スカラー <i>beta</i> を指定する。 <i>beta</i> にゼロを設定した場合には、<i>c</i> を設定する必要はない。</p>
<i>c</i>	<p>REAL (sgemm の場合) DOUBLE PRECISION (dgemm の場合) COMPLEX (cgemm の場合) DOUBLE COMPLEX (zgemm の場合)</p> <p>配列、次元は (<i>ldc</i>, <i>n</i>)。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>c</i> を格納しなければならない。ただし、<i>beta</i> がゼロの場合は、<i>c</i> を設定する必要はない。</p>

ldc INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

c $m \times n$ の行列 ($\alpha * \text{op}(a) * \text{op}(b) + \beta * c$) によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gemm` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>ma</i> , <i>ka</i>) の行列 <i>A</i> を格納する。ここで、 $ka = k$ (<i>transa</i> = 'N' の場合) $ka = m$ (それ以外の場合) $ma = m$ (<i>transa</i> = 'N' の場合) $ma = k$ (それ以外の場合)
<i>b</i>	サイズ (<i>mb</i> , <i>kb</i>) の行列 <i>B</i> を格納する。ここで、 $kb = n$ (<i>transb</i> = 'N' の場合) $kb = k$ (それ以外の場合) $mb = k$ (<i>transb</i> = 'N' の場合) $mb = n$ (それ以外の場合)
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>transa</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>transb</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?hemm

スカラー - 行列 - 行列の積を計算し (行列オペランドのいずれかはエルミート行列)、その結果をスカラー - 行列の積に加える。

構文

Fortran 77:

```
call CHEMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
call zHEMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
```

Fortran 95:

```
call hemm(a, b, c [,side] [,uplo] [,alpha] [,beta])
```

説明

?hemm ルーチンは、エルミート行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * b + \beta * c$$

または

$$c := \alpha * b * a + \beta * c,$$

α と β は、スカラーである。

a は、エルミート行列である。

b と c は、 $m \times n$ の行列である。

入力パラメーター

side CHARACTER*1。次に示すように、演算でエルミート行列 a が左と右のどちらにくるかを指定する。

<i>side</i> の値	実行する演算
L または l	$c := \alpha * a * b + \beta * c$
R または r	$c := \alpha * b * a + \beta * c$

uplo CHARACTER*1。次に示すように、エルミート行列 a の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 a の部分
U または u	エルミート行列の上三角部分だけが参照される。
L または l	エルミート行列の下三角部分だけが参照される。

m INTEGER。行列 c の行数を指定する。 m の値は、ゼロ以上でなければならない。

n INTEGER。行列 c の列数を指定する。 n の値は、ゼロ以上でなければならない。

alpha COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)
スカラー α を指定する。

a COMPLEX (chemm の場合)
DOUBLE COMPLEX (zhemm の場合)

配列、次元は (lda, ka) 。 ka は $side = 'L'$ または $'l'$ の場合に m に、そうでない場合は n になる。 $side = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、次のように、配列 a の $m \times m$ の部分にエルミート行列を格納しなければならない。すなわち、 $uplo = 'U'$ または $'u'$ の場合は、配列 a の先頭の $m \times m$ の上三角部分にエルミート行列の上三角部分を格納しなければならない。また a の厳密な下三角部分は参照されない。 $uplo = 'L'$ または $'l'$ の場合は、配列 a の先頭の $m \times m$ の下三角部分にエルミート行列の下三角部分を格納しなければならない。また a の厳密な上三角部分は参照されない。 $side = 'R'$ また

は 'r' と指定した場合は、このルーチンに入る前に、次のように、配列 a の $n \times n$ の部分にエルミート行列を格納しなければならない。すなわち、 $uplo = 'U'$ または $'u'$ の場合は、配列 a の先頭の $n \times n$ の上三角部分にエルミート行列の上三角部分を格納しなければならない、また a の厳密な下三角部分は参照されない。 $uplo = 'L'$ または $'l'$ の場合は、配列 a の先頭の $n \times n$ の下三角部分にエルミート行列の下三角部分を格納しなければならない、また a の厳密な上三角部分は参照されない。対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 $side = 'L'$ または $'l'$ の場合、 lda は $\max(1, m)$ 以上でなければならない。そうでない場合、 lda は $\max(1, n)$ 以上でなければならない。
<i>b</i>	COMPLEX (chemm の場合) DOUBLE COMPLEX (zhemm の場合) 配列、次元は (ldb, n)。このルーチンに入る前に、配列 b の先頭の $m \times n$ の部分に行列 b を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 b の第 1 次元を指定する。 ldb の値は、 $\max(1, m)$ 以上でなければならない。
<i>beta</i>	COMPLEX (chemm の場合) DOUBLE COMPLEX (zhemm の場合) スカラー $beta$ を指定する。 $beta$ にゼロを設定した場合には、 c を設定する必要はない。
<i>c</i>	COMPLEX (chemm の場合) DOUBLE COMPLEX (zhemm の場合) 配列、次元は (c, n)。このルーチンに入る前に、配列 c の先頭の $m \times n$ の部分に行列 c を格納しなければならない。ただし、 $beta$ がゼロの場合は、 c を設定する必要はない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 c の第 1 次元を指定する。 ldc の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

<i>c</i>	更新された $m \times n$ の行列によって上書きされる。
----------	-----------------------------------

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hemm のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (k, k) の行列 A を格納する。 $k = m$ ($side = 'L'$ の場合) $k = n$ (それ以外の場合)
----------	---

<i>b</i>	サイズ (m,n) の行列 <i>B</i> を格納する。
<i>c</i>	サイズ (m,n) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?herk

エルミート行列の階数 n の更新を実行する。

構文

Fortran 77:

```
CALL CHERK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
CALL ZHERK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

Fortran 95:

```
call herk(a, c [,uplo] [,trans] [,alpha] [,beta])
```

説明

?herk ルーチンは、エルミート行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * \text{conjg}(a') + \beta * c$$

または

$$c := \alpha * \text{conjg}(a') * a + \beta * c$$

alpha と *beta* は、スカラーである。

c は、 $n \times n$ のエルミート行列である。

a は、1 番目の定義では $n \times k$ の行列、2 番目の定義では $n \times k$ の行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 *c* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 <i>c</i> の部分
U または u	<i>c</i> の上三角部分だけが参照される。
L または l	<i>c</i> の下三角部分だけが参照される。

trans CHARACTER*1. 次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$c := \alpha * a * \text{conjg}(a') + \beta * c$
C または c	$c := \alpha * \text{conjg}(a') * a + \beta * c$

n INTEGER。行列 *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。 *trans* = 'N' または 'n' と指定した場合は、*k* には行列 *a* の列数を指定する。また、 *trans* = 'C' または 'c' と指定した場合は、*k* には行列 *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

alpha REAL (cherk の場合)
DOUBLE PRECISION (zherk の場合)
スカラー *alpha* を指定する。

a COMPLEX (cherk の場合)
DOUBLE COMPLEX (zherk の場合)
配列、次元は (*lda*, *ka*)。 *ka* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。 *trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times k$ の部分に行列 *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の $n \times k$ の部分に行列 *a* を格納しなければならない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。 *trans* = 'N' または 'n' の場合、*lda* は $\max(1, n)$ 以上でなければならない。そうでない場合、*lda* は $\max(1, k)$ 以上でなければならない。

beta REAL (cherk の場合)
DOUBLE PRECISION (zherk の場合)
スカラー *beta* を指定する。

c COMPLEX (cherk の場合)
DOUBLE COMPLEX (zherk の場合)
配列、次元は (*ldc*, *n*)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならず、また *c* の厳密な下三角部分は参照されない。
uplo = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の $n \times n$ の下三角部分にエルミート行列の下三角部分を格納しなければならず、また *c* の厳密な上三角部分は参照されない。
対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。

ldc INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。*ldc* の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

- c* *uplo* = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新された行列の上三角部分によって上書きされる。
- uplo* = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新された行列の下三角部分によって上書きされる。
- 対角成分の虚数部は、ゼロに設定される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *herk* のインターフェイスの詳細を以下に示す。

- a* サイズ (*ma*, *ka*) の行列 *A* を格納する。ここで、
 ka = *k* (*transa* = 'N' の場合)
 ka = *n* (それ以外の場合)
 ma = *n* (*transa* = 'N' の場合)
 ma = *k* (それ以外の場合)
- c* サイズ (*n*, *n*) の行列 *C* を格納する。
- uplo* 'U' または 'L' でなければならない。デフォルト値は 'U'。
- trans* 'N' または 'C' でなければならない。デフォルト値は 'N'。
- alpha* デフォルト値は '1' である。
- beta* デフォルト値は '1' である。

?her2k

エルミート行列の階数 $2k$ の更新を実行する。

構文

Fortran 77:

```
CALL CHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL ZHER2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
```

Fortran 95:

```
call her2k(a, b, c [,uplo] [,trans] [,alpha] [,beta])
```

説明

?her2k ルーチンは、エルミート行列を使用して階数 $2k$ の行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * \text{conjg}(b') + \text{conjg}(\alpha) * b * \text{conjg}(a') + \text{beta} * c$$

または

$$c := \alpha * \text{conjg}(b') * a + \text{conjg}(\alpha) * \text{conjg}(a') * b + \beta * c$$

α はスカラー、 β は実数のスカラーである。

c は、 $n \times n$ のエルミート行列である。

a と b は、1 番目の定義では $n \times k$ の行列、2 番目の定義では $n \times k$ の行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 c の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 c の部分
U または u	c の上三角部分だけが参照される。
L または l	c の下三角部分だけが参照される。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$c := \alpha * a * \text{conjg}(b') + \alpha * b * \text{conjg}(a') + \beta * c$
C または c	$c := \alpha * \text{conjg}(a') * b + \alpha * \text{conjg}(b') * a + \beta * c$

n INTEGER。行列 c の次数を指定する。 n の値はゼロ以上でなければならない。

k INTEGER。 $trans = 'N'$ または $'n'$ と指定した場合は、 k には行列 a の列数を指定する。また、 $trans = 'C'$ または $'c'$ と指定した場合は、 k には行列 a の行数を指定する。 k の値はゼロ以上でなければならない。

alpha COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)

スカラー α を指定する。

a COMPLEX (cher2k の場合)
DOUBLE COMPLEX (zher2k の場合)

配列、次元は (lda, ka) 。 ka は $trans = 'N'$ または $'n'$ の場合は k に、そうでない場合は n になる。 $trans = 'N'$ または $'n'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times k$ の部分に行列 a を格納しなければならない。そうでない場合は、配列 a の先頭の $n \times k$ の部分に行列 a を格納しなければならない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 $trans = 'N'$ または $'n'$ の場合、 lda は $\max(1, n)$ 以上でなければならない。そうでない場合、 lda は $\max(1, k)$ 以上でなければならない。

beta REAL (cher2k の場合)
DOUBLE PRECISION (zher2k の場合)

スカラー β を指定する。

- b* COMPLEX (cher2k の場合)
 DOUBLE COMPLEX (zher2k の場合)
- 配列、次元は (*ldb*, *kb*)。 *kb* は *trans* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。 *trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *b* の先頭の $n \times k$ の部分に行列 *b* を格納しなければならない。そうでない場合は、配列 *b* の先頭の $n \times k$ の部分に行列 *b* を格納しなければならない。
- ldb* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*b* の第 1 次元を指定する。 *trans* = 'N' または 'n' の場合、*ldb* は $\max(1, n)$ 以上でなければならない。そうでない場合、*ldb* は $\max(1, k)$ 以上でなければならない。
- c* COMPLEX (cher2k の場合)
 DOUBLE COMPLEX (zher2k の場合)
- 配列、次元は (*ldc*, *n*)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の $n \times n$ の上三角部分に、エルミート行列の上三角部分を格納しなければならない。また *c* の厳密な下三角部分は参照されない。
- uplo* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の $n \times n$ の下三角部分にエルミート行列の下三角部分を格納しなければならない。また *c* の厳密な上三角部分は参照されない。
- 対角成分の虚数部は設定する必要はない。これらは、ゼロであるとみなされる。
- ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。 *ldc* の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

- c* *uplo* = 'U' または 'u' と指定した場合は、配列 *c* の上三角部分が、更新された行列の上三角部分によって上書きされる。
- uplo* = 'L' または 'l' と指定した場合は、配列 *c* の下三角部分が、更新された行列の下三角部分によって上書きされる。
- 対角成分の虚数部は、ゼロに設定される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン her2k のインターフェイスの詳細を以下に示す。

- a* サイズ (*ma*, *ka*) の行列 *A* を格納する。ここで、
 ka = *k* (*trans* = 'N' の場合)
 ka = *n* (それ以外の場合)
 ma = *n* (*trans* = 'N' の場合)
 ma = *k* (それ以外の場合)

<i>b</i>	サイズ (<i>mb, kb</i>) の行列 <i>B</i> を格納する。ここで、 <i>kb</i> = <i>k</i> (<i>trans</i> = 'N' の場合) <i>kb</i> = <i>n</i> (それ以外の場合) <i>mb</i> = <i>n</i> (<i>trans</i> = 'N' の場合) <i>mb</i> = <i>k</i> (それ以外の場合)
<i>c</i>	サイズ (<i>n, n</i>) の行列 <i>C</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?symm

スカラー - 行列 - 行列の積 (行列オペランドのいずれかは対称行列) を計算し、その結果をスカラー - 行列の積に加える。

構文

Fortran 77:

```
CALL sSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL dSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL cSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
CALL zSYMM( SIDE, UPLO, M, N, ALPHA, A, LDA, B, LDB, BETA, C, LDC )
```

Fortran 95:

```
call symm(a, b, c [,side] [,uplo] [,alpha] [,beta])
```

説明

?symm ルーチンは、対称行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$C := \alpha * A * B + \beta * C,$$

または

$$c := \alpha * b * a + \beta * c$$

alpha と *beta* は、スカラーである。

a は、対称行列である。

b と *c* は、 $m \times n$ の行列である。

入力パラメーター

side CHARACTER*1。次に示すように、演算で対称行列 *a* が左と右のどちらにくるか指定する。

<i>side</i> の値	実行する演算
L または l	$c := \alpha * a * b + \beta * c$
R または r	$c := \alpha * b * a + \beta * c$

uplo CHARACTER*1。次に示すように、対称行列 *a* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される配列 <i>a</i> の部分
U または u	対称行列の上三角部分だけが参照される。
L または l	対称行列の下三角部分だけが参照される。

m INTEGER。行列 *c* の行数を指定する。*m* の値は、ゼロ以上でなければならない。

n INTEGER。行列 *c* の列数を指定する。*n* の値はゼロ以上でなければならない。

alpha REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)
スカラー *alpha* を指定する。

a REAL (ssymm の場合)
DOUBLE PRECISION (dsymm の場合)
COMPLEX (csymm の場合)
DOUBLE COMPLEX (zsymm の場合)

配列、次元は (*lda*, *ka*)。 *ka* は *side* = 'L' または 'l' の場合は *m* に、そうでない場合は *n* になる。 *side* = 'L' または 'l' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の $m \times m$ の部分に対称行列を格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の $m \times m$ の上三角部分に対称行列の上三角部分を格納しなければならない、また *a* の厳密な下三角部分は参照されない。また、*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の $m \times m$ の下三角部分に対称行列の下三角部分を格納しなければならない、また *a* の厳密な上三角部分は参照されない。

side = 'R' または 'r' と指定した場合は、このルーチンに入る前に、次のように、配列 *a* の $n \times n$ の部分に対称行列を格納しなければならない。すなわち、*uplo* = 'U' または 'u' の場合は、配列 *a* の先頭の $n \times n$ の上三角部分に対称行列の上三角部分を格納しなければならない、また *a* の厳密な下三角部分は参照されない。また、*uplo* = 'L' または 'l' の場合は、配列 *a* の先頭の $n \times n$ の下三角部分に対称行列の下三角部分を格納しなければならない、また *a* の厳密な上三角部分は参照されない。

<i>lda</i>	INTEGER。呼び出し元の(サブ)プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>side</i> = 'L' または '1' の場合、 <i>lda</i> は $\max(1, m)$ 以上でなければならない。そうでない場合、 <i>lda</i> は $\max(1, n)$ 以上でなければならない。
<i>b</i>	REAL (<i>ssymm</i> の場合) DOUBLE PRECISION (<i>dsymm</i> の場合) COMPLEX (<i>csymm</i> の場合) DOUBLE COMPLEX (<i>zsymm</i> の場合) 配列、次元は (<i>ldb</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>b</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の(サブ)プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。 <i>ldb</i> の値は、 $\max(1, m)$ 以上でなければならない。
<i>beta</i>	REAL (<i>ssymm</i> の場合) DOUBLE PRECISION (<i>dsymm</i> の場合) COMPLEX (<i>csymm</i> の場合) DOUBLE COMPLEX (<i>zsymm</i> の場合) スカラー <i>beta</i> を指定する。 <i>beta</i> にゼロを設定した場合には、 <i>c</i> を設定する必要はない。
<i>c</i>	REAL (<i>ssymm</i> の場合) DOUBLE PRECISION (<i>dsymm</i> の場合) COMPLEX (<i>csymm</i> の場合) DOUBLE COMPLEX (<i>zsymm</i> の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>c</i> を格納しなければならない。ただし、 <i>beta</i> がゼロの場合は、 <i>c</i> を設定する必要はない。
<i>ldc</i>	INTEGER。呼び出し元の(サブ)プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。 <i>ldc</i> の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

c 更新された $m \times n$ の行列によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *symm* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>k</i> , <i>k</i>) の行列 <i>A</i> を格納する。 <i>k</i> = <i>m</i> (<i>side</i> = 'L' の場合) <i>k</i> = <i>n</i> (それ以外の場合)
<i>b</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>B</i> を格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。

<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>alpha</i>	デフォルト値は '1' である。
<i>beta</i>	デフォルト値は '1' である。

?syrk

対称行列の階数 n の更新を実行する。

構文

Fortran 77:

```
CALL sSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
CALL DSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
CALL cSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
CALL zSYRK( UPLO, TRANS, N, K, ALPHA, A, LDA, BETA, C, LDC )
```

Fortran 95:

```
call syrk(a, c [,uplo] [,trans] [,alpha] [,beta])
```

説明

?syrk ルーチンは、対称行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * a' + \beta * c$$

または

$$c := \alpha * a' * a + \beta * c$$

alpha と *beta* は、スカラーである。

c は、 $n \times n$ の対称行列である。

a は、1 番目の定義では $n \times k$ の行列、2 番目の定義では $n \times k$ の行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 *c* の上三角部分と下三角部分のどちらが参照されるかを指定する。

<i>uplo</i> の値	参照される行列 <i>c</i> の部分
U または u	<i>c</i> の上三角部分だけが参照される。
L または l	<i>c</i> の下三角部分だけが参照される。

trans CHARACTER*1. 次に示すように、実行する演算を指定する。

<i>trans</i> の値	実行する演算
N または n	$c := \alpha * a * a' + \beta * c$
T または t	$c := \alpha * a' * a + \beta * c$
C または c	$c := \alpha * a' * a + \beta * c$

n INTEGER。行列 *c* の次数を指定する。*n* の値はゼロ以上でなければならない。

k INTEGER。 *trans* = 'N' または 'n' と指定した場合は、*k* には行列 *a* の列数を指定する。また、 *trans* = 'T' または 't'、あるいは 'C' または 'c' と指定した場合は、*k* には行列 *a* の行数を指定する。*k* の値はゼロ以上でなければならない。

alpha REAL (ssyrk の場合)
 DOUBLE PRECISION (dsyrk の場合)
 COMPLEX (csyrk の場合)
 DOUBLE COMPLEX (zsyrk の場合)
 スカラー *alpha* を指定する。

a REAL (ssyrk の場合)
 DOUBLE PRECISION (dsyrk の場合)
 COMPLEX (csyrk の場合)
 DOUBLE COMPLEX (zsyrk の場合)
 配列、次元は (*lda*, *ka*)。 *ka* は *TRANS* = 'N' または 'n' の場合は *k* に、そうでない場合は *n* になる。 *trans* = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 *a* の先頭の $n \times k$ の部分に行列 *a* を格納しなければならない。そうでない場合は、配列 *a* の先頭の $n \times k$ の部分に行列 *a* を格納しなければならない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*a* の第 1 次元を指定する。
trans = 'N' または 'n' の場合、*lda* は $\max(1, n)$ 以上でなければならない。そうでない場合、*lda* は $\max(1, k)$ 以上でなければならない。

beta REAL (ssyrk の場合)
 DOUBLE PRECISION (dsyrk の場合)
 COMPLEX (csyrk の場合)
 DOUBLE COMPLEX (zsyrk の場合)
 スカラー *beta* を指定する。

c REAL (ssyrk の場合)
 DOUBLE PRECISION (dsyrk の場合)
 COMPLEX (csyrk の場合)
 DOUBLE COMPLEX (zsyrk の場合)
 配列、次元は (*ldc*, *n*)。
uplo = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 *c* の先頭の $n \times n$ の上三角部分に、対称行列の上三角部分を格納しなければならない。また *c* の厳密な下三角部分は参照されない。

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、配列 `c` の先頭の $n \times n$ の下三角部分に対称行列の下三角部分を格納しなければならない。また `c` の厳密な上三角部分は参照されない。

`ldc` INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、`c` の第 1 次元を指定する。`ldc` の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

`c` `uplo = 'U'` または `'u'` と指定した場合は、配列 `c` の上三角部分が、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、配列 `c` の下三角部分が、更新された行列の下三角部分によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syrk` のインターフェイスの詳細を以下に示す。

`a` サイズ (ma, ka) の行列 `A` を格納する。ここで、
 $ka = k$ (`transa = 'N'` の場合)
 $ka = n$ (それ以外の場合)
 $ma = n$ (`transa = 'N'` の場合)
 $ma = k$ (それ以外の場合)

`c` サイズ (n, n) の行列 `C` を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

`trans` 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

`alpha` デフォルト値は '1' である。

`beta` デフォルト値は '1' である。

?syr2k

対称行列の階数 $2k$ の更新を実行する。

構文

Fortran 77:

```
CALL sSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL dSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL cSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
CALL zSYR2K( UPLO, TRANS, N, K, ALPHA, A, LDA, b, ldb, BETA, C, LDC )
```

Fortran 95:

```
call syr2k(a, b, c [,uplo] [,trans] [,alpha] [,beta])
```

説明

?syr2k ルーチンは、対称行列を使用して階数 $2k$ の行列 - 行列演算を実行する。この演算は、次のように定義される。

$$c := \alpha * a * b' + \alpha * b * a' + \beta * c$$

または

$$c := \alpha * a' * b + \alpha * b' * a + \beta * c$$

α と β は、スカラーである。

c は、 $n \times n$ の対称行列である。

a と b は、1 番目の定義では $n \times k$ の行列、2 番目の定義では $n \times k$ の行列である。

入力パラメーター

uplo CHARACTER*1。次に示すように、配列 c の上三角部分と下三角部分のどちらが参照されるかを指定する。

uplo の値	参照される行列 c の部分
U または u	c の上三角部分だけが参照される。
L または l	c の下三角部分だけが参照される。

trans CHARACTER*1。次に示すように、実行する演算を指定する。

trans の値	実行する演算
N または n	$c := \alpha * a * b' + \alpha * b * a' + \beta * c$
T または t	$c := \alpha * a' * b + \alpha * b' * a + \beta * c$
C または c	$c := \alpha * a' * b + \alpha * b' * a + \beta * c$

n INTEGER。行列 c の次数を指定する。 n の値はゼロ以上でなければならない。

k INTEGER。 $trans = 'N'$ または $'n'$ と指定した場合は、 k には行列 a と b の列数を指定する。また、 $trans = 'T'$ または $'t'$ 、あるいは $'C'$ または $'c'$ と指定した場合は、 k には行列 a と b の行数を指定する。 k の値はゼロ以上でなければならない。

alpha REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)
スカラー α を指定する。

a REAL (ssyr2k の場合)
DOUBLE PRECISION (dsyr2k の場合)
COMPLEX (csyr2k の場合)
DOUBLE COMPLEX (zsyr2k の場合)

配列、次元は (lda, ka) 。 ka は $TRANS = 'N'$ または $'n'$ の場合は k に、そうでない場合は n になる。 $trans = 'N'$ または $'n'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times k$ の部分に行列 a を格納しなければならない。そうでない場合は、配列 a の先頭の $n \times k$ の部分に行列 a を格納しなければならない。

- lda** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。
 $trans = 'N'$ または $'n'$ の場合、 lda は $\max(1, n)$ 以上でなければならない。そうでない場合、 lda は $\max(1, k)$ 以上でなければならない。
- b** REAL (ssyr2k の場合)
 DOUBLE PRECISION (dsyr2k の場合)
 COMPLEX (csyr2k の場合)
 DOUBLE COMPLEX (zsyr2k の場合)
- 配列、次元は (ldb, kb) 。 kb は $trans = 'N'$ または $'n'$ の場合は k に、そうでない場合は n になる。 $trans = 'N'$ または $'n'$ と指定した場合は、このルーチンに入る前に、配列 b の先頭の $n \times k$ の部分に行列 b を格納しなければならない。そうでない場合は、配列 b の先頭の $n \times k$ の部分に行列 b を格納しなければならない。
- ldb** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。
 $trans = 'N'$ または $'n'$ の場合、 ldb は $\max(1, n)$ 以上でなければならない。そうでない場合、 ldb は $\max(1, k)$ 以上でなければならない。
- beta** REAL (ssyr2k の場合)
 DOUBLE PRECISION (dsyr2k の場合)
 COMPLEX (csyr2k の場合)
 DOUBLE COMPLEX (zsyr2k の場合)
- スカラー $beta$ を指定する。
- c** REAL (ssyr2k の場合)
 DOUBLE PRECISION (dsyr2k の場合)
 COMPLEX (csyr2k の場合)
 DOUBLE COMPLEX (zsyr2k の場合)
- 配列、次元は (ldc, n) 。
 $uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 c の先頭の $n \times n$ の上三角部分に、対称行列の上三角部分を格納しなければならない。また c の厳密な下三角部分は参照されない。
 $uplo = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 c の先頭の $n \times n$ の下三角部分に、対称行列の下三角部分を格納しなければならない。また c の厳密な上三角部分は参照されない。
- ldc** INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 c の第 1 次元を指定する。 ldc の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

- c** $uplo = 'U'$ または $'u'$ と指定した場合は、配列 c の上三角部分が、更新された行列の上三角部分によって上書きされる。

`uplo = 'L'` または `'l'` と指定した場合は、配列 c の下三角部分が、更新された行列の下三角部分によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syr2k` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (ma, ka) の行列 A を格納する。ここで、 $ka = k$ (<code>trans = 'N'</code> の場合) $ka = n$ (それ以外の場合) $ma = n$ (<code>trans = 'N'</code> の場合) $ma = k$ (それ以外の場合)
<code>b</code>	サイズ (mb, kb) の行列 B を格納する。ここで、 $kb = k$ (<code>trans = 'N'</code> の場合) $kb = n$ (それ以外の場合) $mb = n$ (<code>trans = 'N'</code> の場合) $mb = k$ (それ以外の場合)
<code>c</code>	サイズ (n, n) の行列 C を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<code>alpha</code>	デフォルト値は '1' である。
<code>beta</code>	デフォルト値は '1' である。

?trmm

スカラー - 行列 - 行列の積 (行列オペランドのいずれかは三角行列) を計算する。

構文

Fortran 77:

```
CALL sTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL dTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL cTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL zTRMM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
```

Fortran 95:

```
call trmm(a, b [,side] [,uplo] [,transa] [,diag] [,alpha])
```

説明

?trmm ルーチンは、三角行列を使用して行列 - 行列演算を実行する。この演算は、次のように定義される。

$$b := \alpha * \text{op}(a) * b$$

または

$$B := \alpha * B * \text{op}(A)$$

α は、スカラーである。

b は、 $m \times n$ の行列である。

a は、単位または非単位の、上三角行列または下三角行列である。

$\text{op}(a)$ は、 $\text{op}(a) = a$ 、 $\text{op}(a) = a'$ 、または $\text{op}(a) = \text{conjg}(a')$ のいずれかである。

入力パラメーター

side CHARACTER*1。次に示すように、演算で $\text{op}(a)$ が b を左側と右側のどちらから掛け合わせるかを指定する。

<i>side</i> の値	実行する演算
L または l	$b := \alpha * \text{op}(a) * b$
R または r	$b := \alpha * b * \text{op}(a)$

uplo CHARACTER*1。次に示すように、行列 a が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	行列 a は、上三角行列。
L または l	行列 a は、下三角行列。

transa CHARACTER*1。次に示すように、行列の乗算で使用する $\text{op}(a)$ の形式を指定する。

<i>transa</i> の値	$\text{op}(a)$ の形式
N または n	$\text{op}(a) = a$
T または t	$\text{op}(a) = a'$
C または c	$\text{op}(a) = \text{conjg}(a')$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

m INTEGER。 b の行数を指定する。 m の値は、ゼロ以上でなければならない。

<i>n</i>	INTEGER。 <i>b</i> の列数を指定する。 <i>n</i> の値はゼロ以上でなければならない。
<i>alpha</i>	REAL (<i>strmm</i> の場合) DOUBLE PRECISION (<i>dtrmm</i> の場合) COMPLEX (<i>ctrmm</i> の場合) DOUBLE COMPLEX (<i>ztrmm</i> の場合) スカラー <i>alpha</i> を指定する。 <i>alpha</i> がゼロの場合は <i>a</i> は参照されず、またこのルーチンに入る前に <i>b</i> を設定する必要はない。
<i>a</i>	REAL (<i>strmm</i> の場合) DOUBLE PRECISION (<i>dtrmm</i> の場合) COMPLEX (<i>ctrmm</i> の場合) DOUBLE COMPLEX (<i>ztrmm</i> の場合) 配列、次元は (<i>lda</i> , <i>k</i>)。 <i>k</i> は <i>m</i> (<i>side</i> = 'L' または 'l' の場合) または <i>n</i> (<i>side</i> = 'R' または 'r' の場合)。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の <i>k</i> × <i>k</i> の上三角部分に上三角行列を格納しなければならない。また <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の <i>k</i> × <i>k</i> の下三角部分に下三角行列を格納しなければならない。また <i>a</i> の厳密な上三角部分は参照されない。 <i>diag</i> = 'U' または 'u' と指定した場合は、 <i>a</i> の対角成分も参照されず、1 とみなされる。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>a</i> の第 1 次元を指定する。 <i>side</i> = 'L' または 'l' の場合、 <i>lda</i> は $\max(1, m)$ 以上でなければならない。 <i>side</i> = 'R' または 'r' の場合、 <i>lda</i> は $\max(1, n)$ 以上でなければならない。
<i>b</i>	REAL (<i>strmm</i> の場合) DOUBLE PRECISION (<i>dtrmm</i> の場合) COMPLEX (<i>ctrmm</i> の場合) DOUBLE COMPLEX (<i>ztrmm</i> の場合) 配列、次元は (<i>ldb</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>b</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。 <i>ldb</i> の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

b 変換された行列によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *trmm* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (k,k) の行列 <i>A</i> を格納する。 $k = m$ (<i>side</i> = 'L' の場合) $k = n$ (それ以外の場合)
<i>b</i>	サイズ (m,n) の行列 <i>B</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>transa</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。
<i>alpha</i>	デフォルト値は '1' である。

?trsm

行列式(行列オペランドのいずれかは三角行列)を解く。

構文

Fortran 77:

```
CALL sTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL dTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL cTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
CALL zTRSM( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A, LDA, B, LDB )
```

Fortran 95:

```
call trsm(a, b [,side] [,uplo] [,transa] [,diag] [,alpha])
```

説明

?trsm ルーチンは、次の行列式のいずれかを解く。

$$\text{op}(a) * x = \text{alpha} * b$$

または

$$x * \text{op}(a) = \text{alpha} * b$$

alpha は、スカラーである。

x と *b* は、 $m \times n$ の行列である。

a は、単位または非単位の、上三角行列または下三角行列である。

op(a) は、*op(a) = a*、*op(a) = a'*、または
op(a) = conjg(a') のいずれかである。

行列 *x* は、*b* に上書きされる。

入力パラメーター

side CHARACTER*1。次に示すように、実行する演算で $op(a)$ が x の左と右のどちらにくるかを指定する。

<i>side</i> の値	実行する演算
L または l	$op(a)*x = alpha*b$
R または r	$x*op(a) = alpha*b$

uplo CHARACTER*1。次に示すように、行列 a が上三角行列と下三角行列のどちらであるかを指定する。

<i>uplo</i> の値	行列 a
U または u	行列 a は、上三角行列。
L または l	行列 a は、下三角行列。

transa CHARACTER*1。次に示すように、行列の乗算で使用する $op(a)$ の形式を指定する。

<i>transa</i> の値	$op(a)$ の形式
N または n	$op(a) = a$
T または t	$op(a) = a'$
C または c	$op(a) = conjg(a')$

diag CHARACTER*1。次に示すように、 a が単位三角行列であるかどうかを指定する。

<i>diag</i> の値	行列 a
U または u	行列 a は、単位三角行列とみなされる。
N または n	行列 a は、単位三角行列とはみなされない。

m INTEGER。 b の行数を指定する。 m の値は、ゼロ以上でなければならない。

n INTEGER。 b の列数を指定する。 n の値は、ゼロ以上でなければならない。

alpha REAL (*strsm* の場合)
 DOUBLE PRECISION (*dtrsm* の場合)
 COMPLEX (*ctrsm* の場合)
 DOUBLE COMPLEX (*ztrsm* の場合)

スカラー $alpha$ を指定する。 $alpha$ がゼロの場合は a は参照されず、またこのルーチンに入る前に b を設定する必要はない。

a REAL (*strsm* の場合)
 DOUBLE PRECISION (*dtrsm* の場合)
 COMPLEX (*ctrsm* の場合)
 DOUBLE COMPLEX (*ztrsm* の場合)

配列、次元は (lda, k) 。 k は m ($side = 'L'$ または $'l'$ の場合) または n ($side = 'R'$ または $'r'$ の場合)。 $uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $k \times k$ の上三角部分に上三角行列を格納しなければならない、また a の厳密な下三角部分は参照されない。

$uplo = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $k \times k$ の下三角部分に下三角行列を格納しなければならない、また a の厳密な上三角部分は参照されない。 $diag = 'U'$ または $'u'$ と指定した場合は、 a の対角成分も参照されず、1 とみなされる。

- lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 a の第 1 次元を指定する。 $side = 'L'$ または $'l'$ の場合、 lda は $\max(1, m)$ 以上でなければならない。 $side = 'R'$ または $'r'$ の場合、 lda は $\max(1, n)$ 以上でなければならない。
- b REAL ($strsm$ の場合)
DOUBLE PRECISION ($dtrsm$ の場合)
COMPLEX ($ctrsm$ の場合)
DOUBLE COMPLEX ($ztrsm$ の場合)
- 配列、次元は (ldb, n) 。このルーチンに入る前に、配列 b の先頭の $m \times n$ の部分に右辺の行列 b を格納しなければならない。
- ldb INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 b の第 1 次元を指定する。 ldb の値は、 $\max(1, m)$ 以上でなければならない。

出力パラメーター

- b 解の行列 x によって上書きされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン $trsm$ のインターフェイスの詳細を以下に示す。

- a サイズ (k, k) の行列 A を格納する。
 $k = m$ ($side = 'L'$ の場合)
 $k = n$ (それ以外の場合)
- b サイズ (m, n) の行列 B を格納する。
- $side$ $'L'$ または $'R'$ でなければならない。デフォルト値は $'L'$ 。
- $uplo$ $'U'$ または $'L'$ でなければならない。デフォルト値は $'U'$ 。
- $transa$ $'N'$ 、 $'C'$ 、または $'T'$ でなければならない。デフォルト値は $'N'$ 。
- $diag$ $'N'$ または $'U'$ でなければならない。デフォルト値は $'N'$ 。
- $alpha$ デフォルト値は $'1'$ である。

スパース BLAS レベル 1 のルーチンと関数

このセクションでは、BLAS レベル 1 の拡張であるスパース BLAS について説明する。スパース BLAS は、インテル® マス・カーネル・ライブラリーのリリース 2.1 からライブラリーに組み込まれたもので、圧縮形式で格納されたスパースベクトルに対して多数の一般的なベクトル演算を実行するためのルーチンと関数のグループで構成されている。

スパースベクトルは、その成分の大半がゼロのベクトルである。スパース BLAS のルーチンと関数は、ベクトルが粗 (スパース) であるのを利用するために特に導入された。この特徴を利用すると、計算時間とメモリーを大幅に節約することが可能である。ベクトルの非ゼロの成分の数が nz である場合、スパース BLAS の演算に必要な計算時間は $O(nz)$ になる。

ベクトルの引数

圧縮形式のスパースベクトル: a を配列に格納されているベクトルとし、また a の非ゼロの成分を次のように仮定する。

$$a(k_1), a(k_2), a(k_3) \dots a(k_{nz})$$

nz は、 a 内の非ゼロの成分の合計数である。

スパース BLAS では、このベクトルは 2 つの FORTRAN 配列、 x (値) と $indx$ (インデックス) を使って圧縮形式で表すことができる。それぞれの配列は、 nz 個の成分を持つ。

$$x(1)=a(k_1), x(2)=a(k_2), \dots x(nz)=a(k_{nz}),$$

$$indx(1)=k_1, indx(2)=k_2, \dots indx(nz)=k_{nz}$$

したがって、スパースベクトルは 3 つの組 ($nz, x, indx$) で完全に表せる。 nz の値として負の数やゼロをスパース BLAS ルーチンに渡した場合は、サブルーチンはいずれの配列や変数も変更しない。

フル格納形式のベクトル: スパース BLAS のルーチンでは、1 つの FORTRAN 配列に完全に格納されているベクトル引数 (フル格納形式のベクトル) も使用できる。 y がフル格納形式のベクトルとすると、この成分は連続して格納しなければならない。すなわち、最初の成分を $y(1)$ に、2 番目の成分を $y(2)$ に格納する。これは、BLAS レベル 1 の増分 $incy=1$ に対応する。フル格納形式のベクトルの増分値がスパース BLAS のルーチンや関数に引数で渡されることはない。

命名規則

BLAS ルーチンの場合と同じように、スパース BLAS のサブプログラムの名前の先頭には、関連するデータ型を決定する文字 (単精度と倍精度の実数に対しては s と d 、単精度と倍精度の複素数に対しては c と z) がそれぞれに付く。

スパース BLAS のルーチンが「密」なルーチンの拡張である場合は、サブプログラムの名前は、対応する「密」なサブプログラムの名前の末尾に i (*indexed* を表す) が付く。例えば、スパース BLAS のルーチン `saxpyi` は、BLAS のルーチン `saxpy` に、またスパース BLAS の関数 `cdotci` は BLAS の関数 `cdotc` に対応する。

ルーチンとデータ型

表 2-4 に、インテル MKL に導入されているスパース BLAS のルーチンとデータ型を示す。

表 2-4 スパース BLAS のルーチンおよびそのデータ型

ルーチン / 関数	データ型	説明
?axpyi	s, d, c, z	スカラー - ベクトルの積にベクトルを加算 (ルーチン)
?doti	s, d	ドット積 (関数)
?dotci	c, z	複素共役のドット積 (関数)
?dotui	c, z	複素非共役のドット積 (関数)
?gthr	s, d, c, z	フル格納形式のスパースベクトルを圧縮形式 nz 、 x 、 $indx$ に集積する
?gthrz	s, d, c, z	フル格納形式のスパースベクトルを圧縮形式に集積し、フル格納形式のベクトルに集積した成分にゼロを割り当てる (ルーチン)
?roti	s, d	Givens 回転 (ルーチン)
?sctr	s, d, c, z	ベクトルを圧縮形式からフル格納形式に分散させる (ルーチン)

スパースベクトルで使用できる BLAS レベル 1 のルーチン

以下に示す BLAS レベル 1 のルーチンでは、圧縮形式の配列 x を渡した場合にも正しい結果が得られる (増分は、 $incx = 1$)。

- [?asum](#) ベクトル成分の絶対値の合計。
- [?copy](#) ベクトルのコピー。
- [?nrm2](#) ベクトルのユークリッド・ノルム。
- [?scal](#) ベクトルのスケールリング。
- [i?amax](#) 最大絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最大の合計 $|Re(x(i))| + |Im(x(i))|$ を持つ成分のインデックス。
- [i?amin](#) 最小絶対値を持つ成分のインデックス。あるいは、複素数の場合は、最小の合計 $|Re(x(i))| + |Im(x(i))|$ を持つ成分のインデックス。

$i?amax$ と $i?amin$ によって返される結果 i は、圧縮形式の配列のインデックスとして解釈しなければならない。つまり、最大 (最小) 値が $x(i)$ で、これに対応するフル格納形式の配列のインデックスが $indx(i)$ になる。

また、[?rotq](#) を呼び出して Givens 回転パラメーターを計算し、その後でこれらのパラメーターをスパース BLAS のルーチン [?roti](#) に渡すのも可能である。

?axpyi

圧縮形式のスパースベクトルのスカラー倍を、フル格納形式のベクトルに加える。

構文

Fortran 77:

```
call sAxpypi( Nz, a, X, INdX, y )
call dAxpypi( Nz, a, X, INdX, y )
call cAxpypi( Nz, a, X, INdX, y )
call zAxpypi( Nz, a, X, INdX, y )
```

Fortran 95:

```
call axpyi(x, indx, y [,a])
```

説明

?axpyi ルーチンは、次のように定義されるベクトル - ベクトル演算を実行する。

$$y := a * x + y$$

a はスカラーである。

$(nz, x, indx)$ は、圧縮形式で格納されたスパースベクトルである。

y は、フル格納形式のベクトルである。

?axpyi ルーチンは、インデックスが配列 $indx$ に格納されている y の成分に対してのみ、参照あるいは変更を行う。 $indx$ の値は、明確に指定する。

入力パラメーター

nz	INTEGER。 x と $indx$ の成分の数。
a	REAL (saxpyi の場合) DOUBLE PRECISION (daxpyi の場合) COMPLEX (caxpyi の場合) DOUBLE COMPLEX (zaxpyi の場合) スカラー a を指定する。
x	REAL (saxpyi の場合) DOUBLE PRECISION (daxpyi の場合) COMPLEX (caxpyi の場合) DOUBLE COMPLEX (zaxpyi の場合) 配列、次元は nz 以上。
$indx$	INTEGER。 x の成分に対するインデックスを指定する。 配列、次元は nz 以上。

y REAL (saxpyi の場合)
 DOUBLE PRECISION (daxpyi の場合)
 COMPLEX (caxpyi の場合)
 DOUBLE COMPLEX (zaxpyi の場合)
 配列、次元は $\max_i(\text{indx}(i))$ 以上。

出力パラメーター

y 更新されたベクトル y が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン axpyi のインターフェイスの詳細を以下に示す。

x 長さ (nz) のベクトルを格納する。
 indx 長さ (nz) のベクトルを格納する。
 y 長さ (nz) のベクトルを格納する。
 a デフォルト値は '1' である。

?doti

圧縮形式の実数型スパースベクトルとフル格納形式の実ベクトルのドット積を計算する。

構文

Fortran 77:

```
res = sdoti( Nz, X, INdX, y )
```

```
res = ddoti( Nz, X, INdX, y )
```

Fortran 95:

```
res = doti(x, indx, y)
```

説明

?doti 関数は、次のように定義される x と y のドット積を返す。

$$x(1)*y(\text{indx}(1)) + x(2)*y(\text{indx}(2)) + \dots + x(\text{nz})*y(\text{indx}(\text{nz}))$$

3つの組 (nz , x , indx) には、圧縮形式で格納された実数型スパースベクトルを、 y にはフル格納形式の実ベクトルを定義する。関数は、インデックスが配列 indx に格納されている y の成分だけを参照する。 indx の値は、明確に指定する。

入力パラメーター

nz INTEGER。 x と indx の成分の数。

x	REAL (sdoti の場合) DOUBLE PRECISION (ddoti の場合) 配列、次元は nz 以上。
$indx$	INTEGER。 x の成分に対するインデックスを指定する。 配列、次元は nz 以上。
y	REAL (sdoti の場合) DOUBLE PRECISION (ddoti の場合) 配列、次元は $\max_i(indx(i))$ 以上。

出力パラメーター

res	REAL (sdoti の場合) DOUBLE PRECISION (ddoti の場合) nz が正の場合は、 x と y のドット積が格納される。そうでない場合には、 res には 0 が格納される。
-------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン doti のインターフェイスの詳細を以下に示す。

x	長さ (nz) のベクトルを格納する。
$indx$	長さ (nz) のベクトルを格納する。
y	長さ (nz) のベクトルを格納する。

?dotci

圧縮形式の複素数型スパースベクトルとフル格納形式の複素ベクトルの共役ドット積を計算する。

構文

Fortran 77:

```
res = CDOTCi( Nz, X, INdX, Y )
res = zDOTCi( Nz, X, INdX, Y )
```

Fortran 95:

```
res = dotci(x, indx, y)
```

説明

?dotci 関数は、次のように定義される x と y のドット積を返す。

$$\text{conjg}(x(1)) * y(indx(1)) + \dots + \text{conjg}(x(nz)) * y(indx(nz))$$

3つの組 $(nz, x, indx)$ には圧縮形式で格納された複素数型スパースベクトルを、 y にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列 $indx$ に格納されている y の成分だけを参照する。 $indx$ の値は、明確に指定する。

入力パラメーター

nz	INTEGER。 x と $indx$ の成分の数。
x	COMPLEX ($cdotci$ の場合) DOUBLE COMPLEX ($zdotci$ の場合) 配列、次元は nz 以上。
$indx$	INTEGER。 x の成分に対するインデックスを指定する。 配列、次元は nz 以上。
y	COMPLEX ($cdotci$ の場合) DOUBLE COMPLEX ($zdotci$ の場合) 配列、次元は $\max_i(indx(i))$ 以上。

出力パラメーター

res	COMPLEX ($cdotci$ の場合) DOUBLE COMPLEX ($zdotci$ の場合) x と y の共役ドット積が格納される (nz が正の場合)。 res には 0 が格納される (それ以外の場合)。
-------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン $dotci$ のインターフェイスの詳細を以下に示す。

x	長さ (nz) のベクトルを格納する。
$indx$	長さ (nz) のベクトルを格納する。
y	長さ (nz) のベクトルを格納する。

?dotui

圧縮形式の複素数型スパースベクトルとフル格納形式の複素ベクトルとのドット積を計算する。

構文

Fortran 77:

```
res = CDOTui( Nz, X, INdX, Y )
res = zDOTui( Nz, X, INdX, Y )
```

Fortran 95:

```
res = dotui(x, indx, y)
```

説明

?dotui 関数は、次のように定義される x と y のドット積を返す。

$$x(1)*y(indx(1)) + x(2)*y(indx(2)) + \dots + x(nz)*y(indx(nz))$$

3つの組 $(nz, x, indx)$ には圧縮形式で格納された複素数型スパースベクトルを、 y にはフル格納形式の複素ベクトルを定義する。関数は、インデックスが配列 $indx$ に格納されている y の成分だけを参照する。 $indx$ の値は、明確に指定する。

入力パラメーター

nz	INTEGER。 x と $indx$ の成分の数。
x	COMPLEX (c _{dotui} の場合) DOUBLE COMPLEX (z _{dotui} の場合) 配列、次元は nz 以上。
$indx$	INTEGER。 x の成分に対するインデックスを指定する。 配列、次元は nz 以上。
y	COMPLEX (c _{dotui} の場合) DOUBLE COMPLEX (z _{dotui} の場合) 配列、次元は $\max_i(indx(i))$ 以上。

出力パラメーター

res	COMPLEX (c _{dotui} の場合) DOUBLE COMPLEX (z _{dotui} の場合) x と y のドット積が格納される (nz が正の場合)。 res には 0 が格納される (それ以外の場合)。
-------	--

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン dotui のインターフェイスの詳細を以下に示す。

x	長さ (nz) のベクトルを格納する。
$indx$	長さ (nz) のベクトルを格納する。
y	長さ (nz) のベクトルを格納する。

?gthr

フル格納形式のスパースベクトルの成分を圧縮形式に集積する。

構文

Fortran 77:

```
call sgthr( Nz, y, X, INdX )
call dgthr( Nz, y, X, INdX )
call cgthr( Nz, y, X, INdX )
call zgthr( Nz, y, X, INdX )
```

Fortran 95:

```
res = gthr(x, indx, y)
```

説明

?gthr ルーチンは、フル格納形式のスパースベクトル y の指定の成分を、圧縮形式 (nz , x , $indx$) に集積する。ルーチンは、インデックスが配列 $indx$ に格納されている y の成分だけを参照する。

$$x(i) = y(indx(i)), (i=1, 2, \dots, nz)$$

入力パラメーター

nz	INTEGER。集積する y の成分の数。
$indx$	INTEGER。集積する成分のインデックスを指定する。配列、次元は nz 以上。
y	REAL (sgthr の場合) DOUBLE PRECISION (dgthr の場合) COMPLEX (cgthr の場合) DOUBLE COMPLEX (zgthr の場合) 配列、次元は $\max_i(indx(i))$ 以上。

出力パラメーター

x	REAL (sgthr の場合) DOUBLE PRECISION (dgthr の場合) COMPLEX (cgthr の場合) DOUBLE COMPLEX (zgthr の場合) 配列、次元は nz 以上。 圧縮形式に変換されたベクトルが格納される。
-----	---

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `dthr` のインターフェイスの詳細を以下に示す。

<code>x</code>	長さ (<code>nz</code>) のベクトルを格納する。
<code>indx</code>	長さ (<code>nz</code>) のベクトルを格納する。
<code>y</code>	長さ (<code>nz</code>) のベクトルを格納する。

?gthrz

スパースベクトルの成分を圧縮形式に集積し、これらの成分をゼロで置き換える。

構文

Fortran 77:

```
call sgthrz( Nz, y, X, INdX )
call dgthrz( Nz, y, X, INdX )
call cgthrz( Nz, y, X, INdX )
call zgthrz( Nz, y, X, INdX )
```

Fortran 95:

```
res = gthrz(x, indx, y)
```

説明

?gthrz ルーチンは、配列 `indx` で指定したインデックスを持つ成分に対して、フル格納形式のベクトル `y` から圧縮形式 (`nz, x, indx`) に集積し、その集積した `y` の成分をゼロで上書きする。`y` の指定されていない成分に対しては、参照も変更も行わない(「[?gthr](#)」も参照)。

入力パラメーター

<code>nz</code>	INTEGER。集積する <code>y</code> の成分の数。
<code>indx</code>	INTEGER。集積する成分のインデックスを指定する。配列、次元は <code>nz</code> 以上。
<code>y</code>	REAL (sgthrz の場合) DOUBLE PRECISION (dgthrz の場合) COMPLEX (cgthrz の場合) DOUBLE COMPLEX (zgthrz の場合) 配列、次元は $\max_i(\text{indx}(i))$ 以上。

出力パラメーター

<code>x</code>	REAL (sgthrz の場合) DOUBLE PRECISION (dgthrz の場合) COMPLEX (cgthrz の場合) DOUBLE COMPLEX (zgthrz の場合) 配列、次元は <code>nz</code> 以上。 圧縮形式に変換されたベクトルが格納される。
----------------	--

y 更新されたベクトル y

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `dthrz` のインターフェイスの詳細を以下に示す。

x 長さ (nz) のベクトルを格納する。
 $indx$ 長さ (nz) のベクトルを格納する。
 y 長さ (nz) のベクトルを格納する。

?roti

一方が圧縮形式であるスパーベクトルに *Givens* 回転を適用する。

構文

Fortran 77:

```
CALL SROTi( Nz, X, INdX, Y, C, S )
```

```
CALL dROTi( Nz, X, INdX, Y, C, S )
```

Fortran 95:

```
call roti(x, indx, y [,c] [,s])
```

説明

?roti ルーチンは、2つの実数ベクトル、 x (圧縮形式 $nz, x, indx$) と y (フル格納形式) の成分に *Givens* 回転を適用する。

$$x(i) = c*x(i) + s*y(indx(i))$$

$$y(indx(i)) = c*y(indx(i)) - s*x(i)$$

ルーチンは、インデックスが配列 $indx$ に格納されている y の成分だけを参照する。 $indx$ の値は、明確に指定する。

入力パラメーター

nz INTEGER。 x と $indx$ の成分の数。
 x REAL (sroti の場合)
 DOUBLE PRECISION (droti の場合)
 配列、次元は nz 以上。
 $indx$ INTEGER。 x の成分に対するインデックスを指定する。
 配列、次元は nz 以上。

<i>y</i>	REAL (<i>sroti</i> の場合) DOUBLE PRECISION (<i>droti</i> の場合) 配列、次元は $\max_i(\text{indx}(i))$ 以上。
<i>c</i>	スカラー。REAL (<i>sroti</i> の場合) DOUBLE PRECISION (<i>droti</i> の場合)
<i>s</i>	スカラー。REAL (<i>sroti</i> の場合) DOUBLE PRECISION (<i>droti</i> の場合)

出力パラメーター

x および *y* 更新された配列。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *roti* のインターフェイスの詳細を以下に示す。

<i>x</i>	長さ (<i>nz</i>) のベクトルを格納する。
<i>indx</i>	長さ (<i>nz</i>) のベクトルを格納する。
<i>y</i>	長さ (<i>nz</i>) のベクトルを格納する。
<i>c</i>	デフォルト値は '1' である。
<i>s</i>	デフォルト値は '1' である。

?sctr

圧縮形式のスパースベクトルをフル格納形式に変換する。

構文**Fortran 77:**

```
call ssctr( Nz, X, INdX, y )
call dsctr( Nz, X, INdX, y )
call csctr( Nz, X, INdX, y )
call zsctr( Nz, X, INdX, y )
```

Fortran 95:

```
call sctr(x, indx, y)
```

説明

?sctr ルーチンは、圧縮形式のスパースベクトル ($nz, x, indx$) の成分を、フル格納形式のベクトル y に分散する。ルーチンは、インデックスが配列 $indx$ に格納されている y の成分だけを変更する。

$$y(indx(i)) = x(i) \quad (i=1, 2, \dots, nz)$$

入力パラメーター

nz	INTEGER。分散する x の成分の数。
$indx$	INTEGER。分散する成分のインデックスを指定する。配列、次元は nz 以上。
x	REAL (ssctr の場合) DOUBLE PRECISION (dsctr の場合) COMPLEX (csctr の場合) DOUBLE COMPLEX (zsctr の場合) 配列、次元は nz 以上。 フル格納形式に変換するベクトルを格納する。

出力パラメーター

y	REAL (ssctr の場合) DOUBLE PRECISION (dsctr の場合) COMPLEX (csctr の場合) DOUBLE COMPLEX (zsctr の場合) 配列、次元は $\max_i(indx(i))$ 以上。 更新された成分を持つベクトル y が格納される。
-----	---

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sctr のインターフェイスの詳細を以下に示す。

x	長さ (nz) のベクトルを格納する。
$indx$	長さ (nz) のベクトルを格納する。
y	長さ (nz) のベクトルを格納する。

スパース BLAS レベル 2 およびレベル 3

このセクションでは、インテル MKL に含まれるスパース BLAS レベル 2 およびレベル 3 について説明する。スパース BLAS レベル 2 は、スパース行列と密ベクトルに対して演算を実行するためのルーチンと関数のグループで構成されている。スパース BLAS レベル 3 は、スパース行列と密行列に対して演算を実行するためのルーチンと関数のグループで構成されている。

スパース行列は、その成分の大半がゼロの行列である。インテル MKL スパース BLAS のルーチンと関数は、行列が粗 (スパース) であるのを利用するために特に導入された。この特徴を利用すると、計算時間とメモリーを大幅に節約が可能である。スパース BLAS ルーチンは、本マニュアル第 8 章の「[リバーズ・コミュニケーション・インターフェイス \(RCISS\) に基づく反復法スパースソルバー](#)」の基礎といえる。

スパース BLAS レベル 2 およびレベル 3 における命名規則

それぞれのスパース BLAS ルーチンは、プリフィックスが `mk1_` である 6 文字または 8 文字の基本名を持つ。標準インターフェイスを備えたルーチンの基本名は 6 文字で、簡易インターフェイスを備えたルーチンの基本名はテンプレートに従って 8 文字である。

```
mk1_<character code> <data> <operation>( )
mk1_<character code> <data> <mtype> <operation>( )
```

<character code> は、データ型を示すキャラクター・コードである。

s	実数、単精度
c	複素数、単精度
d	実数、倍精度
z	複素数、倍精度



注: 現在のバージョンのインテル MKL スパース BLAS では、実数データは倍精度のみサポートしている。

<data> フィールドにはスパース行列のデータ構造を指定する (「[スパース行列のデータ構造](#)」のセクションを参照)。

coo	座標形式
csr	行圧縮形式とその変形版
csc	列圧縮形式とその変形版
dia	対角形式
sky	輪郭格納形式

<operation> フィールドには演算のタイプを指定する。

mv	行列 - ベクトルの積 (レベル 2)
mm	行列 - 行列の積 (レベル 3)
sm	単一の三角法の解 (レベル 2)
sm	複数の右辺を持つ三角法の解 (レベル 3)

任意のフィールド <mtype> には行列のタイプを指定する。このフィールドは簡易インターフェイス備えたルーチンで使用される。

ge	一般行列のスパース表現
sy	対称行列の上または下三角部分のスパース表現
tr	三角行列のスパース表現

スパース行列のデータ構造

現在のバージョンのインテル MKL スパース BLAS レベル 2 およびレベル 3 では、次のスパース行列のデータ構造がサポートされる [Duff86]。

- 行圧縮形式 (CSR) とその変形版
- 列圧縮形式 (CSC)
- 座標形式
- 対角形式
- 輪郭格納形式

行列の格納形式の詳細は、付録 A の「[スパース BLAS レベル 2-3 のスパース格納形式](#)」を参照。

ルーチンおよびサポートされる演算

このセクションでは、主要な 2 つのルーチンのタイプとサポートされる演算について説明する。ここでは次の表記を使用する。

- A - スパース行列
- B と C - 密行列
- D - スケーリング用の対角行列
- x と y - 密ベクトル
- $alpha$ と $beta$ - スカラー
- $op(A)$ は次のいずれか。
 - $op(A) = A$
 - $op(A) = A' - A$ の転置
 - $op(A) = conj(A') - A$ の共役転置

[表 2-9](#) にすべてのルーチンの全リストを示す。

標準インターフェイスを備えたルーチン

インテル MKL スパース BLAS ルーチンは次の演算をサポートする。

レベル 2

- スパース行列 - 密ベクトルの積の計算
 $y := alpha * op(A) * x + beta * y$
- 単一の三角法の解の算出
 $y := alpha * inv(op(A)) * x$

レベル 3

- スパース行列 - 密行列の積の計算
 $C := alpha * op(A) * B + beta * C$
- 複数の右辺を持つスパース三角法の解の算出
 $C := alpha * inv(op(A)) * B$

これらのルーチンは、NIST スパース BLAS ライブラリーで使用されるインターフェイスとは異なる、自然なインターフェイスを備えている [Rem05]。これらの違いの詳細は「[インターフェイス考慮事項](#)」を参照のこと。

簡易インターフェイスを備えたルーチン

一部のソフトウェア・パッケージとライブラリー (インテル MKL で使用している [PARDISO パッケージ](#)、*Sparskit 2* [Saad94]、*Compaq Extended Math Library* (CXML)[CXML01]) では、異なる (以前の) CSR 形式を使用しており、簡易インターフェイスを備えたレベル 2 の演算のみサポートしている。インテル MKL は、同様の簡易インターフェイスを備えたレベル 2 ルーチン群を提供している。それぞれのルーチンは固定タイプの行列に対して演算を行う。以下の演算をサポートしている。

$y := \text{op}(A) * x$ (一般および対称行列)
 $y := \text{inv}(\text{op}(A)) * x$ (三角行列)

行列のタイプは、ルーチン名の `<mtype>` フィールドで指定される (「[スパース BLAS レベル 2 およびレベル 3 における命名規則](#)」のセクションを参照)。

これらのルーチンに対するインターフェイスの詳細は、「[インターフェイス考慮事項](#)」のセクションを参照のこと。

これらのルーチンは、次の 3 つのスパースデータ格納形式でのみ演算を行う。

PARDISO と CXML で受け入れられる CSR 形式
 CXML で受け入れられる DIA 形式
 COO 形式

上記のグループに属するルーチンは、特定の内部データ構造で動作する、同じ計算カーネルルーチンを使用する。

インターフェイス考慮事項

インテル MKL インターフェイスと NIST インターフェイスの相違点

インテル MKL スパース BLAS レベル 3 ルーチンは、次のインターフェイスを備えている。

`mkl_xyyyymm(transa, m, n, k, alpha, matdescra, arg(A), b, ldb, beta, c, ldc)` (行列 - 行列の積)
`mkl_xyyyism(transa, m, n, alpha, matdescra, arg(A), b, ldb, c, ldc)` (複数の右辺を持つ三角ソルバー)

これに相当する NIST スパース BLAS (NSB) ライブラリー・ルーチンは、次のインターフェイスを備えている。

`yyyymm(transa, m, n, k, alpha, descra, arg(A), b, ldb, beta, c, ldc, work, lwork)` (行列 - 行列の積)
`yyyism(transa, m, n, unitd, dv, alpha, descra, arg(A), b, ldb, beta, c, ldc, work, lwork)` (複数の右辺を持つ三角ソルバー)

いくつかの類似した引数は両方のライブラリーで使用される。引数 `transa` は行列を使用した演算方法を指定する。NSB ライブラリーではやや異なる ([表 2-5](#) を参照)。引数 `m` と引数 `k` はそれぞれ、行列 `A` の行数と列数を示し、`n` は行列 `C` の列数を示す。引数

α と引数 β はそれぞれ、スカラー α スカラー β を示す (β はインテル MKL の三角ソルバーで使用されていない)。引数 b と引数 c はそれぞれ、第 1 次元が ldb と ldc の矩形配列である。記号 $\text{arg}(A)$ は、 A のスパース表現を表す引数リストを指定する。

表 2-5 パラメーター *transa*

	MKL インターフェイス	NSB インターフェイス	演算
データ型	CHARACTER*1	INTEGER	
値	N または n	0	$\text{op}(A) = A$
	T または t	1	$\text{op}(A) = A'$
	C または c	2	$\text{op}(A) = A'$

引数 *matdescra* は行列 A の関連する特性を示す。これは、NSB ライブラリーの *descra* 引数に相当する (詳細は表 2-6 を参照)。

表 2-6 パラメーター *matdescra* (*descra*) に指定可能な値

	MKL インターフェイス	NSB インターフェイス	行列の性質
データ型	CHARACTER	INTEGER	
1 番目の成分	<i>matdescra</i> (1)	<i>descra</i> (1)	行列の構成
値	G	0	一般
	S	1	対称 ($A=A'$)
	H	2	エルミート ($A=\text{conjg}(A')$)
	T	3	三角
	A	4	非対称 ($A=-A'$)
	D	5	対角
2 番目の成分	<i>matdescra</i> (2)	<i>descra</i> (2)	上 / 下三角インジケーター
値	L	1	下三角
	U	2	上三角
3 番目の成分	<i>matdescra</i> (3)	<i>descra</i> (3)	主対角のタイプ
値	N	0	非単位
	U	1	単位

インテル MKL ルーチンでは、*matdescra* はいくつかの特徴を持っている。行列 - 行列と行列 - ベクトルの演算を実行するルーチンでは次の点に注意する。

一般行列 (*matdescra*(1)='G') の場合、*matdescra*(2) と *matdescra*(3) の値は無視される。

非対称 - 対称行列 (*matdescra*(1)='A') の場合、*matdescra*(3) の値は無視される。

対角行列 (*matdescra*(1)='D') の場合、*matdescra*(2) の値は無視される。

matdescra(1) が 'G' または 'T' に設定されておらず、*matdescra*(2) と *matdescra*(3) が定義されていない場合、次のデフォルト値が割り当てられる。
matdescra(2)='L' および *matdescra*(3)='N'

輪郭格納形式を使用するルーチンでは、*matdescra*(1)='G' はサポートされない。

三角ソルバーの場合、*matdescra*(1)='D' ならば *matdescra*(2) は無視される。

三角ソルバーに対してインテル MKL は `matdescra(1)=T,D` のみサポートする。

乗算ルーチンおよび三角ソルバーでは、`matdescra(3)='U'` かつスパース行列が輪郭形式でない場合、非ゼロの対角成分を非単位であってもスパース表現で格納することができる。スパース行列が輪郭形式の場合、対角成分はゼロであってもスパース表現で格納しなければならない。

現バージョンの NSB ライブラリーは、行列 - 行列の乗算では `descra(1)` のみサポートする。`descra(2)`、`descra(3)` は `descra(1)=3` の場合のみ三角ソルバーでサポートされる。

引数 `work` は `work` 配列であり、`lwork` はその次元である。これらの引数は、インテル MKL では使用されない。

引数 `unitd` と引数 `dv` は、NSB 三角ソルバーでのみ使用される。1 つ目は対角行列 D がユニタリーであるかどうかを指定する。`unitd=1` の場合、 D は単位行列である。`unitd=2` (A の行がスケールリングされている) または `unitd=3` (A の列がスケールリングされている) の場合、線形配列 `dv` はスケールリング用の対角行列 D を含む。

簡易インターフェイス

簡易インターフェイスを持つインテル MKL スパース BLAS レベル 2 ルーチンは、次のインターフェイスを備えている。

`mkl_xyyygemv(transa, m, arg(A), x, y)` (一般スパース行列に対する行列 - ベクトルの積)

`mkl_xyyysymv(uplo, transa, m, arg(A), x, y)` (対称スパース行列に対する行列 - ベクトルの積)

`mkl_xyyytrsv(uplo, transa, diag, m, arg(A), x, y)` (スパース三角行列を使用した連立方程式の解)

引数 `transa` は行列を使用した演算方法を指定する (表 2-5 を参照)。引数 `uplo` はスパース行列の上三角と下三角のどちらを考慮するかを指定する。引数 `diag` は、 A が単位三角行列であるかどうかを指定する。引数 `m` は、行列 A 内の行数であり、`arg(A)` は A のスパース表現を表す引数リストを指定する。配列 `x` は、入力ベクトルを格納し、配列 `y` は演算の実行結果を格納する。

行列 - ベクトルの乗算を行うすべてのルーチンは、行列 A のスパース表現から三角や主対角を抽出することができる。

部分行列の演算

インテル MKL スパース BLAS ルーチンの特徴の 1 つは、パラメーター `matdescra` を指定する入力スパース行列の特定の部分 (三角や主対角) に対してのみ演算を実行できることである。スパース行列 A は次のように分解されると仮定する。

$$A = L + D + U$$

L は A の厳密な下三角、 U は A の厳密な上三角、 D は主対角である。

表 2-7 は行列 - 行列の乗算ルーチンの出力行列とスパース実行列 A の $matdescra$ の値の一致を示す。類似した一致は行列 - ベクトルの乗算ルーチンに対して存在する。

表 2-7 出力行列と $matdescra$ の値の一致 (行列 - 行列の乗算ルーチン)

matdescra(1)	matdescra(2)	matdescra(3)	出力行列
G	無視される	無視される	$\alpha *_{op}(A) * B + \beta * C$
S または H	L	N	$\alpha *_{op}(L+D+L') * B + \beta * C$
S または H	L	U	$\alpha *_{op}(L+I+L') * B + \beta * C$
S または H	U	N	$\alpha *_{op}(U+D+U) * B + \beta * C$
S または H	U	U	$\alpha *_{op}(U+I+U) * B + \beta * C$
T	L	U	$\alpha *_{op}(L+I) * B + \beta * C$
T	L	N	$\alpha *_{op}(L+D) * B + \beta * C$
T	U	U	$\alpha *_{op}(U+I) * B + \beta * C$
T	U	N	$\alpha *_{op}(U+D) * B + \beta * C$
A	L	無視される	$\alpha *_{op}(L-L') * B + \beta * C$
A	U	無視される	$\alpha *_{op}(U-U) * B + \beta * C$
D	無視される	N	$\alpha * D * B + \beta * C$
D	無視される	U	$\alpha * B + \beta * C$

表 2-8 は三角ソルバーの出力行列とスパース実行列 A の $matdescra$ の値の一致を示す。

表 2-8 出力行列と $matdescra$ の値の一致 (三角ソルバー)

matdescra(1)	matdescra(2)	matdescra(3)	出力行列
T	L	N	$\alpha * \text{inv}(op(L+D)) * B$
T	L	U	$\alpha * \text{inv}(op(L+I)) * B$
T	U	N	$\alpha * \text{inv}(op(U+D)) * B$
T	U	U	$\alpha * \text{inv}(op(U+I)) * B$
D	無視される	N	$\alpha * \text{inv}(D) * B$
D	無視される	U	$\alpha * B$

三角ソルバールーチンの制限

すべてのインテル MKL 三角ソルバーには、次のような重要な制限がある。

行圧縮形式の列インデックスは、行ごとに昇順でソートされていなければならない。

列圧縮形式の行インデックスは、列ごとに昇順でソートされていなければならない。

対角形式では、スパース行列の非ゼロの対角成分の個数を格納する配列の成分は、昇順でソートされていなければならない。

スパース BLAS レベル 2 およびレベル 3 のルーチン

このセクションで後述する、スパース BLAS レベル 2 およびレベル 3 のルーチンの一覧を表 2-9 に示す。

表 2-9 スパース BLAS レベル 2 およびレベル 3 のルーチン

ルーチン / 関数	説明
レベル 2	
mkl_dcsrsv	CSR 形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
mkl_dcsrgev	CSR 形式 (PARDISO 版) で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
mkl_dcsrsvmv	CSR 形式 (PARDISO 版) で格納されている対称スパース行列の行列 - ベクトルの積を計算する。
mkl_dcscmv	CSC 形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
mkl_dcoomv	座標形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
mkl_dcoogev	座標形式で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
mkl_dcoosvmv	座標形式で格納されている対称スパース行列の行列 - ベクトルの積を計算する。
mkl_ddiamv	対角形式で格納されているスパース行列の行列 - ベクトルの積を計算する。
mkl_ddiagev	対角形式で格納されているスパース一般行列の行列 - ベクトルの積を計算する。
mkl_ddiasvmv	対角形式で格納されている対称スパース行列の行列 - ベクトルの積を計算する。
mkl_dskymv	輪郭格納形式のスパース行列の行列 - ベクトルの積を計算する。
mkl_dcsrsv	CSR 形式のスパース行列について連立 1 次方程式を解く。
mkl_dcsrtrsv	CSR 形式 (PARDISO 版) のスパース行列に対する簡易インターフェイスを備えた三角ソルバー。
mkl_dcscsv	列圧縮形式のスパース行列について連立 1 次方程式を解く。
mkl_dcoosv	座標形式のスパース行列について連立 1 次方程式を解く。
mkl_dcootrsv	座標形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバー。
mkl_ddiasv	対角形式のスパース行列について連立 1 次方程式を解く。
mkl_ddiatrsv	対角形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバー。
mkl_dskysv	輪郭形式のスパース行列について連立 1 次方程式を解く。
レベル 3	
mkl_dcsrmm	行圧縮形式で格納されたスパース行列の行列 - 行列の積を計算する。
mkl_dcscmm	列圧縮形式で格納されたスパース行列の行列 - 行列の積を計算する。
mkl_dcoomm	座標形式で格納されているスパース行列の行列 - 行列の積を計算する。

表 2-9 スパース BLAS レベル 2 およびレベル 3 のルーチン (続き)

ルーチン / 関数	説明
mkl_ddiamm	対角形式で格納されているスパース行列の行列 - 行列の積を計算する。
mkl_dskymm	輪郭形式で格納されているスパース行列の行列 - 行列の積を計算する。
mkl_dcsrsm	CSR 形式のスパース行列について連立 1 次行列方程式を解く。
dcscsm	CSC 形式のスパース行列について連立 1 次行列方程式を解く。
mkl_dcoosm	座標形式のスパース行列について連立 1 次行列方程式を解く。
mkl_ddiasm	対角形式のスパース行列について連立 1 次行列方程式を解く。
mkl_dskysm	輪郭格納形式で格納されたスパース行列について連立 1 次行列方程式を解く。

mkl_dcsrsmv

CSR 形式で格納されているスパース行列の行列 - ベクトルの積を計算する。

構文

Fortran:

```
call mkl_dcsrsmv(transa, m, k, alpha, matdescra, val, indx, pntreb, pntre,
  x, beta, y)
```

C:

```
mkl_dcsrsmv(&transa, &m, &k, &alpha, matdescra, val, indx, pntreb, pntre,
  x, &beta, y);
```

説明

mkl_dcsrsmv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

α と β は、スカラーである。

x と y は、ベクトルである。

A は CSR 形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。

$transa = 'N'$ または $'n'$ の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$

$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。

$y := \alpha * A' * x + \beta * y$

<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntrb</i> (<i>m</i>) - <i>pntrb</i> (1) である。詳細は、「CSR 形式」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと同じ。詳細は、「CSR 形式」の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが <i>pntrb</i> (<i>i</i>) - <i>pntrb</i> (1)+1 である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが <i>pntrb</i> (<i>i</i>) - <i>pntrb</i> (1) である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerE</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>k</i> 以上 ($transa = 'N'$ または $'n'$ の場合) または <i>m</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>beta</i>	REAL*8。スカラー <i>beta</i> を指定する。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上 ($transa = 'N'$ または $'n'$ の場合) または <i>k</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。

出力パラメーター

y 更新されたベクトル *y* によって上書きされる。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcsrmmv(transa, m, k, alpha, matdescra, val, indx, pntrb,
pntrb, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k
  INTEGER      indx(*), pntrb(m), pntrb(m)
  REAL*8      alpha, beta
  REAL*8      val(*), x(*), y(*)
```

Fortran 95:

```

SUBROUTINE mkl_dcsrmmv(transa, m, k, alpha, matdescra, val, indx, pntreb,
pntre, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntreb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dcsrmmv(char *transa, int *m, int *k, double *alpha, char
*matdescra, double *val, int *indx, int *pntreb, int *pntre, double
*x, double *beta, double *y);

```

mkl_dcsrgev

CSR 形式 (PARDISO 版) で格納されているスパース一般行列の行列-ベクトルの積を計算する。

構文**Fortran:**

```
call mkl_dcsrgev(transa, m, a, ia, ja, x, y)
```

C:

```
mkl_dcsrgev(&transa, &m, a, ia, ja, x, y);
```

説明

mkl_dcsrgev ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := A*x$

または

$y := A'*x$

x と y は、ベクトルである。

A は CSR 形式 (PARDISO 版) の $m \times m$ のスパース正方行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「インターフェイス」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。

transa = 'N' または 'n' の場合、行列-ベクトルの積は、次のように計算される。 $y := A*x$

transa = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。

$$y := A' * x$$

- m* INTEGER。行列 *A* の行数。
- a* REAL*8。行列 *A* の非ゼロの成分を格納する配列。長さは、配列 *A* 内の非ゼロの成分の数と等しい。詳細は、「[スパース行列の格納形式](#)」の *values* 配列の説明を参照のこと。
- ia* INTEGER。長さ *m* + 1 の配列。配列 *a* の成分のインデックスを格納する。*ia*(*i*) は配列 *a* の行 *i* の最初の非ゼロ成分のインデックスである。最後の成分 *ia*(*m* + 1) - 1 の値は、非ゼロの数 + 1 に等しい。詳細は、「[スパース行列の格納形式](#)」の *rowIndex* 配列の説明を参照のこと。
- ja* INTEGER。行列 *A* の非ゼロの各成分の列インデックスを格納する配列。長さは、配列 *a* の長さと等しい。詳細は、「[スパース行列の格納形式](#)」の *columns* 配列の説明を参照のこと。
- x* REAL*8。配列、次元は *m*。このルーチンに入る前に、配列 *x* にベクトル *x* を格納しなければならない。

出力パラメーター

- y* REAL*8。配列、次元は *m* 以上。終了時に、配列 *y* にベクトル *y* を格納しなければならない。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcsrgev( transa, m, a, ia, ja, x, y )
  CHARACTER*1  transa
  INTEGER      m
  INTEGER      ia(*), ja(*)
  REAL*8      a(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcsrgev( transa, m, a, ia, ja, x, y )
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m
  INTEGER, INTENT(IN) :: ia(*), ja(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: a(*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

C:

```
void mkl_dcsrgev(char *transa, int *m, double *a, int *ia, int *ja,
  double *x, double *y);
```

mkl_dcsrsvmv

CSR 形式 (PARDISO 版) で格納されている対称スパース行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
```

C:

```
mkl_dcsrsvmv(&uplo, &m, a, ia, ja, x, y);
```

説明

mkl_dcsrsvmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := A*x$$

または

$$y := A'*x$$

x と y は、ベクトルである。

A は CSR 形式 (PARDISO 版) の対称スパース行列の上または下三角、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<code>uplo</code>	CHARACTER*1。行列 A の上三角と下三角のどちらを使用するかを指定する。 <code>uplo = 'U'</code> または <code>'u'</code> の場合、行列 A の上三角が使用される。 <code>uplo = 'L'</code> または <code>'l'</code> の場合、行列 A の下三角が使用される。
<code>m</code>	INTEGER。行列 A の行数。
<code>a</code>	REAL*8。行列 A の非ゼロの成分を格納する配列。長さは、配列 A 内の非ゼロの成分の数と等しい。詳細は、「 スパース行列の格納形式 」の <code>values</code> 配列の説明を参照のこと。
<code>ia</code>	INTEGER。長さ $m+1$ の配列。配列 a の成分のインデックスを格納する。 $ia(i)$ は配列 a の行 i の最初の非ゼロ成分のインデックスである。最後の成分 $ia(m+1)-1$ の値は、非ゼロの数 $+1$ に等しい。詳細は、「 スパース行列の格納形式 」の <code>rowIndex</code> 配列の説明を参照のこと。
<code>ja</code>	INTEGER。行列 A の非ゼロの各成分の列インデックスを格納する配列。長さは、配列 a の長さと同じ。詳細は、「 スパース行列の格納形式 」の <code>columns</code> 配列の説明を参照のこと。
<code>x</code>	REAL*8。配列、次元は m 。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。

出力パラメーター

y REAL*8。配列、次元は m 以上。終了時に、配列 y にベクトル y を格納しなければならない。

インターフェイス**Fortran 77:**

```
SUBROUTINE mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
  CHARACTER*1  uplo
  INTEGER      m
  INTEGER      ia(*), ja(*)
  REAL*8       a(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcsrsvmv(uplo, m, a, ia, ja, x, y)
  CHARACTER(LEN=1), INTENT(IN):: uplo
  INTEGER, INTENT(IN)  :: m
  INTEGER, INTENT(IN)  :: ia(*), ja(*)
  REAL(KIND(1.0D0)), INTENT(IN)  :: a(*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

C:

```
void mkl_dcsrsvmv(char *uplo, int *m, double *a, int *ia, int *ja, double
*x, double *y);
```

mkl_dcscmv

列圧縮形式のスパース行列の行列-ベクトルの積を計算する。

構文**Fortran:**

```
call mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntreb, pntre,
x, beta, y)
```

C:

```
mkl_dcscmv(&transa, &m, &k, &alpha, matdescra, val, indx, pntreb, pntre,
x, &beta, y);
```

説明

mkl_dcscmv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

α と β は、スカラーである。

x と y は、ベクトルである。

A は列圧縮形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<code>transa</code>	CHARACTER*1。実行する演算を指定する。 <code>transa = 'N'</code> または <code>'n'</code> の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$ <code>transa = 'T'</code> 、 <code>'t'</code> または <code>'C'</code> 、 <code>'c'</code> の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<code>m</code>	INTEGER。行列 A の行数。
<code>k</code>	INTEGER。行列 A の列数。
<code>alpha</code>	REAL*8。スカラー α を指定する。
<code>matdescra</code>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<code>val</code>	REAL*8。行列 A の非ゼロの成分を格納する配列。 長さは、 <code>pntre(k) - pntrb(1)</code> である。詳細は、「 CSC 形式 」の <code>values</code> 配列の説明を参照のこと。
<code>indx</code>	INTEGER。行列 A の非ゼロの各成分の行インデックスを格納する配列。 長さは、 <code>val</code> 配列の長さと同じ。詳細は、「 CSC 形式 」の <code>rows</code> 配列の説明を参照のこと。
<code>pntrb</code>	INTEGER。長さ k の配列。配列 <code>val</code> と配列 <code>indx</code> の列 i の最初のインデックスが <code>pntrb(i) - pntrb(1) + 1</code> である行インデックスを格納する。詳細は、「 CSC 形式 」の <code>pointerB</code> 配列の説明を参照のこと。
<code>pntre</code>	INTEGER。長さ k の配列。配列 <code>val</code> と配列 <code>indx</code> の列 i の最後のインデックスが <code>pntre(i) - pntrb(1)</code> である行インデックスを格納する。詳細は、「 CSC 形式 」の <code>pointerE</code> 配列の説明を参照のこと。
<code>x</code>	REAL*8。配列、次元は k 以上 (<code>transa = 'N'</code> または <code>'n'</code> の場合) または m 以上 (それ以外の場合)。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。
<code>beta</code>	REAL*8。スカラー β を指定する。
<code>y</code>	REAL*8。配列、次元は m 以上 (<code>transa = 'N'</code> または <code>'n'</code> の場合) または k 以上 (それ以外の場合)。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。

出力パラメーター

<code>y</code>	更新されたベクトル y によって上書きされる。
----------------	---------------------------

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntrb,
pntre, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k, ldb, ldc
  INTEGER      indx(*), pntrb(m), pntre(m)
  REAL*8      alpha, beta
  REAL*8      val(*), x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dcscmv(transa, m, k, alpha, matdescra, val, indx, pntrb,
pntre, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dcscmv(char *transa, int *m, int *k, double *alpha, char
 *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
 *x, double *beta, double *y);

```

mkl_dcoomv

座標形式で格納されているスパース行列の行列-ベクトルの積を計算する。

構文

Fortran:

```

call mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind, colind, nnz,
x, beta, y)

```

C:

```

mkl_dcoomv(&transa, &m, &k, &alpha, matdescra, val, rowind, colind, &nnz,
x, &beta, y);

```

説明

mkl_dcoomv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

α と β は、スカラーである。

x と y は、ベクトルである。

A は圧縮座標形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<i>m</i>	INTEGER。行列 A の行数。
<i>k</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 A の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 A の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は k 以上 (<i>transa</i> = 'N' または 'n' の場合) または m 以上 (それ以外の場合)。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。
<i>beta</i>	REAL*8。スカラー β を指定する。
<i>y</i>	REAL*8。配列、次元は m 以上 (<i>transa</i> = 'N' または 'n' の場合) または k 以上 (それ以外の場合)。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。

出力パラメーター

y 更新されたベクトル y によって上書きされる。

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind,
colind, nnz, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8       alpha, beta
  REAL*8       val(*), x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dcoomv(transa, m, k, alpha, matdescra, val, rowind,
colind, nnz, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, k, nnz
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dcoomv(char *transa, int *m, int *k, double *alpha, char
  *matdescra, double *val, int *rowind, int *colind, int *nnz, double
  *x, double *beta, double *y);

```

mkl_dcoogemv

座標形式で格納されているスパース一般行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
```

C:

```
mkl_dcoogemv(&transa, &m, val, rowind, colind, &nnz, x, y);
```

説明

mkl_dcoogemv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := A*x$$

または

$$y := A'*x$$

x と y は、ベクトルである。

A は座標形式の $m \times m$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 $transa = 'N'$ または $'n'$ の場合、行列 - ベクトルの積は、次のように計算される。 $y := A*x$ $transa = 'T'$ 、 $'t'$ または $'c'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。 $y := A'*x$
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 <i>A</i> の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

出力パラメーター

<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。終了時に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。
----------	---

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
  CHARACTER*1  transa
  INTEGER      m, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8      val(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcoogemv(transa, m, val, rowind, colind, nnz, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

C:

```
void mkl_dcoogemv(char *transa, int *m, double *val, int *rowind, int
    *colind, int *nnz, double *x, double *y);
```

mkl_dcoosymv

座標形式で格納されている対称スパース行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
```

C:

```
mkl_dcoosymv(&uplo, &m, val, rowind, colind, &nnz, x, y);
```

説明

mkl_dcoosymv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := A*x$$

または

$$y := A'*x$$

x と y は、ベクトルである。

A は座標形式の対称スパース行列の上または下三角、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>uplo</i>	CHARACTER*1。行列 A の上三角と下三角のどちらを使用するかを指定する。 <i>uplo</i> = 'U' または 'u' の場合、行列 A の上三角が使用される。 <i>uplo</i> = 'L' または 'l' の場合、行列 A の下三角が使用される。
<i>m</i>	INTEGER。行列 A の行数。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 A の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。

<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

出力パラメーター

<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。終了時に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。
----------	---

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
  CHARACTER*1  uplo
  INTEGER      m, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8      val(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcoosymv(uplo, m, val, rowind, colind, nnz, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

C:

```
void mkl_dcoosymv(char *uplo, int *m, double *val, int *rowind, int
  *colind, int *nnz, double *x, double *y);
```

mkl_ddiamv

対角形式で格納されているスパース行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, iddiag, ndiag,
  x, beta, y)
```

C:

```
mkl_ddiamv(&transa, &m, &k, &alpha, matdescra, val, &lval, iddiag, &ndiag,
  x, &beta, y);
```

説明

mkl_ddiamv ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$y := \alpha * A * x + \beta * y$

または

$y := \alpha * A' * x + \beta * y$

α と β は、スカラーである。

x と y は、ベクトルである。

A は対角形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<i>m</i>	INTEGER。行列 A の行数。
<i>k</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescri</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 val , $lval \geq \min(m, k)$ のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ $ndiag$ の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は k 以上 (<i>transa</i> = 'N' または 'n' の場合) または m 以上 (それ以外の場合)。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。
<i>beta</i>	REAL*8。スカラー β を指定する。
<i>y</i>	REAL*8。配列、次元は m 以上 (<i>transa</i> = 'N' または 'n' の場合) または k 以上 (それ以外の場合)。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。

出力パラメーター

y 更新されたベクトル y によって上書きされる。

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, idiag,
ndiag, x, beta, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, k, lval, ndiag
  INTEGER      idiag(*)
  REAL*8       alpha, beta
  REAL*8       val(lval,*), x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_ddiamv(transa, m, k, alpha, matdescra, val, lval, idiag,
ndiag, x, beta, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, k, lval, ndiag
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_ddiamv(char *transa, int *m, int *k, double *alpha, char
*matdescra, double *val, int *lval, int *idiag, int *ndiag, double
*x, double *beta, double *y);

```

mkl_ddiagemv

対角形式で格納されているスパース一般行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_ddiagemv(transa, m, val, lval, idiag, ndiag, x, y)
```

C:

```
mkl_ddiagemv(&transa, &m, val, &lval, idiag, &ndiag, x, y);
```

説明

mkl_ddiagemv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$y := A*x$

または

$y := A'*x$

x と y は、ベクトルである。

A は対角格納形式の $m \times m$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。 $y := A*x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $y := A'*x$
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>val</i>	REAL*8。 <i>lval</i> × <i>ndiag</i> の 2 次元配列。行列 <i>A</i> の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , <i>lval</i> ≥ <i>m</i> のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 <i>A</i> の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 <i>A</i> の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

出力パラメーター

<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。終了時に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。
----------	---

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_ddiagemv(transa, m, val, lval, idiag, ndiag, x, y)
  CHARACTER*1  transa
  INTEGER      m, lval, ndiag
  INTEGER      idiag(*)
  REAL*8      val(lval,*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_ddiagemv(transa, m, val, lval, idiag, ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

C:

```
void mkl_ddiagemv(char *transa, int *m, double *val, int *lval, int
  *idiag, int *ndiag, double *x, double *y);
```

mkl_ddiasymv

対角形式で格納されている対称スパース行列の行列-ベクトルの積を計算する。

構文

Fortran:

```
call mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
```

C:

```
mkl_ddiasymv(&uplo, &m, val, &lval, idiag, &ndiag, x, y);
```

説明

mkl_ddiasymv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := A*x$$

または

$$y := A'*x$$

x と y は、ベクトルである。

A は座標形式の対称スパース行列の上または下三角、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<code>uplo</code>	CHARACTER*1。行列 A の上三角と下三角のどちらを使用するかを指定する。 <code>uplo = 'U'</code> または <code>'u'</code> の場合、行列 A の上三角が使用される。 <code>uplo = 'L'</code> または <code>'l'</code> の場合、行列 A の下三角が使用される。
<code>m</code>	INTEGER。行列 A の行数。
<code>val</code>	REAL*8。 <code>lval × ndiag</code> の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <code>values</code> 配列の説明を参照のこと。
<code>lval</code>	INTEGER。 <code>val</code> , <code>lval ≥ m</code> のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <code>lval</code> 配列の説明を参照のこと。
<code>idiag</code>	INTEGER。長さ <code>ndiag</code> の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <code>distance</code> 配列の説明を参照のこと。
<code>ndiag</code>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<code>x</code>	REAL*8。配列、次元は <code>m</code> 。このルーチンに入る前に、配列 <code>x</code> にベクトル x を格納しなければならない。

出力パラメーター

y REAL*8。配列、次元は m 以上。終了時に、配列 y にベクトル y を格納しなければならない。

インターフェイス**Fortran 77:**

```
SUBROUTINE mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
  CHARACTER*1  uplo
  INTEGER      m, lval, ndiag
  INTEGER      idiag(*)
  REAL*8      val(lval,*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_ddiasymv(uplo, m, val, lval, idiag, ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)
```

C:

```
void mkl_ddiasymv(char *uplo, int *m, double *val, int *lval, int
  *idiag, int *ndiag, double *x, double *y);
```

mkl_dskymv

輪郭格納形式のスパース行列の行列-ベクトルの積を計算する。

構文**Fortran:**

```
call mkl_dskymv(transa, m, k, alpha, matdescra, val, pntr, x, beta, y)
```

C:

```
mkl_dskymv(&transa, &m, &k, &alpha, matdescra, val, pntr, x, &beta, y);
```

説明

mkl_dskymv ルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * A * x + \beta * y$$

または

$$y := \alpha * A' * x + \beta * y$$

α と β は、スカラーである。

x と y は、ベクトルである。

A は輪郭格納形式を使用して格納された $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - ベクトルの積は、次のように計算される。 $y := \alpha * A * x + \beta * y$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $y := \alpha * A' * x + \beta * y$
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の成分のセットが輪郭プロファイル形式で格納された配列。 <i>matdescrsa</i> (2) = 'L' の場合、 <i>val</i> は行列 <i>A</i> の下三角の成分を格納する。 <i>matdescrsa</i> (2) = 'U' の場合、 <i>val</i> は行列 <i>A</i> の上三角の成分を格納する。 詳細は、「 スカイライン格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>pntr</i>	INTEGER。下三角の場合は長さ (<i>m</i> +1)、上三角の場合は長さ (<i>k</i> +1) の配列。 <i>val</i> で指定される、行列 <i>A</i> の各行 (または列) の最初の成分の位置のインデックスを格納する。詳細は、「 スカイライン格納方式 」の <i>pointers</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>k</i> 以上 (<i>transa</i> = 'N' または 'n' の場合) または <i>m</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。
<i>beta</i>	REAL*8。スカラー <i>beta</i> を指定する。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上 (<i>transa</i> = 'N' または 'n' の場合) または <i>k</i> 以上 (それ以外の場合)。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。

出力パラメーター

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dskymv(transa, m, k, alpha, matdescra, val, pntnr, x, beta,
y)
  CHARACTER*1    transa
  CHARACTER      matdescra(*)
  INTEGER        m, k
  INTEGER        pntnr(*)
  REAL*8         alpha, beta
  REAL*8         val(*), x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dskymv(transa, m, k, alpha, matdescra, val, pntnr, x, beta,
y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, k
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntnr(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dskymv (char *transa, int *m, int *k, double *alpha, char
  *matdescra, double *val, int *pntnr, double *x, double *beta, double
  *y);

```

mkl_dcsrcsv

CSR 形式のスパース行列について連立 1 次方程式を解く。

構文

Fortran:

```
call mkl_dcsrcsv(transa, m, alpha, matdescra, val, indx, pntrb, pntre, x, y)
```

C:

```
mkl_dcsrcsv(&transa, &m, &alpha, matdescra, val, indx, pntrb, pntre, x, y);
```

説明

mkl_dcsrcsv ルーチンは、CSR 形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inv}(A) * x$$

または

$$y := \alpha * \text{inv}(A') * x$$

$alpha$ は、スカラーである。

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 $transa = 'N'$ または $'n'$ の場合、 $y := alpha * inv(A) * x$ $transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、 $y := alpha * inv(A') * x$
<i>m</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー $alpha$ を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 A の非ゼロの成分を格納する配列。 長さは、 $pntrb(m) - pntrb(1)$ である。詳細は、「 CSR 形式 」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 A の非ゼロの各成分の列インデックスを格納する配列。 長さは、 <i>val</i> 配列の長さと同じ。詳細は、「 CSR 形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ m の配列。配列 <i>val</i> と配列 <i>indx</i> の行 i の最初のインデックスが $pntrb(i) - pntrb(1) + 1$ である行インデックスを格納する。詳細は、「 CSR 形式 」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrc</i>	INTEGER。長さ m の配列。配列 <i>val</i> と配列 <i>indx</i> の行 i の最後のインデックスが $pntrc(i) - pntrc(1)$ である行インデックスを格納する。詳細は、「 CSR 形式 」の <i>pointerE</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。成分には単位増分を使用してアクセスする。

出力パラメーター

y 解として得られたベクトル x を格納する。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcsrsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntrc, x, y)
```

```

CHARACTER*1   transa
CHARACTER     matdescra(*)
INTEGER       m
INTEGER       indx(*), pntrb(m), pntre(m)
REAL*8        alpha
REAL*8        val(*)
REAL*8        x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dcsrsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntre, x, y)

```

```

CHARACTER(LEN=1), INTENT(IN):: transa
INTEGER, INTENT(IN) :: m
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dcsrsv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *indx, int *pntrb, int *pntre, double *x, double
*y);

```

mkl_dcsrtrsv

CSR 形式(PARDISO 版) のスパース行列に対する
簡易インターフェイスを備えた三角ソルバー。

構文

Fortran:

```

call mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)

```

C:

```

mkl_dcsrtrsv(&uplo, &transa, &diag, &m, a, ia, ja, x, y);

```

説明

mkl_dcsrtrsv ルーチンは、PARDISO で受け入れられる CSR 形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A*y = x$$

または

$$A'*y = x$$

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>uplo</i>	CHARACTER*1。行列 <i>A</i> の上三角と下三角のどちらを使用するかを指定する。 <i>uplo</i> = 'U' または 'u' の場合、行列 <i>A</i> の上三角が使用される。 <i>uplo</i> = 'L' または 'l' の場合、行列 <i>A</i> の下三角が使用される。
<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、 $A*y = x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、 $A'*y = x$
<i>diag</i>	CHARACTER*1。 <i>A</i> が単位三角行列であるかどうかを指定する。 <i>diag</i> = 'U' または 'u' の場合、 <i>A</i> は単位三角行列とみなされる。 <i>diag</i> = 'N' または 'n' の場合、 <i>A</i> は単位三角行列とみなされない。
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>a</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、配列 <i>A</i> 内の非ゼロの成分の数と等しい。詳細は、「 スパース行列の格納形式 」の <i>values</i> 配列の説明を参照のこと。
<i>ia</i>	INTEGER。長さ $m+1$ の配列。配列 <i>a</i> の成分のインデックスを格納する。 <i>ia</i> (<i>i</i>) は配列 <i>a</i> の行 <i>i</i> の最初の非ゼロ成分のインデックスである。最後の成分 <i>ia</i> ($m+1$)-1 の値は、非ゼロの数 +1 に等しい。詳細は、「 スパース行列の格納形式 」の <i>rowIndex</i> 配列の説明を参照のこと。
<i>ja</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。長さは、配列 <i>a</i> の長さと同じ。詳細は、「 スパース行列の格納形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は m_0 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

出力パラメーター

<i>y</i>	REAL*8。配列、次元は m 以上。ベクトル <i>y</i> が格納される。
----------	--

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)
  CHARACTER*1  uplo, transa, diag
  INTEGER      m
  INTEGER      ia(*), ja(*)
  REAL*8      a(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcsrtrsv(uplo, transa, diag, m, a, ia, ja, x, y)
  CHARACTER(LEN=1), INTENT(IN):: uplo, transa, diag
```

```

INTEGER, INTENT(IN) :: m
INTEGER, INTENT(IN) :: ia(*), ja(*)
REAL(KIND(1.0D0)), INTENT(IN) :: a(*), x(*)
REAL(KIND(1.0D0)), INTENT(OUT) :: y(*)

```

C:

```

void mkl_dcsrtrsv(char *uplo, char *transa, char *diag, int *m, double *a,
  int *ia, int *ja, double *x, double *y);

```

mkl_dcscsv

CSC 形式のスパース行列の連立 1 次方程式を解く。

構文

Fortran:

```

call mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntreb, pntre, x, y)

```

C:

```

mkl_dcscsv(&transa, &m, &alpha, matdescra, val, indx, pntreb, pntre, x, y);

```

説明

mkl_dcsrsv ルーチンは、CSC 形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inv}(A) * x$$

または

$$y := \alpha * \text{inv}(A') * x$$

α は、スカラーである。

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、 $y := \alpha * \text{inv}(A) * x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、 $y := \alpha * \text{inv}(A') * x$
<i>m</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー α を指定する。

<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 A の非ゼロの成分を格納する配列。長さは、 $pntrb(m) - pntrb(1)$ である。詳細は、「CSC 形式」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 A の非ゼロの各成分の列インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと等しい。詳細は、「CSC 形式」の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが $pntrb(i) - pntrb(1) + 1$ である行インデックスを格納する。詳細は、「CSC 形式」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrr</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが $pntrr(i) - pntrr(1)$ である行インデックスを格納する。詳細は、「CSC 形式」の <i>pointerE</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。成分には単位増分を使用してアクセスする。

出力パラメーター

<i>y</i>	解として得られたベクトル <i>x</i> を格納する。
----------	------------------------------

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntrr, x, y)
    CHARACTER*1    transa
    CHARACTER      matdescra(*)
    INTEGER        m
    INTEGER        indx(*), pntrb(m), pntrr(m)
    REAL*8        alpha
    REAL*8        val(*)
    REAL*8        x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dcscsv(transa, m, alpha, matdescra, val, indx, pntrb,
pntrr, x, y)
    CHARACTER(LEN=1), INTENT(IN):: transa
    INTEGER, INTENT(IN) :: m
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntrr(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha
    REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```
void mkl_dcscsv(char *transa, int *m, double *alpha, char *matdescra,
               double *val, int *indx, int *pntrb, int *pntre, double *x, double
               *y);
```

mkl_dcoosv

座標形式のスパース行列について連立 1 次方程式を解く。

構文

Fortran:

```
call mkl_dcoosv(transa, m, alpha, matdescra, val, rowind, colind, nnz, x,
               y)
```

C:

```
mkl_dcoosv(&transa, &m, &alpha, matdescra, val, rowind, colind, &nnz, x,
           y);
```

説明

mkl_dcoosv ルーチンは、座標形式のスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inv}(A) * x$$

または

$$y := \alpha * \text{inv}(A') * x$$

α は、スカラーである。

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、 $y := \alpha * \text{inv}(A) * x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、 $y := \alpha * \text{inv}(A') * x$
<i>m</i>	INTEGER。行列 A の行数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。

<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 <i>A</i> の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は <i>m</i> 以上。このルーチンに入る前に、配列 <i>y</i> にベクトル <i>y</i> を格納しなければならない。成分には単位増分を使用してアクセスする。

出力パラメーター

<i>y</i>	解として得られたベクトル <i>x</i> を格納する。
----------	------------------------------

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcoosv(transa, m, alpha, matdescra, val, rowind, colind,
nnz, x, y)
    CHARACTER*1    transa
    CHARACTER      matdescra(*)
    INTEGER        m, nnz
    INTEGER        rowind(*), colind(*)
    REAL*8        alpha
    REAL*8        val(*)
    REAL*8        x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_dcoosv(transa, m, alpha, matdescra, val, rowind, colind,
nnz, x, y)
    CHARACTER(LEN=1), INTENT(IN) :: transa
    INTEGER, INTENT(IN) :: m, nnz
    CHARACTER, INTENT(IN) :: matdescra(*)
    INTEGER, INTENT(IN) :: rowind(*), colind(*)
    REAL(KIND(1.0D0)), INTENT(IN) :: alpha
    REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
    REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dcoosv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *rowind, int *colind, int *nnz, double *x, double *y);

```

mkl_dcootrsv

座標形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバー。

構文

Fortran:

```
call mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz, x, y)
```

C:

```
mkl_dcootrsv(&uplo, &transa, &diag, &m, val, rowind, colind, &nnz, x, y);
```

説明

mkl_dcootrsv ルーチンは、座標形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A*y = x$$

または

$$A'*y = x$$

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

uplo CHARACTER*1。行列 A の上三角と下三角のどちらを使用するかを指定する。

$uplo = 'U'$ または $'u'$ の場合、行列 A の上三角が使用される。

$uplo = 'L'$ または $'l'$ の場合、行列 A の下三角が使用される。

transa CHARACTER*1。実行する演算を指定する。

$transa = 'N'$ または $'n'$ の場合、 $A*y = x$

$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、 $A'*y = x$

diag CHARACTER*1。 A が単位三角行列であるかどうかを指定する。

$diag = 'U'$ または $'u'$ の場合、 A は単位三角行列とみなされる。

$diag = 'N'$ または $'n'$ の場合、 A は単位三角行列とみなされない。

m INTEGER。行列 A の行数。

val REAL*8。長さ nnz の配列。行列 A の非ゼロ成分を任意の順番で格納する。詳細は、「[座標形式](#)」の $values$ 配列の説明を参照のこと。

<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>x</i>	REAL*8。配列、次元は <i>m</i> 。このルーチンに入る前に、配列 <i>x</i> にベクトル <i>x</i> を格納しなければならない。

出力パラメーター

y REAL*8。配列、次元は *m* 以上。ベクトル *y* が格納される。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz,
x, y)
  CHARACTER*1    uplo, transa, diag
  INTEGER        m, nnz
  INTEGER        rowind(*), colind(*)
  REAL*8        val(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dcootrsv(uplo, transa, diag, m, val, rowind, colind, nnz,
x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo, transa, diag
  INTEGER, INTENT(IN) :: m, nnz
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

C:

```
void mkl_dcootrsv(char *uplo, char *transa, char *diag, int *m, double
*alpha, char *matdescra, double *val, int *rowind, int *colind, int
*nnz, double *x, double *y);
```

mkl_ddiasv

対角形式のスパース行列について連立1次方程式を解く。

構文

Fortran:

```
call mkl_ddiasv(transa, m, alpha, matdescra, val, lval, iddiag, ndiag, x, y)
```

C:

```
mk1_ddiasv(&transa, &m, &alpha, matdescra, val, &lval, iddiag, &ndiag, x, y);
```

説明

mk1_ddiasv ルーチンは、対角形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$y := \alpha * \text{inv}(A) * x$$

または

$$y := \alpha * \text{inv}(A') * x$$

α は、スカラーである。

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、 $y := \alpha * \text{inv}(A) * x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、 $y := \alpha * \text{inv}(A') * x$
<i>m</i>	INTEGER。行列 A の行数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , $lval \geq m$ のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。成分には単位増分を使用してアクセスする。
<i>y</i>	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。成分には単位増分を使用してアクセスする。

出力パラメーター

y 解として得られたベクトル x を格納する。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_ddiasv(transa, m, alpha, matdescra, val, lval, iddiag,
ndiag, x, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, lval, ndiag
  INTEGER      iddiag(*)
  REAL*8      alpha
  REAL*8      val(lval,*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_ddiasv(transa, m, alpha, matdescra, val, lval, iddiag,
ndiag, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m, lval, ndiag
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: iddiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)
```

C:

```
void mkl_ddiasv(char *transa, int *m, double *alpha, char *matdescra,
double *val, int *lval, int *iddiag, int *ndiag, double *x, double *y);
```

mkl_ddiatsrv

対角形式のスパース行列に対する簡易インターフェイスを備えた三角ソルバー。

構文

Fortran:

```
call mkl_ddiatsrv(uplo, transa, diag, m, val, lval, iddiag, ndiag, x, y)
```

C:

```
mkl_ddiatsrv(&uplo, &transa, &diag, &m, val, &lval, iddiag, &ndiag, x, y);
```

説明

mkl_ddiatsrv ルーチンは、対角形式で格納されたスパース行列に対する行列 - ベクトル演算による連立 1 次方程式の解を算出する。

$$A*y = x$$

または

$$A' * y = x$$

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>uplo</i>	CHARACTER*1。行列 A の上三角と下三角のどちらを使用するかを指定する。 <i>uplo</i> = 'U' または 'u' の場合、行列 A の上三角が使用される。 <i>uplo</i> = 'L' または 'l' の場合、行列 A の下三角が使用される。
<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、 $A * y = x$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、 $A' * y = x$
<i>diag</i>	CHARACTER*1。 A が単位三角行列であるかどうかを指定する。 <i>diag</i> = 'U' または 'u' の場合、 A は単位三角行列とみなされる。 <i>diag</i> = 'N' または 'n' の場合、 A は単位三角行列とみなされない。
<i>m</i>	INTEGER。行列 A の行数。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , $lval \geq m$ のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<i>x</i>	REAL*8。配列、次元は m 。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。

出力パラメーター

y REAL*8。配列、次元は m 以上。ベクトル y が格納される。

インターフェイス

Fortran 77:

SUBROUTINE mkl_ddiattrsv(*uplo*, *transa*, *diag*, *m*, *val*, *lval*, *idiag*, *ndiag*, *x*, *y*)

CHARACTER*1 *uplo*, *transa*, *diag*
INTEGER *m*, *lval*, *ndiag*

```

INTEGER      inddiag(*)
REAL*8      val(lval,*), x(*), y(*)

```

Fortran 95:

```

SUBROUTINE mkl_ddiattrsv(uplo, transa, diag, m, val, lval, iddiag, ndiag,
x, y)
  CHARACTER(LEN=1), INTENT(IN) :: uplo, transa, diag
  INTEGER, INTENT(IN) :: m, lval, ndiag
  INTEGER, INTENT(IN) :: inddiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(lval,*), x(*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_ddiattrsv(char *uplo, char *transa, char *diag, int *m, double
*val, int *lval, int *iddiag, int *ndiag, double *x, double *y);

```

mkl_dskysv

輪郭形式のスパース行列について連立1次方程式を解く。

構文**Fortran:**

```

call mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)

```

C:

```

mkl_dskysv(&transa, &m, &alpha, matdescra, val, pntr, x, y);

```

説明

mkl_dskysv ルーチンは、輪郭格納形式のスパース行列に対する行列 - ベクトル演算による連立1次方程式の解を算出する。

$y := \alpha * \text{inv}(A) * x$

または

$y := \alpha * \text{inv}(A') * x$

α は、スカラーである。

x と y は、ベクトルである。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

$transa$ CHARACTER*1。実行する演算を指定する。

	$transa = 'N'$ または $'n'$ の場合、 $y := \alpha * inv(A) * x$
	$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、 $y := \alpha * inv(A)' * x$
m	INTEGER。行列 A の行数。
α	REAL*8。スカラー α を指定する。
$matdescra$	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
val	REAL*8。行列 A の成分のセットが輪郭プロファイル形式で格納された配列。 $matdescrsa(2) = 'L'$ の場合、 val は行列 A の下三角の成分を格納する。 $matdescrsa(2) = 'U'$ の場合、 val は行列 A の上三角の成分を格納する。 詳細は、「 スカイライン格納方式 」の $values$ 配列の説明を参照のこと。
$pntr$	INTEGER。下三角の場合は長さ $(m+1)$ 、上三角の場合は長さ $(k+1)$ の配列。 val で指定される、行列 A の各行 (または列) の最初の成分の位置のインデックスを格納する。詳細は、「 対角格納方式 」の $pointers$ 配列の説明を参照のこと。
x	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 x にベクトル x を格納しなければならない。成分には単位増分を使用してアクセスする。
y	REAL*8。配列、次元は m 以上。このルーチンに入る前に、配列 y にベクトル y を格納しなければならない。成分には単位増分を使用してアクセスする。

出力パラメーター

y 解として得られたベクトル x を格納する。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m
  INTEGER      pntr(*)
  REAL*8      alpha
  REAL*8      val(*), x(*), y(*)
```

Fortran 95:

```
SUBROUTINE mkl_dskysv(transa, m, alpha, matdescra, val, pntr, x, y)
  CHARACTER(LEN=1), INTENT(IN) :: transa
  INTEGER, INTENT(IN) :: m
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntr(*)
```

```

REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), x(*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: y(*)

```

C:

```

void mkl_dskysv(char *transa, int *m, double *alpha, char *matdescra,
               double *val, int *pntr, double *x, double *y);

```

mkl_dcsrmm

CSR 形式で格納されているスパース行列の行列-行列の積を計算する。

構文**Fortran:**

```

call mkl_dcsrmm(transa, m, n, k, alpha, matdescra, val, indx, pntrb,
               pntrc, b, ldb, beta, c, ldc)

```

C:

```

mkl_dcsrmm(&transa, &m, &n, &k, &alpha, matdescra, val, indx, pntrb,
           pntrc, b, &ldb, &beta, c, &ldc);

```

説明

mkl_dcsrmm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

α と β は、スカラーである。

B と C は、密行列である。

A は行圧縮形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 A の行数。
<i>n</i>	INTEGER。行列 C の列数。

<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。 長さは、 <i>pntre</i> (<i>m</i>) - <i>pntrb</i> (1) である。詳細は、「CSR 形式」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。 長さは、 <i>val</i> 配列の長さと等しい。詳細は、「CSR 形式」の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが <i>pntrb</i> (<i>i</i>) - <i>pntrb</i> (1)+1 である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntre</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが <i>pntre</i> (<i>i</i>) - <i>pntrb</i> (1) である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (<i>ldb</i> , <i>n</i>)。transa = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の <i>n</i> × <i>k</i> の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラー <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は (<i>ldc</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>c</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の <i>n</i> × <i>k</i> の部分に行列 <i>C</i> を格納しなければならない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

出力パラメーター

<i>c</i>	行列 ($\alpha * A * B + \beta * C$) または ($\alpha * A' * B + \beta * C$) によって上書きされる。
----------	---

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcsrmm(transa, m, n, k, alpha, matdescra, val, indx,
pntrb, pntre, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc
  INTEGER      indx(*), pntrb(m), pntre(m)
  REAL*8      alpha, beta

```

```
REAL*8          val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx,
  pntreb, pntre, b, ldb, beta, c, ldc)
```

```
CHARACTER(LEN=1), INTENT(IN):: transa
INTEGER, INTENT(IN) :: m, n, k, ldb, ldc
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: indx(*), pntreb(*), pntre(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_dcscmm(char *transa, int *m, int *n, int *k, double *alpha, char
  *matdescra, double *val, int *indx, int *pntreb, int *pntre, double
  *b, int *ldb, double *beta, double *c, int *ldc,);
```

mkl_dcscmm

CSC 形式で格納されているスパース行列の行列-
行列の積を計算する。

構文**Fortran:**

```
call mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx, pntreb,
  pntre, b, ldb, beta, c, ldc)
```

C:

```
mkl_dcscmm(&transa, &m, &n, &k, &alpha, matdescra, val, indx, pntreb,
  pntre, b, &ldb, &beta, c, &ldc);
```

説明

mkl_dcscmm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

α と β は、スカラーである。

B と C は、密行列である。

A は列圧縮形式の $m \times k$ のスパース行列、 A' は A の転置である。

入カパラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>A</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 <i>pntrb</i> (<i>k</i>) - <i>pntrb</i> (1) である。詳細は、「CSC 形式」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の行インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと同じ。詳細は、「CSC 形式」の <i>rows</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最初のインデックスが <i>pntrb</i> (<i>i</i>) - <i>pntrb</i> (1)+1 である行インデックスを格納する。詳細は、「CSC 形式」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrc</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最後のインデックスが <i>pntrc</i> (<i>i</i>) - <i>pntrb</i> (1) である行インデックスを格納する。詳細は、「CSC 形式」の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (<i>ldb</i> , <i>n</i>)。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の <i>n</i> × <i>k</i> の部分に行列 <i>B</i> を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラー <i>beta</i> を指定する。
<i>c</i>	REAL*8。配列、次元は (<i>ldc</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>c</i> の先頭の <i>m</i> × <i>n</i> の部分に行列 <i>C</i> を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の <i>n</i> × <i>k</i> の部分に行列 <i>C</i> を格納しなければならない。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

出力パラメーター

<i>c</i>	行列 ($\alpha * A * B + \beta * C$) または ($\alpha * A' * B + \beta * C$) によって上書きされる。
----------	---

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx,
  pntrb, pntre, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc
  INTEGER      indx(*), pntrb(k), pntre(k)
  REAL*8      alpha, beta
  REAL*8      val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dcscmm(transa, m, n, k, alpha, matdescra, val, indx,
  pntrb, pntre, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_dcscmm(char *transa, int *m, int *n, int *k, double *alpha, char
  *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
  *b, int *ldb, double *beta, double *c, int *ldc);
```

mkl_dcoomm

座標形式で格納されているスパース行列の行列-
行列の積を計算する。

構文

Fortran:

```
call mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind, colind,
  nnz, b, ldb, beta, c, ldc)
```

C:

```
mkl_dcoomm(&transa, &m, &n, &k, &alpha, matdescra, val, rowind, colind,
  &nnz, b, &ldb, &beta, c, &ldc);
```

説明

mkl_dcoomm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

α と β は、スカラーである。

B と C は、密行列である。

A は座標形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 A の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>k</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 A の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 A の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 A の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (ldb, n) 。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $n \times k$ の部分に行列 B を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 B を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラー β を指定する。
<i>c</i>	REAL*8。配列、次元は (ldc, n) 。このルーチンに入る前に、配列 <i>c</i> の先頭の $m \times n$ の部分に行列 C を格納しなければならない。そうでない場合は、配列 <i>c</i> の先頭の $n \times k$ の部分に行列 C を格納しなければならない。

ldc INTEGER。呼び出し元の(サブ)プログラムで宣言されているとおりに、*c* の第1次元を指定する。

出力パラメーター

c 行列 ($\alpha * A * B + \beta * C$) または ($\alpha * A' * B + \beta * C$) によって上書きされる。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc, nnz
  INTEGER      rowind(*), colind(*)
  REAL*8       alpha, beta
  REAL*8       val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dcoomm(transa, m, n, k, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, ldb, ldc, nnz
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: rowind(*), colind(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_dcoomm(char *transa, int *m, int *n, int *k, double *alpha, char
 *matdescra, double *val, int *rowind, int *colind, int *nnz, double
 *b, int *ldb, double *beta, double *c, int *ldc);
```

mkl_ddiamm

対角形式で格納されているスパース行列の行列-行列の積を計算する。

構文

Fortran:

```
call mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval, idia,
 ndia, b, ldb, beta, c, ldc)
```

C:

```
mkl_ddiamm(&transa, &m, &n, &k, &alpha, matdescra, val, &lval, idia,
 &ndia, b, &ldb, &beta, c, &ldc);
```

説明

mk1_ddiamm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

α と β は、スカラーである。

B と C は、密行列である。

A は CSR 形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 A の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>k</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の <i>values</i> 配列の説明を参照のこと。
<i>lval</i>	INTEGER。 <i>val</i> , $lval \geq \min(m, k)$ のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の <i>lval</i> 配列の説明を参照のこと。
<i>idiag</i>	INTEGER。長さ <i>ndiag</i> の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の <i>distance</i> 配列の説明を参照のこと。
<i>ndiag</i>	INTEGER。行列 A の非ゼロの対角成分を指定する。
<i>b</i>	REAL*8。配列、次元は (ldb, n) 。 <i>transa</i> = 'N' または 'n' と指定した場合は、このルーチンに入る前に、配列 <i>b</i> の先頭の $n \times k$ の部分に行列 B を格納しなければならない。そうでない場合は、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 B を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>beta</i>	REAL*8。スカラー <i>beta</i> を指定する。

- c* REAL*8。配列、次元は (*ldc*, *n*)。このルーチンに入る前に、配列 *c* の先頭の $m \times n$ の部分に行列 *C* を格納しなければならない。そうでない場合は、配列 *c* の先頭の $n \times k$ の部分に行列 *C* を格納しなければならない。
- ldc* INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、*c* の第 1 次元を指定する。

出力パラメーター

- c* 行列 ($\alpha * A * B + \beta * C$) または ($\alpha * A' * B + \beta * C$) によって上書きされる。

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval,
idiag, ndiag, b, ldb, beta, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, k, ldb, ldc, lval, ndiag
  INTEGER      idiag(*)
  REAL*8      alpha, beta
  REAL*8      val(lval,*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_ddiamm(transa, m, n, k, alpha, matdescra, val, lval,
idiag, ndiag, b, ldb, beta, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, k, lval, ndiag, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: idiag(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_ddiamm(char *transa, int *m, int *n, int *k, double *alpha, char
*matdescra, double *val, int *lval, int *idiag, int *ndiag, double
*b, int *ldb, double *beta, double *c, int *ldc);
```

mkl_dskymm

輪郭形式で格納されているスパース行列の行列-行列の積を計算する。

構文

Fortran:

```
call mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b, ldb,
               beta, c, ldc)
```

C:

```
mkl_dskymm(&transa, &m, &n, &k, &alpha, matdescra, val, pntr, b, &ldb,
           &beta, c, &ldc);
```

説明

mkl_dskymm ルーチンは、次のように定義される行列 - 行列演算を実行する。

$$C := \alpha * A * B + \beta * C$$

または

$$C := \alpha * A' * B + \beta * C$$

α と β は、スカラーである。

B と C は、密行列である。

A は輪郭格納形式の $m \times k$ のスパース行列、 A' は A の転置である。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * A * B + \beta * C$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha * A' * B + \beta * C$
<i>m</i>	INTEGER。行列 A の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>k</i>	INTEGER。行列 A の列数。
<i>alpha</i>	REAL*8。スカラー α を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 A の成分のセットが輪郭プロファイル形式で格納された配列。

`matdescrsa(2) = 'L'` の場合、`val` は行列 A の下三角の成分を格納する。

`matdescrsa(2) = 'U'` の場合、`val` は行列 A の上三角の成分を格納する。

詳細は、「[対角格納方式](#)」の `values` 配列の説明を参照のこと。

<code>pntr</code>	INTEGER。下三角の場合は長さ $(m+1)$ 、上三角の場合は長さ $(k+1)$ の配列。 <code>val</code> で指定される、行列 A の各行 (または列) の最初の成分の位置のインデックスを格納する。詳細は、「 対角格納方式 」の <code>pointers</code> 配列の説明を参照のこと。
<code>b</code>	REAL*8。配列、次元は (ldb, n) 。 <code>transa = 'N'</code> または <code>'n'</code> と指定した場合は、このルーチンに入る前に、配列 b の先頭の $n \times k$ の部分に行列 B を格納しなければならない。そうでない場合は、配列 b の先頭の $m \times n$ の部分に行列 B を格納しなければならない。
<code>ldb</code>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 b の第 1 次元を指定する。
<code>beta</code>	REAL*8。スカラー <code>beta</code> を指定する。
<code>c</code>	REAL*8。配列、次元は (ldc, n) 。このルーチンに入る前に、配列 c の先頭の $m \times n$ の部分に行列 C を格納しなければならない。そうでない場合は、配列 c の先頭の $n \times k$ の部分に行列 C を格納しなければならない。
<code>ldc</code>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 c の第 1 次元を指定する。

出力パラメーター

<code>c</code>	行列 $(\alpha * A * B + \beta * C)$ または $(\alpha * A' * B + \beta * C)$ によって上書きされる。
----------------	---

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b,
ldb, beta, c, ldc)
```

```
CHARACTER*1  transa
CHARACTER    matdescra(*)
INTEGER      m, n, k, ldb, ldc
INTEGER      pntr(*)
REAL*8      alpha, beta
REAL*8      val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dskymm(transa, m, n, k, alpha, matdescra, val, pntr, b,
ldb, beta, c, ldc)
```

```
CHARACTER(LEN=1), INTENT(IN):: transa
INTEGER, INTENT(IN) :: m, n, k, ldb, ldc
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: pntr(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha, beta
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
```

```
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_dskymm(char *transa, int *m, int *n, int *k, double *alpha, char
    *matdescra, double *val, int *pntr, double *b, int *ldb, double
    *beta, double *c, int *ldc);
```

mkl_dcsrsm

CSR 形式のスパース行列について連立 1 次行列方程式を解く。

構文

Fortran:

```
call mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntrb, pntrc,
    b, ldb, c, ldc)
```

C:

```
mkl_dcsrsm(&transa, &m, &n, &alpha, matdescra, val, indx, pntrb, pntrc,
    b, &ldb, c, &ldc);
```

説明

mkl_dcsrsm ルーチンは、CSR 形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$$C := \alpha \cdot \text{inv}(A) \cdot B$$

または

$$C := \alpha \cdot \text{inv}(A') \cdot B$$

α は、スカラーである。

B と C は、密行列である。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

<i>transa</i>	CHARACTER*1。実行する演算を指定する。 <i>transa</i> = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha \cdot \text{inv}(A) \cdot B$ <i>transa</i> = 'T'、't' または 'C'、'c' の場合、行列 - ベクトルの積は次のように計算される。 $C := \alpha \cdot \text{inv}(A') \cdot B$
<i>m</i>	INTEGER。行列 A の列数。

<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。 長さは、 $\text{pntrb}(m) - \text{pntrb}(1)$ である。詳細は、「CSR 形式」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の列インデックスを格納する配列。 長さは、 <i>val</i> 配列の長さと同じ。詳細は、「CSR 形式」の <i>columns</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最初のインデックスが $\text{pntrb}(i) - \text{pntrb}(1) + 1$ である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntrc</i>	INTEGER。長さ <i>m</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の行 <i>i</i> の最後のインデックスが $\text{pntrc}(i) - \text{pntrc}(1)$ である行インデックスを格納する。詳細は、「CSR 形式」の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (ldb, n) 。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

出力パラメーター

<i>c</i>	REAL*8。配列、次元は (ldc, n) 。配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納する。
----------	--

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
pntrc, b, ldb, c, ldc)
```

```
CHARACTER*1 transa
CHARACTER matdescra(*)
INTEGER m, n, ldb, ldc
INTEGER indx(*), pntrb(m), pntrc(m)
REAL*8 alpha
REAL*8 val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dcsrsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
pntrc, b, ldb, c, ldc)
```

```
CHARACTER(LEN=1), INTENT(IN):: transa
INTEGER, INTENT(IN) :: m, n, ldb, ldc
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntrc(*)
```

```

REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

C:

```

void mkl_dcsrcsm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
    *b, int *ldb, double *c, int *ldc);

```

dcscsm

CSC 形式のスパース行列について連立 1 次行列方程式を解く。

構文**Fortran:**

```

call mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb, pntre,
    b, ldb, c, ldc)

```

C:

```

mkl_dcscsm(&transa, &m, &n, &alpha, matdescra, val, indx, pntrb, pntre,
    b, &ldb, c, &ldc);

```

説明

mkl_dcscsm ルーチンは、CSC 形式のスパース行列に対する行列 - 行列演算による連立 1 次方程式の解を算出する。

$C := \alpha * \text{inv}(A) * B$

または

$C := \alpha * \text{inv}(A') * B$

α は、スカラーである。

B と C は、密行列である。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。

$\text{transa} = 'N'$ または $'n'$ の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * \text{inv}(A) * B$

$\text{transa} = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。

$C := \alpha * \text{inv}(A') * B$

<i>m</i>	INTEGER。行列 <i>A</i> の列数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。行列 <i>A</i> の非ゼロの成分を格納する配列。長さは、 $pntre(m) - pntrb(1)$ である。詳細は、「 CSC 形式 」の <i>values</i> 配列の説明を参照のこと。
<i>indx</i>	INTEGER。行列 <i>A</i> の非ゼロの各成分の行インデックスを格納する配列。長さは、 <i>val</i> 配列の長さと同じ。詳細は、「 CSC 形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>pntrb</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最初のインデックスが $pntrb(i) - pntrb(1) + 1$ である行インデックスを格納する。詳細は、「 CSC 形式 」の <i>pointerB</i> 配列の説明を参照のこと。
<i>pntre</i>	INTEGER。長さ <i>k</i> の配列。配列 <i>val</i> と配列 <i>indx</i> の列 <i>i</i> の最後のインデックスが $pntre(i) - pntrb(1)$ である行インデックスを格納する。詳細は、「 CSC 形式 」の <i>pointerE</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (<i>ldb</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

出力パラメーター

<i>c</i>	REAL*8。配列、次元は (<i>ldc</i> , <i>n</i>)。配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納する。
----------	--

インターフェイス

Fortran 77:

```

SUBROUTINE mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
pntre, b, ldb, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, ldb, ldc
  INTEGER      indx(*), pntrb(m), pntre(m)
  REAL*8      alpha
  REAL*8      val(*), b(ldb,*), c(ldc,*)

```

Fortran 95:

```

SUBROUTINE mkl_dcscsm(transa, m, n, alpha, matdescra, val, indx, pntrb,
pntre, b, ldb, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, ldb, ldc

```

```

CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: indx(*), pntrb(*), pntre(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

C:

```

void mkl_dcscsm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *indx, int *pntrb, int *pntre, double
    *b, int *ldb, double *c, int *ldc);

```

mkl_dcoosm

座標形式のスパース行列について連立1次行列方程式を解く。

構文

Fortran:

```

call mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind, colind,
    nnz, b, ldb, c, ldc)

```

C:

```

mkl_dcoosm(&transa, &m, &n, &alpha, matdescra, val, rowind, colind,
    &nnz, b, &ldb, c, &ldc);

```

説明

mkl_dcoosm ルーチンは、座標形式のスパース行列に対する行列 - 行列演算による連立1次方程式の解を算出する。

$$C := \alpha \cdot \text{inv}(A) \cdot B$$

または

$$C := \alpha \cdot \text{inv}(A') \cdot B$$

α は、スカラーである B

B と C は、密行列である。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。

transa = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha \cdot \text{inv}(A) \cdot B$

$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。

$$C := \alpha * \text{inv}(A') * B$$

<i>m</i>	INTEGER。行列 <i>A</i> の行数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>alpha</i>	REAL*8。スカラー <i>alpha</i> を指定する。
<i>matdescra</i>	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
<i>val</i>	REAL*8。長さ <i>nnz</i> の配列。行列 <i>A</i> の非ゼロ成分を任意の順番で格納する。詳細は、「 座標形式 」の <i>values</i> 配列の説明を参照のこと。
<i>rowind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の行インデックスを格納する。詳細は、「 座標形式 」の <i>rows</i> 配列の説明を参照のこと。
<i>colind</i>	INTEGER。長さ <i>nnz</i> の配列。行列 <i>A</i> の各非ゼロ成分の列インデックスを格納する。詳細は、「 座標形式 」の <i>columns</i> 配列の説明を参照のこと。
<i>nnz</i>	INTEGER。行列 <i>A</i> の非ゼロの成分を指定する。詳細は、「 座標形式 」の <i>nnz</i> 配列の説明を参照のこと。
<i>b</i>	REAL*8。配列、次元は (<i>ldb</i> , <i>n</i>)。このルーチンに入る前に、配列 <i>b</i> の先頭の $m \times n$ の部分に行列 <i>B</i> を格納しなければならない。
<i>ldb</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>b</i> の第 1 次元を指定する。
<i>ldc</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 <i>c</i> の第 1 次元を指定する。

出力パラメーター

<i>c</i>	REAL*8。配列、次元は (<i>ldc</i> , <i>n</i>)。配列 <i>c</i> の先頭の $m \times n$ の部分に行列 <i>C</i> を格納する。
----------	--

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, c, ldc)
```

```
CHARACTER*1  transa
CHARACTER    matdescra(*)
INTEGER      m, n, ldb, ldc, nnz
INTEGER      rowind(*), colind(*)
REAL*8      alpha
REAL*8      val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dcoosm(transa, m, n, alpha, matdescra, val, rowind,
colind, nnz, b, ldb, c, ldc)
```

```
CHARACTER(LEN=1), INTENT(IN):: transa
```

```

INTEGER, INTENT(IN) :: m, n, ldb, ldc, nnz
CHARACTER, INTENT(IN) :: matdescra(*)
INTEGER, INTENT(IN) :: rowind(*), colind(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

C:

```

void mkl_dcoosm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *rowind, int *colind, int *nnz, double
    *b, int *ldb, double *c, int *ldc);

```

mkl_ddiasm

対角形式のスパース行列について連立1次行列方程式を解く。

構文

Fortran:

```

call mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, iddiag, ndiag,
    b, ldb, c, ldc)

```

C:

```

mkl_ddiasm(&transa, &m, &n, &alpha, matdescra, val, &lval, iddiag, &ndiag,
    b, &ldb, c, &ldc);

```

説明

mkl_ddiasm ルーチンは、対角形式のスパース行列に対する行列 - 行列演算による連立1次方程式の解を算出する。

$$C := \alpha \cdot \text{inv}(A) \cdot B$$

または

$$C := \alpha \cdot \text{inv}(A') \cdot B$$

α は、スカラーである。

B と C は、密行列である。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。

transa = 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha \cdot \text{inv}(A) \cdot B$

$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。

$$C := \alpha * \text{inv}(A') * B$$

m	INTEGER。行列 A の行数。
n	INTEGER。行列 C の列数。
α	REAL*8。スカラー α を指定する。
$matdescra$	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
val	REAL*8。 $lval \times ndiag$ の 2 次元配列。行列 A の非ゼロ対角成分を格納する。詳細は、「 対角格納方式 」の $values$ 配列の説明を参照のこと。
$lval$	INTEGER。 val , $lval \geq m$ のリーディング・ディメンジョン。詳細は、「 対角格納方式 」の $lval$ 配列の説明を参照のこと。
$idiag$	INTEGER。長さ $ndiag$ の配列。行列 A の主対角と非ゼロ対角の距離を格納する。詳細は、「 対角格納方式 」の $distance$ 配列の説明を参照のこと。
$ndiag$	INTEGER。行列 A の非ゼロの対角成分を指定する。
b	REAL*8。配列、次元は (ldb, n) 。このルーチンに入る前に、配列 b の先頭の $m \times n$ の部分に行列 B を格納しなければならない。
ldb	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 b の第 1 次元を指定する。
ldc	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 c の第 1 次元を指定する。

出力パラメーター

c	REAL*8。配列、次元は (ldc, n) 。配列 c の先頭の $m \times n$ の部分に行列 C を格納する。
-----	--

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, idiag,
ndiag, b, ldb, c, ldc)
  CHARACTER*1  transa
  CHARACTER    matdescra(*)
  INTEGER      m, n, ldb, ldc, lval, ndiag
  INTEGER      idiag(*)
  REAL*8      alpha
  REAL*8      val(lval,*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_ddiasm(transa, m, n, alpha, matdescra, val, lval, idiag,
ndiag, b, ldb, c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, lval, ndiag, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
```

```

INTEGER, INTENT(IN) :: idiag(*)
REAL(KIND(1.0D0)), INTENT(IN) :: alpha
REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)

```

C:

```

void mkl_ddiasm(char *transa, int *m, int *n, double *alpha, char
    *matdescra, double *val, int *lval, int *idiag, int *ndiag, double
    *b, int *ldb, double *c, int *ldc);

```

mkl_dskysm

輪郭格納形式で格納されたスパース行列について
連立1次行列方程式を解く。

構文**Fortran:**

```

call mkl_dskysm(transa, m, n, alpha, matdescra, val, pntr, b, ldb, c,
    ldc)

```

C:

```

mkl_dskysm(&transa, &m, &n, &alpha, matdescra, val, pntr, b, &ldb, c,
    &ldc);

```

説明

mkl_dskysm ルーチンは、輪郭格納形式のスパース行列に対する行列 - 行列演算による連立1次方程式の解を算出する。

$$C := \alpha * \text{inv}(A) * B$$

または

$$C := \alpha * \text{inv}(A') * B$$

α は、スカラーである。

B と C は、密行列である。

A は単位または非単位の主対角であるスパース上三角行列または下三角行列、 A' は A の転置を示す。

入力パラメーター

パラメーターの説明は、Fortran 77 の標準タイプを参照するデータ型を除く、すべての実装されているインターフェイスに共通である。インターフェイスごとに特有のデータ型は、以下の「**インターフェイス**」のセクションで説明する。

transa CHARACTER*1。実行する演算を指定する。
transa= 'N' または 'n' の場合、行列 - 行列の積は、次のように計算される。 $C := \alpha * \text{inv}(A) * B$

$transa = 'T'$ 、 $'t'$ または $'C'$ 、 $'c'$ の場合、行列 - ベクトルの積は次のように計算される。

$$C := \alpha * \text{inv}(A') * B$$

m	INTEGER。行列 A の行数。
n	INTEGER。行列 C の列数。
α	REAL*8。スカラー α を指定する。
$matdescra$	CHARACTER。配列の 6 個の成分は、演算で使用する行列のプロパティを指定する。最初の 3 個の配列成分のみ使用される。 表 2-6 にそれぞれの配列成分に対して設定可能な値を示す。
val	REAL*8。行列 A の成分のセットが輪郭プロファイル形式で格納された配列。 $matdescrsa(2) = 'L'$ の場合、 val は行列 A の下三角の成分を格納する。 $matdescrsa(2) = 'U'$ の場合、 val は行列 A の上三角の成分を格納する。 詳細は、「 対角格納方式 」の $values$ 配列の説明を参照のこと。
$pntr$	INTEGER。長さ $(m+1)$ の配列。 val で指定される、行列 A の各行 i (または列) の最初の非ゼロ成分の位置のインデックスを格納する。 $pointers(i) - pointers(1) + 1$ である。詳細は、「 対角格納方式 」の $pointers$ 配列の説明を参照のこと。
b	REAL*8。配列、次元は (ldb, n) 。このルーチンに入る前に、配列 b の先頭の $m \times n$ の部分に行列 B を格納しなければならない。
ldb	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 b の第 1 次元を指定する。
ldc	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおりに、 c の第 1 次元を指定する。

出力パラメーター

c	REAL*8。配列、次元は (ldc, n) 。配列 c の先頭の $m \times n$ の部分に行列 C を格納する。
-----	--

インターフェイス

Fortran 77:

```
SUBROUTINE mkl_dskysm(transa, m, n, alpha, matdescra, val, pntr, b, ldb,
c, ldc)
```

```
CHARACTER*1  transa
CHARACTER    matdescra(*)
INTEGER      m, n, ldb, ldc
INTEGER      pntr(*)
REAL*8      alpha
REAL*8      val(*), b(ldb,*), c(ldc,*)
```

Fortran 95:

```
SUBROUTINE mkl_dskysm(transa, m, n, alpha, matdescra, val, pntnr, b, ldb,
c, ldc)
  CHARACTER(LEN=1), INTENT(IN):: transa
  INTEGER, INTENT(IN) :: m, n, ldb, ldc
  CHARACTER, INTENT(IN) :: matdescra(*)
  INTEGER, INTENT(IN) :: pntnr(*)
  REAL(KIND(1.0D0)), INTENT(IN) :: alpha
  REAL(KIND(1.0D0)), INTENT(IN) :: val(*), b(ldb,*)
  REAL(KIND(1.0D0)), INTENT(INOUT) :: c(ldc,*)
```

C:

```
void mkl_dskysm(char *transa, int *m, int *n, double *alpha, char
*matdescra, double *val, int *pntnr, double *b, int *ldb, double *c,
int *ldc,);
```


LAPACK ルーチン： 線形方程式

3

本章では、連立線形方程式の解の算出と、それに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) に組み込まれているルーチンについて説明する。このライブラリーには、実データと複素データ用の LAPACK ルーチンが含まれる。

このライブラリーは、以下のタイプの行列を持つ連立方程式用のルーチンをサポートしている。

- 一般行列
- 帯行列
- 対称/エルミート正定値行列 (フル格納および圧縮格納形式)
- 対称/エルミート正定値帯行列
- 対称/エルミート不定値行列 (フル格納および圧縮格納形式)
- 対称/エルミート不定値帯行列
- 三角行列 (フル格納および圧縮格納形式)
- 三角帯行列
- 三重対角行列

上記の行列タイプのそれぞれについて、ライブラリーには、以下の計算を行うためのルーチンが用意されている。

- 行列の因子分解 (三角行列を除く)
- 行列の平衡化
- 連立線形方程式の解の算出
- 行列の条件数の推定
- 線形方程式の解の精度の改善と誤差範囲の計算
- 行列の逆転の計算

特定の問題を解くために、2 つ以上の[計算ルーチン](#)を呼び出すか、1 回の呼び出しで複数のタスクを組み合わせる[ドライバルーチン](#) (例えば、因子分解と解の算出用の `?gesv`) を呼び出せる。したがって、一般行列の連立線形方程式を解く場合、最初に `?getrf` (LU 因子分解) を呼び出し、次に `?getrs` (解の算出) を呼び出せる。さらに、

?gerfs を呼び出して、解の精度を改善し、誤差範囲を計算できる。代わりに、これらのタスクを 1 回の呼び出しで実行するドライバルーチン ?gesvx を使用することもできる。



警告: LAPACK ルーチンは、入力行列に INF 値または NaN 値が含まれないことを前提としている。LAPACK に対する入力データが適当でないとき、コンピューターのハングアップなどの問題が発生する可能性がある。

インテル MKL 8.0 以降では、LAPACK 計算ルーチンおよびドライバルーチンに対する Fortran-77 インターフェイスとともに、より短い引数リストで簡略化したルーチン呼び出しを使用する Fortran-95 インターフェイスもサポートしている。Fortran-95 インターフェイスの呼び出しシーケンスは、ルーチン説明の「構文」セクションで、Fortran-77 呼び出しの説明の後に解説する。

ルーチン命名規則

本章の各ルーチンについては、Fortran-77 プログラムから呼び出す際に、LAPACK 名を使用できる。

表 3-1 と表 3-2 に、**LAPACK 名**の一覧を示す。LAPACK 名は、以下に説明するように、?yyzzz または ?yyzz の構造になっている。

最初の記号 ? は、データ型を示す。

- s 実数、単精度 c 複素数、単精度
- d 実数、倍精度 z 複素数、倍精度

2 番目と 3 番目の文字 yy は、行列のタイプと格納形式を示す。

- ge 一般行列
- gb 一般帯行列
- gt 一般三重対角行列
- po 対称 / エルミート 正定値行列
- pp 対称 / エルミート 正定値行列 (圧縮格納形式)
- pb 対称 / エルミート 正定値帯行列
- pt 対称 / エルミート 正定値三重対角行列
- sy 対称不定値行列
- sp 対称不定値行列 (圧縮格納形式)
- he エルミート不定値行列
- hp エルミート不定値行列 (圧縮格納形式)
- tr 三角行列
- tp 三角行列 (圧縮格納形式)
- tb 三角帯行列

計算ルーチンでは、最後の 3 つの文字 zzz は、実行される処理を示す。

- trf 三角行列の因子分解を実行する
- trs 因子分解された行列を使って連立線形方程式を解く
- con 行列の条件数を推定する
- rfs 解の精度を改善し、誤差範囲を計算する
- tri 因子分解を使用して逆行列を計算する
- equ 行列を平衡化する

例えば、ルーチン `sgetrf` は、単精度実数型の一般行列の三角分解を実行する。これに対応する複素行列用のルーチンは、`cgetrf` である。

ドライバールーチンには、名前の最後に `-sv` (簡易ドライバー) または `-svx` (高度ドライバー) が付く。

インテル MKL において、Fortran-95 インターフェイスの LAPACK 計算ルーチン名およびドライバールーチン名は、最初の文字がデータ型を示す以外は Fortran-77 の名前と同じである。例えば、Fortran-95 インターフェイスで、実数型の一般行列の三角分解を実行するルーチン名は `getrf` である。異なるデータ型を扱うには、単精度および倍精度の名前付き定数でモジュールブロックを参照する特定の入力パラメーターを定義して行う。

Fortran-95 インターフェイス規則

LAPACK に対する Fortran-95 インターフェイスは、それぞれの Fortran-77 ルーチン呼び出すラッパーを通して実装される。このインターフェイスは、形状引継ぎ配列や、より少ない引数で簡略化した LAPACK ルーチンの呼び出しを提供するオプション引数などの Fortran-95 の機能を使用する。

Fortran-95 インターフェイスで使用される主な規則を以下に示す。

- Fortran-95 呼び出しで使用される引数名は、一般的にそれぞれの標準 (Fortran-77) インターフェイスと同じである。ただし、ライブラリーで使用される引数名の数を減らすための、簡略化された引数名を以下に示す。

標準引数名	Fortran-95 引数名
<code>ap</code>	<code>a</code>
<code>ab</code>	<code>a</code>
<code>afb</code>	<code>af</code>
<code>afp</code>	<code>af</code>
<code>bp</code>	<code>b</code>
<code>bb</code>	<code>b</code>
<code>selctg</code>	<code>select</code>

これらの正式引数名の変更はプログラムの動作には影響を与えず、また統一規則に従うものである。

- 配列次元などの入力引数は Fortran-95 では不要であり、呼び出しシーケンスからスキップされる。配列次元は、必要な配列形状に正確に従うユーザーデータから再構築される。

Fortran-95 インターフェイスでスキップされるその他の標準引数のタイプは、`work`、`rwork`、などのワークスペース配列を表す引数である。ただし、ワークスペース配列が出力時に重要な情報を返す場合は例外である。

また、引数は、呼び出しシーケンス内の他の引数の有無により、その値が完全に定義される場合にスキップされる。復元値はスキップされた引数にとってのみ意味のある値である。

- いくつかの標準引数は Fortran-95 インターフェイスではオプションとして宣言され、呼び出しシーケンスから省略される場合がある。以下の条件のいずれかを満たす場合は、引数をオプションとして宣言できる。

1. 引数の値が呼び出しシーケンス内の他の引数の有無により完全に定義される場合、オプションとして宣言できる。この場合のスキップされる引数との違いは、オプション引数はデフォルトで再構築される値とは異なる意味を持つことである。
例えば、*jobz* という引数が 2 つの値を持ち、その中の 1 つの値が直接もう一方の引数の使用を暗黙的に示す場合、*jobz* の値は実際の 2 番目の引数の有無により一意的に再構築され、*jobz* は省略できる。
 2. 入力引数が、ごくわずかの設定可能な値を持つ場合、オプションとして宣言できる。これらの引数のデフォルト値は、通常、リストの最初の値として設定される。この規則における例外のすべてはルーチンの説明で明示的に述べる。
 3. 入力引数が自然デフォルト値を持つ場合は、オプションとして宣言できる。これらのオプション引数のデフォルト値は、自然デフォルト値に設定される。
- *info* 引数は、Fortran-95 インターフェイスではオプションとして宣言される。呼び出しシーケンス内にある場合、*info* に設定された値の解釈を以下に示す。
 1. 値が -1000 よりも大きい場合、Fortran-77 ルーチンと同様の意味を持つ。
 2. 値が -1000 の場合、作業メモリーの不足を意味する。
 3. 値が -1001 の場合、矛盾した引数が呼び出しシーケンスに存在することを意味する。
 - Fortran-95 呼び出し構文では、オプション引数を大括弧 ([]) で表現する。

オプション・パラメーターの値の再構築に使用される具体的な規定は、各ルーチンごとに特有で、ルーチン仕様の最後の各「Fortran-95 ノート」で詳細を説明する。

MKL LAPACK ルーチン用 Fortran-95 インターフェイスと Netlib 実装

インテル MKL LAPACK-95 実装と Netlib アナログとの違いを次に示す。

- インターフェイス名のプリフィックスに *LA_* が含まれていない。
- オプションの配列引数には、常に *target* 属性が含まれている。
- [getrf](#)、[gbtrf](#)、および [potrf](#) インターフェイスにおいて MKL LAPACK-95 ラッパーの機能性は、FORTRAN-77 のオリジナル実装に似ている。
- *jobz* 引数の値が *z* 引数の有無を指定する場合、*z* は常にオプションとして宣言され、*jobz* は *z* の有無により復元される。Netlib バージョンでは常にそうとは限らない(付録 E の「[修正された Netlib インターフェイス](#)」を参照)。
- エラーチェックの重複を避けるために、*info* パラメーターの処理では、割り当てられたメモリーのチェックとオプション・パラメーターのかく乱に制限される。
- 引数リストにある引数が他の引数を完全に定義する場合、定義される引数は常にオプションとして宣言される。

次の 2 つの条件を満たす場合、Netlib の LAPACK インターフェイスを使用するアプリケーションを変換して、インテル MKL のインターフェイスで使用することができる。

- a. アプリケーションが正確である。すなわち、明白でコンパイラーに依存せず、エラーが含まれていない。
- b. 各ルーチン名は、特定の 1 つのルーチンのみ表す。(例えば、プリフィックス *LA_* を取り除いた後に) アプリケーションのルーチン名が Netlib のルーチン名と一致するが、Netlib のルーチンとは別のルーチンを表す場合、このルーチン名はコンテキストの置換によって変更される。

次の 4 つのケースでは、ユーザー・アプリケーションの変換が必要である（各インターフェイスの特徴については、付録 E の「[LAPACK ルーチン用 Fortran-95 インターフェイス特有の機能](#)」を参照）。

1. インテル MKL ルーチンとはプリフィックス `LA_` または配列属性 `target` のみ異なる Netlib ルーチンを使用する場合。この場合、コンテキスト名の置換のみ必要である。詳細は、付録 E の「[Netlib と同一のインターフェイス](#)」を参照。
2. インテル MKL ルーチンとはプリフィックス `LA_`、配列属性 `target`、および正式引数名が異なる Netlib ルーチンを使用する場合。引数の位置を渡す場合、コンテキスト名の置換のみ必要である。引数のキーを渡す場合、コンテキスト名の置換に加え、一致しないキーの名前も変更する必要がある。詳細は、付録 E の「[引数名が置換されるインターフェイス](#)」を参照。
3. インテル MKL ルーチンとはプリフィックス `LA_`、配列属性 `target`、引数のシーケンスが異なり、引数が一致しない (MKL にはあるが Netlib にはないか、または、MKL にはあるが Netlib にはない) Netlib ルーチンを使用する場合。上記の 2 と 3 で指定されているすべての変換に加え、引数のシーケンスおよび範囲の差異を取り除く必要がある。詳細は、付録 E の「[修正された Netlib インターフェイス](#)」を参照。
4. `getrf`、`gbtrf`、および `potrf` インターフェイスを使用する場合。すなわち、MKL で実装されている Netlib にはない新機能を使用する場合。違いを克服するために、新しい MKL インターフェイスでビルドするか、または LAPACK-77 ルーチンに対応する MKL インターフェイスを使用して新しいサブルーチンを作成する。サブルーチンは直接呼び出すことができるが、MKL のインターフェイスを使用するほうが望ましい。詳細は、付録 E の「[Netlib にはないインターフェイス](#)」および「[新機能のインターフェイス](#)」を参照。
変換されたアプリケーションが引数 `rcond` および `norm` を制御せずに `getrf`、`gbtrf` または `potrf` を呼び出す場合、上記の 1 にあるように、MKL インターフェイスへの呼び出しの修正では、コンテキストの置換のみ必要である。この場合、Netlib の機能性は保持される。
5. Netlib 補助ルーチンを使用する場合。この場合、MKL LAPACK-77 インターフェイスを使用して、対応するサブルーチンを直接呼び出す。

ユーザー・アプリケーションの変換手順は次のとおりである。

1. 条件 a. と b. を満たしていることを確認する。
2. Netlib LAPACK-95 の呼び出しを選択する。各呼び出しに対して、次の操作を行う。
 - どのケースに該当するか特定して、必要な変換を行う。
 - 同一の呼び出しの後にある不要なコードやデータを除去する。
3. 正確かつ完全に変換されたことを確認する。

行列の格納形式

LAPACK ルーチンでは、以下の行列格納形式を使用している。

- フル格納では、行列 A は 2 次元配列 a に格納され、行列成分 a_{ij} は配列成分 $a(i, j)$ に格納される。
- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が 1 次元配列の各列に圧縮される。

- 帯格納では、 k_l 個の劣対角成分と k_u 個の優対角成分を持つ $m \times n$ の帯行列は、 (k_l+k_u+1) 行 n 列の 2 次元配列 ab にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納される。

第 4 章と第 5 章では、圧縮格納形式で行列を格納する配列は名前の最後の文字を p で、帯格納形式で行列を格納する配列には名前の最後の文字を b で示している。

行列の格納形式の詳細は、付録 B の「[行列引数](#)」を参照。

数学的表記

LAPACK ルーチンの説明では、以下の表記を使用している。

$Ax = b$	$n \times n$ の行列 $A = \{a_{ij}\}$ 、右辺のベクトル $b = \{b_i\}$ 、未知のベクトル $x = \{x_i\}$ を持つ連立線形方程式。
$AX = B$	共通の行列 A と複数の右辺を持つ連立方程式の集合。 B の各列は右辺のそれぞれに相当する。 X の各列は対応する解である。
$ x $	成分 $ x_i $ (x_i の絶対値) を持つベクトル。
$ A $	成分 $ a_{ij} $ (a_{ij} の絶対値) を持つ行列。
$\ x\ _\infty = \max_i x_i $	ベクトル x の無限ノルム。
$\ A\ _\infty = \max_i \sum_j a_{ij} $	行列 A の無限ノルム。
$\ A\ _1 = \max_j \sum_i a_{ij} $	行列 A の 1- ノルム。 $\ A\ _1 = \ A^T\ _\infty = \ A^H\ _\infty$
$\kappa(A) = \ A\ \ A^{-1}\ $	行列 A の条件数。

誤差の分析

実際には、ほとんどの計算で実行時に丸め誤差が発生する。また、連立方程式 $Ax = b$ で、データ (A と b の成分) が正確にわかっていない場合もある。したがって、データの誤差と丸め誤差が解 x に与える影響を理解する必要がある。

データの摂動。 x が $Ax = b$ の正確な解であり、 $x + \delta x$ が摂動問題 $(A + \delta A)x = (b + \delta b)$ の正確な解である場合は、次の式が成り立つ。

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right), \text{ where } \kappa(A) = \|A\| \|A^{-1}\|.$$

つまり、 A または b の相対誤差は、解のベクトル x 内で係数 $\kappa(A) = \|A\| \|A^{-1}\|$ によって増幅される場合がある。この係数を、 A の条件数と呼ぶ。

丸め誤差は、元のデータの相対摂動 $c(n)\varepsilon$ と同じ効果を持つ。 ε はマシンの精度、 $c(n)$ は行列の次数 n の適度な関数である。対応する解の誤差は、 $\|\delta x\|/\|x\| \leq c(n)\kappa(A)\varepsilon$ になる ($c(n)$ の値が $10n$ より大きくなることはほとんどない)。

したがって、行列 A が悪条件である場合（つまり、条件数 $\kappa(A)$ が非常に大きい場合）は、解 x の誤差も大きくなる。ときには、精度が全く失われてしまう場合もある。LAPACK には、 $\kappa(A)$ を推定するためのルーチンが含まれている（[「条件数を推定するためのルーチン」](#)を参照）。また、実際の解の誤差をより正確に推定する方法もある（[「解の精度の改善と誤差の推定」](#)を参照）。

計算ルーチン

表 3-1 に、実数行列の因子分解、平衡化、逆行列の計算、実数行列の条件数の推定、実数行列を持つ連立方程式の解の算出、解の精度の改善、解の誤差の推定を行うための LAPACK 計算ルーチン (Fortran-77 インターフェイス) を示す。表 3-2 に、これに対応する複素行列用のルーチンを示す。

これらの表では、ルーチン名の最初の記号は省略されている (「[ルーチン命名規則](#)」を参照)。

表 3-1 実数行列を持つ連立方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	方程式の 解の算出	条件数	誤差の 推定	逆行列の 計算
一般	?getrf	?geequ	?getrs	?gecon	?gerfs	?getri
一般帯	?gbtrf	?gbequ	?gbtrs	?gbcon	?gbrfs	
一般三重対角	?qttrf		?qttrs	?qtcon	?qtrfs	
対称正定値	?potrf	?poequ	?potrs	?pocon	?porfs	?potri
対称正定値 (圧縮格納形式)	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
対称正定値 (帯形式)	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	
対称正定値 (三重対角形式)	?pttrf		?pttrs	?ptcon	?ptrfs	
対称不定値	?sytrf		?sytrs	?sycon	?syrrfs	?sytri
対称不定値 (圧縮格納形式)	?sptrf		?sptrs	?spcon	?sprfs	?sptri
三角			?trtrs	?trcon	?trrrfs	?trtri
三角 (圧縮格納形式)			?tptrs	?tpcon	?tprfs	?tptri
三角帯			?tbtrs	?tbcon	?tbrfs	

表中の ? は、Fortran-77 インターフェイスの **s** (単精度) または **d** (倍精度) を示す。

表 3-2 複素行列を持つ連立方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	方程式の 解の算出	条件数	誤差の 推定	逆行列の 計算
一般	?getrf	?geequ	?getrs	?gecon	?gerfs	?getri
一般帯	?qbtrf	?qbequ	?qbtrs	?qbcon	?qbrfs	
一般三重対角	?qttrf		?qttrs	?qtcon	?qtrfs	
エルミート正定値	?potrf	?poequ	?potrs	?pocon	?porfs	?potri
エルミート正定値 (圧縮格納形式)	?pptrf	?ppequ	?pptrs	?ppcon	?pprfs	?pptri
エルミート正定値 (帯形式)	?pbtrf	?pbequ	?pbtrs	?pbcon	?pbrfs	
エルミート正定値 (三重対角形式)	?pttrf		?pttrs	?ptcon	?ptrfs	
エルミート不定値	?hetrf		?hetrs	?hecon	?herfs	?hetri
対称不定値	?sytrf		?sytrs	?sycon	?syrf	?sytri
エルミート不定値 (圧縮格納形式)	?hptrf		?hptrs	?hpcon	?hprfs	?hptri
対称不定値 (圧縮格納形式)	?sptrf		?sptrs	?spcon	?sprfs	?sptri
三角			?trtrs	?trcon	?trrfs	?trtri
三角 (圧縮格納形式)			?tptrs	?tpcon	?tprfs	?tptri
三角帯			?tbtrs	?tbcon	?tbrfs	

表中の ? は、Fortran-77 インターフェイスの **c** (単精度複素数) または **z** (倍精度複素数) を示す。

行列の因子分解用のルーチン

この節では、行列の因子分解用の LAPACK ルーチンについて説明する。以下の因子分解をサポートしている。

- LU 因子分解
- 実対称正定値行列のコレスキー因子分解
- エルミート正定値行列のコレスキー因子分解
- 実対称行列と複素対称行列の Bunch-Kaufman 因子分解
- エルミート行列の Bunch-Kaufman 因子分解

LU 因子分解の計算には、フル格納と帯格納形式の行列を使用できる。コレスキー因子分解では、フル格納、圧縮格納、帯格納形式を使用できる。Bunch-Kaufman 因子分解では、フル格納と圧縮格納形式を使用できる。

?getrf

$m \times n$ の一般行列の LU 因子分解を行う。

構文

Fortran 77:

```

call sgetrf(m, N, a, lda, Ipiv, info)
call dgetrf(m, N, a, lda, Ipiv, info)
call cgetrf(m, N, a, lda, Ipiv, info)
call zgetrf(m, N, a, lda, Ipiv, info)

```

Fortran 95:

```
CALL GETRF(a [,ipiv] [,info])
```

説明

このルーチンは、次の式に従って、 $m \times n$ の一般行列 A の LU 因子分解を実行する。

$$A = PLU,$$

ここで、 P は置換行列、 L は単位対角成分を含む下三角 ($m > n$ の場合は下台形)、 U は上三角 ($m < n$ の場合上台形) である。通常 A は正方行列 ($m = n$) で、 L と U はいずれも三角行列である。このルーチンでは、行を交換し、部分的にピボット演算を行う。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
a	REAL (sgetrf の場合) DOUBLE PRECISION (dgetrf の場合) COMPLEX (cgetrf の場合) DOUBLE COMPLEX (zgetrf の場合) 配列、次元は ($lda, *$)。行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。

出力パラメーター

a	L と U によって上書きされる。 L の単位対角成分は格納されない。
$ipiv$	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス: 行 i は行 $ipiv(i)$ と交換される。

info INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、 u_{ii} は 0 である。因子分解は完了したが *U* は完全に特異である。連立線形方程式の解の算出に係数 *U* を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getrf ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。
ipiv 長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

算出された *L* と *U* は、摂動行列 $A + E$ の正確な係数になる。
 $|E| \leq c(\min(m, n)) \varepsilon P |L| |U|$

c(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(2/3)n^3$ (*m* = *n* の場合)
 $(1/3)n^2(3m-n)$ (*m* > *n* の場合)
 $(1/3)m^2(3n-m)$ (*m* < *n* の場合)

複素数型の演算回数は、この 4 倍になる。

m = *n* でこのルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?getrs](#) $AX = B$ または $A^T X = B$ または $A^H X = B$ を解く。
[?gecon](#) *A* の条件数を推定する。
[?getri](#) *A* の逆行列を計算する。

?gbtrf

m × *n* の一般帯行列の LU 因子分解を行う。

構文

Fortran 77:

```
call sgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call dgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call cgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
call zgbtrf(m, N, kl, ku, ab, ldab, Ipiv, info)
```

Fortran 95:

```
call gbrtrf(a [,kl] [,m] [,Ipivot] [,Info])
```

説明

このルーチンは、 k_l 個の非ゼロの劣対角成分と k_u 個の非ゼロの優対角成分を含む $m \times n$ の帯行列 A の LU 因子分解を実行する。通常は、 A は正方行列 ($m = n$) で、次の式が成り立つ。

$$A = PLU$$

P は置換行列である。 L は、単位対角成分を含む、各列の非ゼロの成分の数が k_l 個以下の下三角行列である。 U は、 $k_l + k_u$ 個の優対角成分を含む上三角帯行列である。このルーチンは、行を交換して、部分的にピボット演算を行う (行の交換によって、 U に k_l 個の優対角成分が追加される)。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
k_l	INTEGER。 A の帯内の劣対角成分の数 ($k_l \geq 0$)。
k_u	INTEGER。 A の帯内の優対角成分の数 ($k_u \geq 0$)。
ab	REAL (sgbrtrf の場合) DOUBLE PRECISION (dgbtrf の場合) COMPLEX (cgbtrf の場合) DOUBLE COMPLEX (zgbtrf の場合) 配列、次元は ($ldab, *$)。 配列 ab には、行列 A が帯形式で格納される (「 行列の格納形式 」を参照)。 ab の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
$ldab$	INTEGER。配列 ab の第 1 次元。 ($ldab \geq 2k_l + k_u + 1$)

出力パラメーター

ab	L と U によって上書きされる。 U の対角成分と $k_l + k_u$ 個の優対角成分は、 ab の最初の $1 + k_l + k_u$ 行に格納される。 L の計算に使用される乗数は、次の k_l 行に格納される。
$ipiv$	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス: 行 i は行 $ipiv(i)$ と交換される。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、 u_{ii} は 0 である。因子分解は完了したが U は完全に特異である。連立線形方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbtrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2*k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>k1</i>	省略した場合、 $k1 = ku$ とみなす。
<i>ku</i>	$ku = lda - 2*k1 - 1$ として復元する。
<i>m</i>	省略した場合、 $m = n$ とみなす。

アプリケーション・ノート

算出された *L* と *U* は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(k1 + ku + 1) \varepsilon P |L| |U|$$

$c(k)$ は、 k の適度な 1 次関数で、 ε はマシンの精度である。

実数型の浮動小数点演算の合計回数は、約 $2n(ku+1)k1$ から $2n(k1+ku+1)k1$ までの範囲である。複素数型の演算回数は、この 4 倍になる。これらの推定では、 $k1$ と ku は $\min(m, n)$ よりはるかに小さいものとする。

$m = n$ でこのルーチンを呼び出した後、以下のルーチンを呼び出せる。

?gbtrs	$AX = B$ または $A^T X = B$ または $A^H X = B$ を解く。
?qbcon	<i>A</i> の条件数を推定する。

?gttrf

三重対角行列の LU 因子分解を行う。

構文

Fortran 77:

```
call sgtrrf(N, dl, d, du, du2, Ipiv, info)
call dgtrrf(N, dl, d, du, du2, Ipiv, info)
call cgtrrf(N, dl, d, du, du2, Ipiv, info)
call zgtrrf(N, dl, d, du, du2, Ipiv, info)
```

Fortran 95:

```
CALL GTTRF(dl, d, du, du2 [,ipiv] [,info])
```

説明

このルーチンは、次の形式で、実数または複素三重対角行列 A の LU 因子分解を実行する。

$$A = PLU,$$

P は置換行列、 L は単位対角成分を持つ下二重対角行列、 U は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角行列である。このルーチンは、部分的なピボット演算で行を交換し、消去を実行する。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
$d1, d, du$	REAL (sgttrf の場合) DOUBLE PRECISION (dgttrf の場合) COMPLEX (cgttrf の場合) DOUBLE COMPLEX (zgttrf の場合) A の成分を格納する配列。 次元 ($n-1$) の配列 $d1$ は、 A の劣対角成分を格納する。 次元 n の配列 d は、 A の対角成分を格納する。 次元 ($n-1$) の配列 du は、 A の優対角成分を格納する。

出力パラメーター

$d1$	A の LU 因子分解で得られた行列 L を定義する、($n-1$) 個の乗数によって上書きされる。
d	A の LU 因子分解で得られた上三角行列 U の n 個の対角成分によって上書きされる。
du	U の最初の優対角成分の ($n-1$) 個の成分によって上書きされる。
$du2$	REAL (sgttrf の場合) DOUBLE PRECISION (dgttrf の場合) COMPLEX (cgttrf の場合) DOUBLE COMPLEX (zgttrf の場合) 配列、次元は ($n-2$)。終了時に、 $du2$ には U の 2 番目の優対角成分の ($n-2$) 個の成分が格納される。
$ipiv$	INTEGER。 配列、次元は (n)。 ピボットのインデックス: 行 i は行 $ipiv(i)$ と交換される。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。 $info=i$ の場合、 u_{ii} は 0 である。因子分解は完了したが U は完全に特異である。連立線形方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gttrf ルーチンのインターフェイス特有の詳細を以下に示す。

<code>d1</code>	長さ (n-1) のベクトルを格納する。
<code>d</code>	長さ (n) のベクトルを格納する。
<code>du</code>	長さ (n-1) のベクトルを格納する。
<code>du2</code>	長さ (n-2) のベクトルを格納する。
<code>ipiv</code>	長さ (n) のベクトルを格納する。

アプリケーション・ノート

?gbtrs	$AX=B$ または $A^T X=B$ または $A^H X=B$ を解く。
?gbcon	A の条件数を推定する。

?potrf

対称(エルミート)正定値行列のコレスキー因子分解を行う。

構文

Fortran 77:

```
call spotrf(uplo, N, a, lda, info)
call dpotrf(uplo, N, a, lda, info)
call cpotrf(uplo, N, a, lda, info)
call zpotrf(uplo, N, a, lda, info)
```

Fortran 95:

```
CALL POTRF(a [,uplo] [,info])
```

説明

このルーチンは、対称(複素データの場合はエルミート)正定値行列 A のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo}='U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo}='L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
-------------------	---

	$uplo = 'U'$ の場合、配列 a には行列 A の上三角部分が格納され、 A は $U^H U$ として因子分解される。
	$uplo = 'L'$ の場合、配列 a には行列 A の下三角部分が格納され、 A は LL^H として因子分解される。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
a	REAL (spotrf の場合) DOUBLE PRECISION (dspotrf の場合) COMPLEX (cspotrf の場合) DOUBLE COMPLEX (zspotrf の場合) 配列、次元は ($lda, *$)。 配列 a には、行列 A の上三角部分または下三角部分を格納する ($uplo$ を参照)。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。

出力パラメーター

a	$uplo$ の指定に従って、 a の上三角部分または下三角部分が、コレスキー係数 U あるいは L によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、次数 i の先頭の Minor 行列式 (および行列 A そのもの) が正定値でないため、因子分解を完了できない。これは、行列 A の構成に誤りがあることを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

potrf ルーチンのインターフェイス特有の詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

$uplo = 'U'$ の場合、算出された係数 U は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon |U|^H |U|, \quad |e_{ij}| \leq c(n)\varepsilon \sqrt{a_{ii} a_{jj}}$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

$uplo = 'L'$ の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ 、複素数型の場合は約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

?potrs	$AX = B$ を解く。
?pocon	A の条件数を推定する。

[?potri](#) A の逆行列を計算する。

?pptrf

圧縮格納形式による対称(エルミート)正定値行列のコレスキー因子分解を行う。

構文

Fortran 77:

```
call spptrf(uplo, N, ap, info)
call dpptrf(uplo, N, ap, info)
call cpptrf(uplo, N, ap, info)
call zpptrf(uplo, N, ap, info)
```

Fortran 95:

```
CALL PPTRF(a [,uplo] [,info])
```

説明

このルーチンは、対称(複素データの場合はエルミート)正定値圧縮行列 A のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
 A の上三角部分と下三角部分のどちらを配列 ap にパックするか、また A をどのように因子分解するかを指定する。
 uplo = 'U' の場合、配列 ap には行列 A の上三角部分が格納され、 A は $U^H U$ として因子分解される。
 uplo = 'L' の場合、配列 ap には行列 A の下三角部分が格納され、 A は LL^H として因子分解される。

n INTEGER。行列 A の次数 ($n \geq 0$)。

ap REAL (spptrf の場合)
 DOUBLE PRECISION (dpptrf の場合)
 COMPLEX (cpptrf の場合)
 DOUBLE COMPLEX (zpptrf の場合)
 配列、次元は $\max(1, n(n+1)/2)$ 以上。
 配列 ap には、(uplo の指定に従って) 行列 A の上三角部分または下三角部分を圧縮格納形式で格納する(「[行列の格納形式](#)」を参照)。

出力パラメーター

<i>ap</i>	<i>uplo</i> の指定に従って、圧縮格納形式の <i>A</i> の上三角部分または下三角部分が、コレスキー係数 <i>U</i> または <i>L</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、次数 <i>i</i> の先頭の小行列式 (および行列 <i>A</i> そのもの) が正定値でないため、因子分解を完了できない。これは、行列 <i>A</i> の構成に誤りがあることを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pptrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

uplo = 'U' の場合、算出された係数 *U* は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon |U|^H |U|, \quad |e_{ij}| \leq c(n)\varepsilon \sqrt{a_{ii}a_{jj}}$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ で、複素数型の場合は約 $(4/3)n^3$ になる。

このルーチン呼び出し後、以下のルーチン呼び出せる。

?pptrs	$AX = B$ を解く。
?ppcon	<i>A</i> の条件数を推定する。
?pptri	<i>A</i> の逆行列を計算する。

?pbtrf

対称(エルミート)正定値帯行列のコレスキー因子分解を行う。

構文

Fortran 77:

```
call spbtrf(uplo, N, kd, ab, ldab, info)
call dpbtrf(uplo, N, kd, ab, ldab, info)
```

```
call cpbtrf(uplo, N, kd, ab, ldab, info)
call zpbtrf(uplo, N, kd, ab, ldab, info)
```

Fortran 95:

```
CALL PBTRF(a [,uplo] [,info])
```

説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値帯行列 A のコレスキー因子分解を実行する。

$$A = U^H U \quad (\text{uplo='U' の場合})$$

$$A = LL^H \quad (\text{uplo='L' の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを配列 <i>ab</i> に格納するか、また A をどのように因子分解するかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> には行列 A の上三角部分が格納され、 A は $U^H U$ として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> には行列 A の下三角部分が格納され、 A は LL^H として因子分解される。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab</i>	REAL (spbtrf の場合) DOUBLE PRECISION (dpbtrf の場合) COMPLEX (cpbtrf の場合) DOUBLE COMPLEX (zpbtrf の場合) 配列、次元は (<i>ldab</i> , *)。 配列 <i>ap</i> には、(<i>uplo</i> で指定された) 行列 A の上三角部分または下三角部分が帯形式で格納される (「 行列の格納形式 」を参照)。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 ($ldab \geq kd + 1$)。

出力パラメーター

<i>ap</i>	<i>uplo</i> の指定に従って、帯格納形式の A の上三角部分または下三角部分が、コレスキー係数 U または L によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、次数 <i>i</i> の先頭の Minor 行列式 (および行列 A そのもの) が正定値でないため、因子分解を完了できない。これは、行列 A の構成に誤りがあることを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbtrf ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ $(kd+1, n)$ の配列 *A* を格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

uplo = 'U' の場合、算出された係数 *U* は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(kd+1)\varepsilon |U^H| |U|, \quad |e_{ij}| \leq c(kd+1)\varepsilon \sqrt{a_{ii}a_{jj}}$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

uplo = 'L' の場合も、同様の推定が成り立つ。

実数型の場合はの浮動小数点演算の合計回数は、約 $n(kd+1)^2$ になる。複素数型の演算回数は、この 4 倍になる。これらの推定では、 kd は n よりはるかに小さいものとする。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?pbtrs](#) $AX = B$ を解く。

[?pbcon](#) *A* の条件数を推定する。

?pttrf

対称 (エルミート) 正定値三重対角行列の因子分解を行う。

構文

Fortran 77:

```
call spttrf(N, d, e, info)
call dpttrf(N, d, e, info)
call cpttrf(N, d, e, info)
call zpttrf(N, d, e, info)
```

Fortran 95:

```
CALL PTTRF(d, e [,info])
```

説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値三重対角行列 *A* の因子分解を実行する。

$A = LDL^H$ で、 D は対角行列、 L は単位下二重対角行列である。この因子分解の形式は、 $A = U^H D U$ とみなせる。 D は単位上二重対角行列である。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
d	REAL (spttrf、cpttrf の場合) DOUBLE PRECISION (dpttrf、zpttrf の場合) 配列、次元は (n)。 A の対角成分を格納する。
e	REAL (spttrf の場合) DOUBLE PRECISION (dpttrf の場合) COMPLEX (cpttrf の場合) DOUBLE COMPLEX (zpttrf の場合) 配列、次元は ($n - 1$)。 A の劣対角成分を格納する。

出力パラメーター

d	A の LDL^H 因子分解で得られた対角行列 D の n 個の対角成分によって上書きされる。
e	A の因子分解で得られた単位二重対角係数 L または U の ($n - 1$) 個の非対角成分の成分によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、次数 i の先頭の <small>小行列式</small> (および行列 A そのもの) が正定値でない。 $i < n$ の場合、因子分解を完了できない。 $i = n$ の場合、因子分解は完了したが、 $d(n) = 0$ である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pttrf ルーチンのインターフェイス特有の詳細を以下に示す。

d	長さ (n) のベクトルを格納する。
e	長さ ($n-1$) のベクトルを格納する。

?sytrf

対称行列の Bunch-Kaufman 因子分解を行う。

構文

Fortran 77:

```
call ssytrf(uplo, N, a, lda, ipiv, work, lwork, info)
call dsytrf(uplo, N, a, lda, ipiv, work, lwork, info)
```

```
call csytrf(uplo, N, a, lda, ipiv, work, lwork, info)
call zsytrf(uplo, N, a, lda, ipiv, work, lwork, info)
```

Fortran 95:

```
CALL SYTRF(a [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、対称行列の **Bunch-Kaufman** 因子分解を実行する。

$uplo='U'$ の場合、 $A = PUDU^T P^T$

$uplo='L'$ の場合、 $A = PLDL^T P^T$

A は入力行列、 P は置換行列、 U と L は単位対角成分を含む上三角行列と下三角行列、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。 U と L は、 D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメーター

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 $uplo='U'$ の場合、行列 A の上三角部分を配列 a に格納し、 A を $PUDU^T P^T$ として因子分解する。 $uplo='L'$ の場合、行列 A の下三角部分を配列 a に格納し、 A を $PLDL^T P^T$ として因子分解する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
a	REAL (ssytrf の場合) DOUBLE PRECISION (dsytrf の場合) COMPLEX (csytrf の場合) DOUBLE COMPLEX (zsytrf の場合) 配列、次元は ($lda, *$)。 配列 a には、行列 A の上三角部分または下三角部分を格納する ($uplo$ を参照)。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
$work$	a と同じタイプ。次元 $lwork$ のワークスペース配列。
$lwork$	INTEGER。配列 $work$ のサイズ ($lwork \geq n$)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「 アプリケーション・ノート 」を参照。

出力パラメーター

<i>a</i>	<i>a</i> の上三角部分または下三角部分は、ブロック対角行列 <i>D</i> の各成分と、係数 <i>U</i> (または <i>L</i>) の計算に使用された乗数によって上書きされる。
<i>work</i> (1)	<i>info</i> =0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv</i> (<i>i</i>) = <i>k</i> > 0 の場合、 <i>d_{ii}</i> は 1×1 のブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列と交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> -1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> +1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d_{ii}</i> は 0 である。因子分解は完了したが、 <i>D</i> は完全に特異で、ゼロである。連立線形方程式の解の算出に <i>D</i> を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sytrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n***blocksize* に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

U と *L* の 2×2 の単位対角ブロックと単位対角成分は格納されない。*U* と *L* のそれ以外の成分は、配列 *a* の対応する列に格納される。ただし、*U* または *L* を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての $i=1 \dots n$ について $ipiv(i) = i$ が成り立つ場合は、 $U(L)$ のすべての非対角成分は、配列 a の対応する成分に明示的に格納される。

$uplo = 'U'$ の場合、算出された係数 U と D は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

$uplo = 'L'$ の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ 、複素数型の場合は約 $(4/3)n^3$ になる。

このルーチン呼び出した後、以下のルーチン呼び出せる。

?sytrs	$AX = B$ を解く。
?sycon	A の条件数を推定する。
?sytri	A の逆行列を計算する。

?hetrf

複素数型エルミート行列の *Bunch-Kaufman* 因子分解を行う。

構文

Fortran 77:

```
call chetrf(uplo, N, a, lda, ipiv, work, lwork, info)
call zhetrf(uplo, N, a, lda, ipiv, work, lwork, info)
```

Fortran 95:

```
CALL HETRF(a [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、エルミート行列の *Bunch-Kaufman* 因子分解を実行する。

$uplo = 'U'$ の場合、 $A = PUDU^H P^T$

$uplo = 'L'$ の場合、 $A = PLDL^H P^T$

A は入力行列、 P は置換行列、 U と L は単位対角成分を含む上三角行列と下三角行列、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。 U と L は、 D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメーター

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。
--------	---

	$uplo = 'U'$ の場合、行列 A の上三角部分を配列 a に格納し、 A を $PUDU^H P^T$ として因子分解する。 $uplo = 'L'$ の場合、行列 A の下三角部分を配列 a に格納し、 A を $PLDL^H P^T$ として因子分解する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
a	COMPLEX (chetrif の場合) DOUBLE COMPLEX (zhetrif の場合) 配列、次元は ($lda, *$)。 配列 a には、行列 A の上三角部分または下三角部分を格納する ($uplo$ を参照)。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
$work$	a と同じタイプ。次元 $lwork$ のワークスペース配列。
$lwork$	INTEGER。配列 $work$ のサイズ。 $lwork \geq n$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「 アプリケーション・ノート 」を参照。

出力パラメーター

a	a の上三角部分または下三角部分は、ブロック対角行列 D の各成分と、係数 U (または L) の計算に使用された乗数によって上書きされる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と D のブロック構造の各成分が格納される。 $ipiv(i) = k > 0$ の場合、 d_{ii} は 1×1 のブロックである。 A の i 番目の行と列は、 k 番目の行と列と交換される。 $uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。 A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。 $uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。 A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、 d_{ii} は 0 である。因子分解は完了したが、 D は完全に特異で、ゼロである。連立線形方程式の解の算出に D を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetrf ルーチンのインターフェイス特有の詳細を以下に示す。

`a` サイズ (n,n) の行列 A を格納する。
`ipiv` 長さ (n) のベクトルを格納する。
`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

このルーチンは、正定値行列かどうかわからないエルミート行列に最適である。 A が実際に正定値行列の場合、このルーチンは交換を実行せず、 D 内に 2×2 の対角ブロックは作成されない。

パフォーマンスを最適化するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ~ 64) である。

必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

U と L の 2×2 の単位対角ブロックと単位対角成分は格納されない。 U と L のそれ以外の成分は、配列 a の対応する列に格納される。ただし、 U または L を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての $i = 1 \dots n$ について $ipiv(i) = i$ が成り立つ場合は、 U (L) のすべての非対角成分は、配列 a の対応する成分に明示的に格納される。

$uplo = 'U'$ の場合、算出された係数 U と D は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(n) \varepsilon P |U| |D| |U^T| P^T$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

$uplo = 'L'$ の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約 $(4/3)n^3$ になる。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?hetrs](#) $AX = B$ を解く。
[?hecon](#) A の条件数を推定する。
[?hetri](#) A の逆行列を計算する。

?sptf

圧縮格納形式による対称行列の *Bunch-Kaufman* 因子分解を行う。

構文

Fortran 77:

```
call ssptf(uplo, N, ap, ipiv, info)
call dsptf(uplo, N, ap, ipiv, info)
call csptf(uplo, N, ap, ipiv, info)
call zsptf(uplo, N, ap, ipiv, info)
```

Fortran 95:

```
CALL SPTRF(a [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、圧縮格納形式による対称行列 A の *Bunch-Kaufman* 因子分解を実行する。

$uplo='U'$ の場合、 $A = PUDU^T P^T$

$uplo='L'$ の場合、 $A = PLDL^T P^T$

P は置換行列、 U と L は単位対角成分を含む上三角行列と下三角行列、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。 U と L は、 D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメーター

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを配列 ap にパックするか、また A をどのように因子分解するかを指定する。 $uplo='U'$ の場合、行列 A の上三角部分を配列 ap に格納し、 A を $PUDU^T P^T$ として因子分解する。 $uplo='L'$ の場合、行列 A の下三角部分を配列 ap に格納し、 A を $PLDL^T P^T$ として因子分解する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
ap	REAL (ssptf の場合) DOUBLE PRECISION (dsptf の場合) COMPLEX (csptf の場合) DOUBLE COMPLEX (zsptf の場合) 配列、次元は $\max(1, n(n+1)/2)$ 以上。 配列 ap には、($uplo$ の指定に従って) 行列 A の上三角部分または下三角部分を圧縮格納形式で格納する (「行列の格納形式」 を参照)。

出力パラメーター

<i>ap</i>	(<i>uplo</i> の指定に従って) <i>A</i> の上三角部分または下三角部分は、ブロック対角行列 <i>D</i> の各成分と、係数 <i>U</i> (または <i>L</i>) の計算に使用された乗数によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv</i> (<i>i</i>) = <i>k</i> > 0 の場合、 <i>d</i> _{<i>ii</i>} は 1×1 のブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列と交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> -1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> +1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d</i> _{<i>ii</i>} は 0 である。因子分解は完了したが、 <i>D</i> は完全に特異で、ゼロである。連立線形方程式の解の算出に <i>D</i> を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sptrf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

U と *L* の 2×2 の単位対角ブロックと単位対角成分は格納されない。*U* と *L* のそれ以外の成分は、行列 *A* の対応する列に格納される。ただし、*U* または *L* を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての $i=1 \dots n$ で *ipiv*(*i*) = *i* が成り立つ場合は、*U* (*L*) のすべての非対角成分は、圧縮形式で明示的に格納される。

uplo = 'U' の場合、算出された係数 *U* と *D* は、摂動行列 *A* + *E* の正確な係数になる。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T$$

c(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。

uplo = 'L' の場合も、算出された *L* と *D* について同様の推定が成り立つ。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ 、複素数型の場合は約 $(4/3)n^3$ になる。

このルーチン呼び出した後、以下のルーチン呼び出せる。

[?spttrs](#) $AX = B$ を解く。
[?spcon](#) A の条件数を推定する。
[?spttri](#) A の逆行列を計算する。

?hptrf

圧縮格納形式による複素数型エルミート行列の
 Bunch-Kaufman 因子分解を行う。

構文

Fortran 77:

```
call chptrf(uplo, N, ap, ipiv, info)
call zhptrf(uplo, N, ap, ipiv, info)
```

Fortran 95:

```
CALL HPTRF(a [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、圧縮格納形式によるエルミート行列の **Bunch-Kaufman** 因子分解を実行する。

$uplo = 'U'$ の場合、 $A = PUDU^H P^T$

$uplo = 'L'$ の場合、 $A = PLDL^H P^T$

A は入力行列、 P は置換行列、 U と L は単位対角成分を含む上三角行列と下三角行列、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。 U と L は、 D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持っている。

入力パラメーター

$uplo$ CHARACTER*1。'U' または 'L' でなければならない。
 A の上三角部分と下三角部分のどちらをパックするか、また A をどのように因子分解するかを指定する。
 $uplo = 'U'$ の場合、行列 A の上三角部分を配列 ap に格納し、 A を $PUDU^H P^T$ として因子分解する。
 $uplo = 'L'$ の場合、行列 A の下三角部分を配列 ap に格納し、 A を $PLDL^H P^T$ として因子分解する。

n INTEGER。行列 A の次数 ($n \geq 0$)。

ap COMPLEX (chptrf の場合)
 DOUBLE COMPLEX (zhptrf の場合)
 配列、次元は $\max(1, n(n+1)/2)$ 以上。

配列 *ap* には、(*uplo* の指定に従って) 行列 *A* の上三角部分または下三角部分を圧縮格納形式で格納する ([「行列の格納形式」](#) を参照)。

出力パラメーター

<i>ap</i>	(<i>uplo</i> の指定に従って) <i>A</i> の上三角部分または下三角部分は、ブロック対角行列 <i>D</i> の各成分と、係数 <i>U</i> (または <i>L</i>) の計算に使用された乗数によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 交換の結果と <i>D</i> のブロック構造の各成分が格納される。 <i>ipiv</i> (<i>i</i>) = <i>k</i> > 0 の場合、 <i>d_{ii}</i> は 1×1 のブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列と交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> -1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> +1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d_{ii}</i> は 0 である。因子分解は完了したが、 <i>D</i> は完全に特異で、ゼロである。連立線形方程式の解の算出に <i>D</i> を使用すると、ゼロ除算が発生する。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、[「Fortran-95 インターフェイス規則」](#) を参照のこと。

hptrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

U と *L* の 2×2 の単位対角ブロックと単位対角成分は格納されない。*U* と *L* のそれ以外の成分は、配列 *a* の対応する列に格納される。ただし、*U* または *L* を明示的に復元するには、さらに行の交換が必要である (その必要はほとんどない)。

すべての $i=1 \dots n$ について *ipiv*(*i*) = *i* が成り立つ場合は、*U* (*L*) のすべての非対角成分は、配列 *a* の対応する成分に明示的に格納される。

$uplo = 'U'$ の場合、算出された係数 U と D は、摂動行列 $A + E$ の正確な係数になる。

$$|E| \leq c(n) \varepsilon P |U| |D| |U^T| P^T$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

$uplo = 'L'$ の場合も、算出された L と D について同様の推定が成り立つ。

浮動小数点演算の合計回数は、約 $(4/3)n^3$ になる。

このルーチン呼び出した後、以下のルーチン呼び出せる。

?hptrs	$AX = B$ を解く。
?hpcon	A の条件数を推定する。
?hptri	A の逆行列を計算する。

連立線形方程式を解くためのルーチン

この節では、連立線形方程式を解くための LAPACK ルーチンについて説明する。通常は、これらのルーチン呼び出す前に、連立方程式の行列を因子分解する必要がある (この章の「[行列の因子分解用のルーチン](#)」を参照)。ただし、解を求める連立方程式が三角行列を持つ場合は、因子分解の必要はない。

?getrs

LU 因子分解された正方行列を使って、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call sgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call dgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call cgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
call zgetrs(trans, N, nrhs, a, lda, Ipiv, b, ldb, info)
```

Fortran 95:

```
CALL GETRS(a, ipiv, b [,trans] [,info])
```

説明

このルーチンは、以下の連立線形方程式を X について解く。

$$AX = B \quad (\text{trans} = 'N' \text{ の場合})$$

$$A^T X = B \quad (\text{trans} = 'T' \text{ の場合})$$

$$A^H X = B \quad (\text{trans} = 'C' \text{ の場合。ただし、複素行列のみ})$$

このルーチン呼び出す前に、[?getrf](#) を呼び出して、 A の LU 因子分解を行う必要がある。

入力パラメーター

<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、 $AX = B$ を X について解く。 <i>trans</i> = 'T' の場合、 $A^T X = B$ を X について解く。 <i>trans</i> = 'C' の場合、 $A^H X = B$ を X について解く。
<i>n</i>	INTEGER。A の次数。B の行数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。

a, b	REAL (sgetrs の場合) DOUBLE PRECISION (dgetrs の場合) COMPLEX (cgetrs の場合) DOUBLE COMPLEX (zgetrs の場合) 配列 : $a(lda, *)$, $b ldb, *$ 。 配列 a には、行列 A が格納される。配列 b には、行列 B が格納される。この行列の 列は、連立方程式の右辺である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b の第 2 次 元は、 $\max(1, nrhs)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?getrf によって返される $ipiv$ 配列。

出力パラメーター

b	解の行列 X によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getrs ルーチンのインターフェイス特有の詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
b	サイズ $(n, nrhs)$ の行列 B を格納する。
$ipiv$	長さ (n) のベクトルを格納する。
$trans$	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon P|L||U|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_{\infty}(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_{\infty}(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$ 、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_{\infty}(A)$ を推定するには、[?gecon](#) を呼び出す。
解の精度を改善して誤差を推定するには、[?gerfs](#) を呼び出す。

?gbtrs

LU 因子分解された帯行列を使って、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call sgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call dgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call cgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call zgbtrs(trans, n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL GBTRS(a, b, ipiv, [,kl] [,trans] [,info])
```

説明

このルーチンは、以下の連立線形方程式を X について解く。

$$AX = B \quad (\text{trans} = \text{'N'} \text{ の場合})$$

$$A^T X = B \quad (\text{trans} = \text{'T'} \text{ の場合})$$

$$A^H X = B \quad (\text{trans} = \text{'C'} \text{ の場合。ただし、複素行列のみ})$$

A は、 kl 個の非ゼロの劣対角成分と ku 個の非ゼロの優対角成分を持つ、LU 因子分解された n の一般帯行列である。このルーチンを呼び出す前に、[?gbtrf](#) を呼び出して、 A の LU 因子分解を行う必要がある。

入力パラメーター

<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。
<i>n</i>	INTEGER。A の次数。B の行数 ($n \geq 0$)。
<i>kl</i>	INTEGER。A の帯内の劣対角成分の数 ($kl \geq 0$)。
<i>ku</i>	INTEGER。A の帯内の優対角成分の数 ($ku \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。

<i>ab, b</i>	REAL (sgbtrs の場合) DOUBLE PRECISION (dgbtrs の場合) COMPLEX (cgbtrs の場合) DOUBLE COMPLEX (zgbtrs の場合) 配列 : <i>ab</i> (<i>ldab</i> , *), <i>b</i> (<i>ldb</i> , *)。 配列 <i>ab</i> には、行列 <i>A</i> が帯形式で格納される (「 行列の格納形式 」を参照)。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 ($ldab \geq 2kl + ku + 1$)。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?qgbtrf によって返される <i>ipiv</i> 配列。

出力パラメーター

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2 * kl + ku + 1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>kl</i>	省略した場合、 $kl = ku$ とみなす。
<i>ku</i>	$lda - 2 * kl - 1$ として復元する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(kl + ku + 1) \varepsilon P|L||U|$$

$c(k)$ は、*k* の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 *x* は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(kl + ku + 1) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \|A^{-1}\|_\infty \|A\|_\infty \|x\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は $2n(ku + 2kl)$ になる。複素数型の演算回数は、この 4 倍になる。これらの推定では、 kl と ku は $\min(m, n)$ よりはるかに小さいものとする。

条件数 $\kappa_\infty(A)$ を推定するには、[?qbccon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?qbrfs](#) を呼び出す。

?gttrs

?gttrf によって行われた LU 因子分解を使用し、三重対角行列を係数行列とする連立線形方程式を解く。

構文

Fortran 77:

```
call sgttrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call dgttrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call cgttrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
call zgttrs(trans, n, nrhs, dl, d, du, du2, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL GTTRS(dl, d, du, du2, b, ipiv [,trans] [,info])
```

説明

このルーチンは、複数の右辺を持つ以下の連立線形方程式を、 X について解く。

$AX = B$ (trans='N' の場合)

$A^T X = B$ (trans='T' の場合)

$A^H X = B$ (trans='C' の場合。ただし、複素行列のみ)

このルーチンを呼び出す前に、[?gttrf](#) を呼び出して、 A の LU 因子分解を行う必要がある。

入力パラメーター

trans CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。

方程式の形式を指定する。

	$trans = 'N'$ の場合、 $AX = B$ を X について解く。
	$trans = 'T'$ の場合、 $A^T X = B$ を X について解く。
	$trans = 'C'$ の場合、 $A^H X = B$ を X について解く。
n	INTEGER。 A の次数 ($n \geq 0$)。
$nrhs$	INTEGER。 右辺の数、すなわち、 B の列数 ($nrhs \geq 0$)。
$d1, d, du, du2, b$	REAL (sgttrs の場合) DOUBLE PRECISION (dgttrs の場合) COMPLEX (cgttrs の場合) DOUBLE COMPLEX (zgttrf の場合) 配列: $d1(n-1), d(n), du(n-1), du2(n-2), b(ldb, nrhs)$ 配列 $d1$ には、 A の LU 因子分解で得られた行列 L を定義する ($n-1$) 個の乗数が格納される。 配列 d には、 A の LU 因子分解で得られた上三角行列 U の n 個の対角成分が格納される。 配列 du には、 U の最初の優対角成分の ($n-1$) 個の成分が格納される。 配列 $du2$ には、 U の 2 番目の優対角成分の ($n-2$) 個の成分が格納される。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。
ldb	INTEGER。 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
$ipiv$	INTEGER。 配列、次元は (n)。 ?gttrf によって返される $ipiv$ 配列。

出力パラメーター

b	解の行列 X によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtrrs ルーチンのインターフェイス特有の詳細を以下に示す。

$d1$	長さ ($n-1$) のベクトルを格納する。
d	長さ (n) のベクトルを格納する。
du	長さ ($n-1$) のベクトルを格納する。
$du2$	長さ ($n-2$) のベクトルを格納する。
b	サイズ ($n, nrhs$) の行列 B を格納する。
$ipiv$	長さ (n) のベクトルを格納する。
$trans$	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。
 $|E| \leq c(n)\varepsilon P|L||U|$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \|A^{-1}\|_\infty \|A\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $7n$ (n 回の除算を含む) で、複素数型の場合は $34n$ ($2n$ 回の除算を含む) である。

条件数 $\kappa_\infty(A)$ を推定するには、[?gtcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?gtrfs](#) を呼び出す。

?potrs

コレスキー因子分解された対称 (エルミート) 正定値行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call spotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call dpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call cpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
call zpotrs(uplo, N, nrhs, a, lda, b, ldb, info)
```

Fortran 95:

```
CALL POTRS(a, b [,uplo] [,info])
```

説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値行列 A を持つ連立線形方程式 $AX = B$ を、 A のコレスキー因子分解に基づいて、 X について解く。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンを呼び出す前に、[?potrf](#) を呼び出して、 A のコレスキー因子分解を行う必要がある。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 a には、コレスキー因子分解 $A = U^H U$ の係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 a には、コレスキー因子分解 $A = LL^H$ の係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>a, b</code>	REAL (<code>spotrs</code> の場合) DOUBLE PRECISION (<code>dpotrs</code> の場合) COMPLEX (<code>cpotrs</code> の場合) DOUBLE COMPLEX (<code>zpotrs</code> の場合) 配列: $a(lda, *)$, $b(ldb, *)$ 。 配列 a には、係数 U または L を格納する (<code>uplo</code> を参照)。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`potrs` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>b</code>	サイズ $(n, nrhs)$ の行列 B を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

`uplo = 'U'` の場合、各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon |U^H| |U|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。`uplo = 'L'` の場合も、同様の推定が成り立つ。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$ 、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?pocon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?porfs](#) を呼び出す。

?pptrs

コレスキー因子分解された圧縮形式の対称 (エルミート) 正定値行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call spptrs(uplo, N, nrhs, ap, b, ldb, info)
call dpptrs(uplo, N, nrhs, ap, b, ldb, info)
call cpptrs(uplo, N, nrhs, ap, b, ldb, info)
call zpptrs(uplo, N, nrhs, ap, b, ldb, info)
```

Fortran 95:

```
CALL PPTRS(a, b [,uplo] [,info])
```

説明

このルーチンは、圧縮形式の対称 (複素データの場合はエルミート) 正定値行列 A を持つ連立線形方程式 $AX = B$ を、 A のコレスキー因子分解に基づいて、 X について解く。

$$A = U^H U \quad (\text{uplo} = 'U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチン呼び出す前に、[?pptrf](#) を呼び出して、 A のコレスキー因子分解を行う必要がある。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 a には、コレスキー因子分解 $A = U^H U$ の係数 U を圧縮形式で格納する。 <code>uplo = 'L'</code> の場合、配列 a には、コレスキー因子分解 $A = LL^H$ の係数 L を圧縮形式で格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ap, b</code>	REAL (<code>spptrs</code> の場合) DOUBLE PRECISION (<code>dpptrs</code> の場合) COMPLEX (<code>cpptrs</code> の場合) DOUBLE COMPLEX (<code>zpptrs</code> の場合) 配列: $ap(*), b(l\text{db}, *)$ ap の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 ap には、 <code>uplo</code> の指定に従って、係数 U または L が圧縮格納形式で格納される(「 行列の格納形式 」を参照)。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pptrs` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<code>b</code>	サイズ $(n, nrhs)$ の行列 B を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

`uplo = 'U'` の場合、各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon |U^H| |U|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

`uplo = 'L'` の場合も、同様の推定が成り立つ。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x) \varepsilon$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $2n^2$ 、複素数型の場合は $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?ppcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?pprfs](#) を呼び出す。

?pbtrs

コレスキー因子分解された対称 (エルミート) 正定値帯行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call spbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call cpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbtrs(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

Fortran 95:

```
CALL PBTRS(a, b [,uplo] [,info])
```

説明

このルーチンは、対称 (複素データの場合はエルミート) 正定値 **帯**行列 A を持つ連立線形方程式 $AX=B$ を、 A のコレスキー因子分解に基づいて、 X について解く。

$$A = U^H U \quad (\text{uplo}='U' \text{ の場合})$$

$$A = LL^H \quad (\text{uplo}='L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチン呼び出す前に、[?pbtrf](#) を呼び出して、 A のコレスキー因子分解を帯格納形式で計算する必要がある。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 a には、因子分解 $A = U^H U$ の係数 U を帯格納形式で格納する。 <code>uplo = 'L'</code> の場合、配列 a には、因子分解 $A = LL^H$ の係数 L を帯格納形式で格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>kd</code>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ab, b</code>	REAL (spbtrs の場合) DOUBLE PRECISION (dpbtrs の場合) COMPLEX (cpbtrs の場合) DOUBLE COMPLEX (zpbtrs の場合) 配列: $ab(ldab, *)$, $b(l db, *)$ 。 配列 ab には、因子分解ルーチンによって返されるコレスキー係数を帯格納形式で格納する。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 ab の第 2 次元は $\max(1, n)$ 以上でなければならない。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldab</code>	INTEGER。配列 ab の第 1 次元。 ($ldab \geq kd + 1$)。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

- a* Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ $(kd+1, n)$ の配列 *A* を格納する。
- b* サイズ $(n, nrhs)$ の行列 *B* を格納する。
- uplo* 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(kd+1)\varepsilon P|U^H||U| \quad \text{または} \quad |E| \leq c(kd+1)\varepsilon P|L^H||L|$$

$c(k)$ は、 k の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(kd+1) \text{cond}(A, x)\varepsilon$$

ただし、 $\text{cond}(A, x) = \|A^{-1}\|_\infty \|A\|_\infty \|x\|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算のおおよその回数は、実数型の場合は $4n*kd$ 、複素数型の場合は $16n*kd$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?pbcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?pbrfs](#) を呼び出す。

?pttrs

?pttrf によって行われた因子分解を使用して、対称(エルミート)正定値三重対角行列を係数行列とする連立線形方程式を解く。

構文

Fortran 77:

```
call spttrs(n, nrhs, d, e, b, ldb, info)
call dpttrs(n, nrhs, d, e, b, ldb, info)
```

```
call cpttrs(uplo, n, nrhs, d, e, b, ldb, info)
call zpttrs(uplo, n, nrhs, d, e, b, ldb, info)
```

Fortran 95:

```
CALL PTTRS(d, e, b [,info])
CALL PTTRS(d, e, b [,uplo] [,info])
```

説明

このルーチンは、対称(エルミート)正定値三重対角行列 A を係数行列とする連立線形方程式 $AX=B$ を、 X について解く。

このルーチン呼び出す前に、[?pttrf](#) を呼び出して、 A の LDL^H または $U^H DU$ 因子分解を行う必要がある。

入力パラメーター

<code>uplo</code>	CHARACTER*1。cpttrs/zpttrs でのみ使用される。 'U' または 'L' でなければならない。 三重対角行列 A の優対角成分と劣対角成分のどちらを格納するかと、 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 e には A の優対角成分が格納され、 A は $U^H DU$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 e には A の劣対角成分が格納され、 A は LDL^H として因子分解される。
<code>n</code>	INTEGER。 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。 右辺の数、すなわち、行列 B の列数 ($nrhs \geq 0$)。
<code>d</code>	REAL (spttrs、cpttrs の場合) DOUBLE PRECISION (dpttrs、zpttrs の場合) 配列、次元は (n)。 ?pttrf による因子分解で得られた対角行列 D の対角成分を格納する。
<code>e, b</code>	REAL (spttrs の場合) DOUBLE PRECISION (dpttrs の場合) COMPLEX (cpttrs の場合) DOUBLE COMPLEX (zpttrs の場合) 配列 : $e(n-1), b(ldb, nrhs)$ 。 配列 e には、 ?pttrf による因子分解で得られた単位二重対角係数 U または L の ($n-1$) 個の非対角成分が格納される (<code>uplo</code> を参照)。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。
<code>ldb</code>	INTEGER。 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pttrfsf ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>uplo</i>	複素型でのみ使用される。'U' または 'L' でなければならない。デフォルト値は 'U'。

?sytrs

UDU または LDL 因子分解された対称行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call ssytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call dsytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call csytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zsytrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL SYTRS(a, b, ipiv [,uplo] [,info])
```

説明

このルーチンは、対称行列 *A* を持つ連立線形方程式 $AX = B$ を、*A* の Bunch-Kaufman 因子分解に基づいて、*X* について解く。

uplo='U' の場合、 $A = PUDU^T P^T$

uplo='L' の場合、 $A = PLDL^T P^T$

P は置換行列、*U* と *L* は単位対角成分を含む上三角行列と下三角行列、*D* は対称ブロック対角行列である。行列 *B* の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?sytrf](#) によって返される係数 *U* (または *L*) と配列 *ipiv* を与える必要がある。

入力パラメーター

uplo CHARACTER*1. 'U' または 'L' でなければならない。
 入力行列 *A* の因子分解の方法を指定する。

`uplo = 'U'` の場合、配列 `a` には、因子分解 $A = PUDU^T P^T$ の上三角係数 U を格納する。
`uplo = 'L'` の場合、配列 `a` には、因子分解 $A = PLDL^T P^T$ の下三角係数 L を格納する。

<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ipiv</code>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?sytrf によって返される <code>ipiv</code> 配列。
<code>a, b</code>	REAL (<code>ssytrs</code> の場合) DOUBLE PRECISION (<code>dsytrs</code> の場合) COMPLEX (<code>csytrs</code> の場合) DOUBLE COMPLEX (<code>zsytrs</code> の場合) 配列: <code>a(lda, *)</code> , <code>b(ldb, *)</code> 。 配列 <code>a</code> には、係数 U または L を格納する (<code>uplo</code> を参照)。 配列 <code>b</code> には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>b</code> の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`sytrs` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>b</code>	サイズ $(n, nrhs)$ の行列 B を格納する。
<code>ipiv</code>	長さ (n) のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。
 $c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_{\infty} / \|x\|_{\infty} \leq \|A^{-1}\|_{\infty} \|A\|_{\infty} = \kappa_{\infty}(A)$ 。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T \quad \text{または} \quad |E| \leq c(n)\varepsilon P|L||D||L^T|P^T$$

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon$$

$\operatorname{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約 $2n^2$ 、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?sycon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?syrrfs](#) を呼び出す。

?hetrs

UDU または LDL 因子分解されたエルミート行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call chetrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
call zhetrs(uplo, n, nrhs, a, lda, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL HETRS(a, b, ipiv [,uplo] [,info])
```

説明

このルーチンは、エルミート行列 A を持つ連立線形方程式 $AX = B$ を、 A の Bunch-Kaufman 因子分解に基づいて、 X について解く。

$uplo = 'U'$ の場合、 $A = PUDU^HP^T$

$uplo = 'L'$ の場合、 $A = PLDL^HP^T$

P は置換行列、 U と L は単位対角成分を含む上三角行列と下三角行列、 D は対称ブロック対角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?hetrf](#) によって返される係数 U (または L) と配列 $ipiv$ を与える必要がある。

入力パラメーター

$uplo$ CHARACTER*1. 'U' または 'L' でなければならない。
入力行列 A の因子分解の方法を指定する。

$uplo = 'U'$ の場合、配列 a には、因子分解 $A = PUDU^H P^T$ の上三角係数 U を格納する。

$uplo = 'L'$ の場合、配列 a には、因子分解 $A = PLDL^H P^T$ の上三角係数 L を格納する。

n	INTEGER。行列 A の次数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数 ($nrhs \geq 0$)。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?hetrf によって返される $ipiv$ 配列。
a, b	COMPLEX ($chetrs$ の場合)。 DOUBLE COMPLEX ($zhetrs$ の場合)。 配列: $a(lda, *)$, $b ldb, *$ 。 配列 a には、係数 U または L を格納する ($uplo$ を参照)。 配列 b には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

b	解の行列 X によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetrs ルーチンのインターフェイス特有の詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
b	サイズ $(n, nrhs)$ の行列 B を格納する。
$ipiv$	長さ (n) のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。 $c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x) \varepsilon$$

$$|E| \leq c(n)\varepsilon P|U||D||U^H|P^T \quad \text{または} \quad |E| \leq c(n)\varepsilon P|L||D||L^H|P^T$$

ただし、 $\text{cond}(A,x) = \| |A^{-1}| |A| |x| \|_{\infty} / \|x\|_{\infty} \leq \|A^{-1}\|_{\infty} \|A\|_{\infty} = \kappa_{\infty}(A)$ 。

$\text{cond}(A,x)$ は、 $\kappa_{\infty}(A)$ よりはるかに小さくなる場合がある。

1つの右辺ベクトルの浮動小数点演算の合計回数は、約 $8n^2$ になる。

条件数 $\kappa_{\infty}(A)$ を推定するには、[?hecon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?herfs](#) を呼び出す。

?sptrs

UDU または LDL 因子分解された圧縮格納形式による対称行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call sspttrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call dspttrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call cspttrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zsptrrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL SPTRRS(a, b, ipiv [,uplo] [,info])
```

説明

このルーチンは、対称行列 A を持つ連立線形方程式 $AX=B$ を、 A の Bunch-Kaufman 因子分解に基づいて、 X について解く。

$\text{uplo}='U'$ の場合、 $A = PUDU^T P^T$

$\text{uplo}='L'$ の場合、 $A = PLDL^T P^T$

P は置換行列、 U と L は単位対角成分を含む圧縮形式の上三角行列と下三角行列、 D は対称ブロック対角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。このルーチンには、因子分解ルーチン [?spttrf](#) によって返される係数 U (または L) と配列 ipiv を与える必要がある。

入力パラメーター

uplo CHARACTER*1. 'U' または 'L' でなければならない。
入力行列 A の因子分解の方法を指定する。

$uplo = 'U'$ の場合、配列 ap には、因子分解 $A = PUDU^T P^T$ の圧縮形式の係数 U を格納する。
 $uplo = 'L'$ の場合、配列 ap には、因子分解 $A = PLDL^T P^T$ の圧縮形式の係数 L を格納する。

n	INTEGER。行列 A の次数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数 ($nrhs \geq 0$)。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?spturf によって返される $ipiv$ 配列。
ap, b	REAL ($ssptrs$ の場合) DOUBLE PRECISION ($dsptrs$ の場合) COMPLEX ($csptrs$ の場合) DOUBLE COMPLEX ($zsptrs$ の場合) 配列 : $ap(*), b(l, db, *)$ ap の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 ap には、 $uplo$ の指定に従って、係数 U または L が圧縮格納形式で格納される (「 行列の格納形式 」を参照)。 配列 b には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

b	解の行列 X によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

$spturs$ ルーチンのインターフェイス特有の詳細を以下に示す。

a	Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
b	サイズ $(n, nrhs)$ の行列 B を格納する。
$ipiv$	長さ (n) のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。 $c(n)$ は n の適度な 1 次関数で、 ϵ はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_{\infty} / \|x\|_{\infty} \leq \|A^{-1}\|_{\infty} \|A\|_{\infty} = \kappa_{\infty}(A)$ 。

$$|E| \leq c(n)\varepsilon P|U||D||U^T|P^T \quad \text{または} \quad |E| \leq c(n)\varepsilon P|L||D||L^T|P^T$$

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon$$

$\operatorname{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1 つの右辺ベクトルの浮動小数点演算の合計回数は、実数型の場合は約 $2n^2$ 、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?spcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?sprfs](#) を呼び出す。

?hptrs

UDU または LDL 因子分解された圧縮格納形式によるエルミート行列を使って、連立線形方程式を解く。

構文

Fortran 77:

```
call chptrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zhptrs(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL HPTRS(a, b, ipiv [,uplo] [,info])
```

説明

このルーチンは、エルミート行列 A を持つ連立線形方程式 $AX = B$ を、 A の Bunch-Kaufman 因子分解に基づいて、 X について解く。

$uplo = 'U'$ の場合、 $A = PUDU^H P^T$

$uplo = 'L'$ の場合、 $A = PLDL^H P^T$

P は置換行列、 U と L は単位対角成分を含む**圧縮形式**の上三角行列と下三角行列、 D は対称ブロック対角行列である。行列 B の各列に格納されている複数の右辺を使用して、この連立方程式を解く。

このルーチンには、(U または L を格納する) 配列 ap と、因子分解ルーチン [?hptrf](#) によって返される形式の配列 $ipiv$ を与える必要がある。

入力パラメーター

<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 入力行列 <i>A</i> の因子分解の方法を指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、因子分解 $A = PUDU^H P^T$ の上三角係数 <i>U</i> を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、因子分解 $A = PLDL^H P^T$ の上三角係数 <i>L</i> を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?hptrf によって返される <i>ipiv</i> 配列。
<i>ap, b</i>	COMPLEX (<i>chptrs</i> の場合)。 DOUBLE COMPLEX (<i>zhptrs</i> の場合)。 配列: <i>ap</i> (*), <i>b</i> (<i>ldb</i> , *) <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <i>ap</i> には、 <i>uplo</i> の指定に従って、係数 <i>U</i> または <i>L</i> が圧縮格納形式で格納される (「 行列の格納形式 」を参照)。 配列 <i>b</i> には、行列 <i>B</i> を格納する。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hptrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。*c*(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 *x* は、次の誤差範囲を満たす。

$$|E| \leq c(n)\varepsilon P|U||D||U^H|P^T \quad \text{または} \quad |E| \leq c(n)\varepsilon P|L||D||L^H|P^T$$

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。

1つの右辺ベクトルの浮動小数点演算の合計回数は、複素数型の場合は約 $8n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?hpcon](#) を呼び出す。

解の精度を改善して誤差を推定するには、[?hprfs](#) を呼び出す。

?trtrs

三角行列を使って、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call strtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call dtrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ctrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ztrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
```

Fortran 95:

```
CALL TRTRS(a, b [,uplo] [,trans] [,diag] [,info])
```

説明

このルーチンは、行列 B に格納された複数の右辺を使用して、三角行列 A を持つ以下の連立線形方程式を X について解く。

$$AX = B \quad (\text{trans} = 'N' \text{ の場合})$$

$$A^T X = B \quad (\text{trans} = 'T' \text{ の場合})$$

$$A^H X = B \quad (\text{trans} = 'C' \text{ の場合。ただし、複素行列のみ})$$

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
A が上三角行列か、下三角行列かを指定する。

	$uplo = 'U'$ の場合、 A は上三角行列である。 $uplo = 'L'$ の場合、 A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N'、'T'、または'C'のいずれかでなければならない。 $trans = 'N'$ の場合、 $AX = B$ を X について解く。 $trans = 'T'$ の場合、 $A^T X = B$ を X について解く。 $trans = 'C'$ の場合、 $A^H X = B$ を X について解く。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 $diag = 'N'$ の場合、 A は単位三角行列ではない。 $diag = 'U'$ の場合、 A は単位三角行列である。 A の対角成分は 1 とみなされ、配列 a 内で参照されない。
<i>n</i>	INTEGER。 A の次数。 B の行数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	REAL (<i>strtrs</i> の場合) DOUBLE PRECISION (<i>dtrtrs</i> の場合) COMPLEX (<i>ctrtrs</i> の場合) DOUBLE COMPLEX (<i>ztrtrs</i> の場合) 配列: $a(lda, *)$, $b(ldb, *)$ 。 配列 a には、行列 A が格納される。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 X によって上書きされる。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

trtrs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の行列 A を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 B を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

`trans` 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

`diag` 'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 b について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \operatorname{cond}(A, x)\varepsilon, \text{ provided } c(n) \operatorname{cond}(A, x)\varepsilon < 1$$

ただし、 $\operatorname{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\operatorname{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は n^2 、複素数型の場合は $4n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?ptrcon](#) を呼び出す。

解の誤差を推定するには、[?trrfs](#) を呼び出す。

?tptrs

圧縮形式の三角行列を使って、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call stptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call dtptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call ctptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
call ztptrs(uplo, trans, diag, n, nrhs, ap, b, ldb, info)
```

Fortran 95:

```
CALL TPTRS(a, b [,uplo] [,trans] [,diag] [,info])
```

説明

このルーチンは、行列 B に格納された複数の右辺を使用して、圧縮形式の三角行列 A を持つ以下の連立線形方程式を X について解く。

$AX = B$ (trans='N' の場合)

$A^T X = B$ (trans='T' の場合)

$A^H X = B$ (trans='C' の場合。ただし、複素行列のみ)

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 <i>trans</i> = 'N' の場合、 $AX = B$ を X について解く。 <i>trans</i> = 'T' の場合、 $A^T X = B$ を X について解く。 <i>trans</i> = 'C' の場合、 $A^H X = B$ を X について解く。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。A の次数。B の行数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>ap, b</i>	REAL (<i>stptrs</i> の場合) DOUBLE PRECISION (<i>dtptrs</i> の場合) COMPLEX (<i>ctptrs</i> の場合) DOUBLE COMPLEX (<i>ztptrs</i> の場合) 配列: <i>ap</i> (*), <i>b</i> (<i>ldb</i> ,*) <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <i>ap</i> は行列 A を圧縮格納形式で格納する。 (「 行列の格納形式 」を参照)。 配列 <i>b</i> には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 X によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tpters ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\varepsilon, \text{ provided } c(n) \text{cond}(A, x)\varepsilon < 1$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル *b* の浮動小数点演算のおおよその回数は、実数型の場合は n^2 、複素数型の場合は $4n^2$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?tpcon](#) を呼び出す。
解の誤差を推定するには、[?tprfs](#) を呼び出す。

?tbtrs

帯三角行列を使って、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call stbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call dtbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ctbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
call ztbtrs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, info)
```

Fortran 95:

```
CALL TBTRS(a, b [,uplo] [,trans] [,diag] [,info])
```

説明

このルーチンは、行列 B に格納された複数の右辺を使用して、帯三角行列 A を持つ以下の連立線形方程式を X について解く。

$$AX = B \quad (\text{trans} = \text{'N'} \text{ の場合})$$

$$A^T X = B \quad (\text{trans} = \text{'T'} \text{ の場合})$$

$$A^H X = B \quad (\text{trans} = \text{'C'} \text{ の場合。ただし、複素行列のみ})$$

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、 A は上三角行列である。 <i>uplo</i> = 'L' の場合、 A は下三角行列である。
<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 <i>trans</i> = 'N' の場合、 $AX = B$ を X について解く。 <i>trans</i> = 'T' の場合、 $A^T X = B$ を X について解く。 <i>trans</i> = 'C' の場合、 $A^H X = B$ を X について解く。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。 A の次数。 B の行数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数 ($nrhs \geq 0$)。
<i>ab, b</i>	REAL (<i>stbtrs</i> の場合) DOUBLE PRECISION (<i>dtbtrs</i> の場合) COMPLEX (<i>ctbtrs</i> の場合) DOUBLE COMPLEX (<i>ztbtrs</i> の場合) 配列: <i>ab</i> (<i>ldab</i> , *), <i>b</i> (<i>ldb</i> , *)。 配列 <i>ab</i> には、行列 A を帯格納形式で格納する。 配列 <i>b</i> には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $ldab \geq kd + 1$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

b 解の行列 X によって上書きされる。

info INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbtrs ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ (*kd*+1, *n*) の配列 *A* を格納する。

b サイズ (*n*, *nrhs*) の行列 *B* を格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

trans 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

diag 'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

各右辺 *b* について算出された解は、摂動連立方程式 $(A + E)x = b$ の正確な解になる。

$$|E| \leq c(n)\varepsilon |A|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。
 x_0 が真の解である場合、算出された解 x は、次の誤差範囲を満たす。

$$\frac{\|x - x_0\|_\infty}{\|x\|_\infty} \leq c(n) \text{cond}(A, x)\varepsilon, \text{ provided } c(n) \text{cond}(A, x)\varepsilon < 1$$

ただし、 $\text{cond}(A, x) = \| |A^{-1}| |A| |x| \|_\infty / \|x\|_\infty \leq \|A^{-1}\|_\infty \|A\|_\infty = \kappa_\infty(A)$ 。

$\text{cond}(A, x)$ は、 $\kappa_\infty(A)$ よりはるかに小さくなる場合がある。 A^T と A^H の条件数は、 $\kappa_\infty(A)$ に等しいことも、等しくないこともある。

1 つの右辺ベクトル b の浮動小数点演算のおおよその回数は、実数型の場合は $2n * kd$ 、複素数型の場合は $8n * kd$ になる。

条件数 $\kappa_\infty(A)$ を推定するには、[?tbcon](#) を呼び出す。
 解の誤差を推定するには、[?tbrfs](#) を呼び出す。

条件数を推定するためのルーチン

この節では、行列の条件数を推定するための LAPACK ルーチンについて説明する。条件数は、連立線形方程式の解の誤差の分析に使用される（「[誤差の分析](#)」を参照）。行列がほとんど特異である（ゼロに近い）場合は、条件数が非常に大きくなる場合がある。このため、これらのルーチンでは、実際には条件数の逆数を計算する。

?gcon

一般行列の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

構文

Fortran 77:

```
call sgecon(norm, n, a, lda, anorm, rcond, work, iwork, info)
call dgecon(norm, n, a, lda, anorm, rcond, work, iwork, info)
call cgecon(norm, n, a, lda, anorm, rcond, work, rwork, info)
call zgecon(norm, n, a, lda, anorm, rcond, work, rwork, info)
```

Fortran 95:

```
CALL GCON(a, anorm, rcond [,norm] [,info])
```

説明

このルーチンは、次のように、一般行列 A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H) \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)\end{aligned}$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?getrf](#) を呼び出して、 A の LU 因子分解を計算する。

入力パラメーター

<i>norm</i>	CHARACTER*1。'1'、'0'、または'I'のいずれかでなければならない。 <i>norm</i> = '1' または '0' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	REAL (sgecon の場合) DOUBLE PRECISION (dgecon の場合) COMPLEX (cgecon の場合) DOUBLE COMPLEX (zgecon の場合) 配列: $a(lda, *)$, $work(*)$ 。

配列 *a* には、[?getrf](#) によって返される、LU 因子分解された行列 *A* を格納する。

a の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 *work* は、このルーチン用のワークスペースである。

work の次元は $\max(1, 4*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (「 説明 」を参照)。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cgecon</i> の場合) DOUBLE PRECISION (<i>zgecon</i> の場合) ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond = 0$ に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gecon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチンを呼び出す場合は、連立線形方程式 $Ax = b$ または $A^H x = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gbcon

帯行列の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

構文

Fortran 77:

```
call sgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, iwork, info)
call dgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, iwork, info)
call cgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, rwork, info)
call zgbcon(norm, n, kl, ku, ab, ldab, ipiv, anorm, rcond, work, rwork, info)
```

Fortran 95:

```
call gbcon(a, ipiv, anorm, rcond [,kl] [,norm] [,info])
```

説明

このルーチンは、次のように、一般帯行列 A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H)$$

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?gbtrf](#) を呼び出して、 A の LU 因子分解を計算する。

入力パラメーター

<i>norm</i>	CHARACTER*1。'1'、'0'、または'I'のいずれかでなければならない。 <i>norm</i> = '1' または '0' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kl</i>	INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。
<i>ku</i>	INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 ($ldab \geq 2kl + ku + 1$)。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?gbtrf によって返される <i>ipiv</i> 配列。
<i>ab, work</i>	REAL (sgbcon の場合) DOUBLE PRECISION (dgbcon の場合) COMPLEX (cgbcon の場合) DOUBLE COMPLEX (zgbcon の場合) 配列: <i>ab</i> (<i>ldab</i> , *), <i>work</i> (*)。

配列 *ab* には、[?gbtrf](#) によって返される、因子分解後の帯行列 *A* を格納する。

ab の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
配列 *work* は、このルーチン用のワークスペースである。

work の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (「 説明 」を参照)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cgbccon</i> の場合) DOUBLE PRECISION (<i>zgbcon</i> の場合) ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2*k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。
<i>k1</i>	省略した場合、 <i>k1</i> = <i>ku</i> とみなす。
<i>ku</i>	<i>ku</i> = $lda - 2*k1 - 1$ として復元する。

アプリケーション・ノート

計算された $rcond$ は、 ρ (真の条件数の逆数) より小さくはない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出す場合は、連立線形方程式 $Ax = b$ または $A^H x = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n(ku + 2k1)$ で、複素数型の場合は約 $8n(ku + 2k1)$ である。

?gtcon

?gttrf によって行われた因子分解を使用して、三重対角行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call sgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork, info)
call dgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, iwork, info)
call cgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info)
call zgtcon(norm, n, dl, d, du, du2, ipiv, anorm, rcond, work, info)
```

Fortran 95:

```
call gtcon(dl, d, du, du2, ipiv, anorm, rcond [,norm] [,info])
```

説明

このルーチンは、実数または複素三重対角行列 A の (1- ノルムまたは無限ノルムの) 条件数の逆数を、次のように推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty\end{aligned}$$

推定値は $\|A^{-1}\|$ について得られる。条件数の逆数は、 $rcond = 1 / (\|A\| \|A^{-1}\|)$ として計算される。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?gttrf](#) を呼び出して、 A の LU 因子分解を計算する。

入力パラメーター

$norm$	CHARACTER*1。'1'、'O'、または'I'のいずれかでなければならない。 $norm = '1'$ または $'O'$ の場合、ルーチンは $\kappa_1(A)$ を推定する。 $norm = 'I'$ の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。

<i>d1, d, du, du2</i>	REAL (sgtcon の場合) DOUBLE PRECISION (dgtcon の場合) COMPLEX (cgtcon の場合) DOUBLE COMPLEX (zgtcon の場合) 配列 : $d1(n-1), d(n), du(n-1), du2(n-2)$ 。 配列 <i>d1</i> には、 ?gttrf による <i>A</i> の LU 因子分解で得られた行列 <i>L</i> を定義する、 $(n-1)$ 個の乗数が格納される。 配列 <i>d</i> には、 <i>A</i> の LU 因子分解で得られた上三角行列 <i>U</i> の n 個の対角成分が格納される。 配列 <i>du</i> には、 <i>U</i> の最初の優対角成分の $(n-1)$ 個の成分が格納される。 配列 <i>du2</i> には、 <i>U</i> の 2 番目の優対角成分の $(n-2)$ 個の成分が格納される。
<i>ipiv</i>	INTEGER。 配列、次元は (n) 。 ?gttrf によって返される、ピボットのインデックスの配列。
<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>元</i> の行列 <i>A</i> のノルム (説明を参照)。
<i>work</i>	REAL (sgtcon の場合) DOUBLE PRECISION (dgtcon の場合) COMPLEX (cgtcon の場合) DOUBLE COMPLEX (zgtcon の場合) ワークスペース配列、次元は $(2*n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (n) 。 実数型でのみ使用される。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond=0$ に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info=0</i> の場合、正常に終了したことを示す。 <i>info=-i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtcon ルーチンのインターフェイス特有の詳細を以下に示す。

d1 長さ $(n-1)$ のベクトルを格納する。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>du</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>du2</i>	長さ (<i>n</i> -2) のベクトルを格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax=b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pocon

対称 (エルミート) 正定値行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call spocon(uplo, n, a, lda, anorm, rcond, work, iwork, info)
call dpocon(uplo, n, a, lda, anorm, rcond, work, iwork, info)
call cpocon(uplo, n, a, lda, anorm, rcond, work, rwork, info)
call zpocon(uplo, n, a, lda, anorm, rcond, work, rwork, info)
```

Fortran 95:

```
CALL POCON(a, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、対称 (エルミート) 正定値行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称またはエルミート行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?potrf](#) を呼び出して、 A のコレスキー因子分解を行う必要がある。

入力パラメーター

uplo CHARACTER*1. 'U' または 'L' でなければならない。
入力行列 A の因子分解の方法を指定する。

uplo = 'U' の場合、配列 *a* には、因子分解 $A = U^H U$ の上三角係数 U を格納する。

`uplo = 'L'` の場合、配列 `a` には、因子分解 $A = LL^H$ の下三角係数 L を格納する。

`n` INTEGER。行列 A の次数 ($n \geq 0$)。

`a, work` REAL (`spocon` の場合)
DOUBLE PRECISION (`dpocon` の場合)
COMPLEX (`cpocon` の場合)
DOUBLE COMPLEX (`zpocon` の場合)
配列: `a(lda, *)`, `work(*)`。

配列 `a` には、[?potrf](#) によって返される、因子分解後の行列 A を格納する。

`a` の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 `work` は、このルーチン用のワークスペースである。

`work` の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

`lda` INTEGER。 `a` の第 1 次元。 $lda \geq \max(1, n)$ 。

`anorm` REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元の行列 A のノルム ([説明を参照](#))。

`iwork` INTEGER。
ワークスペース配列、次元は $\max(1, n)$ 以上。

`rwork` REAL (`cpocon` の場合)
DOUBLE PRECISION (`zpocon` の場合)
ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

`rcond` REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。
推定値のアンダーフローが発生した場合は、`rcond = 0` に設定される。この場合、行列は有効な精度で特異になる。ただし、`rcond` が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。

`info` INTEGER。
`info = 0` の場合、正常に終了したことを示す。
`info = -i` の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pocon` ルーチンのインターフェイス特有の詳細を以下に示す。

`a` サイズ (n, n) の行列 A を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された $rcond$ は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax=b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?ppcon

圧縮格納形式の対称 (エルミート) 正定値行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call sppcon(uplo, n, ap, anorm, rcond, work, iwork, info)
call dppcon(uplo, n, ap, anorm, rcond, work, iwork, info)
call cppcon(uplo, n, ap, anorm, rcond, work, rwork, info)
call zppcon(uplo, n, ap, anorm, rcond, work, rwork, info)
```

Fortran 95:

```
CALL PPCON(a, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、圧縮形式の対称 (エルミート) 正定値行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称またはエルミート行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?pptrf](#) を呼び出して、 A のコレスキー因子分解を行う必要がある。

入力パラメーター

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 ap には、因子分解 $A = U^H U$ の上三角係数 U を格納する。 $uplo = 'L'$ の場合、配列 ap には、因子分解 $A = LL^H$ の下三角係数 L を格納する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
$ap, work$	REAL (sppcon の場合) DOUBLE PRECISION (dppcon の場合) COMPLEX (cppcon の場合) DOUBLE COMPLEX (zppcon の場合) 配列: $ap(*), work(*)$ 。

配列 *ap* には、[?pptrf](#) によって返される、因子分解後の圧縮形式の行列 *A* を格納する。

ap の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。

配列 *work* は、このルーチン用のワークスペースである。

work の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (説明を参照)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cppcon</i> の場合) DOUBLE PRECISION (<i>zppcon</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチンを呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pbcon

対称(エルミート)正定値帯行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call spbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call dpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, iwork, info)
call cpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
call zpbcon(uplo, n, kd, ab, ldab, anorm, rcond, work, rwork, info)
```

Fortran 95:

```
CALL PBCON(a, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、対称(エルミート)正定値帯行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称またはエルミート行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- `anorm` を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?pbtrf](#) を呼び出して、 A のコレスキー因子分解を行う必要がある。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ab</code> には、コレスキー因子分解 $A = U^H U$ の上三角係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ab</code> には、因子分解 $A = LL^H$ の下三角係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>kd</code>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<code>ldab</code>	INTEGER。配列 <code>ab</code> の第 1 次元。 ($ldab \geq kd + 1$)。
<code>ab, work</code>	REAL (spbcon の場合) DOUBLE PRECISION (dpbcon の場合) COMPLEX (cpbcon の場合) DOUBLE COMPLEX (zpbcon の場合) 配列: <code>ab(ldab, *)</code> , <code>work(*)</code> 。 配列 <code>ab</code> には、 ?pbtrf によって返される、因子分解後の帯形式の行列 A を格納する。 <code>ab</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 *work* は、このルーチン用のワークスペースである。
work の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (説明を参照)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cpbcon</i> の場合) DOUBLE PRECISION (<i>zpbcon</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチンを呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $4n(kd + 1)$ で、複素数型の場合は約 $16n(kd + 1)$ である。

?ptcon

対称(エルミート)正定値三重対角行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call sptcon(n, d, e, anorm, rcond, work, info)
call dptcon(n, d, e, anorm, rcond, work, info)
call cptcon(n, d, e, anorm, rcond, work, info)
call zptcon(n, d, e, anorm, rcond, work, info)
```

Fortran 95:

```
CALL PTCON(d, e, anorm, rcond [,info])
```

説明

このルーチンは、[?pttrf](#) による因子分解 $A = LDL^H$ または $A = U^H D U$ を使用して、実対称または複素エルミート正定値三重対角行列の (1- ノルムの) 条件数の逆数を、次のように計算する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称またはエルミート行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

ノルム $\|A^{-1}\|_1$ は直接法で計算される。条件数の逆数は、 $rcond = 1 / (\|A\|_1 \|A^{-1}\|_1)$ として計算される。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- `anorm` を $\|A\|_1 = \max_j \sum_i |a_{ij}|$ として計算する。
- [?pttrf](#) を呼び出して、 A の因子分解を計算する。

入力パラメーター

`n` INTEGER。行列 A の次数 ($n \geq 0$)。

`d, work` REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列、次元は (n) 。
配列 `d` には、[?pttrf](#) による A の因子分解で得られた対角行列 D の n 個の対角成分が格納される。
`work` はワークスペース配列である。

`e` REAL (sptcon の場合)
DOUBLE PRECISION (dptcon の場合)
COMPLEX (cptcon の場合)
DOUBLE COMPLEX (zptcon の場合)
配列、次元は $(n-1)$ 。
[?pttrf](#) による因子分解で得られた単位二重対角係数 U または L の非対角成分を格納する。

anorm REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
元の行列 *A* の 1- ノルム (説明を参照)。

出力パラメーター

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
条件数の逆数の推定値。
推定値のアンダーフローが発生した場合は、*rcond* = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtcon ルーチンのインターフェイス特有の詳細を以下に示す。

d 長さ (*n*) のベクトルを格納する。

e 長さ (*n*-1) のベクトルを格納する。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $4n(kd + 1)$ で、複素数型の場合は約 $16n(kd + 1)$ である。

?sycon

対称行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call ssycon(uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call dsycon(uplo, n, a, lda, ipiv, anorm, rcond, work, iwork, info)
call csycon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
call zsycon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
```

Fortran 95:

```
CALL SYCON(a, ipiv, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、対称行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?sytrf](#) を呼び出して、 A の因子分解を計算する。

入力パラメーター

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 a には、因子分解 $A = PUDU^T P^T$ の上三角係数 U を格納する。 $uplo = 'L'$ の場合、配列 a には、因子分解 $A = PLDL^T P^T$ の下三角係数 L を格納する。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
$a, work$	REAL (ssycon の場合) DOUBLE PRECISION (dsycon の場合) COMPLEX (csycon の場合) DOUBLE COMPLEX (zsycon の場合) 配列: $a(lda, *)$, $work(*)$ 。 配列 a には、 ?sytrf によって返される、因子分解後の行列 A を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 $work$ は、このルーチン用のワークスペースである。 $work$ の次元は $\max(1, 2*n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
$ipiv$	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?sytrf によって返される $ipiv$ 配列。
$anorm$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 A のノルム (説明を参照)。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

$rcond$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 $rcond = 0$ に設定さ
---------	--

れる。この場合、行列は有効な精度で特異になる。ただし、 $rcond$ が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sycon ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。
ipiv 長さ (n) のベクトルを格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された $rcond$ は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。

このルーチンを呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hecon

エルミート行列の条件数の逆数を計算する。

構文

Fortran 77:

```
call checon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
call zhecon(uplo, n, a, lda, ipiv, anorm, rcond, work, info)
```

Fortran 95:

```
CALL HECON(a, ipiv, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、エルミート行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ はエルミート行列のため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- *anorm* を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?hetrf](#) を呼び出して、 A の因子分解を計算する。

入力パラメーター

<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 入力行列 <i>A</i> の因子分解の方法を指定する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> には、因子分解 $A = PUDU^H P^T$ の上三角係数 <i>U</i> を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> には、因子分解 $A = PLDL^H P^T$ の上三角係数 <i>L</i> を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>a, work</i>	COMPLEX (<i>checon</i> の場合) DOUBLE COMPLEX (<i>zhecon</i> の場合) 配列: <i>a(lda, *)</i> , <i>work(*)</i> 。 配列 <i>a</i> には、 ?hetrf によって返される、因子分解後の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。 <i>work</i> の次元は $\max(1, 2*n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?hetrf によって返される <i>ipiv</i> 配列。
<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (説明 を参照)。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hecon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された $rcond$ は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax=b$ の解が複数回計算される。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

?spcon

圧縮格納形式の対称行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call sspcon(uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)
call dspcon(uplo, n, ap, ipiv, anorm, rcond, work, iwork, info)
call cspcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
call zspcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
```

Fortran 95:

```
CALL SPCON(a, ipiv, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、圧縮格納形式の対称行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ は対称行列であるため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- $anorm$ を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?sptrf](#) を呼び出して、 A の因子分解を計算する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には、因子分解 $A = PUDU^T P^T$ の圧縮形式の上三角係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には、因子分解 $A = PLDL^T P^T$ の圧縮形式の下三角係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>ap, work</code>	REAL (sspcon の場合) DOUBLE PRECISION (dspcon の場合) COMPLEX (cspcon の場合) DOUBLE COMPLEX (zspcon の場合) 配列: <code>ap(*)</code> , <code>work(*)</code> 。

配列 *ap* には、[?spturf](#) によって返される、因子分解後の圧縮形式の行列 *A* を格納する。

ap の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。

配列 *work* は、このルーチン用のワークスペースである。

work の次元は $\max(1, 2*n)$ 以上でなければならない。

<i>ipiv</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?spturf によって返される <i>ipiv</i> 配列。
<i>anorm</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 <i>A</i> のノルム (説明を参照)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> =0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax=b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hpcon

圧縮格納形式のエルミート行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call chpcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
call zhpcon(uplo, n, ap, ipiv, anorm, rcond, work, info)
```

Fortran 95:

```
CALL HPCON(a, ipiv, anorm, rcond [,uplo] [,info])
```

説明

このルーチンは、次のように、エルミート行列 A の条件数の逆数を推定する。

$$\kappa_1(A) = \|A\|_1 \|A^{-1}\|_1 \quad (A \text{ はエルミート行列のため、} \kappa_\infty(A) = \kappa_1(A)).$$

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- `anorm` を計算する ($\|A\|_1 = \max_j \sum_i |a_{ij}|$ または $\|A\|_\infty = \max_i \sum_j |a_{ij}|$)。
- [?hptrf](#) を呼び出して、 A の因子分解を計算する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には、因子分解 $A = PUDU^T P^T$ の圧縮形式の上三角係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には、因子分解 $A = PLDL^T P^T$ の圧縮形式の下三角係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>ap, work</code>	COMPLEX (<code>chpcon</code> の場合) DOUBLE COMPLEX (<code>zhpcon</code> の場合) 配列: <code>ap(*)</code> , <code>work(*)</code> 。 配列 <code>ap</code> には、 ?hptrf によって返される、因子分解後の圧縮形式の行列 A を格納する。 <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <code>work</code> は、このルーチン用のワークスペースである。 <code>work</code> の次元は $\max(1, 2*n)$ 以上でなければならない。
<code>ipiv</code>	INTEGER。配列、次元は $\max(1, n)$ 以上。 ?hptrf によって返される <code>ipiv</code> 配列。
<code>anorm</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 元の行列 A のノルム (説明を参照)。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

?trcon

三角行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call strcon(norm, uplo, diag, N, a, lda, rcond, work, iwork, info)
call dtrcon(norm, uplo, diag, N, a, lda, rcond, work, iwork, info)
call ctrcon(norm, uplo, diag, N, a, lda, rcond, work, rwork, info)
call ztrcon(norm, uplo, diag, N, a, lda, rcond, work, rwork, info)
```

Fortran 95:

```
CALL TRCON(a, rcond [,uplo] [,diag] [,norm] [,info])
```

説明

このルーチンは、次のように、三角行列 A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H) \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)\end{aligned}$$

入力パラメーター

<i>norm</i>	CHARACTER*1。'1'、'0'、または'I'のいずれかでなければならない。 <i>norm</i> = '1' または '0' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> には、 A の上三角部分を格納する。その他の配列成分は参照されない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> には、 A の下三角部分を格納する。その他の配列成分は参照されない。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (<i>strcon</i> の場合) DOUBLE PRECISION (<i>dtrcon</i> の場合) COMPLEX (<i>ctrcon</i> の場合) DOUBLE COMPLEX (<i>ztrcon</i> の場合) 配列 : <i>a</i> (<i>lda</i> , *), <i>work</i> (*)。 配列 <i>a</i> には、行列 A を格納する。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。 <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>ctrcon</i> の場合) DOUBLE PRECISION (<i>ztrcon</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

trcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tpcon

圧縮格納形式の三角行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call stpcon(norm, uplo, diag, n, ap, rcond, work, iwork, info)
call dtpcon(norm, uplo, diag, n, ap, rcond, work, iwork, info)
call ctpcon(norm, uplo, diag, n, ap, rcond, work, rwork, info)
call ztpcon(norm, uplo, diag, n, ap, rcond, work, rwork, info)
```

Fortran 95:

```
CALL TPCON(a, rcond [,uplo] [,diag] [,norm] [,info])
```

説明

このルーチンは、次のように、圧縮格納形式の三角行列 A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H) \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)\end{aligned}$$

入力パラメーター

<i>norm</i>	CHARACTER*1。'1'、'0'、または'I'のいずれかでなければならない。 <i>norm</i> = '1' または '0' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、 A の上三角部分を圧縮形式で格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、 A の下三角部分を圧縮形式で格納する。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	REAL (stpcon の場合) DOUBLE PRECISION (dtpcon の場合) COMPLEX (ctpcon の場合) DOUBLE COMPLEX (ztpcon の場合) 配列: <i>ap</i> (*), <i>work</i> (*)。 配列 <i>ap</i> には、圧縮形式の行列 A を格納する。 <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。 <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (ctpcon の場合) DOUBLE PRECISION (ztpcon の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tpcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>norm</i>	'1'、'0'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tbcon

三角帯行列の条件数の逆数を推定する。

構文

Fortran 77:

```
call stbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info)
call dtbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, iwork, info)
call ctbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info)
call ztbcon(norm, uplo, diag, n, kd, ab, ldab, rcond, work, rwork, info)
```

Fortran 95:

```
CALL TBCON(a, rcond [,uplo] [,diag] [,norm] [,info])
```

説明

このルーチンは、次のように、三角帯行列 A の条件数の逆数を、1- ノルムまたは無限ノルムによって推定する。

$$\begin{aligned}\kappa_1(A) &= \|A\|_1 \|A^{-1}\|_1 = \kappa_\infty(A^T) = \kappa_\infty(A^H) \\ \kappa_\infty(A) &= \|A\|_\infty \|A^{-1}\|_\infty = \kappa_1(A^T) = \kappa_1(A^H)\end{aligned}$$

入力パラメーター

<i>norm</i>	CHARACTER*1。'1'、'O'、または'I'のいずれかでなければならない。 <i>norm</i> = '1' または 'O' の場合、ルーチンは $\kappa_1(A)$ を推定する。 <i>norm</i> = 'I' の場合、このルーチンは $\kappa_\infty(A)$ を推定する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、 A の上三角部分を圧縮形式で格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、 A の下三角部分を圧縮形式で格納する。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 A は単位三角行列である。対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	REAL (stbcon の場合) DOUBLE PRECISION (dtbcon の場合) COMPLEX (ctbcon の場合) DOUBLE COMPLEX (ztbcon の場合) 配列: <i>ab</i> (<i>ldab</i> , *), <i>work</i> (*)。 配列 <i>ab</i> には、帯行列 A を格納する。 <i>ab</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>work</i> は、このルーチン用のワークスペースである。 <i>work</i> の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 (<i>ldab</i> $\geq kd + 1$)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (ctbcon の場合) DOUBLE PRECISION (ztbcon の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 条件数の逆数の推定値。 推定値のアンダーフローが発生した場合は、 <i>rcond</i> = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、 <i>rcond</i> が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbcon ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ($kd+1, n$) の配列 <i>A</i> を格納する。
<i>norm</i>	'1'、'O'、または 'I' でなければならない。デフォルト値は '1' である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

計算された *rcond* は、 ρ (真の条件数の逆数) より小さくはならない。この値は、実際にはほぼ常に、 10ρ より小さくなる。このルーチン呼び出すと、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n(kd + 1)$ で、複素数型の場合は約 $8n(kd + 1)$ である。

解の精度の改善と誤差の推定

この節では、算出された連立線形方程式の解の精度を改善し、解の誤差を推定するための LAPACK ルーチンについて説明する。これらのルーチン呼び出す前に、連立方程式の行列を因子分解し、解を計算する必要がある（「[行列の因子分解用のルーチン](#)」および「[連立線形方程式を解くためのルーチン](#)」を参照）。

?gerfs

一般行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call sgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call dgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call cgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
call zgerfs(trans, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
```

Fortran 95:

```
CALL GERFS(a, af, ipiv, b, x [,trans] [,ferr] [,berr] [,info])
```

説明

このルーチンは、一般行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ または $A^T X=B$ または $A^H X=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?getrf](#) を呼び出す。
- 解の算出ルーチン [?getrs](#) を呼び出す。

入力パラメーター

<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX=B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X=B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X=B$ である。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。

<i>a, af, b, x, work</i>	REAL (sgerfs の場合) DOUBLE PRECISION (dgerfs の場合) COMPLEX (cgerfs の場合) DOUBLE COMPLEX (zgerfs の場合) 配列： <i>a(lda, *)</i> には、 ?getrf に入力として与えた元の行列 <i>A</i> を格納する。 <i>af(ldaf, *)</i> には、 ?getrf によって返された、因子分解後の行列 <i>A</i> を格納する。 <i>b(l db, *)</i> には、右辺の行列 <i>B</i> が格納される。 <i>x(ldx, *)</i> には、解の行列 <i>X</i> が格納される。 <i>work(*)</i> は、ワークスペース配列である。 <i>a</i> と <i>af</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?getrf によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cgerfs の場合) DOUBLE PRECISION (zgerfs の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gerfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>af</i>	サイズ (<i>n, n</i>) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n, nrhs</i>) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n^2$ 以上、複素数型の場合は $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gbrfs

一般帯行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call sgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call dgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, iwork, info)
call cgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
call zgbtrfs(trans, n, kl, ku, nrhs, ab, ldab, afb, ldafb, ipiv, b, ldb, x, ldx,
             ferr, berr, work, rwork, info)
```

Fortran 95:

```
call gbrfs(a, af, ipiv, b, x [,kl] [,trans] [,ferr] [,berr] [,info])
```

説明

このルーチンは、帯行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ または $A^T X=B$ または $A^H X=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?gbtrf](#) を呼び出す。
- 解の算出ルーチン [?gbtrs](#) を呼び出す。

入力パラメーター

trans CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。

方程式の形式を指定する。

trans = 'N' の場合、連立方程式の形式は $AX=B$ である。

trans = 'T' の場合、連立方程式の形式は $A^T X=B$ である。

trans = 'C' の場合、連立方程式の形式は $A^H X=B$ である。

n INTEGER。行列 A の次数 ($n \geq 0$)。

kl INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。

ku INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。

nrhs INTEGER。右辺の数 ($nrhs \geq 0$)。

ab,afb,b,x,work REAL (sgbrfs の場合)
DOUBLE PRECISION (dgbrfs の場合)
COMPLEX (cgbrfs の場合)
DOUBLE COMPLEX (zgbrfs の場合)

配列:

*ab(ldab, *)* には、[?gbtrf](#) に入力として与えた元の帯行列 A を格納する。この行列は、行 $1 \sim (kl + ku + 1)$ に格納される。

*afb(ldafb, *)* には、[?gbtrf](#) によって返された、因子分解後の帯行列 A を格納する。

*b(ldb, *)* には、右辺の行列 B が格納される。

*x(ldx, *)* には、解の行列 X が格納される。

work()* は、ワークスペース配列である。

ab と afb の第 2 次元は $\max(1, n)$ 以上でなければならない。 b と x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work$ の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

<i>ldab</i>	INTEGER。 ab の第 1 次元。
<i>ldafb</i>	INTEGER。 afb の第 1 次元。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?qbrtf によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cgbrfs</i> の場合) DOUBLE PRECISION (<i>zgbrfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 X 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 ab を意味する。サイズ $(k1+ku+1, n)$ の配列 A を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 afb を意味する。サイズ $(2*k1*ku+1, n)$ の配列 AF を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 B を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 X を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。

trans 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
k1 省略した場合、 $k1 = ku$ とみなす。
ku $ku = lda - k1 - 1$ として復元する。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n(k1 + ku)$ 以上、複素数型の場合は $16n(k1 + ku)$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、実数型の場合は $2n(4k1 + 3ku)$ で、複素数型の場合は $8n(4k1 + 3ku)$ である。繰り返される回数の範囲は 1 ~ 5 になる。前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?gtrfs

三重対角行列を係数行列とする連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call sgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, iwork, info)
call dgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, iwork, info)
call cgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, rwork, info)
call zgtrfs(trans, n, nrhs, dl, d, du, dlf, df, duf, du2, ipiv, b, ldb, x, ldx,
  ferr, berr, work, rwork, info)
```

Fortran 95:

```
call gtrfs(dl, d, du, dlf, df, duf, du2, ipiv, b, x [,trans] [,ferr] [,berr]
  [,info])
```

説明

このルーチンは、三重対角行列 A を係数行列とする、複数の右辺を持つ連立線形方程式 $AX = B$ または $A^T X = B$ または $A^H X = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, \quad |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?qttrf](#) を呼び出す。
- 解の算出ルーチン [?qttrs](#) を呼び出す。

入力パラメーター

<i>trans</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ である。 <i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ である。 <i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ である。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数、すなわち、行列 <i>B</i> の列数 ($nrhs \geq 0$)。
<i>d1, d, du, dlf, df, duf, du2, b, x, work</i>	REAL (sgtrfs の場合) DOUBLE PRECISION (dgtrfs の場合) COMPLEX (cgtrfs の場合) DOUBLE COMPLEX (zgtrfs の場合) 配列: <i>d1</i> 、次元は $(n - 1)$ 。 <i>A</i> の劣対角成分を格納する。 <i>d</i> 、次元は (n) 。 <i>A</i> の対角成分を格納する。 <i>du</i> 、次元は $(n - 1)$ 。 <i>A</i> の優対角成分を格納する。 <i>dlf</i> 、次元は $(n - 1)$ で、 ?qttrf による <i>A</i> の <i>LU</i> 因子分解で得られた行列 <i>L</i> を定義する $(n - 1)$ 個の乗数を格納する。 <i>df</i> 、次元は (n) で、 <i>A</i> の <i>LU</i> 因子分解で得られた上三角行列 <i>U</i> の n 個の対角成分を格納する。 <i>duf</i> 、次元は $(n - 1)$ で、 <i>U</i> の最初の優対角成分の $(n - 1)$ 個の成分を格納する。 <i>du2</i> 、次元は $(n - 2)$ で、 <i>U</i> の 2 番目の優対角成分の $(n - 2)$ 個の成分を格納する。 <i>b(ldb, nrhs)</i> は、右辺の行列 <i>B</i> を格納する。 <i>x(ldx, nrhs)</i> は、 ?qttrs によって計算された解の行列 <i>X</i> を格納する。 <i>work(*)</i> は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 3 * n)$ (実数型の場合) または $\max(1, 2 * n)$ (複素数の場合) 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?gttrf によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。実数型でのみ使用される。
<i>rwork</i>	REAL (<i>cgtrfs</i> の場合) DOUBLE PRECISION (<i>zgtrfs</i> の場合) ワークスペース配列、次元は (<i>n</i>)。複素型でのみ使用される。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d1</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>du</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>d1f</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>df</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>duf</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>du2</i>	長さ (<i>n-2</i>) のベクトルを格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

?porfs

対称 (エルミート) 正定値行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call sporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call cporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call zporfs(uplo, n, nrhs, a, lda, af, ldaf, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

Fortran 95:

```
call porfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、対称 (エルミート) 正定値行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?potrf](#) を呼び出す。
- 解の算出ルーチン [?potrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>af</code> には、コレスキー因子分解 $A = U^H U$ の係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>af</code> には、コレスキー因子分解 $A = LL^H$ の係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。

<i>a, af, b, x, work</i>	REAL (sporfs の場合) DOUBLE PRECISION (dporfs の場合) COMPLEX (cporfs の場合) DOUBLE COMPLEX (zporfs の場合) 配列： <i>a(lda, *)</i> には、 ?potrf に入力として与えた元の行列 <i>A</i> を格納する。 <i>af(ldaf, *)</i> には、 ?potrf によって返された、因子分解後の行列 <i>A</i> を格納する。 <i>b(l db, *)</i> には、右辺の行列 <i>B</i> が格納される。 <i>x(ldx, *)</i> には、解の行列 <i>X</i> が格納される。 <i>work(*)</i> は、ワークスペース配列である。 <i>a</i> と <i>af</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (cporfs の場合) DOUBLE PRECISION (zporfs の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

porfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (n,n) の行列 <i>A</i> を格納する。
<i>af</i>	サイズ (n,n) の行列 <i>AF</i> を格納する。
<i>b</i>	サイズ $(n,nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n,nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n^2$ 以上、複素数型の場合は $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。前進誤差を推定するには、連立線形方程式 $Ax=b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pprfs

圧縮形式の対称(エルミート)正定値行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call spprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, iwork,
            info)
call dpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, iwork,
            info)
call cpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, rwork,
            info)
call zpprfs(uplo, n, nrhs, ap, afp, b, ldb, x, ldx, ferr, berr, work, rwork,
            info)
```

Fortran 95:

```
call pprfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列 A による、複数の右辺を持つ連立線形方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pptrf](#) を呼び出す。
- 解の算出ルーチン [?pptrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>afp</code> には、コレスキー因子分解 $A = U^H U$ の圧縮形式の係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>afp</code> には、コレスキー因子分解 $A = LL^H$ の圧縮形式の係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ap,afp,b,x,work</code>	REAL (<code>spprfs</code> の場合) DOUBLE PRECISION (<code>dpprfs</code> の場合) COMPLEX (<code>cpprfs</code> の場合) DOUBLE COMPLEX (<code>zpprfs</code> の場合) 配列： <code>ap(*)</code> には、 ?pptrf に入力として与えた元の圧縮形式の行列 A を格納する。 <code>afp(*)</code> には、 ?pptrf によって返された、因子分解後の圧縮形式の行列 A を格納する。 <code>b(ldb, *)</code> には、右辺の行列 B が格納される。 <code>x(ldx, *)</code> には、解の行列 X が格納される。 <code>work(*)</code> は、ワークスペース配列である。 配列 <code>ap</code> と <code>afp</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <code>b</code> と <code>x</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work</code> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>ldx</code>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cpprfs</i> の場合) DOUBLE PRECISION (<i>zpprfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pprfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>apf</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n^2$ 以上、複素数型の場合は $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?pbrfs

対称(エルミート)正定値帯行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call spbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call dpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call cpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
call zpbrfs(uplo, n, kd, nrhs, ab, ldab, afb, ldafb, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
```

Fortran 95:

```
call pbrfs(a, af, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、対称(エルミート)正定値帯行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pbtrf](#) を呼び出す。
- 解の算出ルーチン [?pbtrs](#) を呼び出す。

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
 入力行列 A の因子分解の方法を指定する。
 uplo = 'U' の場合、配列 afb には、コレスキー因子分解 $A = U^H U$ の係数 U を格納する。
 uplo = 'L' の場合、配列 afb には、コレスキー因子分解 $A = LL^H$ の係数 L を格納する。

n INTEGER。行列 A の次数 ($n \geq 0$)。

<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数 ($nrhs \geq 0$)。
<i>ab,afb,b,x,work</i>	REAL (<i>spbrfs</i> の場合) DOUBLE PRECISION (<i>dpbrfs</i> の場合) COMPLEX (<i>cpbrfs</i> の場合) DOUBLE COMPLEX (<i>zpbrfs</i> の場合) 配列: <i>ab(ldab, *)</i> には、 ?pbtrf に入力として与えた元の帯行列 <i>A</i> を格納する。 <i>afb(ldafb, *)</i> には、 ?pbtrf によって返された、因子分解後の帯行列 <i>A</i> を格納する。 <i>b(l db, *)</i> には、右辺の行列 <i>B</i> が格納される。 <i>x(ldx, *)</i> には、解の行列 <i>X</i> が格納される。 <i>work(*)</i> は、ワークスペース配列である。 <i>ab</i> と <i>afb</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $ldab \geq kd + 1$ 。
<i>ldafb</i>	INTEGER。 <i>fb</i> の第 1 次元。 $ldafb \geq kd + 1$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>cpbrfs</i> の場合) DOUBLE PRECISION (<i>zpbrfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afb</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>AF</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $8n*kd$ 以上、複素数型の場合は $32n*kd$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、実数型の場合は $12n*kd$ で、複素数型の場合は $48n*kd$ である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $4n*kd$ で、複素数型の場合は約 $16n*kd$ である。

?ptrfs

対称(エルミート)正定値三重対角行列を係数行列とする連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call sptrfs(n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work, info)
call dptrfs(n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work, info)
call cptrfs(uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
           rwork, info)
call zptrfs(uplo, n, nrhs, d, e, df, ef, b, ldb, x, ldx, ferr, berr, work,
           rwork, info)
```

Fortran 95:

```
call ptrfs(d, df, e, ef, b, x [,ferr] [,berr] [,info])
call ptrfs(d, df, e, ef, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、対称 (エルミート) 正定値三重対角行列 A を係数行列とし複数の右辺を持つ連立線形方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?pttrf](#) を呼び出す。
- 解の算出ルーチン [?pttrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。複素型でのみ使用される。 'U' または 'L' でなければならない。 三重対角行列 A の優対角成分と劣対角成分のどちらを格納するかと、 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 e には A の優対角成分が格納され、 A は $U^H D U$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 e には A の劣対角成分が格納され、 A は LDL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>d,df,rwork</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: $d(n)$, $df(n)$, $rwork(n)$ 。 配列 d には、三重対角行列 A の n 個の対角成分が格納される。 配列 df には、 ?pttrf による A の因子分解で得られた対角行列 D の n 個の対角成分が格納される。 配列 $rwork$ は、複素数型でのみ使用されるワークスペース配列である。
<code>e,ef,b,x,work</code>	REAL (<code>spttrfs</code> の場合) DOUBLE PRECISION (<code>dpttrfs</code> の場合) COMPLEX (<code>cpttrfs</code> の場合) DOUBLE COMPLEX (<code>zpttrfs</code> の場合) 配列: $e(n-1)$, $ef(n-1)$, $b(ldb,nrhs)$, $x(ldx,nrhs)$, $work(*)$ 。 配列 e には、三重対角行列 A の $(n-1)$ 個の非対角成分が格納される (<code>uplo</code> を参照)。 配列 ef には、 ?pttrf による因子分解で得られた単位二重対角係数 U または L の $(n-1)$ 個の非対角成分が格納される (<code>uplo</code> を参照)。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

配列 x には、[?pttrs](#) によって計算された解の行列 X が格納される。

配列 $work$ はワークスペース配列である。 $work$ の次元は、 $2*n$ (実数型の場合) または n (複素数型の場合) 以上でなければならない。

ldb INTEGER。 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
 ldx INTEGER。 x のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。

出力パラメーター

x 精度が改善された解の行列 X 。
 $ferr, berr$ REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
 $info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pttrfs` ルーチンのインターフェイス特有の詳細を以下に示す。

d 長さ (n) のベクトルを格納する。
 df 長さ (n) のベクトルを格納する。
 e 長さ ($n-1$) のベクトルを格納する。
 ef 長さ ($n-1$) のベクトルを格納する。
 b サイズ ($n, nrhs$) の行列 B を格納する。
 x サイズ ($n, nrhs$) の行列 X を格納する。
 $ferr$ 長さ ($nrhs$) のベクトルを格納する。
 $berr$ 長さ ($nrhs$) のベクトルを格納する。
 $uplo$ 複素型でのみ使用される。'U' または 'L' でなければならない。デフォルト値は 'U'。

?syrfs

対称行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call ssyrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call dsyrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, iwork, info)
call csyrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
call zsyrfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
```

Fortran 95:

```
call syrfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、フル格納形式の対称行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?sytrf](#) を呼び出す。
- 解の算出ルーチン [?sytrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。

<i>a, af, b, x, work</i>	REAL (ssyrfs の場合) DOUBLE PRECISION (dsyrfs の場合) COMPLEX (csyrfs の場合) DOUBLE COMPLEX (zsyrfs の場合) 配列: <i>a(lda, *)</i> には、 ?sytrf に入力として与えた元の行列 <i>A</i> を格納する。 <i>af(ldaf, *)</i> には、 ?sytrf によって返された、因子分解後の行列 <i>A</i> を格納する。 <i>b(ldb, *)</i> には、右辺の行列 <i>B</i> が格納される。 <i>x(ldx, *)</i> には、解の行列 <i>X</i> が格納される。 <i>work(*)</i> は、ワークスペース配列である。 <i>a</i> と <i>af</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?sytrf によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (csyrfs の場合) DOUBLE PRECISION (zsyrfs の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`syrfs` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>af</code>	サイズ (n, n) の行列 AF を格納する。
<code>ipiv</code>	長さ (n) のベクトルを格納する。
<code>b</code>	サイズ $(n, nrhs)$ の行列 B を格納する。
<code>x</code>	サイズ $(n, nrhs)$ の行列 X を格納する。
<code>ferr</code>	長さ $(nrhs)$ のベクトルを格納する。
<code>berr</code>	長さ $(nrhs)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

`ferr` に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n^2$ 以上、複素数型の場合は $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?herfs

複素エルミート行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call cherfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
call zherfs(uplo, n, nrhs, a, lda, af, ldaf, ipiv, b, ldb, x, ldx, ferr, berr,
            work, rwork, info)
```

Fortran 95:

```
call herfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、フル格納形式の複素エルミート行列 A による、複数の右辺を持つ連立線形方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?hetrf](#) を呼び出す。
- 解の算出ルーチン [?hetrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>af</code> には、Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>a, af, b, x, work</code>	COMPLEX (<code>cherfs</code> の場合) DOUBLE COMPLEX (<code>zherfs</code> の場合) 配列： <code>a(lda, *)</code> には、 ?hetrf に入力として与えた元の行列 A を格納する。 <code>af(ldaf, *)</code> には、 ?hetrf によって返された、因子分解後の行列 A を格納する。 <code>b(l db, *)</code> には、右辺の行列 B が格納される。 <code>x(ldx, *)</code> には、解の行列 X が格納される。 <code>work(*)</code> は、ワークスペース配列である。 a と af の第 2 次元は $\max(1, n)$ 以上でなければならない。 b と x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work</code> の次元は $\max(1, 2*n)$ 以上でなければならない。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ldaf</code>	INTEGER。 af の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>ldx</code>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?hetrf によって返される <i>ipiv</i> 配列。
<i>rwork</i>	REAL (<i>cherfs</i> の場合) DOUBLE PRECISION (<i>zherfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr</i> , <i>berr</i>	REAL (<i>cherfs</i> の場合) DOUBLE PRECISION (<i>zherfs</i> の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

herfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>af</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、 $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階で必要な演算の回数は、 $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

このルーチンに対応する実数型は、[ssyrfs](#) / [dsyrfs](#) である。

?sprfs

圧縮格納形式の対称行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call ssprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dsprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call csprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call zsprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

Fortran 95:

```
call sprfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、圧縮格納形式の対称行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, \quad |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?spturf](#) を呼び出す。
- 解の算出ルーチン [?sptrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>afp</code> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>afp</code> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。

<i>ap, afp, b, x, work</i>	REAL (<i>ssprfs</i> の場合) DOUBLE PRECISION (<i>dsprfs</i> の場合) COMPLEX (<i>csprfs</i> の場合) DOUBLE COMPLEX (<i>zsprfs</i> の場合) 配列: <i>ap</i> (*) には、 ?sprtf に入力として与えた元の圧縮形式の行列 <i>A</i> を格納する。 <i>afp</i> (*) には、 ?sprtf によって返された、因子分解後の圧縮形式の行列 <i>A</i> を格納する。 <i>b</i> (<i>ldb</i> , *) には、右辺の行列 <i>B</i> が格納される。 <i>x</i> (<i>ldx</i> , *) には、解の行列 <i>X</i> が格納される。 <i>work</i> (*) は、ワークスペース配列である。 配列 <i>ap</i> と <i>afp</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> と <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?sprtf によって返される <i>ipiv</i> 配列。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>csprfs</i> の場合) DOUBLE PRECISION (<i>zsprfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>x</i>	精度が改善された解の行列 <i>X</i> 。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sprfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、実数型の場合は $4n^2$ 以上、複素数型の場合は $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、実数型の場合は $6n^2$ で、複素数型の場合は $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n^2$ で、複素数型の場合は約 $8n^2$ である。

?hprfs

圧縮格納形式の複素エルミート行列による連立線形方程式の解の精度を改善し、解の誤差を推定する。

構文

Fortran 77:

```
call chprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call zhprfs(uplo, n, nrhs, ap, afp, ipiv, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

Fortran 95:

```
call hprfs(a, af, ipiv, b, x [,uplo] [,ferr] [,berr] [,info])
```

説明

このルーチンは、圧縮格納形式の複素エルミート行列 A による、複数の右辺を持つ連立線形方程式 $AX = B$ の解の精度の改善を繰り返し実行する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

最後に、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、以下の手順を実行する必要がある。

- 因子分解ルーチン [?hptrf](#) を呼び出す。
- 解の算出ルーチン [?hptrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>afp</code> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 <code>uplo = 'L'</code> の場合には、配列 <code>afp</code> は、圧縮形式の Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ap,afp,b,x,work</code>	COMPLEX (<code>chprfs</code> の場合) DOUBLE COMPLEX (<code>zhprfs</code> の場合) 配列： <code>ap(*)</code> には、 ?hptrf に入力として与えた元の圧縮形式の行列 A を格納する。 <code>afp(*)</code> には、 ?hptrf によって返された、因子分解後の圧縮形式の行列 A を格納する。 <code>b(ldb, *)</code> には、右辺の行列 B が格納される。 <code>x(ldx, *)</code> には、解の行列 X が格納される。 <code>work(*)</code> は、ワークスペース配列である。 配列 <code>ap</code> と <code>afp</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <code>b</code> と <code>x</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work</code> の次元は $\max(1, 2*n)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>ldx</code>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?hptrf によって返される <code>ipiv</code> 配列。

rwork REAL (*chprfs* の場合)
 DOUBLE PRECISION (*zhprfs* の場合)
 ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

x 精度が改善された解の行列 X 。

ferr, *berr* REAL (*chprfs* の場合)。
 DOUBLE PRECISION (*zhprfs* の場合)。
 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hprfs ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。

af Fortran 77 インターフェイスでの引数 *afp* を意味する。サイズ $(n*(n+1)/2)$ の配列 AF を格納する。

ipiv 長さ (*n*) のベクトルを格納する。

b サイズ (*n*, *nrhs*) の行列 B を格納する。

x サイズ (*n*, *nrhs*) の行列 X を格納する。

ferr 長さ (*nrhs*) のベクトルを格納する。

berr 長さ (*nrhs*) のベクトルを格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

各右辺について、後退誤差の計算に必要な浮動小数点演算の回数は、 $16n^2$ 以上である。さらに、繰り返し行われる解の精度の改善の各段階に必要な演算の回数は、 $24n^2$ である。繰り返される回数の範囲は 1 ~ 5 になる。

前進誤差を推定するには、連立線形方程式 $Ax = b$ を繰り返し解く必要がある。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、約 $8n^2$ である。

このルーチンに対応する実数型は、[ssprfs](#) / [dsprfs](#) である。

?trrfs

三角行列による連立線形方程式の解の誤差を推定する。

構文

Fortran 77:

```
call strrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
           work, iwork, info)
call dtrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
           work, iwork, info)
call ctrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
           work, rwork, info)
call ztrrfs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, x, ldx, ferr, berr,
           work, rwork, info)
```

Fortran 95:

```
call trrfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

説明

このルーチンは、三角行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ または $A^T X=B$ または $A^H X=B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \quad \text{ただし } (A + \delta A)x = (b + \delta b)$$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチンを呼び出す前に、解の算出ルーチン [?trtrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <code>uplo = 'U'</code> の場合、A は上三角行列である。 <code>uplo = 'L'</code> の場合、A は下三角行列である。
<code>trans</code>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <code>trans = 'N'</code> の場合、連立方程式の形式は $AX=B$ である。 <code>trans = 'T'</code> の場合、連立方程式の形式は $A^T X=B$ である。 <code>trans = 'C'</code> の場合、連立方程式の形式は $A^H X=B$ である。
<code>diag</code>	CHARACTER*1。'N' または 'U' でなければならない。

diag = 'N' の場合、*A* は単位三角行列ではない。

diag = 'U' の場合、*A* は単位三角行列である。*A* の対角成分は 1 とみなされ、配列 *a* 内で参照されない。

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>a, b, x, work</i>	REAL (<i>strrfs</i> の場合) DOUBLE PRECISION (<i>dtrrfs</i> の場合) COMPLEX (<i>ctrufs</i> の場合) DOUBLE COMPLEX (<i>ztrufs</i> の場合)

配列：

a(*lda, **) には、*uplo* によって指定される上三角または下三角行列 *A* を格納する。

b(*ldb, **) には、右辺の行列 *B* が格納される。

x(*ldx, **) には、解の行列 *X* が格納される。

work(***) は、ワークスペース配列である。

a の第 2 次元は $\max(1, n)$ 以上でなければならない。*b* と *x* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。*work* の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>ctrufs</i> の場合) DOUBLE PRECISION (<i>ztrufs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`trrfs` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	サイズ (n,n) の行列 A を格納する。
<code>b</code>	サイズ $(n,nrhs)$ の行列 B を格納する。
<code>x</code>	サイズ $(n,nrhs)$ の行列 X を格納する。
<code>ferr</code>	長さ $(nrhs)$ のベクトルを格納する。
<code>berr</code>	長さ $(nrhs)$ のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>trans</code>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<code>diag</code>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

`ferr` に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチン呼び出すと、各右辺について、連立線形方程式 $Ax=b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tprfs

圧縮格納形式の三角行列による連立線形方程式の解の誤差を推定する。

構文

Fortran 77:

```
call stprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call dtprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            iwork, info)
call ctprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
call ztprfs(uplo, trans, diag, n, nrhs, ap, b, ldb, x, ldx, ferr, berr, work,
            rwork, info)
```

Fortran 95:

```
call tprfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

説明

このルーチンは、圧縮格納形式の三角行列 A による、複数の右辺を持つ連立線形方程式 $AX=B$ または $A^T X=B$ または $A^H X=B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。

$$|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|, |\delta b_i|/|b_i| \leq \beta |b_i| \text{ ただし } (A + \delta A)x = (b + \delta b)$$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、解の算出ルーチン [?tpttrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <code>uplo = 'U'</code> の場合、 A は上三角行列である。 <code>uplo = 'L'</code> の場合、 A は下三角行列である。
<code>trans</code>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <code>trans = 'N'</code> の場合、連立方程式の形式は $AX=B$ である。 <code>trans = 'T'</code> の場合、連立方程式の形式は $A^T X=B$ である。 <code>trans = 'C'</code> の場合、連立方程式の形式は $A^H X=B$ である。
<code>diag</code>	CHARACTER*1。'N' または 'U' でなければならない。 <code>diag = 'N'</code> の場合、 A は単位三角行列ではない。 <code>diag = 'U'</code> の場合、 A は単位三角行列である。 A の対角成分は 1 とみなされ、配列 <code>ap</code> 内で参照されない。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<code>ap, b, x, work</code>	REAL (<code>strrfs</code> の場合) DOUBLE PRECISION (<code>dtrrfs</code> の場合) COMPLEX (<code>ctrufs</code> の場合) DOUBLE COMPLEX (<code>ztrrfs</code> の場合) 配列： <code>ap(*)</code> には、 <code>uplo</code> によって指定される上三角または下三角行列 A を格納する。 <code>b(ldb, *)</code> には、右辺の行列 B が格納される。 <code>x(ldx, *)</code> には、解の行列 X が格納される。 <code>work(*)</code> は、ワークスペース配列である。

ap の次元は、 $\max(1, n(n+1)/2)$ 以上でなければならない。
 b と x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work$ の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (ctrrfs の場合) DOUBLE PRECISION (ztrrfs の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tprfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 B を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 X を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチン呼び出すと、各右辺について、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 n^2 で、複素数型の場合は約 $4n^2$ である。

?tbrfs

三角帯行列による連立線形方程式の解の誤差を推定する。

構文

Fortran 77:

```
call stbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call dtbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
           berr, work, iwork, info)
call ctbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
call ztbrfs(uplo, trans, diag, n, kd, nrhs, ab, ldab, b, ldb, x, ldx, ferr,
           berr, work, rwork, info)
```

Fortran 95:

```
call tbrfs(a, b, x [,uplo] [,trans] [,diag] [,ferr] [,berr] [,info])
```

説明

このルーチンは、三角帯行列 A による、複数の右辺を持つ連立線形方程式 $AX = B$ または $A^T X = B$ または $A^H X = B$ の解の誤差を推定する。このルーチンは、算出されたそれぞれの解ベクトル x について、成分ごとに後退誤差 β を計算する。次のように、 x が摂動連立方程式の正確な解である場合、この誤差は、 A と b の成分の最小相対摂動になる。
 $|\delta a_{ij}|/|a_{ij}| \leq \beta |a_{ij}|$, $|\delta b_i|/|b_i| \leq \beta |b_i|$ ただし $(A + \delta A)x = (b + \delta b)$

また、このルーチンは、算出された解 $\|x - x_e\|_\infty / \|x\|_\infty$ の成分ごとに前進誤差を推定する (x_e は正確な解である)。

このルーチン呼び出す前に、解の算出ルーチン [?tbtrs](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <code>uplo = 'U'</code> の場合、A は上三角行列である。 <code>uplo = 'L'</code> の場合、A は下三角行列である。
<code>trans</code>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。

	$trans = 'N'$ の場合、連立方程式の形式は $AX = B$ である。
	$trans = 'T'$ の場合、連立方程式の形式は $A^T X = B$ である。
	$trans = 'C'$ の場合、連立方程式の形式は $A^H X = B$ である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 $diag = 'N'$ の場合、 A は単位三角行列ではない。 $diag = 'U'$ の場合、 A は単位三角行列である。 A の対角成分は 1 とみなされ、配列 <i>ab</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>ab, b, x, work</i>	REAL (<i>stbrfs</i> の場合) DOUBLE PRECISION (<i>dtbrfs</i> の場合) COMPLEX (<i>ctbrfs</i> の場合) DOUBLE COMPLEX (<i>ztbrfs</i> の場合) 配列: $ab(ldab, *)$ には、 <i>uplo</i> によって指定される上三角または下三角行列 A を帯格納形式で格納する。 $b(l db, *)$ には、右辺の行列 B が格納される。 $x(ldx, *)$ には、解の行列 X が格納される。 $work(*)$ は、ワークスペース配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b と x の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。 $work$ の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 ($ldab \geq kd + 1$)。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ldx</i>	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>rwork</i>	REAL (<i>ctbrfs</i> の場合) DOUBLE PRECISION (<i>ztbrfs</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
-------------------	---

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tbrfs ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

ferr に返される誤差範囲は厳密ではないが、実際の誤差はほぼ常にこれより小さくなる。

このルーチンを呼び出すと、各右辺について、連立線形方程式 $Ax = b$ の解が複数回計算される。その回数は、通常は 4 または 5 で、11 を超えることはない。それぞれの解の計算に必要な浮動小数点演算の回数は、実数型の場合は約 $2n \cdot kd$ で、複素数型の場合は約 $8n \cdot kd$ である。

行列の反転用のルーチン

行列の逆行列を明示的に計算することは、実際にはほとんどない。
特に、連立方程式 $Ax = b$ を解く場合に、最初に A^{-1} を計算してから、行列とベクトルの積 $x = A^{-1}b$ を計算してはならない。
代わりに、ソルバールーチン呼び出す ([「連立線形方程式を解くためのルーチン」](#) を参照)。この方が効率が良く、高い精度の結果が得られる。

ただし、逆行列を求める必要が生じた場合に備えて、行列反転用のルーチンが用意されている。

?getri

LU 因子分解された一般行列の逆行列を計算する。

構文

Fortran 77:

```
call sgetri(n, a, lda, ipiv, work, lwork, info)
call dgetri(n, a, lda, ipiv, work, lwork, info)
call cgetri(n, a, lda, ipiv, work, lwork, info)
call zgetri(n, a, lda, ipiv, work, lwork, info)
```

Fortran 95:

```
CALL GETRI(a, ipiv [,info])
```

説明

このルーチンは、一般行列 A の逆行列 (A^{-1}) を計算する。
このルーチン呼び出す前に、[?getrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	REAL (sgetri の場合) DOUBLE PRECISION (dgetri の場合) COMPLEX (cgetri の場合) DOUBLE COMPLEX (zgetri の場合) 配列: $a(lda, *)$, $work(lwork)$ 。 $a(lda, *)$ には、 ?getrf によって返される、行列 A の因子分解 $A = PLU$ を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?getrf によって返される <i>ipiv</i> 配列。

lwork INTEGER。配列 *work* のサイズ ($lwork \geq n$)。
lwork = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエンタリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a $n \times n$ の行列 A^{-1} によって上書きされる。
work(1) *info* = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。
info INTEGER。 *info* = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、係数 U の *i* 番目の対角成分がゼロで、 U が特異になるため、行列の反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

getri ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。
ipiv 長さ (n) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスに必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

算出された逆行列 X は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\epsilon |X|P|L||U|$$

$c(n)$ は n の適度な 1 次関数で、 ϵ はマシンの精度である。
 I は元の行列を示す。 P 、 L 、および U は、行列の因子分解 $A = PLU$ の係数である。

浮動小数点演算の合計回数は、実数型の場合は約 $(4/3)n^3$ で、複素数型の場合は約 $(16/3)n^3$ になる。

?potri

対称 (エルミート) 正定値行列の逆行列を計算する。

構文

Fortran 77:

```
call spotri(uplo, n, a, lda, info)
call dpotri(uplo, n, a, lda, info)
call cpotri(uplo, n, a, lda, info)
call zpotri(uplo, n, a, lda, info)
```

Fortran 95:

```
CALL POTRI(a [,uplo] [,info])
```

説明

このルーチンは、対称 (複素行列の場合はエルミート) 正定値行列 A の逆行列 (A^{-1}) を計算する。

このルーチンを呼び出す前に、[?potrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<code>uplo</code>	CHARACTER*1. 'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 a には、コレスキー因子分解 $A = U^H U$ の係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 a には、コレスキー因子分解 $A = LL^H$ の係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>a</code>	REAL (spotri の場合) DOUBLE PRECISION (dpotri の場合) COMPLEX (cpotri の場合) DOUBLE COMPLEX (zpotri の場合) 配列: $a(lda, *)$ 。 ?potrf によって返される、行列 A の因子分解を格納する。 <code>a</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

<code>a</code>	$n \times n$ の行列 A^{-1} によって上書きされる。
----------------	---------------------------------------

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、コレスキー係数の *i* 番目の対角成分 (したがって、係数そのもの) がゼロのため、反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

potri ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (*n*,*n*) の行列 *A* を格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \leq c(n)\varepsilon\kappa_2(A), \quad \|AX - I\|_2 \leq c(n)\varepsilon\kappa_2(A)$$

c(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。
I は元の行列を示す。

行列 *A* の 2- ノルム $\|A\|_2$ は、 $\|A\|_2 = \max_x \cdot_{x=1} (Ax \cdot Ax)^{1/2}$ によって定義される。条件数 $\kappa_2(A)$ は、 $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?pptri

圧縮格納形式の対称 (エルミート) 正定値行列の逆行列を計算する。

構文

Fortran 77:

```
call spptri(uplo, n, ap, info)
call dpptri(uplo, n, ap, info)
call cpptri(uplo, n, ap, info)
call zpptri(uplo, n, ap, info)
```

Fortran 95:

```
CALL PPTRI(a [,uplo] [,info])
```

説明

このルーチンは、*圧縮形式の対称* (複素行列の場合はエルミート) 正定値行列 A の逆行列 (A^{-1}) を計算する。このルーチン呼び出す前に、[?pptrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には、コレスキー因子分解 $A = U^H U$ の圧縮形式の係数 U を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には、コレスキー因子分解 $A = LL^H$ の圧縮形式の係数 L を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>ap</code>	REAL (<code>spptri</code> の場合) DOUBLE PRECISION (<code>dpptri</code> の場合) COMPLEX (<code>cpptri</code> の場合) DOUBLE COMPLEX (<code>zpptri</code> の場合) 配列、次元は $\max(1, n(n+1)/2)$ 以上。 ?pptrf によって返される、圧縮形式の行列 A の因子分解を格納する。 <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。

出力パラメーター

<code>ap</code>	圧縮形式の $n \times n$ の行列 A^{-1} によって上書きされる。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。 <code>info = i</code> の場合、コレスキー係数の i 番目の対角成分 (したがって、係数そのもの) がゼロのため、反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pptri` ルーチンのインターフェイス特有の詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$\|XA - I\|_2 \leq c(n)\epsilon\kappa_2(A), \quad \|AX - I\|_2 \leq c(n)\epsilon\kappa_2(A)$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。
 I は元の行列を示す。

行列 A の 2- ノルム $\|A\|_2$ は、 $\|A\|_2 = \max_{\mathbf{x} \cdot \mathbf{x}=1} (\mathbf{Ax} \cdot \mathbf{Ax})^{1/2}$ によって定義される。条件数 $\kappa_2(A)$ は、 $\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2$ によって定義される。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?sytri

対称行列の逆行列を計算する。

構文

Fortran 77:

```
call ssytri(uplo, n, a, lda, ipiv, work, info)
call dsytri(uplo, n, a, lda, ipiv, work, info)
call csytri(uplo, n, a, lda, ipiv, work, info)
call zsytri(uplo, n, a, lda, ipiv, work, info)
```

Fortran 95:

```
CALL SYTRI(a, ipiv [,uplo] [,info])
```

説明

このルーチンは、対称行列 A の逆行列 (A^{-1}) を計算する。
 このルーチン呼び出す前に、[?sytrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 a には、Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 $uplo = 'L'$ の場合、配列 a には、Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>a, work</code>	REAL (ssytri の場合) DOUBLE PRECISION (dsytri の場合) COMPLEX (csytri の場合) DOUBLE COMPLEX (zsytri の場合) 配列: $a(lda, *)$ には、 ?sytrf によって返される、行列 A の因子分解を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列。
 $work$ の次元は $\max(1, 2*n)$ 以上でなければならない。

lda INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
 $ipiv$ INTEGER。
 配列、次元は $\max(1, n)$ 以上。
[?sytrf](#) によって返される $ipiv$ 配列。

出力パラメーター

a $n \times n$ の行列 A^{-1} によって上書きされる。
 $info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。
 $info = i$ の場合、 D の i 番目の対角成分がゼロで、 D が特異になるため、行列の反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sytri ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。
 $ipiv$ 長さ (n) のベクトルを格納する。
 $uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$|DU^T P^T X P U - I| \leq c(n) \varepsilon (|D| |U^T| P^T |X| P |U| + |D| |D^{-1}|)$$

$uplo = 'U'$ の場合

$$|DL^T P^T X P L - I| \leq c(n) \varepsilon (|D| |L^T| P^T |X| P |L| + |D| |D^{-1}|)$$

$uplo = 'L'$ の場合 $c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。 I は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?hetri

複素エルミート行列の逆行列を計算する。

構文

Fortran 77:

```
call chetri(uplo, n, a, lda, ipiv, work, info)
call zhetri(uplo, n, a, lda, ipiv, work, info)
```

Fortran 95:

```
CALL HETRI(a, ipiv [,uplo] [,info])
```

説明

このルーチンは、複素エルミート行列 A の逆行列 (A^{-1}) を計算する。
このルーチン呼び出す前に、[?hetrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <code>uplo = 'U'</code> の場合、配列 a には、Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 <code>uplo = 'L'</code> の場合、配列 a には、Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>a, work</code>	COMPLEX (chetri の場合) DOUBLE COMPLEX (zhetri の場合) 配列： <code>a(lda, *)</code> には、 ?hetrf によって返される、行列 A の因子分解を格納する。 <code>a</code> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <code>work(*)</code> は、ワークスペース配列。 <code>work</code> の次元は $\max(1, n)$ でなければならない。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?hetrf によって返される <code>ipiv</code> 配列。

出力パラメーター

`a` $n \times n$ の行列 A^{-1} によって上書きされる。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、行列の反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hetri ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (*n*, *n*) の行列 *A* を格納する。
ipiv 長さ (*n*) のベクトルを格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|DU^H P^T X P U - I| \leq c(n) \varepsilon (|D| |U^H| P^T |X| P |U| + |D| |D^{-1}|)$$

uplo = 'U' の場合

$$|DL^H P^T X P L - I| \leq c(n) \varepsilon (|D| |L^H| P^T |X| P |L| + |D| |D^{-1}|)$$

uplo = 'L' の場合 *c*(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。*I* は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

このルーチンの実数版は、[?sytri](#) である。

?sptri

圧縮格納形式の対称行列の逆行列を計算する。

構文

Fortran 77:

```
call ssptri(uplo, n, ap, ipiv, work, info)
call dsptri(uplo, n, ap, ipiv, work, info)
call csptri(uplo, n, ap, ipiv, work, info)
call zsptri(uplo, n, ap, ipiv, work, info)
```

Fortran 95:

```
CALL SPTRI(a, ipiv [,uplo] [,info])
```

説明

このルーチンは、圧縮格納形式の対称行列 A の逆行列 (A^{-1}) を計算する。
このルーチンを呼び出す前に、[?sptf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> には、Bunch-Kaufman 因子分解 $A = PUDU^T P^T$ を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> には、Bunch-Kaufman 因子分解 $A = PLDL^T P^T$ を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	REAL (ssptri の場合) DOUBLE PRECISION (dsptri の場合) COMPLEX (csptri の場合) DOUBLE COMPLEX (zsptri の場合) 配列： <i>ap</i> (*) には、 ?sptf によって返される、行列 A の因子分解を格納する。 <i>ap</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は $\max(1, n)$ でなければならない。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?sptf によって返される <i>ipiv</i> 配列。

出力パラメーター

<i>ap</i>	<i>ap</i> 圧縮形式の $n \times n$ の行列 A^{-1} によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 D の <i>i</i> 番目の対角成分がゼロで、 D が特異になるため、行列の反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sptri ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$|DU^T P^T X P U - I| \leq c(n) \varepsilon (|D| |U^T| P^T |X| P |U| + |D| |D^{-1}|)$$

$uplo = 'U'$ の場合

$$|DL^T P^T X P L - I| \leq c(n) \varepsilon (|D| |L^T| P^T |X| P |L| + |D| |D^{-1}|)$$

$uplo = 'L'$ の場合 $c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。 I は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

?hptri

圧縮格納形式の複素エルミート行列の逆行列を計算する。

構文

Fortran 77:

```
call chptri(uplo, n, ap, ipiv, work, info)
call zhptri(uplo, n, ap, ipiv, work, info)
```

Fortran 95:

```
CALL HPTRI(a, ipiv [,uplo] [,info])
```

説明

このルーチンは、圧縮格納形式の複素エルミート行列 A の逆行列 (A^{-1}) を計算する。このルーチンを呼び出す前に、[?hptrf](#) を呼び出して、 A を因子分解する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 入力行列 A の因子分解の方法を指定する。 $uplo = 'U'$ の場合、配列 <code>ap</code> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PUDU^H P^T$ を格納する。 $uplo = 'L'$ の場合、配列 <code>ap</code> には、圧縮形式の Bunch-Kaufman 因子分解 $A = PLDL^H P^T$ を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。

ap, work COMPLEX (chptri の場合)
 DOUBLE COMPLEX (zhptri の場合)
 配列：
ap(*) には、[?hptrf](#) によって返される、行列 *A* の因子分解を格納する。
ap の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。
work(*) は、ワークスペース配列。
work の次元は $\max(1, n)$ でなければならない。

ipiv INTEGER。
 配列、次元は $\max(1, n)$ 以上。
[?hptrf](#) によって返される *ipiv* 配列。

出力パラメーター

ap $n \times n$ の行列 A^{-1} によって上書きされる。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、*D* の *i* 番目の対角成分がゼロで、*D* が特異になるため、行列の反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hptri ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ap* を意味する。サイズ $(n*(n+1)/2)$ の配列 *A* を格納する。

ipiv 長さ (*n*) のベクトルを格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|DU^H P^T X P U - I| \leq c(n) \varepsilon (|D| |U^H| |P^T| |X| |P| |U| + |D| |D^{-1}|)$$

uplo = 'U' の場合

$$|DL^H P^T X P L - I| \leq c(n) \varepsilon (|D| |L^H| |P^T| |X| |P| |L| + |D| |D^{-1}|)$$

uplo = 'L' の場合 $c(n)$ は *n* の適度な 1 次関数で、 ε はマシンの精度である。*I* は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(2/3)n^3$ で、複素数型の場合は約 $(8/3)n^3$ になる。

このルーチンの実数版は、[?sptri](#) である。

?trtri

三角行列の逆行列を計算する。

構文

Fortran 77:

```
call strtri(uplo, diag, n, a, lda, info)
call dtrtri(uplo, diag, n, a, lda, info)
call ctrtri(uplo, diag, n, a, lda, info)
call ztrtri(uplo, diag, n, a, lda, info)
```

Fortran 95:

```
CALL TRTRI(a [,uplo] [,diag] [,info])
```

説明

このルーチンは、三角行列 A の逆行列 (A^{-1}) を計算する。

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、 A は上三角行列である。 <i>uplo</i> = 'L' の場合、 A は下三角行列である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、 A は単位三角行列である。 A の対角成分は 1 とみなされ、配列 a 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i>	REAL (<i>strtri</i> の場合) DOUBLE PRECISION (<i>dtrtri</i> の場合) COMPLEX (<i>ctrtri</i> の場合) DOUBLE COMPLEX (<i>ztrtri</i> の場合) 配列、次元は (<i>lda</i> , *)。 行列 A を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

a $n \times n$ の行列 A^{-1} によって上書きされる。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、*A* の *i* 番目の対角成分がゼロで、*A* が特異になるため、反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

trtri ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (*n*,*n*) の行列 *A* を格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。
diag 'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

算出された逆行列 *X* は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon |X||A|$$

$$|X - A^{-1}| \leq c(n)\varepsilon |A^{-1}||A||X|$$

c(*n*) は *n* の適度な 1 次関数で、 ε はマシンの精度である。
I は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ で、複素数型の場合は約 $(4/3)n^3$ になる。

?tptri

圧縮格納形式の三角行列の逆行列を計算する。

構文

Fortran 77:

```
call stptri(uplo, diag, n, ap, info)
call dtptri(uplo, diag, n, ap, info)
call ctptri(uplo, diag, n, ap, info)
call ztptri(uplo, diag, n, ap, info)
```

Fortran 95:

```
CALL TPTRI(a [,uplo] [,diag] [,info])
```

説明

このルーチンは、圧縮格納形式の三角行列 A の逆行列 (A^{-1}) を計算する。

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' の場合、A は上三角行列である。 <i>uplo</i> = 'L' の場合、A は下三角行列である。
<i>diag</i>	CHARACTER*1。'N' または 'U' でなければならない。 <i>diag</i> = 'N' の場合、A は単位三角行列ではない。 <i>diag</i> = 'U' の場合、A は単位三角行列である。A の対角成分は 1 とみなされ、配列 <i>ap</i> 内で参照されない。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap</i>	REAL (<i>stptri</i> の場合) DOUBLE PRECISION (<i>dtptri</i> の場合) COMPLEX (<i>ctptri</i> の場合) DOUBLE COMPLEX (<i>ztptri</i> の場合) 配列、次元は $\max(1, n(n+1)/2)$ 以上。 圧縮形式の三角行列 A を格納する。

出力パラメーター

<i>ap</i>	圧縮形式の $n \times n$ の行列 A^{-1} によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、A の <i>i</i> 番目の対角成分がゼロで、A が特異になるため、反転を完了できない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

tptri ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>diag</i>	'N' または 'U' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

算出された逆行列 X は、以下の誤差範囲を満たす。

$$|XA - I| \leq c(n)\varepsilon |X||A|$$

$$|X - A^{-1}| \leq c(n)\varepsilon |A^{-1}||A||X|$$

$c(n)$ は n の適度な 1 次関数で、 ε はマシンの精度である。

I は元の行列を示す。

浮動小数点演算の合計回数は、実数型の場合は約 $(1/3)n^3$ で、複素数型の場合は約 $(4/3)n^3$ になる。

行列の平衡化

この節では、行列の平衡化に必要なスケール係数の計算に使用されるルーチンについて説明する。ただし、これらのルーチンが行列を実際にスケールリングするわけではない。

?gseequ

行列を平衡化して、条件数を小さくするための行と列のスケール係数を計算する。

構文

Fortran 77:

```
call sseequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call dseequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call cseequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
call zseequ(m, n, a, lda, r, c, rowcnd, colcnd, amax, info)
```

Fortran 95:

```
CALL GEEQU(a, r, c [,rowcnd] [,colcnd] [,amax] [,info])
```

説明

このルーチンは、 $m \times n$ 行列 A を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。出力配列 r に行のスケール係数を返し、配列 c に列のスケール係数を返す。これらの係数は、成分 $b_{ij}=r(i)*a_{ij}*c(j)$ を持つ行列 B の各行と各列の最大の成分の絶対値が 1 になるように選択される。

?gseequ により計算されるスケール係数を使用する [?lagge](#) 補助関数を参照のこと。

入力パラメーター

m	INTEGER。行列 A の行数。 $m \geq 0$ 。
n	INTEGER。行列 A の列数 ($n \geq 0$)。
a	REAL (sseequ の場合) DOUBLE PRECISION (dseequ の場合) COMPLEX (cseequ の場合) DOUBLE COMPLEX (zseequ の場合) 配列、次元は ($lda, *$)。 平衡化係数を計算する $m \times n$ 行列 A を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。

出力パラメーター

<i>r, c</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: <i>r</i> (<i>m</i>), <i>c</i> (<i>n</i>). <i>info</i> = 0 または <i>info</i> > <i>m</i> の場合、配列 <i>r</i> に行列 <i>A</i> の行のスケール係数が格納される。 <i>info</i> = 0 の場合、配列 <i>c</i> に行列 <i>A</i> の列のスケール係数が格納される。
<i>rowcnd</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 または <i>info</i> > <i>m</i> の場合、 <i>rowcnd</i> に、最小の <i>r</i> (<i>i</i>) を最大の <i>r</i> (<i>i</i>) で割った値が格納される。
<i>colcnd</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 の場合、 <i>colcnd</i> に、最小の <i>c</i> (<i>i</i>) を最大の <i>c</i> (<i>i</i>) で割った値が格納される。
<i>amax</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> 、かつ <i>i</i> ≤ <i>m</i> の場合、 <i>A</i> の <i>i</i> 番目の行が完全に 0 である。 <i>i</i> > <i>m</i> の場合、 <i>A</i> の (<i>i</i> - <i>m</i>) 番目の列が完全に 0 である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

geequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>m, n</i>) の行列 <i>A</i> を格納する。
<i>r</i>	長さ (<i>m</i>) のベクトルを格納する。
<i>c</i>	長さ (<i>n</i>) のベクトルを格納する。

アプリケーション・ノート

r と *c* のすべての成分は、SMLNUM = 最小の安全な数値と BIGNUM = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、*A* の条件数が小さくなることは保証されないが、実際には有効に機能する。

rowcnd ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*r* によるスケールリングは不要である。*colcnd* ≥ 0.1 の場合、*c* によるスケールリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

?gbequ

帯行列を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。

構文

Fortran 77:

```
call sgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call dgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call cgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
call zgbequ(m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, info)
```

Fortran 95:

```
CALL GBEQU(a, r, c [,kl] [,rowcnd] [,colcnd] [,amax] [,info])
```

説明

このルーチンは、 $m \times n$ の帯行列 A を平衡化して、条件数を小さくするための、行と列のスケール係数を計算する。出力配列 r に行のスケール係数を返し、配列 c に列のスケール係数を返す。これらの係数は、成分 $b_{ij}=r(i)*a_{ij}*c(j)$ を持つ行列 B の各行と各列の最大の成分の絶対値が 1 になるように選択される。

?gbequ により計算されるスケール係数を使用する [?laqgb](#) 補助関数を参照のこと。

入力パラメーター

m	INTEGER。行列 A の行数。 $m \geq 0$ 。
n	INTEGER。行列 A の列数。 $(n \geq 0)$ 。
kl	INTEGER。 A の帯内の劣対角成分の数 $(kl \geq 0)$ 。
ku	INTEGER。 A の帯内の優対角成分の数 $(ku \geq 0)$ 。
ab	REAL (sgbequ の場合) DOUBLE PRECISION (dgbequ の場合) COMPLEX (cgbequ の場合) DOUBLE COMPLEX (zgbequ の場合) 配列、次元は $(ldab, *)$ 。 行 $1 \sim kl + ku + 1$ に格納される、元の帯行列 A を格納する。 ab の第 2 次元は $\max(1, n)$ 以上でなければならない。
$ldab$	INTEGER。 ab のリーディング・ディメンジョン。 $ldab \geq kl + ku + 1$ 。

出力パラメーター

r, c	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列 : $r(m), c(n)$ 。 $info = 0$ または $info > m$ の場合、配列 r に行列 A の行のスケール係数を返し、配列 c に列のスケール係数を返す。
--------	---

ル係数が格納される。
 $info=0$ の場合、配列 c に行列 A の列のスケール係数が格納される。

rowcnd REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
 $info=0$ または $info>m$ の場合、*rowcnd* に、最小の $r(i)$ を最大の $r(i)$ で割った値が格納される。

colcnd REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
 $info=0$ の場合、*colcnd* に、最小の $c(i)$ を最大の $c(i)$ で割った値が格納される。

amax REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
行列 A の最大の成分の絶対値。

info INTEGER。
 $info=0$ の場合、正常に終了したことを示す。
 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。
 $info=i$ 、かつ
 $i \leq m$ の場合、 A の i 番目の行が完全に 0 である。
 $i > m$ の場合、 A の $(i-m)$ 番目の列が完全に 0 である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbequ ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。サイズ $(k1+ku+1, n)$ の配列 A を格納する。

r 長さ (m) のベクトルを格納する。

c 長さ (n) のベクトルを格納する。

k1 省略した場合、 $k1 = ku$ とみなす。

ku $ku = lda - k1 - 1$ として復元する。

アプリケーション・ノート

r と c のすべての成分は、 $SMLNUM$ = 最小の安全な数値と $BIGNUM$ = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても、 A の条件数が小さくなることは保証されないが、実際には有効に機能する。

$rowcnd \geq 0.1$ であり、 $amax$ が大きすぎる値でも小さすぎる値でもない場合は、 r によるスケールリングは不要である。 $colcnd \geq 0.1$ の場合、 c によるスケールリングは不要である。

$amax$ の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 A をスケールリングする必要がある。

?poequ

対称 (エルミート) 正定値行列を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。

構文

Fortran 77:

```
call spoequ(n, a, lda, s, scond, amax, info)
call dpoequ(n, a, lda, s, scond, amax, info)
call cpoequ(n, a, lda, s, scond, amax, info)
call zpoequ(n, a, lda, s, scond, amax, info)
```

Fortran 95:

```
CALL POEQU(a, s [,scond] [,amax] [,info])
```

説明

このルーチンは、対称 (エルミート) 正定値行列 A を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 s に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 $b_{ij}=s(i)*a_{ij}*s(j)$ を持つスケーリング後の行列 B の対角成分が 1 に等しくなるように選択される。

s をこのように選択すると、 B の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 n を掛けた値の範囲内に収まる。

?poequ により計算されるスケール係数を使用する [?lagsy](#) 補助関数を参照のこと。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
a	REAL (spoequ の場合) DOUBLE PRECISION (dpoequ の場合) COMPLEX (cpoequ の場合) DOUBLE COMPLEX (zpoequ の場合) 配列、次元は ($lda, *$)。 スケール係数を計算する、 $n \times n$ の対称 / エルミート正定値行列 A を格納する。 A の対角成分だけが参照される。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。

出力パラメーター

<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (<i>n</i>)。 <i>info</i> = 0 の場合、配列 <i>s</i> に <i>A</i> のスケール係数が格納される。
<i>scond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 の場合、 <i>scond</i> に、最小の <i>s</i> (<i>i</i>) を最大の <i>s</i> (<i>i</i>) で割った値が格納される。
<i>amax</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分が正の値でない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

poequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>s</i>	長さ (<i>n</i>) のベクトルを格納する。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケールリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

?ppequ

圧縮形式の対称(エルミート)正定値行列を平衡化して、条件数を小さくするための行と列のスケール係数を計算する。

構文

Fortran 77:

```
call sppequ(uplo, n, ap, s, scnd, amax, info)
call dppequ(uplo, n, ap, s, scnd, amax, info)
```

```
call cppequ(uplo, n, ap, s, scnd, amax, info)
call zppequ(uplo, n, ap, s, scnd, amax, info)
```

Fortran 95:

```
CALL PPEQU(a, s [,scnd] [,amax] [,uplo] [,info])
```

説明

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列 A を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 s に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 $b_{ij}=s(i)*a_{ij}*s(j)$ を持つスケーリング後の行列 B の対角成分が 1 に等しくなるように選択される。

s をこのように選択すると、 B の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 n を掛けた値の範囲内に収まる。

?ppequ により計算されるスケール係数を使用する [?laqsp](#) 補助関数を参照のこと。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを配列 <code>ap</code> にパックするかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には行列 A の上三角部分が格納される。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には行列 A の下三角部分が格納される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>ap</code>	REAL (<code>sppequ</code> の場合) DOUBLE PRECISION (<code>dppequ</code> の場合) COMPLEX (<code>cppequ</code> の場合) DOUBLE COMPLEX (<code>zppequ</code> の場合) 配列、次元は $\max(1, n(n+1)/2)$ 以上。 配列 <code>ap</code> には、(<code>uplo</code> の指定に従って) 行列 A の上三角部分または下三角部分を圧縮格納形式で格納する (「 行列の格納形式 」を参照)。

出力パラメーター

<code>s</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (n)。 <code>info = 0</code> の場合、配列 <code>s</code> に A のスケール係数が格納される。
----------------	---

<i>scond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 の場合、 <i>scond</i> に、最小の <i>s</i> (<i>i</i>) を最大の <i>s</i> (<i>i</i>) で割った値が格納される。
<i>amax</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分が正の値でない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppsequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>s</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケールリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

?pbequ

対称(エルミート)正定値帯行列を平衡化して、条件数を小さくするための行と列のスケール係数を計算する。

構文

Fortran 77:

```
call spbequ(uplo, n, kd, ab, ldab, s, scnd, amax, info)
call dpbequ(uplo, n, kd, ab, ldab, s, scnd, amax, info)
call cpbequ(uplo, n, kd, ab, ldab, s, scnd, amax, info)
call zpbequ(uplo, n, kd, ab, ldab, s, scnd, amax, info)
```

Fortran 95:

```
CALL PBEQU(a, s [,scond] [,amax] [,uplo] [,info])
```

説明

このルーチンは、圧縮形式の対称 (エルミート) 正定値行列 A を平衡化して (2- ノルムに関して) 条件数を小さくするための、行と列のスケール係数を計算する。出力配列 s に、次のように計算されたスケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、成分 $b_{ij}=s(i)*a_{ij}*s(j)$ を持つスケーリング後の行列 B の対角成分が 1 に等しくなるように選択される。

s をこのように選択すると、 B の条件数は、すべての可能な対角スケーリングで、可能な最小の条件数に係数 n を掛けた値の範囲内に収まる。

?pbequ により計算されるスケール係数を使用する [?laqsb](#) 補助関数を参照のこと。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを配列 <code>ab</code> にパックするかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ab</code> には行列 A の上三角部分が格納される。 <code>uplo = 'L'</code> の場合、配列 <code>ab</code> には行列 A の下三角部分が格納される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>kd</code>	INTEGER。A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<code>ab</code>	REAL (<code>spbequ</code> の場合) DOUBLE PRECISION (<code>dpbequ</code> の場合) COMPLEX (<code>cpbequ</code> の場合) DOUBLE COMPLEX (<code>zpbequ</code> の場合) 配列、次元は (<code>ldab</code> , *)。 配列 <code>ap</code> には、(<code>uplo</code> で指定された) 行列 A の上三角部分または下三角部分が 帯形式 で格納される (「 行列の格納形式 」を参照)。 <code>ab</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>ldab</code>	INTEGER。ab のリーディング・ディメンジョン。 ($ldab \geq kd + 1$)。

出力パラメーター

<code>s</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (n)。 <code>info = 0</code> の場合、配列 <code>s</code> に A のスケール係数が格納される。
----------------	---

<i>scond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 の場合、 <i>scond</i> に、最小の <i>s</i> (<i>i</i>) を最大の <i>s</i> (<i>i</i>) で割った値が格納される。
<i>amax</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 <i>A</i> の最大の成分の絶対値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>A</i> の <i>i</i> 番目の対角成分が正の値でない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbequ ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ (<i>kd+1, n</i>) の配列 <i>A</i> を格納する。
<i>s</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

scond ≥ 0.1 であり、*amax* が大きすぎる値でも小さすぎる値でもない場合は、*s* によるスケールリングは不要である。

amax の値がオーバーフローまたはアンダーフローに非常に近い場合は、行列 *A* をスケールリングする必要がある。

ドライバールーチン

表 3-3 に、実数または複素数行列を係数行列とする連立線形方程式を解くための LAPACK ドライバールーチンを示す。

表 3-3 連立線形方程式の解の算出用のドライバールーチン

行列のタイプ、 格納形式	簡易ドライバー	高度ドライバー
一般	?gesv	?gesvx
一般帯	?qbsv	?qbsvx
一般三重対角	?qtsv	?qtsvx
対称 / エルミート 正定値	?posv	?posvx
対称 / エルミート 正定値 (圧縮格納形式)	?ppsv	?ppsvx
対称 / エルミート 正定値 (帯形式)	?pbsv	?pbsvx
対称 / エルミート 正定値 (三重対角形式)	?ptsv	?ptsvx
対称 / エルミート 不定値	?sysv/?hesv	?sysvx/?hesvx
対称 / エルミート 不定値 (圧縮格納形式)	?spsv/?hpsv	?spsvx/?hpsvx
複素対称	?sysv	?sysvx
複素対称 (圧縮格納形式)	?spsv	?spsvx

表中の ? は、**s** (単精度実数)、**d** (倍精度実数)、**c** (単精度複素数)、または **z** (倍精度複素数) を示す。

?gesv

正方行列 A を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call sgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call cgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
call zgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL GESV(a, b [,ipiv] [,info])
```

説明

このルーチンは、連立線形方程式 $AX=B$ を X について解く。 A は $n \times n$ 行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

部分的なピボット演算と行の交換による LU 分解を使用して、 A を $A=PLU$ として因子分解する。 P は置換行列、 L は単位下三角行列、 U は上三角行列である。次に、 A の因子分解された形式を使用して、連立方程式 $AX=B$ を解く。

入力パラメーター

n	INTEGER。 A の次数。 B の行数 ($n \geq 0$)。
$nrhs$	INTEGER。 右辺の数。 B の列数 ($nrhs \geq 0$)。
a, b	REAL (sgesv の場合) DOUBLE PRECISION (dgesv の場合) COMPLEX (cgesv の場合) DOUBLE COMPLEX (zgesv の場合) 配列 : $a(lda, *)$, $b ldb, *$ 。 配列 a には、行列 A が格納される。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 b の第 2 次元は、 $\max(1, nrhs)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

a	$A=PLU$ の因子分解で得られた係数 L と U によって上書きされる。 L の単位対角成分は格納されない。
b	解の行列 X によって上書きされる。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 置換行列 P を定義するピボットのインデックス。行列の行 i は、行 $ipiv(i)$ と交換される。
$info$	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。 $info=i$ の場合、 $U(i, i)$ が完全に 0 である。因子分解は完了したが、係数 U が完全に特異であるため、解を計算できなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gesv ルーチンのインターフェイス特有の詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
-----	-----------------------------

<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。

?gesvx

正方行列 *A* を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
call zgesvx(FACT, TRANS, N, NRHS, A, LDA, AF, LDAF, IPIV, EQUED, R, C, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL GESVX(a, b, x [,af] [,ipiv] [,fact] [,trans] [,equed] [,r] [,c]
           [,ferr] [,berr] [,rcond] [,rpvgrw] [,info])
```

説明

このルーチンは、LU 因子分解を使用して、実数または複素数連立線形方程式 $AX = B$ の解を計算する。A は $n \times n$ 行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gesvx は、以下の手順を実行する。

1. *fact* = 'E' の場合、連立方程式を平衡化するための実スケール係数 *r* と *c* を計算する。

$$\text{trans} = \text{'N'}: \text{diag}(r) * A * \text{diag}(c) * \text{diag}(c)^{-1} * X = \text{diag}(r) * B$$

$$\text{trans} = \text{'T'}: (\text{diag}(r) * A * \text{diag}(c))^T * \text{diag}(r)^{-1} * X = \text{diag}(c) * B$$

$$\text{trans} = \text{'C'}: (\text{diag}(r) * A * \text{diag}(c))^H * \text{diag}(r)^{-1} * X = \text{diag}(c) * B$$

連立方程式が平衡化されるかどうかは、行列 *A* のスケーリングによって決まる。平衡化が行われる場合、*A* は $\text{diag}(r) * A * \text{diag}(c)$ によって上書きされ、*B* は $\text{diag}(r) * B$ (*trans* = 'N' の場合) または $\text{diag}(c) * B$ (*trans* = 'T' または 'C' の場合) によって上書きされる。

2. *fact* = 'N' または 'E' の場合、LU 分解を使用して、(*fact* = 'E' の場合は平衡化の後に) 行列 *A* を $A = PLU$ として因子分解する。*P* は置換行列、*L* は単位下三角行列、*U* は上三角行列である。

3. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
4. A の因子分解された形式を使用して、連立方程式を X について解く。
5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。
6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $diag(c)$ ($trans = 'N'$ の場合) または $diag(r)$ ($trans = 'T'$ または $'C'$ の場合) によって、行列 X を事前乗算する。

入力パラメーター

<i>fact</i>	<p>CHARACTER*1。'F'、'N'、または'E' でなければならない。</p> <p>ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。</p> <p><i>fact</i> = 'F' の場合、開始時に、<i>af</i> と <i>ipiv</i> に A の因子分解された形式が格納される。<i>equed</i> が 'N' でない場合は、行列 A は、r と c で指定されたスケール係数によって平衡化されている。a、<i>af</i>、<i>ipiv</i> は変更されない。</p> <p><i>fact</i> = 'N' の場合、行列 A を <i>af</i> にコピーし、因子分解を実行する。</p> <p><i>fact</i> = 'E' の場合、必要に応じて行列 A を平衡化してから、<i>af</i> にコピーし、因子分解を実行する。</p>
<i>trans</i>	<p>CHARACTER*1。'N'、'T'、または'C' でなければならない。</p> <p>連立方程式の形式を指定する。</p> <p><i>trans</i> = 'N' の場合、連立方程式の形式は $AX = B$ (転置なし) である。</p> <p><i>trans</i> = 'T' の場合、連立方程式の形式は $A^T X = B$ (転置) である。</p> <p><i>trans</i> = 'C' の場合、連立方程式の形式は $A^H X = B$ (共役転置) である。</p>
<i>n</i>	INTEGER。線形方程式の数。行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。行列 B と X の列数 ($nrhs \geq 0$)。
<i>a, af, b, work</i>	<p>REAL (sgesvx の場合)</p> <p>DOUBLE PRECISION (dgesvx の場合)</p> <p>COMPLEX (cgesvx の場合)</p> <p>DOUBLE COMPLEX (zgesvx の場合)</p> <p>配列: $a(lda, *)$, $af(ldaf, *)$, $b ldb, *)$, $work(*)$。</p> <p>配列 a には、行列 A が格納される。<i>fact</i> = 'F' で、<i>equed</i> が 'N' でない場合は、A は、r または c、あるいはその両方のスケール係数によって平衡化されている。a の第 2 次元は $\max(1, n)$ 以上でなければならない。</p> <p>配列 af は、<i>fact</i> = 'F' の場合は入力引数になる。この配列には、</p>

行列 A の因子分解された形式、すなわち、[?getrf](#) による因子分解 $A = PLU$ で得られた係数 L と U が格納される。*equed* が 'N' でない場合は、*af* は、平衡化された行列 A の因子分解された形式になる。*af* の第 2 次元は $\max(1, n)$ 以上でなければならない。配列 *b* には、行列 B が格納される。この行列の列は、連立方程式の右辺である。*b* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

work()* は、ワークスペース配列。

work の次元は $\max(1, 4*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この配列には、 ?getrf による因子分解 $A = PLU$ で得られたピボットのインデックスが格納される。行列の行 i は、行 <i>ipiv</i> (i) と交換される。
<i>equed</i>	CHARACTER*1。 'N'、'R'、'C'、または 'B' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N' の場合、常に真)。 <i>equed</i> = 'R' の場合、行の平衡化が行われ、 A は $\text{diag}(r)$ によって事前乗算されている。 <i>equed</i> = 'C' の場合、列の平衡化が行われ、 A は $\text{diag}(c)$ によって事後乗算されている。 <i>equed</i> = 'B' の場合、行と列の平衡化が行われ、 A は $\text{diag}(r)*A*\text{diag}(c)$ で置き換えられている。
<i>r, c</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: $r(n), c(n)$ 。 配列 <i>r</i> には A の行のスケール係数が格納され、配列 <i>c</i> には A の列のスケール係数が格納される。これらの配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'R' または 'B' の場合、 A は $\text{diag}(r)$ によって左辺で乗算される。 <i>equed</i> = 'N' または 'C' の場合、 <i>r</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'R' または 'B' の場合、 <i>r</i> の各成分は正の値でなければならない。 <i>equed</i> = 'C' または 'B' の場合、 A は $\text{diag}(c)$ によって右辺で乗算される。 <i>equed</i> = 'N' または 'R' の場合、 <i>c</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'C' または 'B' の場合、 <i>c</i> の各成分は正の値でなければならない。
<i>ldx</i>	INTEGER。出力配列 x の第 1 次元。 $ldx \geq \max(1, n)$ 。

<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) ワークスペース配列、次元は $\max(1, 2*n)$ 以上。複素数型でのみ使用される。

出力パラメーター

<i>x</i>	REAL (<i>sgesvx</i> の場合) DOUBLE PRECISION (<i>dgesvx</i> の場合) COMPLEX (<i>cgesvx</i> の場合) DOUBLE COMPLEX (<i>zgesvx</i> の場合) 配列、次元は (<i>ldx</i> , *)。 <i>info</i> = 0 または <i>info</i> = <i>n</i> +1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。ただし、 <i>equed</i> ≠ 'N' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は次のようになる。 <i>diag(c)</i> ⁻¹ * <i>X</i> (<i>trans</i> = 'N' で、 <i>equed</i> = 'C' または 'B' の場合) <i>diag(r)</i> ⁻¹ * <i>X</i> (<i>trans</i> = 'T' または 'C' で、 <i>equed</i> = 'R' または 'B' の場合)。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>a</i>	配列 <i>a</i> は、 <i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。 <i>equed</i> ≠ 'N' の場合、 <i>A</i> は終了時に次のようにスケールされる。 <i>equed</i> = 'R': $A = \text{diag}(r)*A$ <i>equed</i> = 'C': $A = A*\text{diag}(c)$ <i>equed</i> = 'B': $A = \text{diag}(r)*A*\text{diag}(c)$
<i>af</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>af</i> は出力引数になる。この引数は、終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) の因子分解 $A = PLU$ で得られた係数 <i>L</i> と <i>U</i> を返す。平衡化された行列の形式については、 <i>a</i> の説明を参照のこと。
<i>b</i>	<i>diag(r)*B</i> によって上書きされる (<i>trans</i> = 'N' で、 <i>equed</i> = 'R' または 'B' の場合) <i>diag(c)*B</i> によって上書きされる (<i>trans</i> = 'T' で、 <i>equed</i> = 'C' または 'B' の場合) 変更されない (<i>equed</i> = 'N' の場合)
<i>r, c</i>	これらの配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。「入力引数」セクションの <i>r</i> , <i>c</i> の説明を参照のこと。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。推定値のアンダーフローが発生した場合、 <i>rcond</i> = 0 に設定

される。この場合、行列は有効な精度で特異になる。ただし、 $rcond$ が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。

<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>ipiv</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>ipiv</i> は出力引数になる。この引数には、終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) の因子分解 $A = PLU$ で得られたピボットのインデックスが格納される。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>work, rwork</i>	終了時に、 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) には、ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。「最大絶対成分」ノルムが使用される。 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) が 1 よりはるかに小さい場合は、(平衡化された) 行列 <i>A</i> の <i>LU</i> 因子分解の安定性は低くなる。このため、解 <i>x</i> 、条件推定子 <i>rcond</i> 、前進誤差範囲 <i>ferr</i> の信頼性も低くなる。 $0 < info \leq n$ で因子分解に失敗した場合は、 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) には、 <i>A</i> の先頭の <i>info</i> 列のピボット演算成長係数の逆数が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、 $U(i, i)$ は完全にゼロである。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 $rcond = 0$ が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gesvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n, nrhs</i>) の行列 <i>X</i> を格納する。
<i>af</i>	サイズ (<i>n, n</i>) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。

<i>r</i>	長さ (<i>n</i>) のベクトルを格納する。各成分のデフォルト値は、 $r(i) = 1.0_WP$ である。
<i>c</i>	長さ (<i>n</i>) のベクトルを格納する。各成分のデフォルト値は、 $c(i) = 1.0_WP$ である。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>equed</i>	'N'、'B'、'C'、または 'R' でなければならない。デフォルト値は 'N'。
<i>rpvgrw</i>	実数値。ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。

?gbsv

帯行列 *A* を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call sgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call dgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call cgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
call zgbsv(n, kl, ku, nrhs, ab, ldab, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL GBSV(a, b [,kl] [,ipiv] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX = B$ を、 X について解く。 A は k_l 個の劣対角成分と k_u 個の優対角成分を持つ $n \times n$ の帯行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

部分的なピボット演算と行の交換による LU 分解を使用して、 A を $A = LU$ として因子分解する。 L は置換行列と k_l 個の劣対角成分を持つ単位下三角行列の積、 U は $k_l + k_u$ 個の優対角成分を持つ上三角行列である。次に、 A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメーター

n INTEGER。 A の次数。 B の行数 ($n \geq 0$)。

<i>k1</i>	INTEGER。 <i>A</i> の帯内の劣対角成分の数 ($k1 \geq 0$)。
<i>ku</i>	INTEGER。 <i>A</i> の帯内の優対角成分の数 ($ku \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数。 <i>B</i> の列数 ($nrhs \geq 0$)。
<i>ab, b</i>	REAL (sgbsv の場合) DOUBLE PRECISION (dgbv の場合) COMPLEX (cgbsv の場合) DOUBLE COMPLEX (zgbv の場合) 配列: <i>ab(ldab, *)</i> , <i>b(ldb, *)</i> 。 配列 <i>ab</i> には、行列 <i>A</i> が帯形式で格納される (「 行列の格納形式 」を参照)。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 配列 <i>ab</i> の第 1 次元。 ($ldab \geq 2k1 + ku + 1$)
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>ab</i>	<i>L</i> と <i>U</i> によって上書きされる。 <i>U</i> の対角成分と $k1 + ku$ 個の優対角成分は、 <i>ab</i> の最初の $1 + k1 + ku$ 行に格納される。 <i>L</i> の計算に使用される乗数は、次の <i>k1</i> 行に格納される。
<i>b</i>	解の行列 <i>X</i> によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス: 行 <i>i</i> は行 <i>ipiv(i)</i> と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>U</i> (<i>i, i</i>) が完全に 0 である。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解を計算できなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ $(2*k1+ku+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>k1</i>	省略した場合、 $k1 = ku$ とみなす。
<i>ku</i>	$ku = lda - 2*k1 - 1$ として復元する。

?gbsvx

帯行列 A を係数行列とする、複数の右辺を持つ実数または複素数連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
call zgbsvx(FACT, TRANS, N, kl, ku, NRHS, Ab, LDAb, AFb, LDAFb, IPIV,
            EQUED, R, C, B, LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL GBSVX (a, b, x [,kl] [,af] [,ipiv] [,fact] [,trans] [,equed] [,r]
            [,c] [,ferr] [,berr] [,rcond] [,rpvgrw] [,info])
```

説明

このルーチンは、 LU 因子分解を使用して、実数または複素数連立線形方程式 $AX = B$ 、 $A^T X = B$ 、または $A^H X = B$ の解を計算する。 A は kl 個の劣対角成分と ku 個の優対角成分を持つ次数 n の帯行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gbsvx は、以下の手順を実行する。

1. $fact = 'E'$ の場合、連立方程式を平衡化するための実スケール係数 r と c を計算する。

$$trans = 'N': \quad \text{diag}(r) * A * \text{diag}(c) * \text{diag}(c)^{-1} * X = \text{diag}(r) * B$$

$$trans = 'T': \quad (\text{diag}(r) * A * \text{diag}(c))^T * \text{diag}(r)^{-1} * X = \text{diag}(c) * B$$

$$trans = 'C': \quad (\text{diag}(r) * A * \text{diag}(c))^H * \text{diag}(r)^{-1} * X = \text{diag}(c) * B$$

連立方程式が平衡化されるかどうかは、行列 A のスケーリングによって決まる。平衡化が行われる場合、 A は $\text{diag}(r) * A * \text{diag}(c)$ によって上書きされ、 B は $\text{diag}(r) * B$ ($trans = 'N'$ の場合) または $\text{diag}(c) * B$ ($trans = 'T'$ または $'C'$ の場合) によって上書きされる。

2. $fact = 'N'$ または $'E'$ の場合、 LU 分解を使用して、($fact = 'E'$ の場合は平衡化の後に) 行列 A を $A = LU$ として因子分解する。 L は置換行列と kl 個の劣対角成分を持つ単位下三角行列の積、 U は $kl+ku$ 個の優対角成分を持つ上三角行列である。

3. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。
5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。
6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(c)$ ($\text{trans} = 'N'$ の場合) または $\text{diag}(r)$ ($\text{trans} = 'T'$ または $'C'$ の場合) によって、行列 X を事前乗算する。

入力パラメーター

<i>fact</i>	<p>CHARACTER*1. 'F'、'N'、または 'E' でなければならない。</p> <p>ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。</p> <p>$fact = 'F'$ の場合、開始時に、afb と $ipiv$ に A の因子分解された形式が格納される。$equed$ が 'N' でない場合は、行列 A は、r と c で指定されたスケール係数によって平衡化されている。ab、afb、$ipiv$ は変更されない。</p> <p>$fact = 'N'$ の場合、行列 A を afb にコピーし、因子分解を実行する。</p> <p>$fact = 'E'$ の場合、必要に応じて行列 A を平衡化してから、afb にコピーし、因子分解を実行する。</p>
<i>trans</i>	<p>CHARACTER*1. 'N'、'T'、または 'C' でなければならない。</p> <p>連立方程式の形式を指定する。</p> <p>$trans = 'N'$ の場合、連立方程式の形式は $A X = B$ (転置なし) である。</p> <p>$trans = 'T'$ の場合、連立方程式の形式は $A^T X = B$ (転置) である。</p> <p>$trans = 'C'$ の場合、連立方程式の形式は $A^H X = B$ (共役転置) である。</p>
<i>n</i>	INTEGER。線形方程式の数。行列 A の次数 ($n \geq 0$)。
<i>kl</i>	INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。
<i>ku</i>	INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。行列 B と X の列数 ($nrhs \geq 0$)。
<i>ab,afb,b,work</i>	<p>REAL ($sgesvx$ の場合)</p> <p>DOUBLE PRECISION ($dgesvx$ の場合)</p> <p>COMPLEX ($cgesvx$ の場合)</p> <p>DOUBLE COMPLEX ($zgesvx$ の場合)</p> <p>配列: $a(lda, *)$, $af(ldaf, *)$, $b(ldb, *)$, $work(*)$。</p> <p>配列 ab には、行列 A が帯形式で格納される (「行列の格納形式」を参照)。</p> <p>ab の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p>$fact = 'F'$ で、$equed$ が 'N' でない場合は、A は、r または c、あるいはその両方のスケール係数によって平衡化されている。</p>

配列 *afb* は、*fact* = 'F' の場合は入力引数になる。
afb の第 2 次元は $\max(1, n)$ 以上でなければならない。
 この配列には、行列 *A* の因子分解された形式、すなわち、
[?gbtrf](#) による因子分解 $A = LU$ で得られた係数 *L* と *U* が格納される。
U は、 $k_l + k_u$ 個の優対角成分を持つ上三角帯行列として、
afb の最初の $1 + k_l + k_u$ 行に格納される。因子分解に使用した
 乗数は、次の k_l 行に格納される。
equed が 'N' でない場合は、*afb* は、平衡化された行列 *A* の因子
 分解された形式になる。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程
 式の右辺である。*b* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

work(*) は、ワークスペース配列。
work の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$
 (複素数型の場合) 以上でなければならない。

<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $ldab \geq k_l + k_u + 1$ 。
<i>ldafb</i>	INTEGER。 <i>afb</i> の第 1 次元。 $ldafb \geq 2 * k_l + k_u + 1$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この配列には、 ?gbtrf による因子分解 $A = LU$ で得られたピ ボットのインデックスが格納される。行列の行 <i>i</i> は、行 <i>ipiv</i> (<i>i</i>) と交換される。
<i>equed</i>	CHARACTER*1。 'N'、'R'、'C'、または 'B' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実 行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N' の場 合、常に真)。 <i>equed</i> = 'R' の場合、行の平衡化が行われ、 <i>A</i> は $\text{diag}(r)$ によって 事前乗算されている。 <i>equed</i> = 'C' の場合、列の平衡化が行われ、 <i>A</i> は $\text{diag}(c)$ によって 事後乗算されている。 <i>equed</i> = 'B' の場合、行と列の平衡化が行われ、 <i>A</i> は $\text{diag}(r) * A * \text{diag}(c)$ で置き換えられている。
<i>r, c</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: <i>r</i> (<i>n</i>), <i>c</i> (<i>n</i>)。 配列 <i>r</i> には <i>A</i> の行のスケール係数が格納され、配列 <i>c</i> には <i>A</i> の 列のスケール係数が格納される。これらの配列は、 <i>fact</i> = 'F' の 場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'R' または 'B' の場合、 <i>A</i> は $\text{diag}(r)$ によって左辺で乗算 される。 <i>equed</i> = 'N' または 'C' の場合、 <i>r</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'R' または 'B' の場合、 <i>r</i> の各成分は正の 値でなければならない。

$equed = 'C'$ または $'B'$ の場合、 A は $diag(c)$ によって右辺で乗算される。 $equed = 'N'$ または $'R'$ の場合、 c は使用されない。 $fact = 'F'$ で、 $equed = 'C'$ または $'B'$ の場合、 c の各成分は正の値でなければならない。

ldx	INTEGER。出力配列 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
$rwork$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

x	REAL (sgbsvx の場合) DOUBLE PRECISION (dgbvsx の場合) COMPLEX (cgbsvx の場合) DOUBLE COMPLEX (zgbvsx の場合) 配列、次元は $(ldx, *)$ 。 $info = 0$ または $info = n+1$ の場合、配列 x には、元の連立方程式の解の行列 X が格納される。ただし、 $equed \neq 'N'$ の場合、 A と B は終了時に修正され、平衡化された連立方程式の解は次のようになる。 $diag(c)^{-1} * X$ ($trans = 'N'$ で、 $equed = 'C'$ または $'B'$ の場合) $diag(r)^{-1} * X$ ($trans = 'T'$ または $'C'$ で、 $equed = 'R'$ または $'B'$ の場合)。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
ab	配列 ab は、 $fact = 'F'$ または $'N'$ の場合、あるいは $fact = 'E'$ で $equed = 'N'$ の場合、終了時に変更されない。 $equed \neq 'N'$ の場合、 A は終了時に次のようにスケーリングされる。 $equed = 'R'$: $A = diag(r) * A$ $equed = 'C'$: $A = A * diag(c)$ $equed = 'B'$: $A = diag(r) * A * diag(c)$
afb	$fact = 'N'$ または $'E'$ の場合、 afb は出力引数になる。この引数は、終了時に、元の行列 A ($fact = 'N'$ の場合) または平衡化された行列 A ($fact = 'E'$ の場合) の LU 因子分解の詳細を返す。平衡化された行列の形式については、 ab の説明を参照のこと。
b	$diag(r) * b$ によって上書きされる ($trans = 'N'$ で、 $equed = 'R'$ または $'B'$ の場合) $diag(c) * b$ によって上書きされる ($trans = 'T'$ で、 $equed = 'C'$ または $'B'$ の場合) 変更されない ($equed = 'N'$ の場合)
r, c	これらの配列は、 $fact \neq 'F'$ の場合は出力引数になる。「入力引数」セクションの r, c の説明を参照のこと。

<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>ipiv</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>ipiv</i> は出力引数になる。この引数には、終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) の因子分解 $A = LU$ で得られたピボットのインデックスが格納される。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>work</i> , <i>rwork</i>	終了時に、 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) には、ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。「最大絶対成分」ノルムが使用される。 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) が 1 よりはるかに小さい場合は、(平衡化された) 行列 <i>A</i> の <i>LU</i> 因子分解の安定性は低くなる。このため、解 <i>x</i> 、条件推定子 <i>rcond</i> 、前進誤差範囲 <i>ferr</i> の信頼性も低くなる。0 < <i>info</i> ≤ <i>n</i> で因子分解に失敗した場合は、 <i>work</i> (1) (実数型の場合) または <i>rwork</i> (1) (複素数型の場合) には、 <i>A</i> の先頭の <i>info</i> 列のピボット演算成長係数の逆数が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>U</i> (<i>i</i> , <i>i</i>) は完全にゼロである。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gbsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ($k1+ku+1, n$) の配列 <i>A</i> を格納する。
----------	--

<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n, nrhs</i>) の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ($2*k1+ku+1, n$) の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>r</i>	長さ (<i>n</i>) のベクトルを格納する。各成分のデフォルト値は、 $r(i)=1.0_WP$ である。
<i>c</i>	長さ (<i>n</i>) のベクトルを格納する。各成分のデフォルト値は、 $c(i)=1.0_WP$ である。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>equed</i>	'N'、'B'、'C'、または 'R' でなければならない。デフォルト値は 'N'。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。
<i>rpgvgrw</i>	実数値。ピボット成長係数の逆数 $\text{norm}(A)/\text{norm}(U)$ が格納される。
<i>k1</i>	省略した場合、 $k1 = ku$ とみなす。
<i>ku</i>	$ku = lda - k1 - 1$ として復元する。

?gtsv

三重対角行列 *A* を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call sgtsv(n, nrhs, dl, d, du, b, ldb, info)
call dgtsv(n, nrhs, dl, d, du, b, ldb, info)
call cgtsv(n, nrhs, dl, d, du, b, ldb, info)
call zgtsv(n, nrhs, dl, d, du, b, ldb, info)
```

Fortran 95:

```
CALL GTSV(dl, d, du, b [,info])
```

説明

このルーチンは、連立線形方程式 $AX=B$ を *X* について解く。*A* は $n \times n$ の三重対角行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。
このルーチンは、部分的なピボット演算によるガウス消去を使用する。

方程式 $A^T X = B$ の解は、引数 du と $d1$ の順序を入れ替えて求められる。

入力パラメーター

n	INTEGER。 A の次数。 B の行数 ($n \geq 0$)。
$nrhs$	INTEGER。 右辺の数。 B の列数 ($nrhs \geq 0$)。
$d1, d, du, b$	REAL (sgtsv の場合) DOUBLE PRECISION (dgtsv の場合) COMPLEX (cgtsv の場合) DOUBLE COMPLEX (zgtsv の場合) 配列 : $d1(n-1), d(n), du(n-1), b(l, db, *)$ 。 配列 $d1$ には、 A の $(n-1)$ 個の劣対角成分が格納される。配列 d には A の対角成分を格納する。 配列 du には、 A の $(n-1)$ 個の優対角成分が格納される。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

$d1$	A の LU 因子分解で得られた上三角行列 U の 2 番目の優対角成分の $(n-2)$ 個の成分によって上書きされる。これらの成分は、 $d1(1), \dots, d1(n-2)$ に格納される。
d	U の n 個の対角成分によって上書きされる。
du	U の最初の優対角成分の $(n-1)$ 個の成分によって上書きされる。
b	解の行列 X によって上書きされる。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、 $U(i, i)$ が完全に 0 であるため、解は計算されなかった。 $i = n$ でない限り、因子分解は完了していない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtsv ルーチンのインターフェイス特有の詳細を以下に示す。

$d1$	長さ $(n-1)$ のベクトルを格納する。
d	長さ (n) のベクトルを格納する。
$d1$	長さ $(n-1)$ のベクトルを格納する。
b	サイズ $(n, nrhs)$ の行列 B を格納する。

?gtsvx

三重対角行列 A を係数行列とする、複数の右辺を持つ実数または複素数連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call dgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call cgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
call zgtsvx(FACT, TRANS, N, NRHS, dl, d, du, dlf, df, duf, du2, IPIV, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL GTSVX (dl, d, du, b, x [,dlf] [,df] [,duf] [,du2] [,ipiv] [,fact]
           [,trans] [,ferr] [,berr] [,rcond] [,info])
```

説明

このルーチンは、 LU 因子分解を使用して、実数または複素数連立線形方程式 $AX=B$ 、 $A^T X=B$ 、または $A^H X=B$ の解を計算する。 A は次数 n の三重対角行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?gtsvx は、以下の手順を実行する。

1. $fact = 'N'$ の場合、 LU 分解を使用して、行列 A を $A=LU$ として因子分解する。 L は置換行列と単位下二重対角行列の積で、 U は主対角成分と最初の 2 つの優対角成分にのみ 0 でない値を持つ上三角行列である。
2. $U_{i,i} = 0$ の場合、つまり U が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

fact

CHARACTER*1。'F' または 'N' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。

fact = 'F' の場合、開始時に、*d1f*、*df*、*duf*、*du2*、*ipiv* に、*A* の因子分解された形式が格納される。配列 *d1*、*d*、*du*、*d1f*、*df*、*duf*、*du2*、*ipiv* は変更されない。

fact = 'N' の場合、行列 *A* を *d1f*、*df*、*duf* にコピーし、因子分解を実行する。

trans CHARACTER*1。'N'、'T'、または 'C' でなければならない。
連立方程式の形式を指定する。

trans = 'N' の場合、連立方程式の形式は $AX = B$ (転置なし) である。
trans = 'T' の場合、連立方程式の形式は $A^T X = B$ (転置) である。
trans = 'C' の場合、連立方程式の形式は $A^H X = B$ (共役転置) である。

n INTEGER。線形方程式の数。行列 *A* の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。行列 *B* と *X* の列数 ($nrhs \geq 0$)。

d1, *d*, *du*, *d1f*, *df*, *duf*,
du2, *b*, *x*, *work*

REAL (sgtsvx の場合)
DOUBLE PRECISION (dgtsvx の場合)
COMPLEX (cgtsvx の場合)
DOUBLE COMPLEX (zgtsvx の場合)

配列：

d1、次元は $(n-1)$ 。*A* の劣対角成分を格納する。

d、次元は (n) 。*A* の対角成分を格納する。

du、次元は $(n-1)$ 。*A* の優対角成分を格納する。

d1f、次元は $(n-1)$ 。*fact* = 'F' の場合、*d1f* は入力引数になる。この引数には、開始時に、[?gttrf](#) による *A* の LU 因子分解で得られた行列 *L* を定義する、 $(n-1)$ 個の乗数が格納される。

df、次元は (n) 。*fact* = 'F' の場合、*df* は入力引数になる。この引数には、開始時に、*A* の LU 因子分解で得られた上三角行列 *U* の n 個の対角成分が格納される。

duf、次元は $(n-1)$ 。*fact* = 'F' の場合、*duf* は入力引数になる。この引数には、開始時に、*U* の最初の優対角成分の $(n-1)$ 個の成分が格納される。

du2、次元は $(n-2)$ 。*fact* = 'F' の場合、*du2* は入力引数になる。この引数には、開始時に、*U* の 2 番目の優対角成分の $(n-2)$ 個の成分が格納される。

b(*ldb*, *) には、右辺の行列 *B* が格納される。*b* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

x(*ldx*, *) には、解の行列 *X* が格納される。*x* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列である。

$work$ の次元は、 $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数の場合) 以上でなければならない。

ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
ldx	INTEGER。 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 $fact = 'F'$ の場合、 $ipiv$ は入力引数になる。この配列には、開始時に、 ?qtrf によって返されたピボットのインデックスが格納される。
$iwork$	INTEGER。 ワークスペース配列、次元は (n) 。実数型でのみ使用される。
$rwork$	REAL ($cgtsvx$ の場合) DOUBLE PRECISION ($zgtsvx$ の場合) ワークスペース配列、次元は (n) 。複素型でのみ使用される。

出力パラメーター

x	REAL ($sgtsvx$ の場合) DOUBLE PRECISION ($dgtsvx$ の場合) COMPLEX ($cgtsvx$ の場合) DOUBLE COMPLEX ($zgtsvx$ の場合) 配列、次元は $(ldx, *)$ 。 $info = 0$ または $info = n+1$ の場合、配列 x には、解の行列 X が格納される。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
$d1f$	$fact = 'N'$ の場合、 $d1f$ は出力引数になる。この引数には、終了時に、 A の LU 因子分解で得られた行列 L を定義する、 $(n-1)$ 個の乗数が格納される。
df	$fact = 'N'$ の場合、 df は出力引数になる。この引数には、終了時に、 A の LU 因子分解で得られた上三角行列 U の n 個の対角成分が格納される。
duf	$fact = 'N'$ の場合、 duf は出力引数になる。この引数には、終了時に、 U の最初の優対角成分の $(n-1)$ 個の成分が格納される。
$du2$	$fact = 'N'$ の場合、 $du2$ は出力引数になる。この引数には、終了時に、 U の 2 番目の優対角成分の $(n-2)$ 個の成分が格納される。
$ipiv$	配列 $ipiv$ は、 $fact = 'N'$ の場合は出力引数になる。この配列は、終了時に、因子分解 $A = LU$ で得られたピボットのインデックスが格納される。行列の行 i は、行 $ipiv(i)$ と交換される。 $ipiv(i)$ の値は、常に i または $i+1$ になる。 $ipiv(i)=i$ は、行の交換が不要であったことを示す。
$rcond$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 A の条件数の逆数の推定値。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。

<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と後退誤差を格納する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、 $U(i, i)$ は完全にゼロである。 $i = n$ でない限り、因子分解は完了していない。係数 U が完全に特異であるため、解と誤差範囲は計算できなかった。 $rcond = 0$ が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 U は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

gtsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d1</i>	長さ ($n-1$) のベクトルを格納する。
<i>d</i>	長さ (n) のベクトルを格納する。
<i>du</i>	長さ ($n-1$) のベクトルを格納する。
<i>b</i>	サイズ ($n, nrhs$) の行列 B を格納する。
<i>x</i>	サイズ ($n, nrhs$) の行列 X を格納する。
<i>d1f</i>	長さ ($n-1$) のベクトルを格納する。
<i>df</i>	長さ (n) のベクトルを格納する。
<i>duf</i>	長さ ($n-1$) のベクトルを格納する。
<i>du2</i>	長さ ($n-2$) のベクトルを格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>ferr</i>	長さ ($nrhs$) のベクトルを格納する。
<i>berr</i>	長さ ($nrhs$) のベクトルを格納する。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>d1f</i> 、 <i>df</i> 、 <i>duf</i> 、 <i>du2</i> 、および <i>ipiv</i> を指定しなければならない。指定しない場合は、エラーが返される。
<i>trans</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

?posv

対称/エルミート正定値行列 A を係数行列とする、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call sposv(uplo, n, nrhs, a, lda, b, ldb, info)
call dposv(uplo, n, nrhs, a, lda, b, ldb, info)
call cposv(uplo, n, nrhs, a, lda, b, ldb, info)
call zposv(uplo, n, nrhs, a, lda, b, ldb, info)
```

Fortran 95:

```
CALL POSV(a, b [,uplo] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX = B$ を、 X について解く。 A は $n \times n$ の対称/エルミート正定値行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

コレスキー因子分解を使用して、 A を $A = U^H U$ ($uplo = 'U'$ の場合)

または $A = LL^H$ ($uplo = 'L'$ の場合) として因子分解する。 U は上三角行列、 L は下三角行列である。次に、 A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>a</code> には行列 A の上三角部分が格納され、A は $U^H U$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>a</code> には行列 A の下三角部分が格納され、A は LL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。B の列数 ($nrhs \geq 0$)。
<code>a, b</code>	REAL (sposv の場合) DOUBLE PRECISION (dposv の場合) COMPLEX (cpovs の場合) DOUBLE COMPLEX (zposv の場合) 配列: <code>a(lda, *)</code> , <code>b(ldb, *)</code> 。 配列 <code>a</code> には、行列 A の上三角部分または下三角部分を格納する (<code>uplo</code> を参照)。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

lda INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。

ldb INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

a $info=0$ の場合、 $uplo$ の指定に従って、 a の上三角部分または下三角部分が、コレスキー係数 U または L によって上書きされる。

b 解の行列 X によって上書きされる。

$info$ INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。
 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。
 $info=i$ の場合、次数 i の先頭の小行列式(したがって、行列 A そのもの)が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

posv ルーチンのインターフェイス特有の詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。

b サイズ $(n, nrhs)$ の行列 B を格納する。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

?posvx

コレスキー因子分解を使用して、対称/エルミート正定値行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call dposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call cposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

```
call zposvx(FACT, uplo, N, NRHS, A, LDA, AF, LDAF, EQUED, s, B, LDB, X,
            LDX, RCOND, FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL POSVX (a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]
           [,rcond] [,info])
```

説明

このルーチンは、コレスキー因子分解 $A=U^H U$ または $A=LL^H$ を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。A は $n \times n$ の実対称/エルミート正定値行列、行列 B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?posvx` は、以下の手順を実行する。

1. `fact = 'E'` の場合、連立方程式を平衡化するための実スケール係数 s を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列 A のスケーリングによって決まる。平衡化が行われる場合は、A は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされ、B は $\text{diag}(s) * B$ によって上書きされる。

2. `fact = 'N'` または `'E'` の場合、コレスキー分解を使用して、(`fact = 'E'` の場合は平衡化の後に) 行列 A を次のように因子分解する。

$$A = U^H U \text{ (uplo = 'U' の場合)}$$

$$A = LL^H \text{ (uplo = 'L' の場合)}$$

ここで、U は上三角行列、L は下三角行列である。

3. 先頭の $i \times i$ の主小行列式が正定値でない場合は、ルーチンは `info = i` を返す。それ以外の場合、A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として `info = n + 1` を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(s)$ によって行列 X を事前に乗算する。

入力パラメーター

`fact` CHARACTER*1. 'F'、'N'、または 'E' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。

`fact = 'F'` の場合、開始時に、`af` に A の因子分解された形式が格納される。`equed = 'Y'` の場合、行列 A は、 s で指定されたスケール係数によって平衡化されている。

`a` と `af` は変更されない。

fact = 'N' の場合、行列 *A* を *af* にコピーし、因子分解を実行する。

fact = 'E' の場合、必要に応じて行列 *A* を平衡化してから、*af* にコピーし、因子分解を実行する。

uplo CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また *A* をどのように因子分解するかを指定する。

uplo = 'U' の場合、配列 *a* には行列 *A* の上三角部分が格納され、*A* は $U^H U$ として因子分解される。

uplo = 'L' の場合、配列 *a* には行列 *A* の下三角部分が格納され、*A* は LL^H として因子分解される。

n INTEGER。行列 *A* の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数。 *B* の列数 ($nrhs \geq 0$)。

a, af, b, work REAL (*sposvx* の場合)
DOUBLE PRECISION (*dposvx* の場合)
COMPLEX (*cposvx* の場合)
DOUBLE COMPLEX (*zposvx* の場合)
配列 : *a(lda, *)*, *af(ldaf, *)*, *b(ldb, *)*, *work(*)*。

配列 *a* には、*uplo* の指定に従って、行列 *A* が格納される。
fact = 'F' で *equed* = 'Y' の場合、*A* は *s* のスケール係数によって平衡化され、*a* には平衡化された行列 $\text{diag}(s) * A * \text{diag}(s)$ が格納されている。*a* の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 *af* は、*fact* = 'F' の場合は入力引数になる。
この配列には、*A* のコレスキー因子分解で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。*equed* が 'N' でない場合は、*af* は、平衡化された行列 $\text{diag}(s) * A * \text{diag}(s)$ の因子分解された形式になる。*af* の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。*b* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

work()* は、ワークスペース配列。
work の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

lda INTEGER。 *a* の第 1 次元。 $lda \geq \max(1, n)$ 。

ldaf INTEGER。 *af* の第 1 次元。 $ldaf \geq \max(1, n)$ 。

ldb INTEGER。 *b* の第 1 次元。 $ldb \geq \max(1, n)$ 。

equed CHARACTER*1。 'N' または 'Y' でなければならない。
equed は、*fact* = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。
equed = 'N' の場合、平衡化は行われていない (*fact* = 'N' の場合、常に真)。
equed = 'Y' の場合、平衡化が行われ、*A* は $\text{diag}(s) * A * \text{diag}(s)$ で置き換えられている。

<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (<i>n</i>)。 配列 <i>s</i> には、 <i>A</i> のスケール係数が格納される。この配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'N' の場合、 <i>s</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'Y' の場合、 <i>s</i> の各成分は正の値でなければならない。
<i>ldx</i>	INTEGER。出力配列 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (<i>cposvx</i> の場合) DOUBLE PRECISION (<i>zposvx</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

<i>x</i>	REAL (<i>sposvx</i> の場合) DOUBLE PRECISION (<i>dposvx</i> の場合) COMPLEX (<i>cposvx</i> の場合) DOUBLE COMPLEX (<i>zposvx</i> の場合) 配列、次元は (<i>ldx</i> , *)。 <i>info</i> = 0 または <i>info</i> = <i>n</i> +1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。ただし、 <i>equed</i> = 'Y' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は $\text{diag}(s)^{-1} * X$ になる。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>a</i>	配列 <i>a</i> は、 <i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされる。
<i>af</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>af</i> は出力引数になる。この引数は終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) のコレスキー因子分解 $A = U^H U$ または $A = LL^H$ で得られた三角係数 <i>U</i> または <i>L</i> を返す。平衡化された行列の形式については、 <i>a</i> の説明を参照のこと。
<i>b</i>	$\text{diag}(s) * B$ によって上書きされる (<i>equed</i> = 'Y' の場合) 変更されない (<i>equed</i> = 'N' の場合)
<i>s</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>s</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場

合)。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond=0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info>0$ のリターンコードで示される。

<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	$fact \neq 'F'$ の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。 $info=i$ で、 $i \leq n$ の場合、次数 i の先頭の <small>小</small> 行列式 (したがって、行列 A そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 $rcond=0$ が返される。 $info=i$ で、 $i = n + 1$ の場合、 U は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

posvxx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 B を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 X を格納する。
<i>af</i>	サイズ (n, n) の行列 AF を格納する。
<i>s</i>	長さ (n) のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0_WP$ である。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N'。 $fact = 'F'$ の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equed</i>	'N' または 'Y' でなければならない。デフォルト値は 'N'。

?ppsv

圧縮形式の対称(エルミート)正定値行列 A を係数行列とする、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call sppsv(uplo, n, nrhs, ap, b, ldb, info)
call dppsv(uplo, n, nrhs, ap, b, ldb, info)
call cppsv(uplo, n, nrhs, ap, b, ldb, info)
call zppsv(uplo, n, nrhs, ap, b, ldb, info)
```

Fortran 95:

```
CALL PPSV(a, b [,uplo] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX = B$ を、 X について解く。 A は圧縮形式で格納される $n \times n$ の実対称/エルミート正定値行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

コレスキー因子分解を使用して、 A を $A = U^H U$ ($uplo = 'U'$ の場合)

または $A = LL^H$ ($uplo = 'L'$ の場合) として因子分解する。 U は上三角行列、 L は下三角行列である。次に、 A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>a</code> には行列 A の上三角部分が格納され、 A は $U^H U$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>a</code> には行列 A の下三角部分が格納され、 A は LL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>ap, b</code>	REAL (sppsv の場合) DOUBLE PRECISION (dppsv の場合) COMPLEX (cppsv の場合) DOUBLE COMPLEX (zppsv の場合) 配列: <code>ap(*)</code> , <code>b(ldb, *)</code> 。 配列 <code>ap</code> には、(<code>uplo</code> の指定に従って) 行列 A の上三角部分または下三角部分を圧縮格納形式で格納する(「 行列の格納形式 」を参照)。 <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。

配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

ldb INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

ap $info=0$ の場合、 $uplo$ の指定に従って、圧縮格納形式の A の上三角部分または下三角部分が、コレスキー係数 U または L によって上書きされる。

b 解の行列 X によって上書きされる。

$info$ INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。
 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。
 $info=i$ の場合、次数 i の先頭の Minor 行列式 (したがって、行列 A そのもの) が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppsv ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。

b サイズ $(n, nrhs)$ の行列 B を格納する。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

?ppsvx

コレスキー因子分解を使用して、圧縮形式の対称 (エルミート) 正定値行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call dppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call cppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, rWORK, INFO)
```

```
call zppsvx(FACT, uplo, N, NRHS, Ap, Afp, EQUED, s, B, LDB, X, LDX,
           rCOND, FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL PPSVX (a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]
           [,rcond] [,info])
```

説明

このルーチンは、コレスキー因子分解 $A=U^H U$ または $A=LL^H$ を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。A は圧縮形式で格納される $n \times n$ の対称 / エルミート正定値行列、行列 B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?ppsvx は、以下の手順を実行する。

1. $fact = 'E'$ の場合、連立方程式を平衡化するための実スケール係数 s を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列 A のスケーリングによって決まる。平衡化が行われる場合は、A は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされ、B は $\text{diag}(s) * B$ によって上書きされる。

2. $fact = 'N'$ または $'E'$ の場合、コレスキー分解を使用して、($fact = 'E'$ の場合は平衡化の後に) 行列 A を次のように因子分解する。

$$A = U^H U \text{ (uplo = 'U' の場合)}$$

$$A = L L^H \text{ (uplo = 'L' の場合)}$$

ここで、U は上三角行列、L は下三角行列である。

3. 先頭の $i \times i$ の主小行列式が正定値でない場合は、ルーチンは $info = i$ を返す。それ以外の場合、A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(s)$ によって行列 X を事前に乗算する。

入力パラメーター

fact CHARACTER*1. 'F'、'N'、または 'E' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。

$fact = 'F'$ の場合、開始時に、 afp に A の因子分解された形式が格納される。 $equed = 'Y'$ の場合、行列 A は、 s で指定されたスケール係数によって平衡化されている。

ap と afp は変更されない。

	<p>$fact = 'N'$ の場合、行列 A を afp にコピーし、因子分解を実行する。</p> <p>$fact = 'E'$ の場合、必要に応じて行列 A を平衡化してから、afp にコピーし、因子分解を実行する。</p>
$uplo$	<p>CHARACTER*1。'U' または 'L' でなければならない。</p> <p>A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。</p> <p>$uplo = 'U'$ の場合、配列 ap には行列 A の上三角部分が格納され、A は $U^H U$ として因子分解される。</p> <p>$uplo = 'L'$ の場合、配列 ap には行列 A の下三角部分が格納され、A は LL^H として因子分解される。</p>
n	INTEGER。行列 A の次数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
$ap, afp, b, work$	<p>REAL ($sppsvx$ の場合)</p> <p>DOUBLE PRECISION ($dppsvx$ の場合)</p> <p>COMPLEX ($cppsvx$ の場合)</p> <p>DOUBLE COMPLEX ($zppsvx$ の場合)</p> <p>配列 : $ap(*), afp(*), b(l, db, *), work(*)$。</p> <p>配列 ap には、元の対称 / エルミート行列 A の上三角部分または下三角部分が圧縮格納形式で格納される (「行列の格納形式」を参照)。$fact = 'F'$ で $equed = 'Y'$ の場合、ap には平衡化された行列 $diag(s)*A*diag(s)$ が格納されている。</p> <p>配列 afp は、$fact = 'F'$ の場合は入力引数になる。この配列には、A のコレスキー因子分解で得られた三角係数 U または L が、A と同じ格納形式で格納される。$equed$ が 'N' でない場合は、afp は、平衡化された行列 A の因子分解された形式になる。</p> <p>配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。</p> <p>$work(*)$ は、ワークスペース配列である。</p> <p>配列 ap と afp の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。$work$ の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。</p>
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
$equed$	<p>CHARACTER*1。'N' または 'Y' でなければならない。</p> <p>$equed$ は、$fact = 'F'$ の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。</p> <p>$equed = 'N'$ の場合、平衡化は行われていない ($fact = 'N'$ の場合は常に真)。</p> <p>$equed = 'Y'$ の場合、平衡化が行われ、A は $diag(s)*A*diag(s)$ で置き換えられている。</p>
s	<p>REAL (単精度の場合)</p> <p>DOUBLE PRECISION (倍精度の場合)</p> <p>配列、次元は (n)。</p>

配列 s には、 A のスケール係数が格納される。この配列は、 $fact = 'F'$ の場合にのみ入力引数になり、それ以外の場合は出力引数になる。

$equed = 'N'$ の場合、 s は使用されない。

$fact = 'F'$ で、 $equed = 'Y'$ の場合、 s の各成分は正の値でなければならない。

ldx	INTEGER。出力配列 x の第 1 次元。 $ldx \geq \max(1, n)$ 。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
$rwork$	REAL ($cppsvx$ の場合) DOUBLE PRECISION ($zppsvx$ の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

x	REAL ($sppsvx$ の場合) DOUBLE PRECISION ($dppsvx$ の場合) COMPLEX ($cppsvx$ の場合) DOUBLE COMPLEX ($zppsvx$ の場合) 配列、次元は $(ldx, *)$ 。 $info = 0$ または $info = n+1$ の場合、配列 x には、元の連立方程式の解の行列 X が格納される。ただし、 $equed = 'Y'$ の場合、 A と B は終了時に修正され、平衡化された連立方程式の解は $diag(s)^{-1} * X$ になる。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
ap	配列 ap は、 $fact = 'F'$ または $'N'$ の場合、または $fact = 'E'$ で $equed = 'N'$ の場合、終了時に変更されない。 $fact = 'E'$ で $equed = 'Y'$ の場合、 A は $diag(s) * A * diag(s)$ によって上書きされる。
afp	$fact = 'N'$ または $'E'$ の場合、 afp は出力引数になる。この引数は終了時に、元の行列 A ($fact = 'N'$ の場合) または平衡化された行列 A ($fact = 'E'$ の場合) のコレスキー因子分解 $A = U^H U$ または $A = LL^H$ で得られた三角係数 U または L を返す。平衡化された行列の形式は、 ap の説明を参照のこと。
b	$diag(s) * B$ によって上書きされる ($equed = 'Y'$ の場合) 変更されない ($equed = 'N'$ の場合)
s	この配列は、 $fact \neq 'F'$ の場合は出力引数になる。 「入力引数」セクションの s の説明を参照。
$rcond$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 A の条件数の逆数の推定値 (平衡化が行われる場合)。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。

<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する(「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、次数 <i>i</i> の先頭の <small>小行列式</small> (したがって、行列 <i>A</i> そのもの)が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ppsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>s</i>	長さ (n) のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0_WP$ である。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equed</i>	'N' または 'Y' でなければならない。デフォルト値は 'N'。

?pbsv

対称/エルミート正定値帯行列 A を係数行列とする、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call spbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call dpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call cpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
call zpbsv(uplo, n, kd, nrhs, ab, ldab, b, ldb, info)
```

Fortran 95:

```
CALL PBSV(a, b [,uplo] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX = B$ を、 X について解く。 A は $n \times n$ の対称/エルミート正定値帯行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

コレスキー因子分解を使用して、 A を $A = U^H U$ ($uplo = 'U'$ の場合)

または $A = LL^H$ ($uplo = 'L'$ の場合) として因子分解する。 U は A と同じ数の優対角成分を持つ上三角帯行列で、 L は A と同じ数の劣対角成分を持つ下三角帯行列である。次に、 A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを配列 <code>ab</code> に格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ab</code> には行列 A の上三角部分が格納され A は $U^H U$ として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>ab</code> には行列 A の下三角部分が格納され、 A は LL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>kd</code>	INTEGER。行列 A の優対角成分の数 ($uplo = 'U'$ の場合) または劣対角成分の数 ($uplo = 'L'$ の場合)。($kd \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>ab, b</code>	REAL (<code>spbsv</code> の場合) DOUBLE PRECISION (<code>dpbsv</code> の場合) COMPLEX (<code>cpbsv</code> の場合) DOUBLE COMPLEX (<code>zpbsv</code> の場合) 配列: <code>ab(ldab, *)</code> , <code>b(ldb, *)</code> 。 配列 <code>ab</code> には、 <code>uplo</code> の指定に従って、行列 A の上三角部分また

は下三角部分が帯形式で格納される（「[行列の格納形式](#)」を参照）。

ab の第 2 次元は、 $\max(1, n)$ 以上でなければならない。配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

$ldab$ INTEGER。配列 ab の第 1 次元 ($ldab \geq kd + 1$)。

ldb INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

ab $uplo$ の指定に従って、(帯形式の) A の上三角部分または下三角部分が、コレスキー係数 U または L によって、 A と同じ格納形式で上書きされる。

b 解の行列 X によって上書きされる。

$info$ INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。
 $info = i$ の場合、次数 i の先頭の小行列式(したがって、行列 A そのもの)が正定値になっていないため、因子分解を実行できず、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

`pbsv` ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 ab を意味する。サイズ $(kd+1, n)$ の配列 A を格納する。

b サイズ $(n, nrhs)$ の行列 B を格納する。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

?pbsvx

コレスキー因子分解を使用して、対称(エルミート)正定値帯行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call spbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call dpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
            LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

```
call cpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
call zpbsvx(FACT, uplo, N, kd, NRHS, Ab, ldab, Afb, ldafb, EQUED, s, B,
           LDB, X, LDX, RCOND, FERR, BERR, WORK, IWORK, INFO)
```

Fortran 95:

```
CALL PBSVX(a, b, x [,uplo] [,af] [,fact] [,equed] [,s] [,ferr] [,berr]
           [,rcond] [,info])
```

説明

このルーチンは、コレスキー因子分解 $A=U^H U$ または $A=LL^H$ を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。A は $n \times n$ の対称/エルミート正定値帯行列、行列 B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?pbsvx は、以下の手順を実行する。

1. *fact* = 'E' の場合、連立方程式を平衡化するための実スケール係数 *s* を計算する。

$$\text{diag}(s) * A * \text{diag}(s) * \text{diag}(s)^{-1} * X = \text{diag}(s) * B$$

連立方程式が平衡化されるかどうかは、行列 A のスケールリングによって決まる。平衡化が行われる場合は、A は $\text{diag}(s) * A * \text{diag}(s)$ によって上書きされ、B は $\text{diag}(s) * B$ によって上書きされる。

2. *fact* = 'N' または 'E' の場合、コレスキー分解を使用して、(*fact* = 'E' の場合は平衡化の後に) 行列 A を次のように因子分解する。

$$A = U^H U \text{ (uplo = 'U' の場合)}$$

$$A = LL^H \text{ (uplo = 'L' の場合)}$$

ここで、U は上三角帯行列、L は下三角帯行列である。

3. 先頭の $i \times i$ の主小行列式が正定値でない場合は、ルーチンは *info* = *i* を返す。それ以外の場合、A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として *info* = $n+1$ を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合は、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(s)$ によって行列 X を事前に乗算する。

入力パラメーター

fact CHARACTER*1. 'F'、'N'、または 'E' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。

fact = 'F' の場合、開始時に、*afb* に *A* の因子分解された形式が格納される。*equed* = 'Y' の場合、行列 *A* は、*s* で指定されたスケール係数によって平衡化されている。
ab と *afb* は変更されない。

fact = 'N' の場合、行列 *A* を *afb* にコピーし、因子分解を実行する。

fact = 'E' の場合、必要に応じて行列 *A* を平衡化してから、*afb* にコピーし、因子分解を実行する。

uplo CHARACTER*1。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらを格納するか、また *A* をどのように因子分解するかを指定する。

uplo = 'U' の場合、配列 *ab* には行列 *A* の上三角部分が格納され、*A* は $U^H U$ として因子分解される。
uplo = 'L' の場合、配列 *ab* には行列 *A* の下三角部分が格納され、*A* は LL^H として因子分解される。

n INTEGER。行列 *A* の次数 ($n \geq 0$)。

kd INTEGER。*A* の優対角成分または劣対角成分の数 ($kd \geq 0$)。

nrhs INTEGER。右辺の数。*B* の列数 ($nrhs \geq 0$)。

ab, afb, b, work REAL (spbsvx の場合)
DOUBLE PRECISION (dpbsvx の場合)
COMPLEX (cpbsvx の場合)
DOUBLE COMPLEX (zpbsvx の場合)
配列: *ab*(*ldab*, *), *afb*(*ldab*, *), *b*(*ldb*, *), *work*(*)。

配列 *ab* には、行列 *A* の上三角部分または下三角部分が帯形式で格納される(「[行列の格納形式](#)」を参照)。
fact = 'F' で *equed* = 'Y' の場合、*ab* には平衡化された行列 $\text{diag}(s) * A * \text{diag}(s)$ が格納されている。*ab* の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

配列 *afb* は、*fact* = 'F' の場合は入力引数になる。
この配列には、帯行列 *A* のコレスキー因子分解で得られた三角係数 *U* または *L* が、*A* と同じ格納形式で格納される。*equed* = 'Y' の場合、*afb* は、平衡化された行列 *A* の因子分解された形式になる。
afb の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 *b* には、行列 *B* が格納される。この行列の列は、連立方程式の右辺である。*b* の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

work(*) は、ワークスペース配列。
work の次元は $\max(1, 3 * n)$ (実数型の場合) または $\max(1, 2 * n)$ (複素数型の場合) 以上でなければならない。

ldab INTEGER。*ab* の第 1 次元。 $ldab \geq kd + 1$ 。

ldafb INTEGER。*afb* の第 1 次元。 $ldafb \geq kd + 1$ 。

ldb INTEGER。*b* の第 1 次元。 $ldb \geq \max(1, n)$ 。

<i>equed</i>	CHARACTER*1. 'N' または 'Y' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N' の場合、常に真)。 <i>equed</i> = 'Y' の場合、平衡化が行われ、 <i>A</i> は $\text{diag}(s)*A*\text{diag}(s)$ で置き換えられている。
<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は (<i>n</i>)。 配列 <i>s</i> には、 <i>A</i> のスケール係数が格納される。この配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'N' の場合、 <i>s</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'Y' の場合、 <i>s</i> の各成分は正の値でなければならない。
<i>ldx</i>	INTEGER。出力配列 <i>x</i> の第 1 次元。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (<i>cpbsvx</i> の場合) DOUBLE PRECISION (<i>zpbsvx</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

<i>x</i>	REAL (<i>spbsvx</i> の場合) DOUBLE PRECISION (<i>dpbsvx</i> の場合) COMPLEX (<i>cpbsvx</i> の場合) DOUBLE COMPLEX (<i>zpbsvx</i> の場合) 配列、次元は (<i>ldx</i> , *)。 <i>info</i> = 0 または <i>info</i> = <i>n</i> +1 の場合、配列 <i>x</i> には、元の連立方程式の解の行列 <i>X</i> が格納される。ただし、 <i>equed</i> = 'Y' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は $\text{diag}(s)^{-1}*X$ になる。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ab</i>	終了時に、 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>A</i> は $\text{diag}(s)*A*\text{diag}(s)$ によって上書きされる。
<i>afb</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>afb</i> は出力引数になる。この引数は終了時に、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) のコレスキー因子分解 $A=U^H U$ または $A=LL^H$ で得られた三角係数 <i>U</i> または <i>L</i> を返す。平衡化された行列の形式については、 <i>ab</i> の説明を参照のこと。
<i>b</i>	$\text{diag}(s)*B$ によって上書きされる (<i>equed</i> = 'Y' の場合) 変更されない (<i>equed</i> = 'N' の場合)

<i>s</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>s</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、 $i \leq n$ の場合、次数 <i>i</i> の先頭の <small>小行列式</small> (したがって、行列 <i>A</i> そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 <i>rcond</i> = 0 が返される。 <i>info</i> = <i>i</i> で、 $i = n + 1$ の場合、 <i>U</i> は特異でないが、 <i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 <i>rcond</i> の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

pbsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。サイズ ($kd+1, n$) の配列 <i>A</i> を格納する。
<i>b</i>	サイズ ($n, nrhs$) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ ($n, nrhs$) の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afb</i> を意味する。サイズ ($kd+1, n$) の配列 <i>AF</i> を格納する。
<i>s</i>	長さ (<i>n</i>) のベクトルを格納する。各成分のデフォルト値は、 $s(i) = 1.0_WP$ である。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

<i>fact</i>	'N'、'E'、または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、 <i>af</i> を指定しなければならない。指定しない場合はエラーが返される。
<i>equed</i>	'N' または 'Y' でなければならない。デフォルト値は 'N'。

?ptsv

対称/エルミート正定値三重対角行列 A を係数行列とする、複数の右辺を持つ連立線形方程式を解く。

構文

Fortran 77:

```
call sptsv(n, nrhs, d, e, b, ldb, info)
call dptsv(n, nrhs, d, e, b, ldb, info)
call cptsv(n, nrhs, d, e, b, ldb, info)
call zptsv(n, nrhs, d, e, b, ldb, info)
```

Fortran 95:

```
CALL PTSV(d, e, b [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX=B$ を、 X について解く。 A は $n \times n$ の対称/エルミート正定値三重対角行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

A を $A=LDL^H$ として因子分解する。次に、 A の因子分解された形式を使用して、連立方程式 $AX=B$ を解く。

入力パラメーター

<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<i>d</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, n)$ 以上。三重対角行列 A の対角成分を格納する。
<i>e, b</i>	REAL (sptsv の場合) DOUBLE PRECISION (dptsv の場合) COMPLEX (cptsv の場合) DOUBLE COMPLEX (zptsv の場合) 配列: $e(n-1)$, $b(ldb, *)$ 。 配列 e には、 A の $(n-1)$ 個の劣対角成分が格納される。

配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

ldb INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

d A の LDL^H 因子分解で得られた対角行列 D の n 個の対角成分によって上書きされる。

e A の因子分解で得られた単位二重対角係数 L の $(n-1)$ 個の劣対角成分によって上書きされる。

b 解の行列 X によって上書きされる。

$info$ INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。
 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。
 $info=i$ の場合、次数 i の先頭の小行列式(したがって、行列 A そのもの)が正定値になっていないため、解は計算されなかった。 $i=n$ でない限り、因子分解は完了していない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ptsv ルーチンのインターフェイス特有の詳細を以下に示す。

d 長さ (n) のベクトルを格納する。

e 長さ $(n-1)$ のベクトルを格納する。

b サイズ $(n, nrhs)$ の行列 B を格納する。

?ptsvx

因子分解 $A=LDL^H$ を使用して、対称(エルミート)正定値三重対角行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
           BERR, WORK, INFO)
call dptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
           BERR, WORK, INFO)
call cptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
           BERR, WORK, rWORK, INFO)
call zptsvx(FACT, N, NRHS, d, e, df, ef, B, LDB, X, LDX, RCOND, FERR,
           BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL PTSVX(d, e, b, x [,df] [,ef] [,fact] [,ferr] [,berr] [,rcond]
          [,info])
```

説明

このルーチンは、コレスキー因子分解 $A=LDL^H$ を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。A は $n \times n$ の対称/エルミート正定値三重対角行列、行列 B の列は個々の右辺、X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン `?ptsvx` は、以下の手順を実行する。

1. `fact = 'N'` の場合、行列 A を $A = LDL^H$ として因子分解する。L は単位下二重対角行列で、D は対角行列である。この因子分解の形式は、 $A = U^H D U$ とみなせる。
2. 先頭の $i \times i$ の主小行列式が正定値でない場合は、ルーチンは `info = i` を返す。それ以外の場合、A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として `info = n + 1` を返すが、後で説明するようにルーチンは続行され、X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

<code>fact</code>	CHARACTER*1。'F' または 'N' でなければならない。 ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。 <code>fact = 'F'</code> の場合、開始時に、 <code>df</code> と <code>ef</code> に A の因子分解された形式が格納される。配列 <code>d</code> 、 <code>e</code> 、 <code>df</code> 、 <code>ef</code> は変更されない。 <code>fact = 'N'</code> の場合、行列 A を <code>df</code> と <code>ef</code> にコピーし、因子分解を実行する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。B の列数 ($nrhs \geq 0$)。
<code>d,df,rwork</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: <code>d(n)</code> 、 <code>df(n)</code> 、 <code>rwork(n)</code> 。 配列 <code>d</code> には、三重対角行列 A の n 個の対角成分が格納される。 配列 <code>df</code> は、 <code>fact = 'F'</code> の場合は入力引数になる。この配列には、開始時に、A の LDL^H 因子分解で得られた対角行列 D の n 個の対角成分が格納される。 配列 <code>rwork</code> は、複素数型でのみ使用されるワークスペース配列である。

e, ef, b, work REAL (sptsvx の場合)
DOUBLE PRECISION (dptsvx の場合)
COMPLEX (cptsvx の場合)
DOUBLE COMPLEX (zptsvx の場合)
配列: $e(n-1)$, $ef(n-1)$, $b(ldb,*)$, $work(*)$ 。
配列 e には、三重対角行列 A の $(n-1)$ 個の劣対角成分が格納される。

配列 ef は、 $fact = 'F'$ の場合は入力引数になる。この配列には、開始時に、 A の LDL^H 因子分解で得られた単位二重対角係数 L の $(n-1)$ 個の劣対角成分が格納される。

配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

配列 $work$ はワークスペース配列である。 $work$ の次元は、 $2*n$ (実数型の場合) または n (複素数型の場合) 以上でなければならない。

ldb INTEGER。 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

ldx INTEGER。 x のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。

出力パラメーター

x REAL (sptsvx の場合)
DOUBLE PRECISION (dptsvx の場合)
COMPLEX (cptsvx の場合)
DOUBLE COMPLEX (zptsvx の場合)
配列、次元は $(ldx, *)$ 。

$info = 0$ または $info = n+1$ の場合、配列 x には、連立方程式の解の行列 X が格納される。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

df, ef これらの配列は、 $fact = 'N'$ の場合は出力引数になる。「入力引数」セクションの df , ef の説明を参照。

rcond REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
平衡化後の行列 A の条件数の逆数の推定値 (平衡化が行われる場合)。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。

ferr, berr REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。

info INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ で、 $i \leq n$ の場合、次数 i の先頭の最小行列式 (したがって、行列 A そのもの) が正定値になっていないため、因子分解を実行できず、解と誤差範囲は計算できなかった。 $rcond = 0$ が返される。

$info = i$ で、 $i = n + 1$ の場合、 U は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ptsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>b</i>	サイズ (<i>n, nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n, nrhs</i>) の行列 <i>X</i> を格納する。
<i>df</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>ef</i>	長さ (<i>n-1</i>) のベクトルを格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

?sysv

実対称または複素対称行列 *A* を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call ssysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call dsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call csysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zsysv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

Fortran 95:

```
CALL SYSV(a, b [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX=B$ を、 X について解く。 A は $n \times n$ の対称行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

対角ピボット演算法を使用して、 A を $A=UDU^T$ または $A=LDL^T$ として因子分解する。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。

次に、 A の因子分解された形式を使用して、連立方程式 $AX=B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>a</code> には行列 A の上三角部分が格納され、 A は UDU^T として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>a</code> には行列 A の下三角部分が格納され、 A は LDL^T として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>a, b, work</code>	REAL (<code>ssysv</code> の場合) DOUBLE PRECISION (<code>dsysv</code> の場合) COMPLEX (<code>csysv</code> の場合) DOUBLE COMPLEX (<code>zsysv</code> の場合) 配列: <code>a(lda, *)</code> , <code>b(ldb, *)</code> , <code>work(lwork)</code> 。 配列 <code>a</code> には、対称行列 A の上三角部分または下三角部分が格納される (<code>uplo</code> を参照)。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 <code>b</code> には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 <code>b</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $lda \geq \max(1, n)$ 。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ、 $lwork \geq 1$ 。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエンタリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。 <code>lwork</code> の詳細と推奨値は、以下の「アプリケーション・ノート」を参照。

出力パラメーター

<code>a</code>	<code>info = 0</code> の場合、 <code>a</code> は <code>?sytrf</code> による A の因子分解で得られたブロック対角行列 D と、係数 U (または L) の計算に使用された乗数によって上書きされる。
----------------	---

<i>b</i>	<i>info</i> = 0 の場合、 <i>b</i> は解の行列 <i>X</i> によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?sytrf によって決まる、交換操作と <i>D</i> のブロック構造の詳細を格納する。 <i>ipiv</i> (<i>i</i>) = <i>k</i> > 0 の場合、 <i>d</i> _{<i>i</i><i>i</i>} は 1×1 の対角ブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列に交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> -1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> +1) 列に 2×2 のブロックが作成される。 <i>A</i> の (<i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d</i> _{<i>i</i><i>i</i>} は 0 である。因子分解は完了したが、 <i>D</i> が完全に特異であるため、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sysv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* = *n***blocksize* に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスに必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* = -1 に設定する。この設定は、ワークスペースのクエリーとみなされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリー *work*(1) として返す。[xerbla](#) は、*lwork* に関するエラーメッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?sysvx

対角ピボット演算による因子分解を使用して、実対称または複素対称行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call ssysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, LWORK, IWORK, INFO)
call dsysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, LWORK, IWORK, INFO)
call csysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, LWORK, rWORK, INFO)
call zsysvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
           RCOND, FERR, BERR, WORK, LWORK, rWORK, INFO)
```

Fortran 95:

```
CALL SYSVX(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
           [,info])
```

説明

このルーチンは、対角ピボット演算による因子分解を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。 A は $n \times n$ の対称行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?sysvx は、以下の手順を実行する。

1. $fact = 'N'$ の場合、対角ピボット演算法を使用して、行列 A を因子分解する。因子分解の形式は、 $A = U D U^T$ または $A = L D L^T$ である。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。
2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

fact CHARACTER*1. 'F' または 'N' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。

$fact = 'F'$ の場合、開始時に、 af と $ipiv$ に A の因子分解された形式が格納される。配列 a 、 af 、 $ipiv$ は変更されない。

$fact = 'N'$ の場合、行列 A を af にコピーし、因子分解を実行する。

$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 $uplo = 'U'$ の場合、配列 a には対称行列 A の上三角部分が格納され、 A は UDU^T として因子分解される。 $uplo = 'L'$ の場合、配列 a には対称行列 A の下三角部分が格納され、 A は LDL^T として因子分解される。
n	INTEGER。行列 A の次数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
$a, af, b, work$	REAL (ssysvx の場合) DOUBLE PRECISION (dsysvx の場合) COMPLEX (csysvx の場合) DOUBLE COMPLEX (zsysvx の場合) 配列: $a(lda, *)$, $af(ldaf, *)$, $b(ldb, *)$, $work(*)$ 。 配列 a には、対称行列 A の上三角部分または下三角部分が格納される ($uplo$ を参照)。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 af は、 $fact = 'F'$ の場合は入力引数になる。この配列には、 ?sytrf による因子分解 $A = UDU^T$ または $A = LDL^T$ で得られた、ブロック対角行列 D と、係数 U または L の計算に使用された乗数が格納される。 af の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work(*)$ は、次元 ($lwork$) のワークスペース配列。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
$ldaf$	INTEGER。 af の第 1 次元。 $ldaf \geq \max(1, n)$ 。
ldb	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
$ipiv$	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 $ipiv$ は、 $fact = 'F'$ の場合は入力引数になる。 この配列には、 ?sytrf によって決まる、交換操作と D のブロック構造の詳細が格納される。 $ipiv(i) = k > 0$ の場合、 d_{ii} は 1×1 の対角ブロックである。 A の i 番目の行と列は、 k 番目の行と列に交換される。

$uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。 A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。

$uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。 A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。

<i>ldx</i>	INTEGER。出力配列 x のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。work 配列のサイズ。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエントリーとして返し、xerbla は lwork に関するエラーメッセージを生成しない。lwork の詳細と推奨値は、以下の「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (csysvx の場合) DOUBLE PRECISION (zsysvx の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

x	REAL (ssysvx の場合) DOUBLE PRECISION (dsysvx の場合) COMPLEX (csysvx の場合) DOUBLE COMPLEX (zsysvx の場合) 配列、次元は ($ldx, *$)。 $info = 0$ または $info = n+1$ の場合、配列 x には、連立方程式の解の行列 X が格納される。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>af, ipiv</i>	これらの配列は、 $fact = 'N'$ の場合は出力引数になる。「入力引数」セクションの <i>af, ipiv</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 A の条件数の逆数の推定値。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>work(1)</i>	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。

info INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* で、 $i \leq n$ の場合、 d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角行列 *D* が完全に特異であるため、解と誤差範囲は計算できなかった。*rcond* = 0 が返される。
info = *i* で、 $i = n + 1$ の場合、*D* は特異でないが、*rcond* がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

sysvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>X</i> を格納する。
<i>af</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>ferr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>berr</i>	長さ (<i>nrhs</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

アプリケーション・ノート

lwork は、実数型の場合は $3*n$ 以上、複素数型の場合は $2*n$ 以上でなければならない。パフォーマンスを向上させるには、 $lwork = n*blocksize$ に設定する。*blocksize* は、*?sytrf* に最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* = -1 に設定する。この設定は、ワークスペースのクエリーとみなされる。ルーチンは、*work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリー *work*(1) として返す。[xerbla](#) は、*lwork* に関するエラーメッセージを生成しない。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?hesv

エルミート行列 A を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call chesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
call zhesv(uplo, n, nrhs, a, lda, ipiv, b, ldb, work, lwork, info)
```

Fortran 95:

```
CALL HESV(a, b [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX=B$ を、 X について解く。 A は $n \times n$ の対称行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

対角ピボット演算法を使用して、 A を $A=UDU^H$ または $A=LDL^H$ として因子分解する。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。

次に、 A の因子分解された形式を使用して、連立方程式 $AX=B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 a には行列 A の上三角部分が格納され、 A は UDU^H として因子分解される。 <code>uplo = 'L'</code> の場合、配列 a には行列 A の下三角部分が格納され、 A は LDL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>a, b, work</code>	COMPLEX (chesv の場合) DOUBLE COMPLEX (zhesv の場合) 配列: $a(lda, *)$, $b(ldb, *)$, $work(lwork)$ 。 配列 a には、エルミート行列 A の上三角部分または下三角部分が格納される (<code>uplo</code> を参照)。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。

<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ ($lwork \geq 1$)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の詳細と推奨値は、以下の「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	<i>info</i> = 0 の場合、 <i>a</i> は ?hetrf による <i>A</i> の因子分解で得られたブロック対角行列 <i>D</i> と、係数 <i>U</i> (または <i>L</i>) の計算に使用された乗数によって上書きされる。
<i>b</i>	<i>info</i> = 0 の場合、 <i>b</i> は解の行列 <i>X</i> によって上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?hetrf によって決まる、交換操作と <i>D</i> のブロック構造の詳細を格納する。 <i>ipiv</i> (<i>i</i>) = <i>k</i> > 0 の場合、 <i>d</i> _{<i>ii</i>} は 1 × 1 の対角ブロックである。 <i>A</i> の <i>i</i> 番目の行と列は、 <i>k</i> 番目の行と列に交換される。 <i>uplo</i> = 'U' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> -1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> -1) 列に 2 × 2 のブロックが作成される。 <i>A</i> の (<i>i</i> -1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。 <i>uplo</i> = 'L' で <i>ipiv</i> (<i>i</i>) = <i>ipiv</i> (<i>i</i> +1) = - <i>m</i> < 0 の場合、 <i>D</i> の <i>i</i> 行 (<i>i</i> +1) 列に 2 × 2 のブロックが作成される。 <i>A</i> の (<i>i</i> +1) 番目の行と列は、 <i>m</i> 番目の行と列と交換される。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>d</i> _{<i>ii</i>} は 0 である。因子分解は完了したが、 <i>D</i> が完全に特異であるため、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hesv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>nrhs</i>) の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスに必要な値で、マシンによって異なる（通常は 16 ~ 64）。

必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork = -1$ に設定する。この設定は、ワークスペースのクエリーとみなされる。ルーチンは、 $work$ 配列の最適なサイズだけを計算し、この値を $work$ 配列の最初のエントリー $work(1)$ として返す。[xerbla](#) は、 $lwork$ に関するエラーメッセージを生成しない。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

?hesvx

対角ピボット演算による因子分解を使用して、エルミート行列 A を係数行列とする複素数連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call chesvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, LWORK, rWORK, INFO)
call zhesvx(FACT, uplo, N, NRHS, A, lda, Af, ldaf, ipiv, B, LDB, X, LDX,
            RCOND, FERR, BERR, WORK, LWORK, rWORK, INFO)
```

Fortran 95:

```
CALL HESVX(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
           [,info])
```

説明

このルーチンは、対角ピボット演算による因子分解を使用して、複素数連立線形方程式 $AX=B$ の解を計算する。 A は $n \times n$ のエルミート行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hesvx は、以下の手順を実行する。

1. $fact = 'N'$ の場合、対角ピボット演算法を使用して、行列 A を因子分解する。因子分解の形式は、 $A = UDU^H$ または $A = LDL^H$ である。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。
2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。

4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

<i>fact</i>	CHARACTER*1。'F' または 'N' でなければならない。 ルーチンの開始時に行列 <i>A</i> の因子分解された形式が与えられるかどうかを指定する。 <i>fact</i> = 'F' の場合、開始時に、 <i>af</i> と <i>ipiv</i> に <i>A</i> の因子分解された形式が格納される。配列 <i>a</i> 、 <i>af</i> 、および <i>ipiv</i> は変更されない。 <i>fact</i> = 'N' の場合、行列 <i>A</i> を <i>af</i> にコピーし、因子分解を実行する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>A</i> の上三角部分と下三角部分のどちらを格納するか、また <i>A</i> をどのように因子分解するかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> にはエルミート行列 <i>A</i> の上三角部分が格納され、 <i>A</i> は UDU^H として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> にはエルミート行列 <i>A</i> の下三角部分が格納され、 <i>A</i> は LDL^H として因子分解される。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。 <i>B</i> の列数 ($nrhs \geq 0$)。
<i>a, af, b, work</i>	COMPLEX (<i>chesvx</i> の場合) DOUBLE COMPLEX (<i>zhesvx</i> の場合) 配列: <i>a(lda, *)</i> , <i>af(ldaf, *)</i> , <i>b(ldb, *)</i> , <i>work(*)</i> 。 配列 <i>a</i> には、エルミート行列 <i>A</i> の上三角部分または下三角部分が格納される (<i>uplo</i> を参照)。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>af</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この配列には、 ?hetrf による因子分解 $A = UDU^H$ または $A = LDL^H$ で得られた、ブロック対角行列 <i>D</i> と、係数 <i>U</i> または <i>L</i> の計算に使用された乗数が格納される。 <i>af</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 配列 <i>b</i> には、行列 <i>B</i> が格納される。この行列の列は、連立方程式の右辺である。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work(*)</i> は、次元 (<i>lwork</i>) のワークスペース配列。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldaf</i>	INTEGER。 <i>af</i> の第 1 次元。 $ldaf \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 配列 <i>ipiv</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。 この配列には、 ?hetrf によって決まる、交換操作と <i>D</i> のブロッ

ク構造の詳細が格納される。

$ipiv(i) = k > 0$ の場合、 d_{ii} は 1×1 の対角ブロックである。A の i 番目の行と列は、 k 番目の行と列に交換される。

$uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。

$uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。

<code>ldx</code>	INTEGER。出力配列 x のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<code>lwork</code>	INTEGER。work 配列のサイズ。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエントリーとして返し、xerbla は lwork に関するエラーメッセージを生成しない。lwork の詳細と推奨値は、以下の「アプリケーション・ノート」を参照。
<code>rwork</code>	REAL (chesvx の場合) DOUBLE PRECISION (zhesvx の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<code>x</code>	COMPLEX (chesvx の場合) DOUBLE COMPLEX (zhesvx の場合) 配列、次元は ($ldx, *$)。 $info = 0$ または $info = n+1$ の場合、配列 x には、連立方程式の解の行列 X が格納される。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>af, ipiv</code>	これらの配列は、 $fact = 'N'$ の場合は出力引数になる。「入力引数」セクションの <code>af, ipiv</code> の説明を参照。
<code>rcond</code>	REAL (chesvx の場合) DOUBLE PRECISION (zhesvx の場合) 行列 A の条件数の逆数の推定値。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。
<code>ferr, berr</code>	REAL (chesvx の場合) DOUBLE PRECISION (zhesvx の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<code>work(1)</code>	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な lwork の最小値が <code>work(1)</code> に格納される。これ以降の実行には、この lwork の値を使用する。
<code>info</code>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ で、 $i \leq n$ の場合、 d_{ii} は完全に 0 である。因子分解は

完了したが、ブロック対角行列 D が完全に特異であるため、解と誤差範囲は計算できなかった。 $rcond=0$ が返される。
 $info=i$ で、 $i=n+1$ の場合、 D は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。
 しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hesvxx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 B を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 X を格納する。
<i>af</i>	サイズ (n, n) の行列 AF を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 $fact = 'F'$ の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

アプリケーション・ノート

lwork の値は、 $2*n$ 以上でなければならない。パフォーマンスを向上させるには、 $lwork = n*blocksize$ に設定する。 *blocksize* は、 `?hetrf` に最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork=-1$ に設定する。この設定は、ワークスペースのクエリーとみなされる。ルーチンは、 *work* 配列の最適なサイズだけを計算し、この値を *work* 配列の最初のエントリー $work(1)$ として返す。 [xerbla](#) は、 *lwork* に関するエラーメッセージを生成しない。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

?spsv

圧縮形式で格納される実対称または複素対称行列 A を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call sspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call dspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call cspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zspsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL SPSV(a, b [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、実数または複素数連立線形方程式 $AX=B$ を、 X について解く。 A は圧縮形式で格納される $n \times n$ の対称行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

対角ピボット演算法を使用して、 A を $A=UDU^T$ または $A=LDL^T$ として因子分解する。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。

次に、 A の因子分解された形式を使用して、連立方程式 $AX=B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には行列 A の上三角部分が格納され、 A は UDU^T として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には行列 A の下三角部分が格納される。 A は LDL^T として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>ap, b</code>	REAL (sspsv の場合) DOUBLE PRECISION (dspsv の場合) COMPLEX (cspsv の場合) DOUBLE COMPLEX (zspsv の場合) 配列: <code>ap(*), b(ldb,*)</code>

ap の次元は、 $\max(1, n(n+1)/2)$ 以上でなければならない。
 配列 ap には、 $uplo$ の指定に従って、係数 U または L が圧縮格納形式で格納される (「[行列の格納形式](#)」を参照)。
 配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。
 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。

ldb INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

ap [?spturf](#) による A の因子分解で得られた、ブロック対角行列 D と、係数 U (または L) の計算に使用された乗数が、 A と同じ格納形式で、圧縮形式の三角行列として格納される。

b $info=0$ の場合、 b は解の行列 X によって上書きされる。

$ipiv$ INTEGER。
 配列、次元は $\max(1, n)$ 以上。
[?spturf](#) によって決まる、交換操作と D のブロック構造の詳細を格納する B
 $ipiv(i) = k > 0$ の場合、 d_{ii} は 1×1 のブロックである。 A の i 番目の行と列は、 k 番目の行と列と交換される。

$uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。 A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。

$uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。 A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。

$info$ INTEGER。 $info=0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。
 $info = i$ の場合、 d_{ii} は 0 である。因子分解は完了したが、 D が完全に特異であるため、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spsv ルーチンのインターフェイス特有の詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 ap を意味する。サイズ $(n*(n+1)/2)$ の配列 A を格納する。

b サイズ $(n, nrhs)$ の行列 B を格納する。

$ipiv$ 長さ (n) のベクトルを格納する。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

?spsvx

対角ピボット演算による因子分解を使用して、圧縮形式で格納される実対称または複素対称行列 A を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call sspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, iWORK, INFO)
call dspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, iWORK, INFO)
call cspsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
call zpsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL SPSVX(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
           [,info])
```

説明

このルーチンは、対角ピボット演算による因子分解を使用して、実数または複素数連立線形方程式 $AX=B$ の解を計算する。 A は圧縮形式で格納される $n \times n$ の対称行列、行列 B の列は個々の右辺、 X の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?spsvx は、以下の手順を実行する。

1. $fact = 'N'$ の場合、対角ピボット演算法を使用して、行列 A を因子分解する。因子分解の形式は、 $A = U D U^T$ または $A = L D L^T$ である。 U (または L) は置換行列と単位上(下)三角行列の積で、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。
2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

fact CHARACTER*1. 'F' または 'N' でなければならない。

ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。

$fact = 'F'$ の場合、開始時に、 afp と $ipiv$ に A の因子分解された形式が格納される。配列 ap 、 afp 、 $ipiv$ は変更されない。

$fact = 'N'$ の場合、行列 A を afp にコピーし、因子分解を実行する。

$uplo$

CHARACTER*1。'U' または 'L' でなければならない。

A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。

$uplo = 'U'$ の場合、配列 ap には対称行列 A の上三角部分が格納され、 A は UDU^T として因子分解される。

$uplo = 'L'$ の場合、配列 ap には対称行列 A の下三角部分が格納され、 A は LDL^T として因子分解される。

n

INTEGER。行列 A の次数 ($n \geq 0$)。

$nrhs$

INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。

$ap, afp, b, work$

REAL (sspsvx の場合)

DOUBLE PRECISION (dpsvx の場合)

COMPLEX (cspvx の場合)

DOUBLE COMPLEX (zspvx の場合)

配列: $ap(*)$, $afp(*)$, $b(ldb, *)$, $work(*)$ 。

配列 ap には、対称行列 A の上三角部分または下三角部分が圧縮格納形式で格納される ([「行列の格納形式」](#) を参照)。

配列 afp は、 $fact = 'F'$ の場合は入力引数になる。この配列には、[?sptrf](#) による因子分解 $A = UDU^T$ または $A = LDL^T$ で得られた、ブロック対角行列 D と、係数 U または L の計算に使用された乗数が、 A と同じ格納形式で格納される。

配列 b には、行列 B が格納される。この行列の列は、連立方程式の右辺である。

$work(*)$ は、ワークスペース配列である。

配列 ap と afp の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work$ の次元は $\max(1, 3*n)$ (実数型の場合) または $\max(1, 2*n)$ (複素数型の場合) 以上でなければならない。

ldb

INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

$ipiv$

INTEGER。

配列、次元は $\max(1, n)$ 以上。

配列 $ipiv$ は、 $fact = 'F'$ の場合は入力引数になる。

この配列には、[?sptrf](#) によって決まる、交換操作と D のブロック構造の詳細が格納される。

$ipiv(i) = k > 0$ の場合、 d_{ii} は 1×1 の対角ブロックである。 A の i 番目の行と列は、 k 番目の行と列に交換される。

$uplo = 'U'$ で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。 A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。

$uplo = 'L'$ で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。 A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。

<i>ldx</i>	INTEGER。出力配列 x のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (<i>cspsvx</i> の場合) DOUBLE PRECISION (<i>zspsvx</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

x	REAL (<i>sspsvx</i> の場合) DOUBLE PRECISION (<i>dspsvx</i> の場合) COMPLEX (<i>cspsvx</i> の場合) DOUBLE COMPLEX (<i>zspsvx</i> の場合) 配列、次元は ($ldx, *$)。 $info = 0$ または $info = n+1$ の場合、配列 x には、連立方程式の解の行列 X が格納される。 x の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>afp, ipiv</i>	これらの配列は、 $fact = 'N'$ の場合は出力引数になる。「入力引数」セクションの <i>afp, ipiv</i> の説明を参照。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 行列 A の条件数の逆数の推定値。 $rcond$ がマシンの精度より小さい場合 (特に、 $rcond = 0$ の場合) は、行列は有効な精度で特異になる。この条件は、 $info > 0$ のリターンコードで示される。
<i>ferr, berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ で、 $i \leq n$ の場合、 d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角行列 D が完全に特異であるため、解と誤差範囲は計算できなかった。 $rcond = 0$ が返される。 $info = i$ で、 $i = n + 1$ の場合、 D は特異でないが、 $rcond$ がマシンの精度より小さいため、行列は有効な精度で特異になる。しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、 $rcond$ の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

spsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>afp</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

?hpsv

圧縮形式で格納されるエルミート行列 *A* を係数行列とする、複数の右辺を持つ連立線形方程式の解を計算する。

構文

Fortran 77:

```
call chpsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
call zhpsv(uplo, n, nrhs, ap, ipiv, b, ldb, info)
```

Fortran 95:

```
CALL HPSV(a, b [,uplo] [,ipiv] [,info])
```

説明

このルーチンは、連立線形方程式 $AX=B$ を *X* について解く。*A* は圧縮形式で格納される $n \times n$ のエルミート行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

対角ピボット演算法を使用して、*A* を $A=UDU^H$ または $A=LDL^H$ として因子分解する。*U* (または *L*) は置換行列と単位上(下)三角行列の積で、*D* は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。

次に、 A の因子分解された形式を使用して、連立方程式 $AX = B$ を解く。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <code>uplo = 'U'</code> の場合、配列 <code>ap</code> には行列 A の上三角部分が格納され、 A は UDU^H として因子分解される。 <code>uplo = 'L'</code> の場合、配列 <code>ap</code> には行列 A の下三角部分が格納され、 A は LDL^H として因子分解される。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>nrhs</code>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<code>ap, b</code>	COMPLEX (<code>chpsv</code> の場合) DOUBLE COMPLEX (<code>zhpsv</code> の場合) 配列 : <code>ap(*), b(ldb,*)</code> <code>ap</code> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 配列 <code>ap</code> には、 <code>uplo</code> の指定に従って、係数 U または L が圧縮格納形式で格納される (「 行列の格納形式 」を参照)。 配列 <code>b</code> には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 <code>b</code> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<code>ldb</code>	INTEGER。 <code>b</code> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>ap</code>	?hptrf による A の因子分解で得られた、ブロック対角行列 D と、係数 U (または L) の計算に使用された乗数が、 A と同じ格納形式で、圧縮形式の三角行列として格納される。
<code>b</code>	<code>info = 0</code> の場合、 <code>b</code> は解の行列 X によって上書きされる。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ?hptrf によって決まる、交換操作と D のブロック構造の詳細を格納する。 <code>ipiv(i) = k > 0</code> の場合、 d_{ii} は 1×1 のブロックである。 A の i 番目の行と列は、 k 番目の行と列と交換される。 <code>uplo = 'U'</code> で <code>ipiv(i) = ipiv(i-1) = -m < 0</code> の場合、 D の i 行 ($i-1$) 列に 2×2 のブロックが作成される。 A の ($i-1$) 番目の行と列は、 m 番目の行と列と交換される。 <code>uplo = 'L'</code> で <code>ipiv(i) = ipiv(i+1) = -m < 0</code> の場合、 D の i 行 ($i+1$) 列に 2×2 のブロックが作成される。 A の ($i+1$) 番目の行と列は、 m 番目の行と列と交換される。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。 <code>info = i</code> の場合、 d_{ii} は 0 である。因子分解は完了したが、 D が完全に特異であるため、解は計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hpsv ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

?hpsvx

対角ピボット演算による因子分解を使用して、圧縮形式で格納されるエルミート行列 *A* を係数行列とする連立線形方程式の解を計算し、解の誤差範囲を示す。

構文

Fortran 77:

```
call chpsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
call zhpsvx(FACT, uplo, N, NRHS, Ap, Afp, ipiv, B, LDB, X, LDX, RCOND,
            FERR, BERR, WORK, rWORK, INFO)
```

Fortran 95:

```
CALL HPSVX(a, b, x [,uplo] [,af] [,ipiv] [,fact] [,ferr] [,berr] [,rcond]
           [,info])
```

説明

このルーチンは、対角ピボット演算による因子分解を使用して、複素数連立線形方程式 $AX=B$ の解を計算する。*A* は圧縮形式で格納される $n \times n$ のエルミート行列、行列 *B* の列は個々の右辺、*X* の列はそれに対応する解である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

ルーチン ?hpsvx は、以下の手順を実行する。

1. *fact* = 'N' の場合、対角ピボット演算法を使用して、行列 *A* を因子分解する。因子分解の形式は、 $A = U D U^H$ または $A = L D L^H$ である。*U* (または *L*) は置換行列と単位上 (下) 三角行列の積で、*D* は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。

2. $d_{i,i} = 0$ の場合、つまり D が完全に特異である場合は、ルーチンは $info = i$ を返す。それ以外の場合、 A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合は、警告として $info = n + 1$ を返すが、後で説明するようにルーチンは続行され、 X について解を算出し、誤差範囲を計算する。
3. A の因子分解された形式を使用して、連立方程式を X について解く。
4. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

入力パラメーター

<i>fact</i>	CHARACTER*1。'F' または 'N' でなければならない。 ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうかを指定する。 <i>fact</i> = 'F' の場合、開始時に、 <i>afp</i> と <i>ipiv</i> に A の因子分解された形式が格納される。配列 <i>ap</i> 、 <i>afp</i> 、 <i>ipiv</i> は変更されない。 <i>fact</i> = 'N' の場合、行列 A を <i>afp</i> にコピーし、因子分解を実行する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらを格納するか、また A をどのように因子分解するかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> にはエルミート行列 A の上三角部分が格納され、 A は UDU^H として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> にはエルミート行列 A の下三角部分が格納され、 A は LDL^H として因子分解される。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<i>ap, afp, b, work</i>	COMPLEX (chpsvx の場合) DOUBLE COMPLEX (zhpsvx の場合) 配列: <i>ap</i> (*), <i>afp</i> (*), <i>b</i> (ldb, *), <i>work</i> (*)。 配列 <i>ap</i> には、エルミート行列 A の上三角部分または下三角部分が圧縮格納形式で格納される(「 行列の格納形式 」を参照)。 配列 <i>afp</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この配列には、 ?hptrf による因子分解 $A = UDU^H$ または $A = LDL^H$ で得られた、ブロック対角行列 D と、係数 U または L の計算に使用された乗数が、 A と同じ格納形式で格納される。 配列 <i>b</i> には、行列 B が格納される。この行列の列は、連立方程式の右辺である。 <i>work</i> (*) は、ワークスペース配列である。 配列 <i>ap</i> と <i>afp</i> の次元は $\max(1, n(n+1)/2)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> の次元は $\max(1, 2*n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。

<i>ipiv</i>	<p>INTEGER。</p> <p>配列、次元は $\max(1, n)$ 以上。</p> <p>配列 <i>ipiv</i> は、<i>fact</i> = 'F' の場合は入力引数になる。</p> <p>この配列には、zhptrf によって決まる、交換操作と <i>D</i> のブロック構造の詳細が格納される。</p> <p>$ipiv(i) = k > 0$ の場合、d_{ii} は 1×1 の対角ブロックである。<i>A</i> の <i>i</i> 番目の行と列は、<i>k</i> 番目の行と列に交換される。</p> <p><i>uplo</i> = 'U' で $ipiv(i) = ipiv(i-1) = -m < 0$ の場合、<i>D</i> の <i>i</i> 行 (<i>i-1</i>) 列に 2×2 のブロックが作成される。<i>A</i> の (<i>i-1</i>) 番目の行と列は、<i>m</i> 番目の行と列と交換される。</p> <p><i>uplo</i> = 'L' で $ipiv(i) = ipiv(i+1) = -m < 0$ の場合、<i>D</i> の <i>i</i> 行 (<i>i+1</i>) 列に 2×2 のブロックが作成される。<i>A</i> の (<i>i+1</i>) 番目の行と列は、<i>m</i> 番目の行と列と交換される。</p>
<i>ldx</i>	<p>INTEGER。出力配列 <i>x</i> のリーディング・ディメンジョン。</p> <p>$ldx \geq \max(1, n)$。</p>
<i>rwork</i>	<p>REAL (<i>chpsvx</i> の場合)</p> <p>DOUBLE PRECISION (<i>zhpsvx</i> の場合)</p> <p>ワークスペース配列、次元は $\max(1, n)$ 以上。</p>

出力パラメーター

<i>x</i>	<p>COMPLEX (<i>chpsvx</i> の場合)</p> <p>DOUBLE COMPLEX (<i>zhpsvx</i> の場合)</p> <p>配列、次元は (<i>ldx</i>, *)。</p> <p><i>info</i> = 0 または <i>info</i> = <i>n</i>+1 の場合、配列 <i>x</i> には、連立方程式の解の行列 <i>X</i> が格納される。<i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。</p>
<i>afp, ipiv</i>	<p>これらの配列は、<i>fact</i> = 'N' の場合は出力引数になる。</p> <p>「入力引数」セクションの <i>afp, ipiv</i> の説明を参照。</p>
<i>rcond</i>	<p>REAL (<i>chpsvx</i> の場合)</p> <p>DOUBLE PRECISION (<i>zhpsvx</i> の場合)</p> <p>行列 <i>A</i> の条件数の逆数の推定値。<i>rcond</i> がマシンの精度より小さい場合 (特に、<i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、<i>info</i> > 0 のリターンコードで示される。</p>
<i>ferr, berr</i>	<p>REAL (<i>chpsvx</i> の場合)</p> <p>DOUBLE PRECISION (<i>zhpsvx</i> の場合)</p> <p>配列、次元は $\max(1, nrhs)$ 以上。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。</p>
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。</p> <p><i>info</i> = <i>i</i> で、$i \leq n$ の場合、d_{ii} は完全に 0 である。因子分解は完了したが、ブロック対角行列 <i>D</i> が完全に特異であるため、解と誤差範囲は計算できなかった。<i>rcond</i> = 0 が返される。</p> <p><i>info</i> = <i>i</i> で、$i = n + 1$ の場合、<i>D</i> は特異でないが、<i>rcond</i> がマシンの精度より小さいため、行列は有効な精度で特異になる。</p>

しかし、この場合は、解と誤差範囲が計算される。これは、計算された解の精度が、*rcond* の値から想定される精度より高くなる場合があるためである。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または再構築可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

hpsvx ルーチンのインターフェイス特有の詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	サイズ $(n, nrhs)$ の行列 <i>B</i> を格納する。
<i>x</i>	サイズ $(n, nrhs)$ の行列 <i>X</i> を格納する。
<i>af</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>AF</i> を格納する。
<i>ipiv</i>	長さ (n) のベクトルを格納する。
<i>ferr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>berr</i>	長さ $(nrhs)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>fact</i>	'N' または 'F' でなければならない。デフォルト値は 'N'。 <i>fact</i> = 'F' の場合、引数 <i>af</i> および <i>ipiv</i> の両方を指定しなければならない。指定しない場合は、エラーが返される。

LAPACK ルーチン： 最小二乗問題および 固有値問題

4

この章では、線形の最小二乗問題、固有値および特異値問題の解決や、それらに関連する一連の計算タスクを実行するための、LAPACK パッケージからインテル[®] マス・カーネル・ライブラリーに組み込まれているルーチンについて説明する。

この章の各セクションで、LAPACK の[計算ルーチン](#)と[ドライバルーチン](#)について説明する。LAPACK ルーチンの完全なリファレンスと関連情報は、[\[LUG\]](#) を参照。

最小二乗問題：典型的な最小二乗問題とは、行列 A とベクトル b が与えられたときに、二乗和 $\sum_i ((Ax)_i - b_i)^2$ を最小にするベクトル x を見つける、または 2- ノルム $\|Ax - b\|_2$ を最小にするベクトル x を見つけることである。

通常、 A は $m \times n$ の行列 ($m \geq n$) で、 $\text{rank}(A) = n$ である。また、この問題は、*優決定* の連立 1 次方程式 (未知数より方程式の個数の方が多い) に対して *最小二乗解* を見つけることとも言える。この問題を解くには、行列 A の *QR* 因子分解 ([「QR 因子分解」](#) を参照) を使用する。

$m < n$ で $\text{rank}(A) = m$ の場合、 $Ax = b$ を満たす (つまり、ノルム $\|Ax - b\|_2$ が最小になる) 解 x が無限に存在する。この場合、 $\|x\|_2$ を最小にする一意な解を見つける方が有用である。この問題は、*劣決定* の連立 1 次方程式 (方程式より未知数の個数の方が多い) に対して *最小ノルム解* を見つけることとも言える。この問題を解くには、行列 A の *LQ* 因子分解 ([「LQ 因子分解」](#) を参照) を使用する。

一般には、 $\text{rank}(A) < \min(m, n)$ となり、*階数不足の最小二乗問題* になるときがあるので、 $\|x\|_2$ も $\|Ax - b\|_2$ も最小にする *最小ノルム最小二乗解* を見つける必要がある。その場合 (または A の階数が不明な場合) は、*QR* 因子分解と、ピボット演算または *特異値分解* ([「特異値分解」](#) を参照) を併用する。

固有値問題：固有値 (eigenvalue) 問題 (*eigen* は、"固有" の意味のドイツ語) は、「行列 A が与えられたときに、下記のいずれかの方程式を満たす *固有値* λ と、それに対応する *固有ベクトル* z を見つけること」である。

$$Az = \lambda z \text{ (右固有ベクトル } z)$$

または

$$z^H A = \lambda z^H \text{ (左固有ベクトル } z)$$

A が実対称 / 複素エルミート行列の場合、上記の 2 つの方程式が等価になり、上記の問題は *対称固有値問題* と呼ばれる。このタイプの問題を解決するためのルーチンについては、この章の [「対称固有値問題」](#) のセクションで説明する。

非対称 / 非エルミート行列を使って固有値問題を解決するためのルーチンについては、この章の [「非対称固有値問題」](#) のセクションで説明する。

本ライブラリーには、汎用対称固有値問題(次のいずれかの方程式を満たす固有値 λ と、それに対応する固有ベクトル z を見つけること)を処理するためのルーチンも含まれている。

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

A は、対称/エルミート行列である。 B は、対称/エルミート正定値行列である。これらの問題を標準的な対称固有値問題にするためのルーチンについては、この章の「[汎用対称固有値問題](#)」のセクションで説明する。

個々の問題を解決するには、通常、いくつかの計算ルーチン呼び出す。本章のルーチンを、第3章で説明している他の LAPACK ルーチンや第2章で説明している BLAS ルーチンと組み合わせて使用しなければならない場合もある。

例えば、与えられた行列 B のすべての列 b に対して $\|Ax - b\|_2$ が最小になる一連の最小二乗問題 (A も B も実行列) を解くには、`?geqrf` を呼び出して $A = QR$ の因子分解を行った後、`?ormqr` を呼び出して $C = Q^H B$ を求める。最後に、BLAS ルーチンの `?trsm` を呼び出して、連立方程式 $RX = C$ を X について解く。

あるいは、適切なドライバールーチンを使用すれば、複数のタスクを一度に実行できる。例えば、最小二乗問題を解く場合は、ドライバールーチン `?gels` を使用する。



警告: LAPACK ルーチンは、入力行列に INF 値または NaN 値が含まれないことを前提としている。LAPACK に対する入力データが適当でないとき、コンピューターのハングアップなどの問題が発生する可能性がある。

インテル MKL 8.0 以降では、LAPACK 計算ルーチンおよびドライバールーチンに対する Fortran-77 インターフェイスとともに、より短い引数リストで簡略化したルーチン呼び出しを使用する Fortran-95 インターフェイスもサポートしている。Fortran-95 インターフェイスの呼び出しシーケンスは、ルーチン説明の「構文」セクションで、Fortran-77 呼び出しの説明の後に解説する。

ルーチン命名規則

この章の各ルーチンについて、Fortran-77 プログラムからの呼び出し時に LAPACK 名を使用できる。

LAPACK 名は、以下に示すように、`xyyzzz` 構造になっている。

最初の文字 x は、データ型を示す。

s	実数、単精度	c	複素数、単精度
d	実数、倍精度	z	複素数、倍精度

2番目と3番目の文字 yy は、行列のタイプと格納形式を示す。

bd	二重対角行列
ge	一般行列
gb	一般帯行列
hs	上 Hessenberg 行列
or	(実数)直交行列
op	(実数)直交行列(圧縮格納形式)

un	(複素数)ユニタリー行列
up	(複素数)ユニタリー行列(圧縮格納形式)
pt	対称/エルミート正定値三重対角行列
sy	対称行列
sp	対称行列(圧縮格納形式)
sb	(実数)対称帯行列
st	(実数)対称三重対角行列
he	エルミート行列
hp	エルミート行列(圧縮格納形式)
hb	(複素数)エルミート帯行列
tr	三角または準三角行列

最後の3文字 zzz は、実行される処理を示す。

qrf *QR* 因子分解を行う。

lqf *LQ* 因子分解を行う。

例えば、ルーチン `sgeqrf` は、単精度の一般実行列の *QR* 因子分解を行う。これに対応する複素行列用のルーチンは、`cgeqrf` である。

インテル MKL において、Fortran-95 インターフェイスの LAPACK 計算ルーチン名およびドライバルーチン名は、最初の文字がデータ型を示す以外は Fortran-77 の名前と同じである。例えば、Fortran-95 インターフェイスで、一般実行列の *QR* 因子分解を行うルーチン名は `geqrf` である。異なるデータ型を扱うには、単精度および倍精度の名前付き定数でモジュールブロックを参照する特定の入力パラメーターを定義して行う。

インテル MKL に含まれている LAPACK 計算ルーチンとドライバルーチン用の Fortran-95 インターフェイスの詳細、およびオプションの引数の使用についての一般的な情報は、第3章の「[Fortran-95 インターフェイス規則](#)」を参照。

行列の格納形式

LAPACK ルーチンでは、以下の行列格納形式を使用している。

- フル格納では、行列 A は2次元配列 a に格納され、行列成分 a_{ij} は配列成分 $a(i, j)$ に格納される。
- 圧縮格納では、対称行列、エルミート行列、または三角行列をコンパクトに格納できる。行列の上三角または下三角が1次元配列の各列に圧縮される。
- 帯格納では、 k_l 個の劣対角成分と k_u 個の優対角成分を持つ $m \times n$ の帯行列は、 $(k_l + k_u + 1)$ 行 n 列の2次元配列 ab にコンパクトに格納される。行列の各列は配列の対応する列に格納され、行列の各対角成分は配列の各行に格納される。

第3章と第4章では、圧縮格納形式で行列を格納する配列は名前の最後の文字を p で、帯格納形式で行列を格納する配列には名前の最後の文字を b で示している。行列の格納形式の詳細は、付録Bの「[行列引数](#)」を参照。

数学的表記

これまでの章で使用した数学表記以外に、本章では以下に示す表記も使用している。

λ_i	行列 A の固有値 (固有値の定義は、「 固有値問題 」を参照)。
σ_i	行列 A の特異値。 $A^H A$ の固有値の平方根に等しい (詳細は、「 特異値分解 」を参照)。
$\ x\ _2$	ベクトル x の 2- ノルム $x: \ x\ _2 = (\sum_i x_i ^2)^{1/2} = \ x\ _E$
$\ A\ _2$	行列 A の 2- ノルム (またはスペクトルノルム)。 $\ A\ _2 = \max_i \sigma_i, \ A\ _2^2 = \max_{ x =1} (Ax Ax)$
$\ A\ _E$	行列 A のユークリッド・ノルム。 $\ A\ _E^2 = \sum_i \sum_j a_{ij} ^2$ (ベクトルの場合、ユークリッド・ノルムと 2- ノルムが等しくなるので、 $\ x\ _E = \ x\ _2$)。
$q(x, y)$	ベクトル x と y の間の鋭角。 $\cos q(x, y) = x \cdot y / (\ x\ _2 \ y\ _2)$ 。

計算ルーチン

以下の各セクションでは、LAPACK 計算ルーチンについて説明する。LAPACK 計算ルーチンは、次の目的の計算タスクを個別に実行する。

[直交因子分解](#)

[特異値分解](#)

[対称固有値問題](#)

[汎用対称固有値問題](#)

[非対称固有値問題](#)

[汎用非対称固有値問題](#)

[汎用特異値分解](#)

各[ドライバールーチン](#)も参照のこと。

直交因子分解

このセクションでは、行列の QR (RQ) 因子分解と LQ (QL) 因子分解を行うための LAPACK ルーチンについて説明する。一般化された QR 因子分解と RQ 因子分解と同様に RZ 因子分解用のルーチンも含まれている。

QR 因子分解 : A を、因子分解する $m \times n$ の行列とする。

$m \geq n$ の場合、 QR 因子分解は次の式で与えられる。

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} = (Q_1, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix}$$

R は、実数型の対角成分を持つ $n \times n$ の上三角行列である。 Q は、 $m \times m$ の直交 (またはユニタリー) 行列である。

QR 因子分解を用いると、「 A が $m \times n$ の最大階数の行列 ($m \geq n$) である場合に、 $\|Ax - b\|_2$ を最小の値にする」という最小二乗問題が解ける。該当する行列の因子分解を行ってから、 $Rx = (Q_1)^T b$ を解いて、解 x を求める。

$m < n$ の場合、 QR 因子分解は次の式で与えられる。

$$A = QR = Q(R_1 R_2)$$

R は台形行列、 R_1 は上三角行列、 R_2 は矩形行列である。

LAPACK ルーチンでは、行列 Q を明示的な形式で表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

LQ 因子分解 : LQ 因子分解 A を $m \times n$ の行列とすると、 $m \leq n$ の場合、次の式で与えられる。

$$A = (L, 0)Q = (L, 0) \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} = LQ_1$$

L は、実数型の対角成分を持つ $m \times m$ の下三角行列である。 Q は、 $n \times n$ の直交 (またはユニタリー) 行列である。

$m > n$ の場合、 LQ 因子分解は次の式で与えられる。

$$A = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} Q$$

L_1 は $n \times n$ の下三角行列、 L_2 は矩形行列、 Q は $n \times n$ の直交 (またはユニタリー) 行列である。

LQ 因子分解を用いると、劣決定の連立 1 次方程式 $Ax = b$ (A は、階数 m ($m < n$) の $m \times n$ の行列) の最小ノルム解を求めることができる。該当する行列の因子分解を行ってから、 $Ly = b$ を y について解いた後、 $x = (Q_1)^H y$ を計算し、解ベクトル x を求める。

表 4-1 に、行列の直交因子分解を行うための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 4-1 直交因子分解用の計算ルーチン

行列のタイプ、因子分解	因子分解 (ピボット演算なし)	因子分解 (ピボット演算あり)	行列 Q の生成	行列 Q の適用
一般行列、 QR 因子分解	?sgeqrf	?dgeqrf ?cgeqrf3	?orgqr ?ungqr	?ormqr ?unmqr
一般行列、 RQ 因子分解	?sgerqf		?orgrq ?ungrq	?ormrq ?unmrq
一般行列、 LQ 因子分解	?sgeqlf		?orqlq ?unqlq	?ormlq ?unmlq
一般行列、 QL 因子分解	?sgeqlf		?orqql ?unqql	?ormql ?unmql
台形行列、 RZ 因子分解	?tZRzrf			?ormrz ?unmrz
行列のペア、 汎用 QR 因子分解	?sgqrff			
行列のペア、 汎用 RQ 因子分解	?sgqrff			

?sgeqrf

$m \times n$ の一般行列の QR 因子分解を行う。

構文

Fortran 77:

```
call sgeqrf(m, n, a, lda, tau, work, lwork, info)
call dgeqrf(m, n, a, lda, tau, work, lwork, info)
call cgeqrf(m, n, a, lda, tau, work, lwork, info)
call zgeqrf(m, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call geqrf(a [,tau] [,info])
```

説明

このルーチンは、 $m \times n$ の一般行列 A の QR 因子分解を行う。
(「[直交因子分解](#)」を参照)。ピボット演算は行わない。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合) COMPLEX (cgeqrf の場合) DOUBLE COMPLEX (zgeqrf の場合)。 配列: $a(lda, *)$ には、行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ ($lwork \geq n$)。 $lwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返す。 xerbla は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「 アプリケーション・ノート 」を参照。

出力パラメーター

a	因子分解のデータによって、次のように上書きされる。 $m \geq n$ の場合、対角線より下の成分は、ユニタリー行列 Q の各成分によって上書きされる。上三角は、上三角行列 R の対応する成分によって上書きされる。 $m < n$ の場合、厳密な下三角部分は、ユニタリー行列 Q の各成分によって上書きされる。残りの成分は、 $m \times n$ の上台形行列 R の対応する成分によって上書きされる。
tau	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合) COMPLEX (cgeqrf の場合) DOUBLE COMPLEX (zgeqrf の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 Q に関するその他の情報も格納される。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gexrf` のインターフェイスの詳細を以下に示す。

`a` サイズ (m, n) の行列 A を格納する。
`tau` 長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork = n * blocksize` に設定する。`blocksize` は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた因子分解は、行列 $A + E$ ($\|E\|_2 = O(\epsilon) \|A\|_2$) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(4/3)n^3$ ($m = n$ の場合)、
 $(2/3)n^2(3m - n)$ ($m > n$ の場合)、
 $(2/3)m^2(3n - m)$ ($m < n$ の場合)。

複素数型の演算回数は、この 4 倍になる。

与えられた行列 B のすべての列 b に対して $\|Ax - b\|_2$ が最小にする一連の最小二乗問題を解くには、以下の手順でルーチン呼び出す。

`?gexrf` (このルーチン) 因子分解 $A = QR$ を行う。
`?ormqr` $C = Q^T B$ を計算する (実行列の場合)。
`?unmqr` $C = Q^H B$ を計算する (複素行列の場合)。
`?trsm` (BLAS ルーチン) $RX = C$ を解く。

(計算で求めた X の列は、最小二乗の解ベクトル x である。)

Q の成分を明示的に計算するには、次のルーチン呼び出す。

`?orgqr` (実行列の場合)
`?ungqr` (複素行列の場合)

?geqpf

$m \times n$ の一般行列の QR 因子分解をピボット演算付きで行う。

構文

Fortran 77:

```
call sgeqpf(m, n, a, lda, jpvt, tau, work, info)
call dgeqpf(m, n, a, lda, jpvt, tau, work, info)
call cgeqpf(m, n, a, lda, jpvt, tau, work, rwork, info)
call zgeqpf(m, n, a, lda, jpvt, tau, work, rwork, info)
```

Fortran 95:

```
call geqpf(a, jpvt [,tau] [,info])
```

説明

このルーチンは [?geqp3](#) に置き換えられている。

このルーチン ?geqpf は、 $m \times n$ の一般行列 A の QR 因子分解 $AP = QR$ を列ピボット演算を用いて行う (「[直交因子分解](#)」を参照)。 P は、 $n \times n$ の置換行列である。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgeqpf の場合) DOUBLE PRECISION (dgeqpf の場合) COMPLEX (cgeqpf の場合) DOUBLE COMPLEX (zgeqpf の場合)。 配列： $a(lda, *)$ には、行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, 3*n)$ 以上でなければならない。
$jpvt$	INTEGER。配列、次元は $\max(1, n)$ 以上。 呼び出し時に、 $jpvt(i) > 0$ の場合、計算の開始前に、 A の i 番目の列が AP の先頭に移され、計算中はその位置に保持される。 $jpvt(i) = 0$ の場合、 A の i 番目の列はフリー列なので、計算中に他のフリー列と交換される場合がある。

rwork REAL (cgeqpf の場合)
DOUBLE PRECISION (zgeqpf の場合)。
ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

出力パラメーター

a 因子分解のデータによって、次のように上書きされる。
 $m \geq n$ の場合、対角線より下の成分は、ユニタリー (直交) 行列 Q の各成分によって上書きされる。上三角は、上三角行列 R の対応する成分によって上書きされる。
 $m < n$ の場合、厳密な下三角部分は、行列 Q の各成分によって上書きされる。残りの成分は、 $m \times n$ の上台形行列 R の対応する成分によって上書きされる。

tau REAL (sgeqpf の場合)
DOUBLE PRECISION (dgeqpf の場合)
COMPLEX (cgeqpf の場合)
DOUBLE COMPLEX (zgeqpf の場合)。
配列、次元は $\max(1, \min(m, n))$ 以上。
行列 Q に関するその他の情報も格納される。

jpvt 因子分解 $AP = QR$ における置換行列 P の各成分によって上書きされる。すなわち、 AP の列は、 A の列を次の順に並べたものになる。
 $jpvt(1), jpvt(2), \dots, jpvt(n)$ 。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geqpf` のインターフェイスの詳細を以下に示す。

a サイズ (m, n) の行列 A を格納する。
jpvt 長さ (n) のベクトルを格納する。
tau 長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

計算で求めた因子分解は、行列 $A + E$ ($\|E\|_2 = O(\epsilon) \|A\|_2$) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$$(4/3)n^3 \quad (m = n \text{ の場合}),$$

$$(2/3)n^2(3m-n) \quad (m > n \text{ の場合}),$$

$$(2/3)m^2(3n-m) \quad (m < n \text{ の場合}).$$

複素数型の演算回数は、この 4 倍になる。

与えられた行列 B のすべての列 b に対して $\|Ax - b\|_2$ が最小にする一連の最小二乗問題を解くには、以下の手順でルーチン呼び出す。

?geqpf (このルーチン) 因子分解 $AP = QR$ を行う。
[?ormqr](#) $C = Q^T B$ を計算する (実行列の場合)。
[?unmqr](#) $C = Q^H B$ を計算する (複素行列の場合)。
[?trsm](#) (BLAS ルーチン) $RX = C$ を解く。

(計算で求めた X の列は、最小二乗解ベクトル x が置換されたものである。置換順序は、配列 $jpvt$ に出力される。)

Q の成分を明示的に計算するには、次のルーチン呼び出す。

[?orgqr](#) (実行列の場合)
[?ungqr](#) (複素行列の場合)

?geqp3

レベル3 BLAS を使用して、 $m \times n$ の一般行列の QR 因子分解をピボット演算を用いて行う。

構文

Fortran 77:

```
call sgeqp3(m, n, a, lda, jpvt, tau, work, lwork, info)
call dgeqp3(m, n, a, lda, jpvt, tau, work, lwork, info)
call cgeqp3(m, n, a, lda, jpvt, tau, work, lwork, rwork, info)
call zgeqp3(m, n, a, lda, jpvt, tau, work, lwork, rwork, info)
```

Fortran 95

```
call geqp3(a, jpvt [,tau] [,info])
```

説明

このルーチンは、レベル3 BLAS を使用して、 $m \times n$ の一般行列 A の QR 因子分解 ($AP = QR$ 、[「直交因子分解」](#) を参照) をピボット演算を用いて行う。 P は、 $n \times n$ の置換行列である。

このルーチンを使用すると、?geqpf より高いパフォーマンスが得られる。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

m INTEGER。行列 A の行数 ($m \geq 0$)。
 n INTEGER。 A の列数 ($n \geq 0$)。

<i>a, work</i>	REAL (sgeqp3 の場合) DOUBLE PRECISION (dgeqp3 の場合) COMPLEX (cgeqp3 の場合) DOUBLE COMPLEX (zgeqp3 の場合)。 配列: <i>a(lda,*)</i> には、行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(lwork)</i> は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ。 $\max(1, 3*n+1)$ 以上 (実数型の場合) または $\max(1, n+1)$ 以上 (複素数型の場合) でなければならない。 <i>lwork = -1</i> の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>jpvt</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 呼び出し時に、 <i>jpvt(i) ≠ 0</i> の場合、計算の開始前に、 <i>A</i> の <i>i</i> 番目の列が <i>AP</i> の先頭に移され、計算中はその位置に保持される。 <i>jpvt(i) = 0</i> の場合、 <i>A</i> の <i>i</i> 番目の列はフリー列なので、計算中に他のフリー列と交換される場合がある。
<i>rwork</i>	REAL (cgeqp3 の場合) DOUBLE PRECISION (zgeqp3 の場合)。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。 複素数型でのみ使用される。

出力パラメーター

<i>a</i>	因子分解のデータによって、次のように上書きされる。 $m \geq n$ の場合、対角線より下の成分は、ユニタリー (直交) 行列 <i>Q</i> の各成分によって上書きされる。上三角は、上三角行列 <i>R</i> の対応する成分によって上書きされる。 $m < n$ の場合、厳密な下三角部分は、行列 <i>Q</i> の各成分によって上書きされる。残りの成分は、 $m \times n$ の上台形行列 <i>R</i> の対応する成分によって上書きされる。
<i>tau</i>	REAL (sgeqp3 の場合) DOUBLE PRECISION (dgeqp3 の場合) COMPLEX (cgeqp3 の場合) DOUBLE COMPLEX (zgeqp3 の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 <i>Q</i> に対する基本リフレクターのスカラー係数が格納される。
<i>jpvt</i>	因子分解 $AP = QR$ における置換行列 <i>P</i> の各成分によって上書きされる。すなわち、 <i>AP</i> の列は、 <i>A</i> の列を次の順に並べたものになる。 <i>jpvt(1), jpvt(2), ..., jpvt(n)</i> 。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geqp3` のインターフェイスの詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。
jpvt 長さ (*n*) のベクトルを格納する。
tau 長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

与えられた行列 *B* のすべての列 *b* に対して $\|Ax - b\|_2$ が最小にする一連の最小二乗問題を解くには、以下の手順でルーチンを呼び出す。

?`geqp3` (このルーチン) 因子分解 $AP = QR$ を行う。
[?ormqr](#) $C = Q^T B$ を計算する (実行列の場合)。
[?unmqr](#) $C = Q^H B$ を計算する (複素行列の場合)。
[?trsm](#) (BLAS ルーチン) $RX = C$ を解く。

(計算で求めた *X* の列は、最小二乗解ベクトル *x* が置換されたものである。置換順序は、配列 *jpvt* に出力される。)

Q の成分を明示的に計算するには、次のルーチンを呼び出す。

[?orgqr](#) (実行列の場合)
[?ungqr](#) (複素行列の場合)

?orgqr

?`geqrf` で求めた *QR* 因子分解の実直交行列 *Q* を生成する。

構文

Fortran 77:

```
call sorgqr(m, n, k, a, lda, tau, work, lwork, info)
call dorgqr(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orgqr(a, tau [,info])
```

説明

このルーチンは、ルーチン [sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) によって求めた QR 因子分解の直交行列 Q ($m \times m$) の全部または一部を生成する。このルーチンは、[sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) を呼び出した後で使用する。

Q は、通常、 $m \times p$ の行列 A ($m \geq p$) の QR 因子分解によって決定される。行列 Q の全体を計算するには、次のように呼び出す。

```
call ?orgqr(m, m, p, a, lda, tau, work, lwork, info)
```

Q の先頭から p 個の列 (A の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orgqr(m, p, p, a, lda, tau, work, lwork, info)
```

A の先頭から k 個の列に関して QR 因子分解の行列 Q^k を求めるには、次のように呼び出す。

```
call ?orgqr(m, m, k, a, lda, tau, work, lwork, info)
```

Q^k の先頭から k 個の列 (A の先頭から k 個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr(m, k, k, a, lda, tau, work, lwork, info)
```

入力パラメーター

m	INTEGER。直交行列 Q の次数 ($m \geq 0$)。
n	INTEGER。計算する Q の列数 ($0 \leq n \leq m$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($0 \leq k \leq n$)。
$a, \tau, work$	REAL (sorgqr の場合) DOUBLE PRECISION (dorgqr の場合) 配列: $a(lda, *)$ と $\tau(*)$ は、 sgeqrf/dgeqrf または sgeqpf/dgeqpf から返された配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ ($lwork \geq n$)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$m \times m$ の直交行列 Q の先頭から n 個の列によって上書きされる。
-----	---

`work(1)` `info=0` の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の `lwork` の値が `work(1)` に出力される。これ以降の実行には、この `lwork` の値を使用する。

`info` INTEGER。
`info=0` の場合、正常に終了したことを示す。
`info=-i` の場合、`i` 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgqr` のインターフェイスの詳細を以下に示す。

`a` サイズ (m, n) の行列 A を格納する。
`tau` 長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork=n*blocksize` に設定する。`blocksize` は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ~ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\epsilon) \|A\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $4*m*n*k - 2*(m+n)*k^2 + (4/3)*k^3$ である。
 $n=k$ の場合、この値は約 $(2/3)*n^2*(3m-n)$ になる。

このルーチンの複素数版は、[?ungqr](#) である。

?ormqr

実行列に `?geqrf` または `?geqpf` で求めた QR 因子分解の直交行列 Q を掛ける。

構文

Fortran 77:

```
call sormqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call ormqr(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、実行列 C に Q または Q^T を掛ける。 Q は、ルーチン [sgeqrf/dgeqrf](#) または [sgeqpf/dgeqpf](#) で求めた QR 因子分解の直交行列 Q である。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^T は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^T は、 C に右側から適用される。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'T'$ の場合、 C に Q^T を掛ける。
m	INTEGER。行列 C の行数 ($m \geq 0$)。
n	INTEGER。行列 C の列数 ($n \geq 0$)。
k	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合) $0 \leq k \leq n$ ($side = 'R'$ の場合)
$a, work, tau, c$	REAL (sgeqrf の場合) DOUBLE PRECISION (dgeqrf の場合)。 配列: $a(lda, *)$ と $tau(*)$ は、sgeqrf/dgeqrf または sgeqpf/dgeqpf から返された配列である。 a の第 2 次元は、 $\max(1, k)$ 以上でなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。次の制約がある。 $lda \geq \max(1, m)$ ($side = 'L'$ の場合)。 $lda \geq \max(1, n)$ ($side = 'R'$ の場合)。
ldc	INTEGER。 c の第 1 次元。次の制約がある: $ldc \geq \max(1, m)$ 。
$lwork$	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、xerbla は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormqr` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>r</i> , <i>k</i>) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> (<i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> (<i>side</i> = 'R' の場合)。
<i>tau</i>	長さ (<i>k</i>) のベクトルを格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (*side* = 'L' の場合) または $lwork = m * blocksize$ (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?unmqr](#) である。

?ungqr

?geqrf で求めた QR 因子分解の複素ユニタリー行列 Q を生成する。

構文

Fortran 77:

```
call cungqr(m, n, k, a, lda, tau, work, lwork, info)
call zungqr(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call ungqr(a, tau [,info])
```

説明

このルーチンは、ルーチン [cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) によって求めた QR 因子分解のユニタリー行列 Q ($m \times m$) の全部または一部を生成する。このルーチンは、[cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) を呼び出した後で使用する。

Q は、通常、 $m \times p$ の行列 A ($m \geq p$) の QR 因子分解によって決定される。行列 Q の全体を計算するには、次のように呼び出す。

```
call ?ungqr(m, m, p, a, lda, tau, work, lwork, info)
```

Q の先頭から p 個の列 (A の列で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?ungqr(m, p, p, a, lda, tau, work, lwork, info)
```

A の先頭から k 個の列に関して QR 因子分解の行列 Q^k を求めるには、次のように呼び出す。

```
call ?ungqr(m, m, k, a, lda, tau, work, lwork, info)
```

Q^k の先頭から k 個の列 (A の先頭から k 個の列で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr(m, k, k, a, lda, tau, work, lwork, info)
```

入力パラメーター

m	INTEGER。ユニタリー行列 Q の次数 ($m \geq 0$)。
n	INTEGER。計算する Q の列数 ($0 \leq n \leq m$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($0 \leq k \leq n$)。
$a, \tau, work$	COMPLEX (cungqr の場合) DOUBLE COMPLEX (zungqr の場合) 配列: $a(lda, *)$ と $\tau(*)$ は、 cgeqrf/zgeqrf または cgeqpf/zgeqpf から返された配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

lwork INTEGER。 配列 $work$ のサイズ ($lwork \geq n$)。
 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a $m \times m$ のユニタリー行列 Q の先頭から n 個の列によって上書きされる。

$work(1)$ $info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungqr` のインターフェイスの詳細を以下に示す。

a サイズ (m, n) の行列 A を格納する。

tau 長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた Q は、正確なユニタリー行列と行列 E ($\|E\|_2 = O(\epsilon) \|A\|_2$, ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $16 * m * n * k - 8 * (m + n) * k^2 + (16/3) * k^3$ である。
 $n = k$ の場合、この値は約 $(8/3) * n^2 * (3m - n)$ になる。

このルーチンの実数版は、[?orgqr](#) である。

?unmqr

複素行列に ?geqrf で求めた QR 因子分解のユニタリー行列 Q を掛ける。

構文

Fortran 77:

```
call cunmqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmqr(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call unmqr(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、矩形の複素行列 C に Q または Q^H を掛ける。 Q は、ルーチン [cgeqrf/zgeqrf](#) または [cgeqpf/zgeqpf](#) で求めた QR 因子分解のユニタリー行列 Q である。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 Q^HC 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

$side$	CHARACTER*1. 'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^H は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^H は、 C に右側から適用される。
$trans$	CHARACTER*1. 'N' または 'C' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'C'$ の場合、 C に Q^H を掛ける。
m	INTEGER。行列 C の行数 ($m \geq 0$)。
n	INTEGER。行列 C の列数 ($n \geq 0$)。
k	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合) $0 \leq k \leq n$ ($side = 'R'$ の場合)
$a, work, tau, c$	COMPLEX (cgeqrf の場合) DOUBLE COMPLEX (zgeqrf の場合)。 配列: $a(lda, *)$ と $tau(*)$ は、cgeqrf / zgeqrf または cgeqpf / zgeqpf から返された配列である。 a の第 2 次元は、 $\max(1, k)$ 以上でなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。次の制約がある。 $lda \geq \max(1, m)$ (<i>side</i> = 'L' の場合)。 $lda \geq \max(1, n)$ (<i>side</i> = 'R' の場合)。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。次の制約がある： $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ (<i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *unmqr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>r</i> , <i>k</i>) の行列 <i>A</i> を格納する。 $r = m$ (<i>side</i> = 'L' の場合)。 $r = n$ (<i>side</i> = 'R' の場合)。
<i>tau</i>	長さ (<i>k</i>) のベクトルを格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * \text{blocksize}$ (*side* = 'L' の場合) または $lwork = m * \text{blocksize}$ (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?ormqr](#) である。

?gelqf

$m \times n$ の一般行列の LQ 因子分解を行う。

構文

Fortran 77:

```
call sgelqf(m, n, a, lda, tau, work, lwork, info)
call dgelqf(m, n, a, lda, tau, work, lwork, info)
call cgelqf(m, n, a, lda, tau, work, lwork, info)
call zgelqf(m, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call gelqf(a [,tau] [,info])
```

説明

このルーチンは、 $m \times n$ の一般行列 A の LQ 因子分解を行う (「[直交因子分解](#)」を参照)。ピボット演算は行わない。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgelqf の場合) DOUBLE PRECISION (dgelqf の場合) COMPLEX (cgelqf の場合) DOUBLE COMPLEX (zgelqf の場合)。 配列: $a(lda,*)$ には、行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	因子分解のデータによって、次のように上書きされる。 $m \leq n$ の場合、対角線より上の成分は、ユニタリー (直交) 行列 Q の各成分によって上書きされる。下三角は、下三角行列 L の対応する成分によって上書きされる。 $m > n$ の場合、厳密な上三角部分は、行列 Q の各成分によって上書きされる。残りの成分は、 $m \times n$ の下台形行列 L の対応する成分によって上書きされる。
<i>tau</i>	REAL (sgelqf の場合) DOUBLE PRECISION (dgelqf の場合) COMPLEX (cgelqf の場合) DOUBLE COMPLEX (zgelqf の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 Q に関するその他の情報も格納される。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gelqf` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>tau</i>	長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた因子分解は、行列 $A + E$ ($\|E\|_2 = O(\epsilon) \|A\|_2$) を因子分解したそのものである。

実数型の浮動小数点演算のおおよその回数は次のとおりである。

$(4/3)n^3$	($m = n$ の場合)、
$(2/3)n^2(3m-n)$	($m > n$ の場合)、
$(2/3)m^2(3n-m)$	($m < n$ の場合)。

複素数型の演算回数は、この 4 倍になる。

与えられた行列 B のすべての列 b に対して $\|Ax - b\|_2$ が最小になるような劣決定最小二乗問題の最小ノルム解を見つけるには、以下の手順でルーチン呼び出す。

`?gelqf` (このルーチン) 因子分解 $A = LQ$ を行う。

`?trsm` (BLAS ルーチン) $LY = B$ を Y について解く。

`?ormlq` $X = (Q_1)^T Y$ を計算する (実行列の場合)。

`?unmlq` $X = (Q_1)^H Y$ を計算する (複素行列の場合)。

計算で求めた X の列は、最小ノルムの解ベクトル x である。ここで、 A は、 $m \times n$ の行列 ($m < n$) である。 Q_1 は、 Q の先頭から m 個の列を表す。

Q の成分を明示的に計算するには、次のルーチン呼び出す。

`?orglq` (実行列の場合)

`?unglq` (複素行列の場合)

?orglq

`?gelqf` で求めた LQ 因子分解の実直交行列 Q を生成する。

構文

Fortran 77:

```
call sorglq(m, n, k, a, lda, tau, work, lwork, info)
call dorglq(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orglq(a, tau [,info])
```

説明

このルーチンは、ルーチン `sgelqf/dgelqf` によって求めた LQ 因子分解の直交行列 Q ($n \times n$) の全部または一部を生成する。このルーチンは、`sgelqf/dgelqf` を呼び出した後で使用される。

Q は、通常、 $p \times n$ の行列 A ($n \geq p$) の LQ 因子分解によって決定される。行列 Q の全体を計算するには、次のように呼び出す。

```
call ?orglq(n, n, p, a, lda, tau, work, lwork, info)
```

Q の先頭から p 個の行 (A の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?orglq(p, n, p, a, lda, tau, work, lwork, info)
```

A の先頭から k 個の行に関して LQ 因子分解の行列 Q^k を求めるには、次のように呼び出す。

```
call ?orglq(n, n, k, a, lda, tau, work, lwork, info)
```

Q^k の先頭から k 個の行 (A の先頭から k 個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?orgqr(k, n, k, a, lda, tau, work, lwork, info)
```

入力パラメーター

m	INTEGER。計算する Q の行数 ($0 \leq m \leq n$)。
n	INTEGER。直交行列 Q の次数 ($n \geq m$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($0 \leq k \leq m$)。
$a, \tau, work$	REAL (sorglq の場合) DOUBLE PRECISION (dorglq の場合) 配列: $a(lda,*)$ と $\tau(*)$ は、sgelqf/dgelqf から返された配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$n \times n$ の直交行列 Q の先頭から m 個の行によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orglq` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
τ	長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\epsilon) \|A\|_2$, ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $4 * m * n * k - 2 * (m + n) * k^2 + (4/3) * k^3$ である。 $m = k$ の場合、この値は約 $(2/3) * m^2 * (3n - m)$ になる。

このルーチンの複素数版は、[?ungqlq](#) である。

?ormlq

実行列に [?gelqf](#) で求めた LQ 因子分解の直交行列 Q を掛ける。

構文

Fortran 77:

```
call sormlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call ormlq(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の実行列 C に Q または Q^T を掛ける。 Q は、ルーチン [sgelqf/dgelqf](#) で求めた LQ 因子分解の直交行列 Q である。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

$side$	CHARACTER*1. 'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^T は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^T は、 C に右側から適用される。
$trans$	CHARACTER*1. 'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'T'$ の場合、 C に Q^T を掛ける。
m	INTEGER. 行列 C の行数 ($m \geq 0$).
n	INTEGER. 行列 C の列数 ($n \geq 0$).

k	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合) $0 \leq k \leq n$ ($side = 'R'$ の場合)
$a, work, tau, c$	REAL (sormlq の場合) DOUBLE PRECISION (dormlq の場合)。 配列： $a(lda, *)$ と $tau(*)$ は、?gelqf から返された配列である。 a の第 2 次元は、 $\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ 以上 ($side = 'R'$ の場合) でなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, k)$
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
$lwork$	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c	($side$ と $trans$ の値に従って) QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormlq` のインターフェイスの詳細を以下に示す。

a	サイズ (k, m) の行列 A を格納する。
tau	長さ (k) のベクトルを格納する。
c	サイズ (m, n) の行列 C を格納する。

side 'L' または 'R' でなければならない。デフォルト値は 'L'。

trans 'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?unmlq](#) である。

?unglq

?gelqf で求めた *LQ* 因子分解の複素ユニタリー行列 *Q* を生成する。

構文

Fortran 77:

```
call cunglq(m, n, k, a, lda, tau, work, lwork, info)
call zunglq(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call unglq(a, tau [,info])
```

説明

このルーチンは、ルーチン [cgelqf/zgelqf](#) で求めた *LQ* 因子分解のユニタリー行列 *Q* ($n \times n$) の全部または一部を生成する。このルーチンは、[cgelqf/zgelqf](#) を呼び出した後で使用される。

Q は、通常、 $p \times n$ の行列 *A* ($n \geq p$) の *LQ* 因子分解によって決定される。行列 *Q* の全体を計算するには、次のように呼び出す。

```
call ?unglq(n, n, p, a, lda, tau, work, lwork, info)
```

Q の先頭から *p* 個の行 (*A* の行で張られた空間内で直交基を形成) を計算するには、次のように呼び出す。

```
call ?unglq(p, n, p, a, lda, tau, work, lwork, info)
```

A の先頭から *k* 個の行に関して *LQ* 因子分解の行列 Q^k を求めるには、次のように呼び出す。

```
call ?unglq(n, n, k, a, lda, tau, work, lwork, info)
```

Q^k の先頭から *k* 個の行 (*A* の先頭から *k* 個の行で張られた空間内で直交基を形成) を求めるには、次のように呼び出す。

```
call ?ungqr(k, n, k, a, lda, tau, work, lwork, info)
```

入力パラメーター

<i>m</i>	INTEGER。計算する Q の行数 ($0 \leq m \leq n$)。
<i>n</i>	INTEGER。ユニタリー行列 Q の次数 ($n \geq m$)。
<i>k</i>	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($0 \leq k \leq m$)。
<i>a, tau, work</i>	COMPLEX (cunglq の場合) DOUBLE COMPLEX (zunglq の場合) 配列： <i>a</i> (<i>lda</i> ,*) と <i>tau</i> (*) は、sgelqf/dgelqf から返された配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 $\max(1, m)$ 以上。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	$n \times n$ のユニタリー行列 Q の先頭から <i>m</i> 個の行によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *unglq* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ (<i>k</i>) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた Q は、正確なユニタリ行列と行列 E ($\|E\|_2 = O(\epsilon) \|A\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $16 * m * n * k - 8 * (m + n) * k^2 + (16/3) * k^3$ である。 $m = k$ の場合、この値は約 $(8/3) * m^2 * (3n - m)$ になる。

このルーチンの実数版は、[?orgqlq](#) である。

?unmlq

複素行列に [?gelqf](#) で求めた LQ 因子分解のユニタリ行列 Q を掛ける。

構文

Fortran 77:

```
call cunmlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmlq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call unmlq(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の複素行列 C に Q または Q^H を掛ける。 Q は、ルーチン [cgelqf/zgelqf](#) で求めた LQ 因子分解のユニタリ行列 Q である。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

$side$	CHARACTER*1. 'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^H は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^H は、 C に右側から適用される。
$trans$	CHARACTER*1. 'N' または 'C' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'C'$ の場合、 C に Q^H を掛ける。
m	INTEGER. 行列 C の行数 ($m \geq 0$).
n	INTEGER. 行列 C の列数 ($n \geq 0$).

k	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合) $0 \leq k \leq n$ ($side = 'R'$ の場合)
$a, work, tau, c$	COMPLEX (cunmlq の場合) DOUBLE COMPLEX (zunmlq の場合)。 配列： $a(lda, *)$ と $tau(*)$ は、?gelqf から返された配列である。 a の第 2 次元は、 $\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ 以上 ($side = 'R'$ の場合) でなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c ldc, *$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, k)$
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
$lwork$	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c	($side$ と $trans$ の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン unmlq のインターフェイスの詳細を以下に示す。

a	サイズ (k, m) の行列 A を格納する。
tau	長さ (k) のベクトルを格納する。
c	サイズ (m, n) の行列 C を格納する。

`side` 'L' または 'R' でなければならない。デフォルト値は 'L'。
`trans` 'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (`side = 'L'` の場合) または $lwork = m * blocksize$ (`side = 'R'` の場合) に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?ormlq](#) である。

?gqqlf

$m \times n$ の一般行列の QL 因子分解を行う。

構文

Fortran 77:

```
call sgeqlf(m, n, a, lda, tau, work, lwork, info)
call dgeqlf(m, n, a, lda, tau, work, lwork, info)
call cgeqlf(m, n, a, lda, tau, work, lwork, info)
call zgeqlf(m, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call gqqlf(a [,tau] [,info])
```

説明

このルーチンは、 $m \times n$ の一般行列 A の QL 因子分解を行う。ピボット演算は行わない。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

<code>m</code>	INTEGER。行列 A の行数 ($m \geq 0$)。
<code>n</code>	INTEGER。 A の列数 ($n \geq 0$)。
<code>a, work</code>	REAL (sgeqlf の場合) DOUBLE PRECISION (dgeqlf の場合) COMPLEX (cgeqlf の場合) DOUBLE COMPLEX (zgeqlf の場合)。

配列：

$a(lda,*)$ には、行列 A を格納する。

a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。 配列 $work$ のサイズ。 $\max(1, n)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	終了時に、因子分解データによって次のように上書きされる。 $m \geq n$ の場合、部分配列 $a(m-n+1:m, 1:n)$ の下三角部分に、 $n \times n$ の下三角行列 L が格納される。 $m \leq n$ の場合、 $(n-m)$ 番目の優対角成分とその下の各成分に、 $m \times n$ の下台形行列 L が格納される。 どちらの場合も、残りの成分は、配列 tau とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表現する。
tau	REAL (sgeqlf の場合) DOUBLE PRECISION (dgeqlf の場合) COMPLEX (cgeqlf の場合) DOUBLE COMPLEX (zgeqlf の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 Q に対する基本リフレクターのスカラー係数が格納される。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geqlf` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
tau	長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

関連ルーチン:

?orgql	行列 Q を生成する (実行列の場合)。
?ungql	行列 Q を生成する (複素行列の場合)。
?ormql	行列 Q を適用する (実行列の場合)。
?unmql	行列 Q を適用する (複素行列の場合)。

?orgql

?geqlf で求めた QL 因子分解の実行列 Q を生成する。

構文

Fortran 77:

```
call sorgql(m, n, k, a, lda, tau, work, lwork, info)
call dorgql(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orgql(a, tau [,info])
```

説明

このルーチンは、直交列を持つ $m \times n$ の実行列 Q を生成する。これは、ルーチン [sgeqlf/dgeqlf](#) から返された k 個の基本リフレクター H_1 (次数は m) の積の最終 n 列で定義される ($Q = H_k \cdots H_2 H_1$)。このルーチンは、[sgeqlf/dgeqlf](#) を呼び出した後で使用する。

入力パラメーター

m	INTEGER。行列 Q の行数 ($m \geq 0$)。
n	INTEGER。行列 Q の列数 ($m \geq n \geq 0$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
$a, \tau, work$	REAL (sorgql の場合) DOUBLE PRECISION (dorgql の場合) 配列: $a(lda, *)$, $\tau(*)$, $work(lwork)$ 。

呼び出し時に、`sgeqlf/dgeqlf` で配列引数 a の最終 k 列に返されたとおりに、 a の $(n-k+i)$ 番目の列に、基本リフレクター H_i (ただし、 $i=1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。

$\tau(i)$ には、`sgeqlf/dgeqlf` から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。

a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

τ の次元は、 $\max(1, k)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。 配列 $work$ のサイズ。 $\max(1, n)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 <code>xerbla</code> は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$m \times n$ の行列 Q によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgql` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
τ	長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、`?ungql` である。

?ungql

?geqlf で求めた QL 因子分解の複素行列を生成する。

構文

Fortran 77:

```
call cungql(m, n, k, a, lda, tau, work, lwork, info)
call zungql(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call ungql(a, tau [,info])
```

説明

このルーチンは、直交列を持つ $m \times n$ の複素行列 Q を生成する。これは、ルーチン [cgeqlf/zgeqlf](#) から返された k 個の基本リフレクター H_i (次数は m) の積の最終 n 列で定義される ($Q = H_k \cdots H_2 H_1$)。このルーチンは、[cgeqlf/zgeqlf](#) を呼び出した後で使用する。

入力パラメーター

m	INTEGER。行列 Q の行数 ($m \geq 0$)。
n	INTEGER。行列 Q の列数 ($m \geq n \geq 0$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
$a, \tau, work$	COMPLEX (cungql の場合) DOUBLE COMPLEX (zungql の場合) 配列: $a(lda, *)$, $\tau(*)$, $work(lwork)$ 。 呼び出し時に、 cgeqlf/zgeqlf で配列引数 a の最終 k 列に返されたとおりに、 a の $(n - k + i)$ 番目の列に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 $\tau(i)$ には、 cgeqlf/zgeqlf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, n)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 xerbla は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	$m \times n$ の行列 Q によって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungql` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>tau</i>	長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?orgql](#) である。

?ormql

実行列に `?geqlf` で求めた QL 因子分解の直交行列 Q を掛ける。

構文**Fortran 77:**

```
call sormql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call ormql(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の実行列 C に Q または Q^T を掛ける。 Q は、ルーチン [sgeqlf/dgeqlf](#) で求めた QL 因子分解の直交行列 Q である。

このルーチンを使用すれば、パラメーター *side* と *trans* の値に従って、 QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 Q または Q^T は、 C に左側から適用される。 <i>side</i> = 'R' の場合、 Q または Q^T は、 C に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 C に Q を掛ける。 <i>trans</i> = 'T' の場合、 C に Q^T を掛ける。
<i>m</i>	INTEGER。行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 C の列数 ($n \geq 0$)。
<i>k</i>	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ (<i>side</i> = 'L' の場合) $0 \leq k \leq n$ (<i>side</i> = 'R' の場合)
<i>a, tau, c, work</i>	REAL (sormql の場合) DOUBLE PRECISION (dormql の場合)。 配列: $a(lda, *)$, $tau(*)$, $c ldc, *$, $work(lwork)$ 呼び出し時に、sgeqlf/dgeqlf で配列引数 a の最終 k 列に返されたとおりに、 a の i 番目の列に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 a の第 2 次元は、 $\max(1, k)$ 以上でなければならない。 $tau(i)$ には、sgeqlf/dgeqlf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、 $m \times n$ の行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第 1 次元。 <i>side</i> = 'L' の場合、 $lda \geq \max(1, m)$ 。 <i>side</i> = 'R' の場合、 $lda \geq \max(1, n)$ 。
<i>ldc</i>	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ (<i>side</i> = 'R' の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、xerbla は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの積によって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormql` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>r</i> , <i>k</i>) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> (<i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> (<i>side</i> = 'R' の場合)。
<i>tau</i>	長さ (<i>k</i>) のベクトルを格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (*side* = 'L' の場合) または $lwork = m * blocksize$ (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?unmql](#) である。

?unmql

複素行列に `?geqlf` で求めた *QL* 因子分解のユニタリー行列 *Q* を掛ける。

構文**Fortran 77:**

```
call cunmql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmql(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call unmq1(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の複素行列 C に Q または Q^H を掛ける。 Q は、ルーチン [cgeqlf/zgeqlf](#) で求めた QL 因子分解のユニタリー行列 Q である。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^H は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^H は、 C に右側から適用される。
$trans$	CHARACTER*1。'N' または 'C' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'C'$ の場合、 C に Q^H を掛ける。
m	INTEGER。行列 C の行数 ($m \geq 0$)。
n	INTEGER。行列 C の列数 ($n \geq 0$)。
k	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合) $0 \leq k \leq n$ ($side = 'R'$ の場合)
$a, tau, c, work$	COMPLEX (cunmq1 の場合) DOUBLE COMPLEX (zunmq1 の場合)。 配列: $a(lda, *)$, $tau(*)$, $c(ldc, *)$, $work(lwork)$ 呼び出し時に、cgeqlf/zgeqlf で配列引数 a の最終 k 列に返されたとおりに、 a の i 番目の列に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 a の第 2 次元は、 $\max(1, k)$ 以上でなければならない。 $tau(i)$ には、cgeqlf/zgeqlf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、 $m \times n$ の行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $side = 'L'$ の場合、 $lda \geq \max(1, m)$ 。 $side = 'R'$ の場合、 $lda \geq \max(1, n)$ 。
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
$lwork$	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。

$lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c	($side$ と $trans$ の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmql` のインターフェイスの詳細を以下に示す。

a	サイズ (r, k) の行列 A を格納する。 $r = m$ ($side = 'L'$ の場合)。 $r = n$ ($side = 'R'$ の場合)。
tau	長さ (k) のベクトルを格納する。
c	サイズ (m, n) の行列 C を格納する。
$side$	'L' または 'R' でなければならない。デフォルト値は 'L'。
$trans$	'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合) に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?ormql](#) である。

?gerqf

$m \times n$ の一般行列の RQ 因子分解を行う。

構文

Fortran 77:

```
call sgerqf(m, n, a, lda, tau, work, lwork, info)
call dgerqf(m, n, a, lda, tau, work, lwork, info)
call cgerqf(m, n, a, lda, tau, work, lwork, info)
call zgerqf(m, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call gerqf(a [,tau] [,info])
```

説明

このルーチンは、 $m \times n$ の一般行列 A の RQ 因子分解を行う。ピボット演算は行わない。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgerqf の場合) DOUBLE PRECISION (dgerqf の場合) COMPLEX (cgerqf の場合) DOUBLE COMPLEX (zgerqf の場合)。 配列： $a(lda, *)$ には、 $m \times n$ の行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $lwork \geq \max(1, m)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「 アプリケーション・ノート 」を参照。

出力パラメーター

<i>a</i>	終了時に、因子分解データによって次のように上書きされる。 $m \leq n$ の場合、部分配列 $a(1:m, n-m+1:n)$ の上三角部分に、 $m \times m$ の上三角行列 R が格納される。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の各成分に、 $m \times n$ の上台形行列 R が格納される。 どちらの場合も、残りの成分は、配列 <i>tau</i> とともに、 $\min(m, n)$ 個の基本リフレクターの積として直交/ユニタリー行列 Q を表現する。
<i>tau</i>	REAL (sgerqf の場合) DOUBLE PRECISION (dgerqf の場合) COMPLEX (cgerqf の場合) DOUBLE COMPLEX (zgerqf の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 Q に対する基本リフレクターのスカラー係数が格納される。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gerqf* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>tau</i>	長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = m * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

関連ルーチン：

?orgqr	行列 Q を生成する (実行列の場合)。
?ungqr	行列 Q を生成する (複素行列の場合)。
?ormqr	行列 Q を適用する (実行列の場合)。
?unmqr	行列 Q を適用する (複素行列の場合)。

?orgrq

?gerqf で求めた RQ 因子分解の実行列 Q を生成する。

構文

Fortran 77:

```
call sorgrq(m, n, k, a, lda, tau, work, lwork, info)
call dorgrq(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orgrq(a, tau [,info])
```

説明

このルーチンは、直交行を持つ $m \times n$ の実行列 Q を生成する。これは、ルーチン [sgerqf/dgerqf](#) から返された k 個の基本リフレクター H_i (次数は n) の積の最終 m 行で定義される ($Q = H_1 H_2 \dots H_k$)。このルーチンは、[sgerqf/dgerqf](#) を呼び出した後で使用される。

入力パラメーター

m	INTEGER。行列 Q の行数 ($m \geq 0$)。
n	INTEGER。行列 Q の列数 ($n \geq m$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
$a, \tau, work$	REAL (sorgrq の場合) DOUBLE PRECISION (dorgrq の場合) 配列: $a(lda, *)$, $\tau(*)$, $work(lwork)$ 。 呼び出し時に、 sgerqf/dgerqf で配列引数 a の最終 k 行に返されたとおりに、 a の $(m - k + i)$ 番目の行に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 $\tau(i)$ には、 sgerqf/dgerqf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$m \times n$ の行列 Q によって上書きされる。
$work(1)$	$info=0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgrq` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
τ	長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork=m*blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?ungrq](#) である。

?ungrq

?gerqf で求めた RQ 因子分解の複素行列 Q を生成する。

構文

Fortran 77:

```
call cungrq(m, n, k, a, lda, tau, work, lwork, info)
call zungrq(m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call ungrq(a, tau [,info])
```

説明

このルーチンは、直交行を持つ $m \times n$ の複素行列 Q を生成する。これは、ルーチン [sgerqf/dgerqf](#) から返された k 個の基本リフレクター H_i (次数は n) の積の最終 m 行で定義される ($Q = H_1^H H_2^H \dots H_k^H$)。このルーチンは、[sgerqf/dgerqf](#) を呼び出した後で使用する。

入力パラメーター

m	INTEGER。行列 Q の行数 ($m \geq 0$)。
n	INTEGER。行列 Q の列数 ($n \geq m$)。
k	INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
$a, \tau, work$	REAL (cungrq の場合) DOUBLE PRECISION (zungrq の場合) 配列: $a(lda, *)$, $\tau(*)$, $work(lwork)$ 。 呼び出し時に、 sgerqf/dgerqf で配列引数 a の最終 k 行に返されたとおりに、 a の ($m - k + i$) 番目の行に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 $\tau(i)$ には、 sgerqf/dgerqf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 τ の次元は、 $\max(1, k)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, m)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 <code>xerbla</code> は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$m \times n$ の行列 Q によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungrq` のインターフェイスの詳細を以下に示す。

`a` サイズ (m, n) の行列 A を格納する。
`tau` 長さ (k) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork = m * blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?orgrq](#) である。

?ormrq

実行列に `?gerqf` で求めた RQ 因子分解の直交行列 Q を掛ける。

構文

Fortran 77:

```
call sormrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call dormrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call ormrmq(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の実行列 C に Q または Q^T を掛ける。 Q は、 RQ 因子分解ルーチン [sgerqf/dgerqf](#) から返された k 個の基本リフレクター H_i の積で定義される直交行列である ($Q = H_1 H_2 \cdots H_k$)。

このルーチンを使用すれば、パラメーター `side` と `trans` の値に従って、 QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

`side` CHARACTER*1。'L' または 'R' でなければならない。
 `side = 'L'` の場合、 Q または Q^T は、 C に左側から適用される。
 `side = 'R'` の場合、 Q または Q^T は、 C に右側から適用される。

<i>trans</i>	CHARACTER*1. 'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 <i>C</i> に <i>Q</i> を掛ける。 <i>trans</i> = 'T' の場合、 <i>C</i> に <i>Q^T</i> を掛ける。
<i>m</i>	INTEGER。行列 <i>C</i> の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 <i>C</i> の列数 ($n \geq 0$)。
<i>k</i>	INTEGER。その積が行列 <i>Q</i> を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ (<i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ (<i>side</i> = 'R' の場合)。
<i>a, tau, c, work</i>	REAL (<i>sormrq</i> の場合) DOUBLE PRECISION (<i>dormrq</i> の場合)。 配列: <i>a(lda,*)</i> , <i>tau(*)</i> , <i>c ldc,*)</i> , <i>work(lwork)</i> 呼び出し時に、 <i>sgerqf/dgerqf</i> で配列引数 <i>a</i> の最終 <i>k</i> 行に返されたとおりに、 <i>a</i> の <i>i</i> 番目の行に、基本リフレクター <i>H_i</i> (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 (<i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 (<i>side</i> = 'R' の場合) でなければならない。 <i>tau(i)</i> には、 <i>sgerqf/dgerqf</i> から返されたとおりに、基本リフレクター <i>H_i</i> のスカラー係数が格納されていなければならない。 <i>tau</i> の次元は、 $\max(1, k)$ 以上でなければならない。 <i>c(ldc,*)</i> には、 $m \times n$ の行列 <i>C</i> を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(lwork)</i> は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ (<i>side</i> = 'R' の場合)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) <i>QC</i> 、 <i>Q^TC</i> 、 <i>CQ</i> 、または <i>CQ^T</i> のいずれかの積によって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormrq` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (k, m) の行列 A を格納する。
<code>tau</code>	長さ (k) のベクトルを格納する。
<code>c</code>	サイズ (m, n) の行列 C を格納する。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<code>trans</code>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合) に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?unmrq](#) である。

?unmrq

複素行列に `?gerqf` で求めた RQ 因子分解のユニタリー行列 Q を掛ける。

構文

Fortran 77:

```
call cunmrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
call zunmrq(side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork, info)
```

Fortran 95:

```
call unmrq(a, tau, c [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の複素行列 C に Q または Q^H を掛ける。 Q は、 RQ 因子分解ルーチン [cgerqf/zgerqf](#) から返された k 個の基本リフレクター H_i の積で定義される複素ユニタリー行列である ($Q = H_1^H H_2^H \dots H_k^H$)。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

<i>side</i>	CHARACTER*1. 'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 Q または Q^H は、 C に左側から適用される。 <i>side</i> = 'R' の場合、 Q または Q^H は、 C に右側から適用される。
<i>trans</i>	CHARACTER*1. 'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 C に Q を掛ける。 <i>trans</i> = 'C' の場合、 C に Q^H を掛ける。
<i>m</i>	INTEGER。行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 C の列数 ($n \geq 0$)。
<i>k</i>	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ (<i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ (<i>side</i> = 'R' の場合)。
<i>a, tau, c, work</i>	COMPLEX (cunmrq の場合) DOUBLE COMPLEX (zunmrq の場合)。 配列: $a(lda, *)$, $tau(*)$, $c ldc, *$, $work(lwork)$ 呼び出し時に、cgerqf/zgerqf で配列引数 a の最終 k 行に返されたとおりに、 a の i 番目の行に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 a の第 2 次元は、 $\max(1, m)$ 以上 (<i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 (<i>side</i> = 'R' の場合) でなければならない。 $tau(i)$ には、cgerqf/zgerqff から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、 $m \times n$ の行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ (<i>side</i> = 'R' の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、xerbla は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。
----------	---

work(1) *info*=0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER。
info=0 の場合、正常に終了したことを示す。
info=-*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmrq` のインターフェイスの詳細を以下に示す。

a サイズ (*k,m*) の行列 *A* を格納する。

tau 長さ (*k*) のベクトルを格納する。

c サイズ (*m,n*) の行列 *C* を格納する。

side 'L' または 'R' でなければならない。デフォルト値は 'L'。

trans 'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (*side* = 'L' の場合) または $lwork = m * blocksize$ (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?ormrq](#) である。

?tzzrf

上台形行列 *A* を上三角形式に縮退させる。

構文

Fortran 77:

```
call stzzrf(m, n, a, lda, tau, work, lwork, info)
call dtzzrf(m, n, a, lda, tau, work, lwork, info)
call ctzzrf(m, n, a, lda, tau, work, lwork, info)
call ztzzrf(m, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call tzzrf(a [,tau] [,info])
```

説明

このルーチンは、直交 / ユニタリー変換を用いて、 $m \times n$ ($m \leq n$) の実 / 複素上台形行列 A を、上三角形式に縮退させる。上台形行列 A は、次のように因子分解される。

$$A = (R \ 0) * Z$$

Z は $n \times n$ の直交 / ユニタリー行列、 R は $m \times m$ の上三角行列である。

?tzrzf が返したとおりの基本リフレクターを一般行列に適用する。[?larz](#) を参照。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。行列 A の列数 ($n \geq m$)。
$a, work$	REAL (stzrzf の場合) DOUBLE PRECISION (dtzrzf の場合) COMPLEX (ctzrzf の場合) DOUBLE COMPLEX (ztzrzf の場合)。 配列: $a(lda, *)$, $work(lwork)$ 。 配列 a の先頭の $m \times n$ の上台形部分に、因子分解する行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work$ は、ワークスペース配列。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $lwork \geq \max(1, m)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	終了時に、因子分解データによって次のように上書きされる。 a の先頭の $m \times m$ の上三角部分に、上三角行列 R が格納される。 a の最初の m 行の $m+1$ から n までの成分は、配列 tau とともに、基本リフレクター m の積として直交行列 Z を表現する。
tau	REAL (stzrzf の場合) DOUBLE PRECISION (dtzrzf の場合) COMPLEX (ctzrzf の場合) DOUBLE COMPLEX (ztzrzf の場合)。 配列、次元は $\max(1, m)$ 以上。 行列 Z に対する基本リフレクターのスカラー係数が格納される。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tzrzf` のインターフェイスの詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。
tau 長さ (*m*) のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、`lwork = m * blocksize` に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

関連ルーチン:

[?ormrz](#) 行列 *Q* を適用する (実行列の場合)。
[?unmrz](#) 行列 *Q* を適用する (複素行列の場合)。

?ormrz

実行列に ?tzrzf で求めた直交行列を掛ける。

構文

Fortran 77:

```
call sormrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork,
           info)
call dormrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork,
           info)
```

Fortran 95:

```
call ormrz(a, tau, c, l [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の実行列 *C* に *Q* または Q^T を掛ける。*Q* は、ルーチン [stzrzf/dtzrzf](#) から返された *k* 個の基本リフレクター H_i の積で定義される直交実行列である ($Q = H_1 H_2 \cdots H_k$)。

このルーチンを使用すれば、パラメーター *side* と *trans* の値に従って、 QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。
行列 Q の次数は、 m ($side = 'L'$ の場合) または n ($side = 'R'$ の場合)。

入力パラメーター

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 Q または Q^T は、 C に左側から適用される。 <i>side</i> = 'R' の場合、 Q または Q^T は、 C に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 C に Q を掛ける。 <i>trans</i> = 'T' の場合、 C に Q^T を掛ける。
<i>m</i>	INTEGER。行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 C の列数 ($n \geq 0$)。
<i>k</i>	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ ($side = 'L'$ の場合)。 $0 \leq k \leq n$ ($side = 'R'$ の場合)。
<i>l</i>	INTEGER。 Householder リフレクターとして意味を持った部分が格納されている行列 A の列数。次の制約がある。 $0 \leq l \leq m$ ($side = 'L'$ の場合)。 $0 \leq l \leq n$ ($side = 'R'$ の場合)。
<i>a, tau, c, work</i>	REAL (<i>sormrz</i> の場合) DOUBLE PRECISION (<i>dormrz</i> の場合)。 配列: $a(lda, *)$, $tau(*)$, $c(ldc, *)$, $work(lwork)$ 呼び出し時に、 <i>stzrzf/dtzrzf</i> で配列引数 a の最終 k 行に返されたとおりに、 a の i 番目の行に、基本リフレクター H_i (ただし、 $i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 a の第 2 次元は、 $\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ 以上 ($side = 'R'$ の場合) でなければならない。 $tau(i)$ には、 <i>stzrzf/dtzrzff</i> から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 tau の次元は、 $\max(1, k)$ 以上でなければならない。 $c(ldc, *)$ には、 $m \times n$ の行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, k)$ 。
<i>ldc</i>	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>lwork</i>	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルー

チンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>side</i> と <i>trans</i> の値に従って) QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの積によって上書きされる。
<i>work</i> (1)	<i>info</i> =0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ormrz* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>k</i> , <i>m</i>) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ (<i>k</i>) のベクトルを格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (*side* = 'L' の場合) または $lwork = m * blocksize$ (*side* = 'R' の場合) に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの複素数版は、[?unmrz](#) である。

?unmrz

複素行列に ?tzrzf で求めた因子分解のユニタリー行列 Q を掛ける。

構文

Fortran 77:

```
call cunmrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork,
           info)
call zunmrz(side, trans, m, n, k, l, a, lda, tau, c, ldc, work, lwork,
           info)
```

Fortran 95:

```
call unmrz(a, tau, c, l [,side] [,trans] [,info])
```

説明

このルーチンは、 $m \times n$ の複素行列 C に Q または Q^H を掛ける。 Q は、ルーチン [ctzrzf/ztzrzf](#) から返された k 個の基本リフレクター H_i の積で定義されるユニタリー行列である ($Q = H_1^H H_2^H \dots H_k^H$)。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

行列 Q の次数は、 m ($side = 'L'$ の場合) または n ($side = 'R'$ の場合)。

入力パラメーター

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 Q または Q^H は、 C に左側から適用される。 <i>side</i> = 'R' の場合、 Q または Q^H は、 C に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 C に Q を掛ける。 <i>trans</i> = 'C' の場合、 C に Q^H を掛ける。
<i>m</i>	INTEGER。行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 C の列数 ($n \geq 0$)。
<i>k</i>	INTEGER。その積が行列 Q を定義する基本リフレクターの個数。 次の制約がある。 $0 \leq k \leq m$ (<i>side</i> = 'L' の場合)。 $0 \leq k \leq n$ (<i>side</i> = 'R' の場合)。
<i>l</i>	INTEGER。 Householder リフレクターとして意味を持った部分が格納されている行列 A の列数。次の制約がある。 $0 \leq l \leq m$ (<i>side</i> = 'L' の場合)。 $0 \leq l \leq n$ (<i>side</i> = 'R' の場合)。

$a, \tau, c, work$	<p>COMPLEX (cunmrz の場合) DOUBLE COMPLEX (zunmrz の場合)。 配列: $a(lda, *)$, $\tau(*)$, $c ldc, *$, $work(lwork)$</p> <p>呼び出し時に、ctzrzf/ztzrzf で配列引数 a の最終 k 行に返されたとおりに、a の i 番目の行に、基本リフレクター H_i (ただし、$i = 1, 2, \dots, k$) を定義するベクトルが格納されていなければならない。 a の第 2 次元は、$\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ 以上 ($side = 'R'$ の場合) でなければならない。</p> <p>$\tau(i)$ には、ctzrzf/ztzrzf から返されたとおりに、基本リフレクター H_i のスカラー係数が格納されていなければならない。 τ の次元は、$\max(1, k)$ 以上でなければならない。</p> <p>$c(ldc, *)$ には、$m \times n$ の行列 C を格納する。c の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p>$work(lwork)$ は、ワークスペース配列である。</p>
lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, k)$ 。
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
$lwork$	<p>INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、xerbla は $lwork$ に関するエラーメッセージを生成しない。</p> <p>$lwork$ の推奨値は、「アプリケーション・ノート」を参照。</p>

出力パラメーター

c	($side$ と $trans$ の値に従って) QC , $Q^H C$, CQ , または CQ^H のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	<p>INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、i 番目のパラメーターの値が不正である。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン unmrz のインターフェイスの詳細を以下に示す。

a	サイズ (k, m) の行列 A を格納する。
τ	長さ (k) のベクトルを格納する。

<code>c</code>	サイズ (m,n) の行列 C を格納する。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<code>trans</code>	'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合) に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

このルーチンの実数版は、[?ormrz](#) である。

?ggqrf

2 つの行列に対して汎用 QR 因子分解を行う。

構文

Fortran 77:

```
call sggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggqrf(n, m, p, a, lda, taua, b, ldb, taub, work, lwork, info)
```

Fortran 95:

```
call ggqrf(a, b [,taua] [,taub] [,info])
```

説明

このルーチンは、 $n \times m$ の行列 A と $n \times p$ の行列 B に対して、 $A = QR$ 、 $B = QTZ$ で表される汎用 QR 因子分解を行う。 Q は $n \times n$ の直交 / ユニタリー行列、 Z は $p \times p$ の直交 / ユニタリー行列である。また、 R と T は次のどちらかの形式である。

$$R = \begin{matrix} & & m \\ & m & \begin{pmatrix} R_{11} \\ \phantom{R_{11}} \end{pmatrix} \\ n-m & & \begin{pmatrix} \phantom{R_{11}} \\ 0 \end{pmatrix} \end{matrix}, \quad (n \geq m \text{ の場合})$$

または

$$R = \begin{matrix} & n & m-n \\ n & \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \end{matrix}, \quad (n < m \text{ の場合})$$

R_{11} は上三角行列である。

$$T = \begin{matrix} & p-n & n \\ n & (0 & T_{12}) \end{matrix}, \quad (n \leq p \text{ の場合}), \text{ または}$$

$$T = \begin{matrix} & & p \\ n-p & \left(\begin{matrix} T_{11} \\ T_{21} \end{matrix} \right) \\ p & & \end{matrix}, \quad (n > p \text{ の場合})$$

T_{12} または T_{21} は $p \times p$ の上三角行列である。

特に、行列 B が正方かつ非特異の場合、 A と B の GQR 因子分解によって、 $B^{-1}A$ の QR 因子分解が次のように得られる。

$$B^{-1}A = Z^H (T^{-1} R)$$

入力パラメーター

n	INTEGER。行列 A と B の行数 ($n \geq 0$)。
m	INTEGER。行列 A の列数 ($m \geq 0$)。
p	INTEGER。行列 B の列数 ($p \geq 0$)。
$a, b, work$	REAL (sggqrf の場合) DOUBLE PRECISION (dggqrf の場合) COMPLEX (cggqrf の場合) DOUBLE COMPLEX (zggqrf の場合)。 配列： $a(lda, *)$ には、行列 A を格納する。 a の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 $b(l db, *)$ には、行列 B を格納する。 b の第 2 次元は、 $\max(1, p)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, n, m, p)$ 以上でなければならない。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a, b	因子分解データによって次のように上書きされる。 終了時に、配列 a の対角成分とその上の成分は、 $\min(n, m) \times m$ の上台形行列 R ($n \geq m$ の場合、 R は上三角行列) によって上書きされる。対角成分よりも下の成分は、配列 $taua$ とともに、 $\min(n, m)$ 個の基本リフレクターの積として直交/ユニタリー行列 Q を表現する。
--------	--

$n \leq p$ の場合、部分配列 $b(1:n, p-n+1:p)$ の上三角部分に、 $n \times n$ の上三角行列 T が格納される。

$n > p$ の場合、 $(n-p)$ 番目の劣対角成分とその上の各成分に、 $n \times p$ の上台形行列 T が格納される。残りの各成分は、配列 $taub$ とともに、基本リフレクターの積として直交/ユニタリー行列 Z を表現する。

taua, *taub*

REAL (sggqrf の場合)

DOUBLE PRECISION (dggqrf の場合)

COMPLEX (cggqrf の場合)

DOUBLE COMPLEX (zggqrf の場合)。

配列、次元は $\max(1, \min(n, m))$ 以上 (*taua* の場合) または $\max(1, \min(n, p))$ 以上 (*taub* の場合)。

配列 *taua* には、直交/ユニタリー行列 Q を表す基本リフレクターのスカラー係数が格納される。

配列 *taub* には、直交/ユニタリー行列 Z を表す基本リフレクターのスカラー係数が格納される。

work(1)

info = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。

info

INTEGER。

info = 0 の場合、正常に終了したことを示す。

info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggqrf` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, m) の行列 A を格納する。
<i>b</i>	サイズ (n, p) の行列 B を格納する。
<i>taua</i>	長さ $\min(n, m)$ のベクトルを格納する。
<i>taub</i>	長さ $\min(n, p)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を次のように設定する。

$lwork \geq \max(n, m, p) * \max(nb1, nb2, nb3)$ 。

nb1 は $n \times m$ の行列の QR 因子分解における最適ブロックサイズ、*nb2* は $n \times p$ の行列の RQ 因子分解における最適ブロックサイズ、*nb3* は `?ormqr`/`?unmqr` を呼び出すときの最適ブロックサイズである。

?ggrqf

2つの行列に対して汎用RQ因子分解を行う。

構文

Fortran 77:

```

call sggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call dggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call cggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)
call zggrqf (m, p, n, a, lda, taua, b, ldb, taub, work, lwork, info)

```

Fortran 95:

```
call ggrqf(a, b [,taua] [,taub] [,info])
```

説明

このルーチンは、 $m \times n$ の行列 A と $p \times n$ の行列 B に対して、 $A = RQ$, $B = ZTQ$ で表される汎用RQ因子分解を行う。 Q は $n \times n$ の直交/ユニタリー行列、 Z は $p \times p$ の直交/ユニタリー行列である。また、 R と T は次のどちらかの形式である。

$$R = \begin{pmatrix} n-m & m \\ 0 & R_{12} \end{pmatrix}, \quad (m \leq n \text{ の場合}),$$

または

$$R = \begin{pmatrix} m-n & n \\ R_{11} & \\ n & R_{21} \end{pmatrix}, \quad (m > n \text{ の場合})$$

R_{11} と R_{21} は上三角行列である。

$$T = \begin{pmatrix} n & \\ n & T_{11} \\ p-n & 0 \end{pmatrix}, \quad (p \geq n \text{ の場合})$$

または

$$T = \begin{pmatrix} p & n-p \\ T_{11} & T_{12} \end{pmatrix}, \quad (p < n \text{ の場合})$$

T_{11} は上三角行列である。

特に、行列 B が正方かつ非特異の場合、 A と B のGRQ因子分解によって、 AB^{-1} のRQ因子分解が次のように得られる。

$$AB^{-1} = (R \ T^{-1}) Z^H$$

入力パラメーター

<i>m</i>	INTEGER。行列 <i>A</i> の行数 ($m \geq 0$)。
<i>p</i>	INTEGER。行列 <i>B</i> の行数 ($p \geq 0$)。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の列数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sggrqf の場合) DOUBLE PRECISION (dggrqf の場合) COMPLEX (cggrqf の場合) DOUBLE COMPLEX (zggrqf の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、 $m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、 $p \times n$ の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, p)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 $\max(1, n, m, p)$ 以上でなければならない。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i> , <i>b</i>	因子分解データによって次のように上書きされる。 終了時に、 $m \leq n$ の場合、部分配列 <i>a</i> (1: <i>m</i> , <i>n</i> - <i>m</i> +1: <i>n</i>) の上三角部分は、 $m \times m$ の上三角行列 <i>R</i> によって上書きされる。 $m > n$ の場合、(<i>m</i> - <i>n</i>) 番目の劣対角成分とその上の成分は、 $m \times n$ の上台形行列 <i>R</i> によって上書きされる。残りの各成分は、配列 <i>taua</i> とともに、基本リフレクターの積として直交/ユニタリー行列 <i>Q</i> を表現する。 配列 <i>b</i> の対角成分とその上の成分は、 $\min(p, n) \times n$ の上台形行列 <i>T</i> ($p \geq n$ の場合、 <i>T</i> は上三角行列) によって上書きされる。対角成分よりも下の成分は、配列 <i>taub</i> とともに、基本リフレクターの積として直交/ユニタリー行列 <i>Z</i> を表現する。
<i>taua</i> , <i>taub</i>	REAL (sggrqf の場合) DOUBLE PRECISION (dggrqf の場合) COMPLEX (cggrqf の場合) DOUBLE COMPLEX (zggrqf の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上 (<i>taua</i> の場合) または $\max(1, \min(p, n))$ 以上 (<i>taub</i> の場合)。 配列 <i>taua</i> には、直交/ユニタリー行列 <i>Q</i> を表す基本リフレクターのスカラー係数が格納される。

配列 *taub* には、直交/ユニタリー行列 *Z* を表す基本リフレクターのスカラー係数が格納される。

work(1) *info* = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ggqrqf* のインターフェイスの詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。
b サイズ (*p*, *n*) の行列 *A* を格納する。
taua 長さ $\min(m, n)$ のベクトルを格納する。
taub 長さ $\min(p, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork \geq \max(n, m, p) * \max(nb1, nb2, nb3)$ のように設定する。

nb1 は $m \times n$ の行列の *RQ* 因子分解における最適なブロックサイズ、*nb2* は $p \times n$ の行列の *QR* 因子分解における最適なブロックサイズ、*nb3* は *?ormrq/?unmrq* を呼び出すときの最適なブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

特異値分解

このセクションでは、一般行列 A ($m \times n$) を次のように特異値分解 (SVD) するための LAPACK ルーチンについて説明する。

$$A = U\Sigma V^H$$

この分解処理では、 U と V は、ユニタリ行列 (A が複素行列の場合) または直交行列 (A が実行列の場合) である。 Σ は、次に示す対角成分 σ_i を持つ $m \times n$ の対角行列である。

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_{\min(m, n)} \geq 0$$

対角成分 σ_i は、 A の特異値である。行列 U と V の先頭から $\min(m, n)$ の列は、それぞれ、 A の左特異ベクトルと右特異ベクトルである。対応する特異値と特異ベクトルは、次の条件を満たす。

$$Av_i = \sigma_i u_i \quad \text{かつ} \quad A^H u_i = \sigma_i v_i$$

u_i と v_i は、それぞれ、 U と V の i 番目の成分である。

一般行列 A の SVD を求めるには、まず、LAPACK ルーチンの `?gebrd` または `?gbbbrd` を呼び出し、ユニタリ (直交) 変換 $A = QBP^H$ によって A を二重対角行列 B に縮退させる。次に `?bdsqr` を呼び出し、二重対角行列 B から $B = U_1 \Sigma V_1^H$ となるような SVD を求める。

そのため、求めるべき A の SVD は、 $A = U\Sigma V^H = (QU_1) \Sigma (V_1^H P^H)$ のように表現できる。

表 4-2 に、行列の特異値分解を実行するための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 4-2 特異値分解 (SVD) 用のルーチン

演算	実行列	複素行列
A を二重対角行列 B に縮退させる。 $A = QBP^H$ (フル格納)	?gebrd	?gebrd
A を二重対角行列 B に縮退させる。 $A = QBP^H$ (帯格納)	?gbbbrd	?gbbbrd
直交 (ユニタリ) 行列 Q または P を生成する	?orgbr	?ungbr
直交 (ユニタリ) 行列 Q または P を適用する	?ormbr	?unmbr
二重対角行列 B から次の特異値分解を行う。 $B = U\Sigma V^H$?bdsqr ?bdsdc	?bdsqr

図 4-1 デシジョンツリー：特異値分解

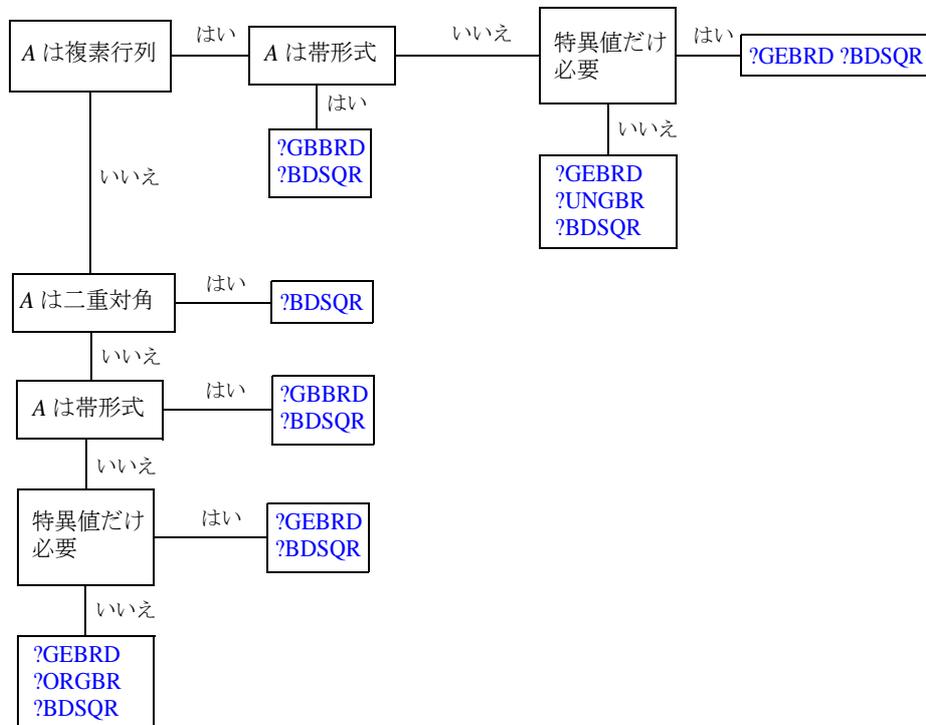


図 4-1 のデシジョンツリーを参考にすれば、特異値だけ必要な場合、特異値と特異ベクトルの両方が必要な場合、あるいは A が実数または複素数のどちらであるかなどといった条件に応じて、SVD を求めるための一連のルーチンを正しく選択できる。

SVD を使用すれば、 $\|Ax - b\|_2$ が最小になるような階数不足の最小二乗問題に対する最小ノルム解を得られる（可能な場合）。行列 A の有効な階数 k は、適切なしきい値を超えるような特異値の個数によって決定できる。最小ノルム解は、次のとおりである。

$$x = V_k(\Sigma_k)^{-1}c$$

Σ_k は、 Σ の先頭から $k \times k$ の部分行列である。行列 V_k は $V = PV_1$ の先頭から k 個の列で構成され、ベクトル c は $U^H b = U^H Q^H b$ の先頭から k 個の成分で構成される。

?gebrd

一般行列を二重対角形式に縮退させる。

構文

Fortran 77:

```
call sgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call dgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call cgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
call zgebrd(m, n, a, lda, d, e, tauq, taup, work, lwork, info)
```

Fortran 95:

```
call gebrd(a [,d] [,e] [,tauq] [,taup] [,info])
```

説明

このルーチンは、直交 (ユニタリー) 変換によって、 $m \times n$ の一般行列 A を二重対角行列 B に縮退させる。

$m \geq n$ の場合、縮退は次の式で与えられる。 $A = QBP^H = Q \begin{pmatrix} B_1 \\ 0 \end{pmatrix} P^H = Q_1 B_1 P^H$

B_1 は、 $n \times n$ の上三角行列である。 Q と P は、直交行列またはユニタリー行列 (A が複素行列の場合) である。 Q_1 は、 Q の先頭から n 個の列によって構成される。

$m < n$ の場合、縮退は次の式で与えられる。

$$A = QBP^H = Q(B_1 0)P^H = Q_1 B_1 P_1^H,$$

B_1 は、 $m \times m$ の下対角行列である。 Q と P は、直交行列またはユニタリー行列 (A が複素行列の場合) である。 P_1 は、 P の先頭から m 個の行によって構成される。

このルーチンでは、行列 Q と P を明示的な形式では表現せず、基本リフレクターの積として表現する。一連のルーチンでは、この形式で表現される行列 Q と P を操作する。

行列 A が実数型の場合、

- Q と P を明示的に求めるには、[?orgbr](#) を呼び出す。
- 一般行列に Q または P を掛けるには、[?ormbr](#) を呼び出す。

行列 A が複素数型の場合、

- Q と P を明示的に求めるには、[?ungbr](#) を呼び出す。
- 一般行列に Q または P を掛けるには、[?unmbr](#) を呼び出す。

入力パラメーター

m INTEGER。 行列 A の行数 ($m \geq 0$)。

n INTEGER。 A の列数 ($n \geq 0$)。

<i>a, work</i>	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。 <i>work</i> の次元。 $\max(1, m, n)$ 以上。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
出力パラメーター	
<i>a</i>	$m \geq n$ の場合、 <i>a</i> の対角線と第 1 の優対角成分は、上二重対角行列 <i>B</i> によって上書きされる。対角線より下の成分は <i>Q</i> の各成分によって、残りの成分は <i>P</i> の各成分によって上書きされる。 $m < n$ の場合、 <i>a</i> の対角線と第 1 の劣対角成分は、下二重対角行列 <i>B</i> によって上書きされる。対角線より上の成分は <i>P</i> の各成分によって、残りの成分は <i>Q</i> の各成分によって上書きされる。
<i>d</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 <i>B</i> の対角成分が格納される。
<i>e</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n) - 1)$ 以上。 <i>B</i> の非対角成分が格納される。
<i>tauq, taup</i>	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合)。 配列、次元は $(1, \min(m, n))$ 以上。 行列 <i>Q</i> と <i>P</i> の各成分が格納される。
<i>work</i> (1)	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gebrd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (m, n) の行列 A を格納する。
<code>d</code>	長さ $\min(m, n)$ のベクトルを格納する。
<code>e</code>	長さ $\min(m, n)-1$ のベクトルを格納する。
<code>tauq</code>	長さ $\min(m, n)$ のベクトルを格納する。
<code>taup</code>	長さ $\min(m, n)$ のベクトルを格納する。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (m + n) * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 Q 、 B 、 P は、 $QBP^H = A + E$ を満たす。

$\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、 $c(n)$ は漸増する n の関数で、 ε はマシンによって異なる値である。

実数型の場合の浮動小数演算のおおよその総数は、次のとおりである。

$(4/3) * n^2 * (3 * m - n)$ ($m \geq n$ の場合)、

$(4/3) * m^2 * (3 * n - m)$ ($m < n$ の場合)。

複素数型の場合、この 4 倍になる。

n が m に比べてはるかに小さい場合は、まず [?gqgrf](#) を呼び出して A の QR 因子分解を求めた後、因数 R を二重対角形式に縮退した方が効率が良くなる。その場合、約 $2 * n^2 * (m + n)$ 回の浮動小数演算が必要になる。

一方、 m が n に比べてはるかに小さい場合は、まず [?qlqf](#) を呼び出して A の LQ 因子分解を求めた後、因数 L を二重対角形式に縮退した方が効率が良くなる。その場合、約 $2 * m^2 * (m + n)$ 回の浮動小数演算が必要になる。

?gbbrd

一般帯行列を二重対角形式に縮退させる。

構文

Fortran 77:

```

call sgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, info)
call dgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, info)
call cgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, rwork, info)
call zgbbrd(vect, m, n, ncc, kl, ku, ab, ldab, d, e, q, ldq, pt,
           ldpt, c, ldc, work, rwork, info)

```

Fortran 95:

```
call gbbrd(a [,c] [,d] [,e] [,q] [,pt] [,kl] [,m] [,info])
```

説明

このルーチンは、 $m \times n$ の帯行列 A を $A = QBP^H$ に縮退させ、上二重対角行列 B を求める。行列 Q と P は、直交行列 (A が実数の場合) またはユニタリ行列 (A が複素数の場合) である。これらは、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使って明示的に求めることができる。このルーチンでは、 $C = Q^H C$ として行列 C の更新もできる。

入力パラメーター

<i>vect</i>	CHARACTER*1. 'N'、'Q'、'P'、または 'B' のいずれかでなければならない。 <i>vect</i> = 'N' の場合、 Q も P^H も生成されない。 <i>vect</i> = 'Q' の場合、行列 Q が生成される。 <i>vect</i> = 'P' の場合、行列 P^H が生成される。 <i>vect</i> = 'B' の場合、 Q も P^H も生成される。
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。 A の列数 ($n \geq 0$)。
<i>ncc</i>	INTEGER。行列 C の列数 ($ncc \geq 0$)。
<i>kl</i>	INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。
<i>ku</i>	INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。
<i>ab,c,work</i>	REAL (sgbbrd の場合) DOUBLE PRECISION (dgbbrd の場合) COMPLEX (cgbbrd の場合) DOUBLE COMPLEX (zgbbrd の場合)。

配列:

$ab(ldab, *)$ には、帯格納形式の行列 A (「[行列の格納形式](#)」を参照) を格納する。

a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$c(ldc, *)$ には、 $m \times ncc$ の行列 C を格納する。

$ncc = 0$ の場合、配列 c は参照されない。 c の第 2 次元は、 $\max(1, ncc)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列。

$work$ の次元は、 $2 \cdot \max(m, n)$ 以上 (実数型の場合) または $\max(m, n)$ 以上 (複素数型の場合) でなければならない。

$ldab$	INTEGER。配列 ab の第 1 次元。 $(ldab \geq kl + ku + 1)$ 。
ldq	INTEGER。出力配列 q の第 1 次元。 $vect = 'Q'$ または $'B'$ の場合、 $ldq \geq \max(1, m)$ そうでない場合、 $ldq \geq 1$ 。
$ldpt$	INTEGER。出力配列 pt の第 1 次元。 $vect = 'P'$ または $'B'$ の場合、 $ldpt \geq \max(1, n)$ そうでない場合、 $ldpt \geq 1$ 。
ldc	INTEGER。配列 c の第 1 次元。 $ldc \geq \max(1, m)$ ($ncc > 0$ の場合)、 $ldc \geq 1$ ($ncc = 0$ の場合)。
$rwork$	REAL (cgbbird の場合) DOUBLE PRECISION (zgbbrd の場合)。 ワークスペース配列、次元は $\max(m, n)$ 以上。

出力パラメーター

ab	縮退中に生成された値によって上書きされる。
d	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 行列 B の対角成分が格納される。
e	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n) - 1)$ 以上。 B の非対角成分が格納される。
q, pt	REAL (sgebrd の場合) DOUBLE PRECISION (dgebrd の場合) COMPLEX (cgebrd の場合) DOUBLE COMPLEX (zgebrd の場合)。 配列:

$q(ldq, *)$ には、 $m \times m$ の出力行列 Q が格納される。
 q の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$p(ldpt, *)$ には、 $n \times n$ の出力行列 P^H が格納される。
 pt の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gbbbrd` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。
 サイズ $(k_l + k_u + 1, n)$ の配列 *A* を格納する。

c サイズ (m, n_{cc}) の行列 *C* を格納する。

d 長さ $\min(m, n)$ のベクトルを格納する。

e 長さ $\min(m, n) - 1$ のベクトルを格納する。

q サイズ (m, m) の行列 *Q* を格納する。

pt サイズ (n, n) の行列 *PT* を格納する。

m 省略した場合、 $m = n$ とみなす。

k_l 省略した場合、 $k_l = k_u$ とみなす。

k_u $k_u = l_{da} - k_l - 1$ として復元する。

vect 引数 *q* および *pt* の存在に基づいて以下のように復元される。
vect = 'B' (*q* と *pt* の両方が存在する場合)、
vect = 'Q' (*q* が存在し、*pt* が省略された場合)、
vect = 'P' (*q* が省略され、*pt* が存在する場合)、
vect = 'N' (*q* と *pt* の両方が省略された場合)。

アプリケーション・ノート

計算で求めた行列 *Q*、*B*、*P* は、 $QB P^H = A + E$ を満たす。
 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ であり、 $c(n)$ は漸増する n の関数で、 ε はマシンによって異なる値である。

$m = n$ の場合、実数型の場合の浮動小数演算のおおよその総数は、次の合計になる。

$6 * n^2 * (k_l + k_u)$ (*vect* = 'N' かつ $n_{cc} = 0$ の場合)

$3 * n^2 * n_{cc} * (k_l + k_u - 1) / (k_l + k_u)$ (*C* を更新する場合)

$3 * n^3 * (k_l + k_u - 1) / (k_l + k_u)$ (*Q* または P^H を生成する場合)
 (両方とも生成する場合は、2 倍の値になる)。

複素数型の場合の演算回数を見積もる場合は、同じ式で係数を (6 と 3 ではなく) 20 と 10 とする。

?orgbr

?gebrd で求めた実直交行列 Q または P^T を生成する。

構文

Fortran 77:

```
call sorgbr(vect, m, n, k, a, lda, tau, work, lwork, info)
call dorgbr(vect, m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orgbr(a, tau [,vect] [,info])
```

説明

このルーチンは、ルーチン [sgebrd/dgebrd](#) で求めた直交行列 Q と P^T の全部または一部を生成する。このルーチンは、[sgebrd/dgebrd](#) を呼び出した後で使用する。引数の有効な組み合わせについては、入力パラメーターを参照。ほとんどの場合、次のように呼び出す必要がある。

$m \times m$ の行列 Q の全体を求めるには、
 call ?orgbr ('Q', m, m, n, a ...)
 (配列 a には、列が m 個以上なければならない)。

Q の先頭から n 列を求めるには ($m > n$ の場合)、
 call ?orgbr ('Q', m, n, n, a ...)

$n \times n$ の行列 P^T の全体を求めるには、
 call ?orgbr ('P', n, n, m, a ...)
 (配列 a には、行が n 個以上なければならない)。

P^T の先頭から m 行を求めるには ($m < n$ の場合)、
 call ?orgbr ('P', m, n, m, a ...)

入力パラメーター

vect	CHARACTER*1。'Q' または 'P' でなければならない。 vect = 'Q' の場合、行列 Q が生成される。 vect = 'P' の場合、行列 P^T が生成される。
m	INTEGER。 Q または P^T に必要な行数。
n	INTEGER。 Q または P^T に必要な列数。
k	INTEGER。 ?gebrd から返された A の次元のいずれか。 vect = 'Q' の場合、A の列数。 vect = 'P' の場合、A の行数。

次の制約がある。
 $m \geq 0, n \geq 0, k \geq 0$ 。
 vect = 'Q' の場合、 $k \leq n \leq m$ ($m > k$ の場合) または

	$m = n$ ($m \leq k$ の場合)。 $\text{vect} = 'P'$ の場合、 $k \leq m \leq n$ ($n > k$ の場合) または $m = n$ ($n \leq k$ の場合)。
a, work	REAL (sorgbr の場合) DOUBLE PRECISION (dorgbr の場合)。 配列： $a(\text{lda}, *)$ は、?gebrd から返された配列 a である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $\text{work}(\text{lwork})$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
tau	REAL (sorgbr の場合) DOUBLE PRECISION (dorgbr の場合)。 $\text{vect} = 'Q'$ の場合、 tauq は ?gebrd から返された配列である。 $\text{vect} = 'P'$ の場合、 taup は ?gebrd から返された配列である。 tau の次元は、 $\max(1, \min(m, k))$ 以上 ($\text{vect} = 'Q'$ の場合)、または $\max(1, \min(m, k))$ 以上 ($\text{vect} = 'P'$ の場合) でなければならない。 い。
lwork	INTEGER。配列 work のサイズ。 $\text{lwork} = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエンタリーとして返し、 xerbla は lwork に関するエラーメッセージを生成しない。 lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	vect 、 m 、 n の値に従って、直交行列 Q または P^T (あるいは対応する先頭からの一連の行または列) によって上書きされる。
$\text{work}(1)$	$\text{info} = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の lwork の値が $\text{work}(1)$ に出力される。これ以降の実行には、この lwork の値を使用する。
info	INTEGER。 $\text{info} = 0$ の場合、正常に終了したことを示す。 $\text{info} = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgbr` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
tau	長さ $\min(m, k)$ のベクトルを格納する。ここで $k = m$ ($\text{vect} = 'P'$ の場合)、 $k = n$ ($\text{vect} = 'Q'$ の場合)。
vect	'Q' または 'P' でなければならない。デフォルト値は 'Q'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = \min(m, n) * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。計算で求めた行列 Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\epsilon)$) だけ異なる。

説明で示した場合ごとの浮動小数演算のおおよその総数は、次のようになる。

Q の全体を求める場合、

$$\begin{aligned} (4/3)n(3m^2 - 3m*n + n^2) & \quad (m > n \text{ の場合}), \\ (4/3)m^3 & \quad (m \leq n \text{ の場合}). \end{aligned}$$

Q の先頭から n 列を求める場合 ($m > n$)、

$$(2/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

P^T の全体を求める場合、

$$\begin{aligned} (4/3)n^3 & \quad (m \geq n \text{ の場合}), \\ (4/3)m(3n^2 - 3m*n + m^2) & \quad (m < n \text{ の場合}). \end{aligned}$$

P^T の先頭から m 列を求める場合 ($m < n$)、

$$(2/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

このルーチンの複素数版は、[?ungbr](#) である。

?ormbr

任意の実行列に ?gebrd で求めた実直交行列 Q または P^T を掛ける。

構文

Fortran 77:

```
call sormbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
           info)
call dormbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
           info)
```

Fortran 95:

```
call ormbr(a, tau, c [,vect] [,side] [,trans] [,info])
```

説明

このルーチンは、任意の実行列 C に対して、 QC 、 Q^TC 、 CQ 、 CQ^T 、 PC 、 P^TC 、 CP 、または CP^T のいずれかの行列積を生成する。 Q と P は、ルーチン [sgebrd/dgebrd](#) を呼び出して求めた直交行列である。このルーチンを実行すると、 C の積が上書きされる。

入力パラメーター

下記の説明で、 r は Q または P^T の次数を表す。

$side = 'L'$ の場合、 $r = m$ 、 $side = 'R'$ の場合、 $r = n$ 。

<i>vect</i>	CHARACTER*1。'Q' または 'P' でなければならない。 <i>vect</i> = 'Q' の場合、 Q または Q^T が C に適用される。 <i>vect</i> = 'P' の場合、 P または P^T が C に適用される。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、乗算は C に左側から適用される。 <i>side</i> = 'R' の場合、乗算は C に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 Q または P が C に適用される。 <i>trans</i> = 'T' の場合、 Q^T または P^T が C に適用される。
<i>m</i>	INTEGER。 C の行数。
<i>n</i>	INTEGER。 C の列数。
<i>k</i>	INTEGER。 ?gebrd から返された A の次元のいずれか。 <i>vect</i> = 'Q' の場合、 A の列数。 <i>vect</i> = 'P' の場合、 A の行数。 次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。
<i>a, c, work</i>	REAL (sormbr の場合) DOUBLE PRECISION (dormbr の場合)。 配列： $a(lda, *)$ は、?gebrd から返された配列 a である。 その第 2 次元は、 $\max(1, \min(r, k))$ 以上 (<i>vect</i> = 'Q' の場合)、または $\max(1, r)$ 以上 (<i>vect</i> = 'P' の場合) でなければならない。 $c(ldc, *)$ には、行列 C を格納する。 その第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第 1 次元。次の制約がある。 $lda \geq \max(1, r)$ (<i>vect</i> = 'Q' の場合)。 $lda \geq \max(1, \min(r, k))$ (<i>vect</i> = 'P' の場合)。
<i>ldc</i>	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
<i>tau</i>	REAL (sormbr の場合) DOUBLE PRECISION (dormbr の場合) 配列、次元は $\max(1, \min(r, k))$ 以上。 <i>vect</i> = 'Q' の場合、 τ_{uq} は ?gebrd から返された配列である。 <i>vect</i> = 'P' の場合、 τ_{up} は ?gebrd から返された配列である。
<i>lwork</i>	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<i>side</i> = 'L' の場合)。 $lwork \geq \max(1, m)$ (<i>side</i> = 'R' の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルー

チンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエンタリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	(<i>vect</i> 、 <i>side</i> 、および <i>trans</i> の値に従って) QC 、 $Q^T C$ 、 CQ 、 CQ^T 、 PC 、 $P^T C$ 、 CP 、または CP^T のいずれかの積によって書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ormbr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ ($r, \min(nq, k)$) の行列 <i>A</i> を格納する。ここで $r = nq$ (<i>vect</i> = 'Q' の場合) $r = \min(nq, k)$ (<i>vect</i> = 'P' の場合) $nq = m$ (<i>side</i> = 'L' の場合) $nq = n$ (<i>side</i> = 'R' の場合) $k = m$ (<i>vect</i> = 'P' の場合) $k = n$ (<i>vect</i> = 'Q' の場合)。
<i>tau</i>	長さ $\min(nq, k)$ のベクトルを格納する。
<i>c</i>	サイズ (m, n) の行列 <i>C</i> を格納する。
<i>vect</i>	'Q' または 'P' でなければならない。デフォルト値は 'Q'。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、

$lwork = n * blocksize$ (*side* = 'L' の場合) または $lwork = m * blocksize$ (*side* = 'R' の場合)

に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\epsilon) \|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、

$2 * n * k(2 * m - k)$	(<i>side</i> = 'L' かつ $m \geq k$ の場合)、
$2 * m * k(2 * n - k)$	(<i>side</i> = 'R' かつ $n \geq k$ の場合)、
$2 * m^2 * n$	(<i>side</i> = 'L' かつ $m < k$ の場合)、
$2 * n^2 * m$	(<i>side</i> = 'R' かつ $n < k$ の場合)。

このルーチンの複素数版は、[?unmbr](#) である。

?ungbr

?gebrd で求めた複素ユニタリ行列 Q または P^H を生成する。

構文

Fortran 77:

```
call cungbr(vect, m, n, k, a, lda, tau, work, lwork, info)
call zungbr(vect, m, n, k, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call ungbr(a, tau [,vect] [,info])
```

説明

このルーチンは、ルーチン [cgebrd/zgebrd](#) で求めたユニタリ行列 Q と P^H の全部または一部を生成する。このルーチンは、[cgebrd/zgebrd](#) を呼び出した後で使用される。引数の有効な組み合わせについては、[入力パラメーター](#)を参照。ほとんどの場合、次のように呼び出す必要がある。

$m \times m$ の行列 Q の全体を求めるには、
`call ?ungbr ('Q', m, m, n, a ...)`
 (配列 a には、列が m 個以上なければならない)。

Q の先頭から n 列を求めるには ($m > n$ の場合)、
`call ?ungbr ('Q', m, n, n, a ...)`

$n \times n$ の行列 P^H の全体を求めるには、
`call ?ungbr ('P', n, n, m, a ...)`
 (配列 a には、行が n 個以上なければならない)。

P^H の先頭から m 行を求めるには ($m < n$ の場合)、
`call ?ungbr ('P', m, n, m, a ...)`

入力パラメーター

<i>vect</i>	CHARACTER*1. 'Q' または 'P' でなければならない。 <i>vect</i> = 'Q' の場合、行列 Q が生成される。 <i>vect</i> = 'P' の場合、行列 P^H が生成される。
<i>m</i>	INTEGER。 Q または P^H に必要な行数。
<i>n</i>	INTEGER。 Q または P^H に必要な列数。
<i>k</i>	INTEGER。 ?gebrd から返された A の次元のいずれか。 <i>vect</i> = 'Q' の場合、 A の列数。 <i>vect</i> = 'P' の場合、 A の行数。 次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。 <i>vect</i> = 'Q' の場合、 $k \leq n \leq m$ ($m > k$ の場合) または $m = n$ ($m \leq k$ の場合)。 <i>vect</i> = 'P' の場合、 $k \leq m \leq n$ ($n > k$ の場合) または $m = n$ ($n \leq k$ の場合)。
<i>a, work</i>	COMPLEX (cungbr の場合) DOUBLE COMPLEX (zungbr の場合)。 配列: <i>a</i> (<i>lda</i> ,*) は、?gebrd から返された配列 <i>a</i> である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>tau</i>	COMPLEX (cungbr の場合) DOUBLE COMPLEX (zungbr の場合)。 <i>vect</i> = 'Q' の場合、 <i>tauq</i> は ?gebrd から返された配列である。 <i>vect</i> = 'P' の場合、 <i>taup</i> は ?gebrd から返された配列である。 <i>tau</i> の次元は、 $\max(1, \min(m, k))$ 以上 (<i>vect</i> = 'Q' の場合)、または $\max(1, \min(m, k))$ 以上 (<i>vect</i> = 'P' の場合) でなければならない。
<i>lwork</i>	INTEGER。 配列 <i>work</i> のサイズ。 次の制約がある。 $lwork \geq \max(1, \min(m, n))$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、xerbla は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	<i>vect</i> 、 <i>m</i> 、 <i>n</i> の値に従って、直交行列 Q または P^T (あるいは対応する先頭からの一連の行または列) によって上書きされる。
<i>work</i> (1)	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungbr` のインターフェイスの詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。
tau 長さ $\min(m, k)$ のベクトルを格納する。ここで
 $k = m$ (*vect* = 'P' の場合)、
 $k = n$ (*vect* = 'Q' の場合)。
vect 'Q' または 'P' でなければならない。デフォルト値は 'Q'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = \min(m, n) * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。計算で求めた行列 *Q* は、正確な直交行列と行列 *E* ($\|E\|_2 = O(\epsilon)$) だけ異なる。

説明で示した場合ごとの浮動小数演算のおおよその総数は、次のようになる。

Q の全体を求める場合、

$$(16/3)n(3m^2 - 3m*n + n^2) \quad (m > n \text{ の場合}),$$

$$(16/3)m^3 \quad (m \leq n \text{ の場合}).$$

Q の先頭から *n* 列を求める場合 ($m > n$)、

$$(8/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

P^T の全体を求める場合、

$$(16/3)n^3 \quad (m \geq n \text{ の場合}),$$

$$(16/3)m(3n^2 - 3m*n + m^2) \quad (m < n \text{ の場合}).$$

P^T の先頭から *m* 列を求める場合 ($m < n$)、

$$(8/3)n^2(3m - n^2) \quad (m > n \text{ の場合}).$$

このルーチンの実数版は、`?orgbr` である。

?unmbr

任意の複素行列に ?gebrd で求めた
ユニタリー行列 Q または P を掛ける。

構文

Fortran 77:

```
call cunmbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
           info)
call zunmbr(vect, side, trans, m, n, k, a, lda, tau, c, ldc, work, lwork,
           info)
```

Fortran 95:

```
call unmbr(a, tau, c [,vect] [,side] [,trans] [,info])
```

説明

このルーチンは、任意の複素行列 C に対して、 QC 、 $Q^H C$ 、 CQ 、 CQ^H 、 PC 、 $P^H C$ 、 CP 、
または CP^H のいずれかの行列積を生成する。 Q と P は、ルーチン [cgebrd/zgebrd](#) を呼
び出して求めた直交行列である。このルーチンを実行すると、 C の積が上書きされる。

入力パラメーター

下記の説明で、 r は Q または P^H の次数を表す。

$side = 'L'$ の場合、 $r = m$ 、 $side = 'R'$ の場合、 $r = n$ 。

<i>vect</i>	CHARACTER*1。'Q' または 'P' でなければならない。 <i>vect</i> = 'Q' の場合、 Q または Q^H が C に適用される。 <i>vect</i> = 'P' の場合、 P または P^H が C に適用される。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、乗算は C に左側から適用される。 <i>side</i> = 'R' の場合、乗算は C に右側から適用される。
<i>trans</i>	CHARACTER*1。'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 Q または P が C に適用される。 <i>trans</i> = 'C' の場合、 Q^H または P^H が C に適用される。
<i>m</i>	INTEGER。 C の行数。
<i>n</i>	INTEGER。 C の列数。
<i>k</i>	INTEGER。 ?gebrd から返された A の次元のいずれか。 <i>vect</i> = 'Q' の場合、 A の列数。 <i>vect</i> = 'P' の場合、 A の行数。 次の制約がある。 $m \geq 0, n \geq 0, k \geq 0$ 。
<i>a, c, work</i>	COMPLEX (cunmbr の場合) DOUBLE COMPLEX (zunmbr の場合)。

配列：

$a(lda,*)$ は、`?gebrd` から返された配列 a である。
その第 2 次元は、 $\max(1, \min(r, k))$ 以上 ($vect = 'Q'$ の場合)、または $\max(1, r)$ 以上 ($vect = 'P'$ の場合) でなければならない。

$c(ldc,*)$ には、行列 C を格納する。
その第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda	INTEGER。 a の第 1 次元。次の制約がある。 $lda \geq \max(1, r)$ ($vect = 'Q'$ の場合)。 $lda \geq \max(1, \min(r, k))$ ($vect = 'P'$ の場合)。
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, m)$ 。
tau	COMPLEX (<code>cunmbr</code> の場合) DOUBLE COMPLEX (<code>zunmbr</code> の場合)。 配列、次元は $\max(1, \min(r, k))$ 以上。 $vect = 'Q'$ の場合、 $tauq$ は <code>?gebrd</code> から返された配列である。 $vect = 'P'$ の場合、 $taup$ は <code>?gebrd</code> から返された配列である。
$lwork$	INTEGER。配列 $work$ のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 <code>xerbla</code> は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c	($vect$ 、 $side$ 、および $trans$ の値に従って) QC 、 $Q^H C$ 、 CQ 、 CQ^H 、 PC 、 $P^H C$ 、 CP 、または CP^H のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmbr` のインターフェイスの詳細を以下に示す。

a	サイズ $(r, \min(nq, k))$ の行列 A を格納する。ここで $r = nq$ ($vect = 'Q'$ の場合) $r = \min(nq, k)$ ($vect = 'P'$ の場合)
-----	---

$nq = m$ ($side = 'L'$ の場合)
 $nq = n$ ($side = 'R'$ の場合)
 $k = m$ ($vect = 'P'$ の場合)
 $k = n$ ($vect = 'Q'$ の場合)。

tau 長さ $\min(nq, k)$ のベクトルを格納する。
 c サイズ (m, n) の行列 C を格納する。
 $vect$ ' Q ' または ' P ' でなければならない。デフォルト値は ' Q '。
 $side$ ' L ' または ' R ' でなければならない。デフォルト値は ' L '。
 $trans$ ' N ' または ' C ' でなければならない。デフォルト値は ' N '。

アプリケーション・ノート

パフォーマンスを改善するには、

$lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合)

に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\epsilon) \|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、

$8 * n * k (2 * m - k)$ ($side = 'L'$ かつ $m \geq k$ の場合)、
 $8 * m * k (2 * n - k)$ ($side = 'R'$ かつ $n \geq k$ の場合)、
 $8 * m^2 * n$ ($side = 'L'$ かつ $m < k$ の場合)、
 $8 * n^2 * m$ ($side = 'R'$ かつ $n < k$ の場合)。

このルーチンの実数版は、[?ormbr](#) である。

?bdsqr

二重対角形式に縮退された一般行列の特異値分解を求める。

構文

Fortran 77:

```

call sbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,
            c, ldc, work, info)
call dbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,
            c, ldc, work, info)
call cbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,
            c, ldc, work, info)
call zbdsqr(uplo, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu,
            c, ldc, work, info)

```

Fortran 95:

```

call rbdsqr(d, e [,vt] [,u] [,c] [,uplo] [,info])
call bdsqr(d, e [,vt] [,u] [,c] [,uplo] [,info])

```

説明

このルーチンは、暗黙的ゼロシフト *QR* アルゴリズムを使用して、 $n \times n$ の実上/下二重対角行列 B に対して **特異値分解 (SVD)** を行い、その特異値を求める。オプションで、右と左の特異ベクトルを求めることができる。行列 B の SVD は、 $B = Q * S * P^H$ と表せられる。 S は特異値を持つ対角行列、 Q は左特異ベクトルを持つ直交行列、 P は右特異ベクトルを持つ直交行列である。このサブルーチンは、与えられた実/複素入力行列 U と VT に関して、左特異ベクトルが要求されると Q ではなく $U * Q$ を返し、右特異ベクトルが要求されると P^H ではなく $P^H * VT$ を返す。 U と VT が、`?gebrd` で計算されるような、一般行列 A を対角形式に縮退させた直交/ユニタリー行列の場合、($A = U * B * VT$)、 A の SVD は次のようになる。 $A = (U * Q) * S * (P^H * VT)$ このサブルーチンは、オプションで、実/複素入力行列 C に対する $Q^H * C$ を計算できる。

参照

[?lasq1](#)、[?lasq2](#)、[?lasq3](#)、[?lasq4](#)、[?lasq5](#)、[?lasq6](#)。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 B は上二重対角行列である。 <code>uplo = 'L'</code> の場合、 B は下二重対角行列である。
<code>n</code>	INTEGER。行列 B の次数 ($n \geq 0$)。
<code>ncvt</code>	INTEGER。行列 VT の列数、つまり、右特異ベクトルの個数 ($ncvt \geq 0$)。 右特異ベクトルが不要な場合、 <code>ncvt = 0</code> に設定する。

<i>nru</i>	INTEGER。 <i>U</i> の行数、つまり左特異ベクトルの個数 ($nru \geq 0$)。左特異ベクトルが不要な場合、 $nru = 0$ に設定する。
<i>ncc</i>	INTEGER。積 $Q^H C$ を計算するために使用する行列 <i>C</i> の列数 ($ncc \geq 0$)。行列 <i>C</i> を指定しない場合、 $ncc = 0$ に設定する。
<i>d, e, work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列： <i>d</i> (*) には、 <i>B</i> の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、 <i>B</i> の ($n-1$) の非対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (<i>n</i>) はワークスペース用に使用される。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、 $\max(1, 2*n)$ 以上 ($ncvt = nru = ncc = 0$ の場合) または $\max(1, 4*(n-1))$ 以上 (それ以外の場合) でなければならない。
<i>vt, u, c</i>	REAL (sbdsqr の場合) DOUBLE PRECISION (dbdsqr の場合) COMPLEX (cbdsqr の場合) DOUBLE COMPLEX (zbdbsqr の場合)。 配列： <i>vt</i> (<i>ldvt</i> , *) には、 $n \times ncvt$ の行列 <i>VT</i> を格納する。 <i>vt</i> の第 2 次元は、 $\max(1, ncvt)$ 以上でなければならない。 $ncvt = 0$ の場合、 <i>vt</i> は参照されない。 <i>u</i> (<i>ldu</i> , *) には、 $nru \times n$ の単位行列 <i>U</i> を格納する。 <i>u</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $nru = 0$ の場合、 <i>u</i> は参照されない。 <i>c</i> (<i>ldc</i> , *) には、積 $Q^H * C$ を計算するための行列 <i>C</i> を格納する。 <i>c</i> の第 2 次元は、 $\max(1, ncc)$ 以上でなければならない。 $ncc = 0$ の場合、対応する配列は参照されない。
<i>ldvt</i>	INTEGER。 <i>vt</i> の第 1 次元。次の制約がある。 $ldvt \geq \max(1, n)$ ($ncvt > 0$ の場合)。 $ldvt \geq 1$ ($ncvt = 0$ の場合)。
<i>ldu</i>	INTEGER。 <i>u</i> の第 1 次元。次の制約がある。 $ldu \geq \max(1, nru)$ 。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。次の制約がある。 $ldc \geq \max(1, n)$ ($ncc > 0$ の場合)。 $ldc \geq 1$ (その他の場合)。

出力パラメーター

<i>d</i>	終了時に、 <i>info</i> = 0 の場合、特異値によって降順に上書きされる (<i>info</i> を参照)。
<i>e</i>	終了時に、 <i>info</i> = 0 の場合、 <i>e</i> は破棄される。後述の <i>info</i> も参照。

<i>c</i>	積 $Q^H * C$ によって上書きされる。
<i>vt</i>	終了時に、 $P^H * VT$ によって上書きされる。
<i>u</i>	終了時に、 $U * Q$ によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、アルゴリズムが収束していない。 <i>i</i> は、収束しなかった非対角成分の個数を示す。 その場合、終了時に、 <i>d</i> と <i>e</i> に、二重対角行列 (<i>B</i> の直交行列と等価) の対角成分と非対角成分が格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `bdsqr` のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>vt</i>	サイズ (<i>n</i> , <i>ncvt</i>) の行列 <i>VT</i> を格納する。
<i>u</i>	サイズ (<i>nru</i> , <i>n</i>) の行列 <i>U</i> を格納する。
<i>c</i>	サイズ (<i>n</i> , <i>ncc</i>) の行列 <i>C</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>ncvt</i>	引数 <i>vt</i> を指定する場合、 <i>ncvt</i> は行列 <i>VT</i> の列数と同じ。それ以外の場合、 <i>ncvt</i> にはゼロが設定される。
<i>nru</i>	引数 <i>u</i> を指定する場合、 <i>nru</i> は行列 <i>U</i> の行数と同じ。それ以外の場合、 <i>nru</i> にはゼロが設定される。
<i>ncc</i>	引数 <i>c</i> を指定する場合、 <i>ncc</i> は行列 <i>C</i> の列数と同じ。それ以外の場合、 <i>ncc</i> にはゼロが設定される。

vt、*u*、および *c* が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`bdsqr` ルーチンには2つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rbdsqr`、複素数の場合 (単精度または倍精度) は `bdsqr` を使用する。

アプリケーション・ノート

それぞれの特異値や特異ベクトルは、比較的高い精度で計算される。ただし、(このルーチンが呼び出される前に実行される) 二重対角形式への縮退によって、特異値の絶対値が大きく変動するような場合には、元の行列の特異値が小さいときに相対精度が低下する場合もある。

σ_i が *B* の正確な特異値で、 s_i が対応する計算で求めた値の場合、次のようになる。

$$|s_i - \sigma_i| \leq p(m, n) \epsilon \sigma_i$$

$p(m, n)$ は、漸増する m と n の関数で、 ε はマシン精度である。特異値だけを計算するほうが、特異ベクトルも一緒に計算する (つまり、関数 $p(m, n)$ が小さい) よりも結果は正確である。

u_i が B の対応する正確な左特異ベクトルで、 w_i が対応する計算で求めた左特異ベクトルの場合、それらの間の角 $\theta(u_i, w_i)$ は次のようになる。

$$\theta(u_i, w_i) \leq p(m, n)\varepsilon / \min_{i \neq j} (|\sigma_i - \sigma_j| / |\sigma_i + \sigma_j|)$$

$\min_{i \neq j} (|\sigma_i - \sigma_j| / |\sigma_i + \sigma_j|)$ は、 σ_i と他の特異値との間の相対誤差である。右特異ベクトルに関しても、これと同様に誤差が制限される。

特異値だけを計算する場合、実数型の浮動小数演算のおおよその総数は、 n^2 に比例する。左特異ベクトルを計算する場合は約 $6n^2 * nru$ (複素数型の場合は約 $12n^2 * nru$) 回だけ、右特異ベクトルを計算する場合は約 $6n^2 * ncvt$ (複素数型の場合は約 $12n^2 * ncvt$) 回だけ、余分な計算が必要になる。

?bdsdc

分割統治法を使用して、実二重対角行列の特異値分解を実行する。

構文

Fortran 77:

```
call sbdsdc(uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work,
            iwork, info)
call dbdsdc(uplo, compq, n, d, e, u, ldu, vt, ldvt, q, iq, work,
            iwork, info)
```

Fortran 95:

```
call bdsdc(d, e [,u] [,vt] [,q] [,iq] [,uplo] [,info])
```

説明

このルーチンは、分割統治法を使用して、 $n \times n$ の実上/二重対角行列 B に対する [特異値分解](#) (SVD)、すなわち $B = U \Sigma V^T$ を計算する。 Σ は非負の対角成分 (B の特異値) を持つ対角行列、 U と V は、それぞれ左と右の特異ベクトルを持つ直交行列である。?bdsdc を使用して、すべての特異値を計算し、オプションで特異ベクトル (またはコンパクト形式の特異ベクトル) を求めることができる。

参照

[?lasd0](#)、[?lasd1](#)、[?lasd2](#)、[?lasd3](#)、[?lasd4](#)、[?lasd5](#)、[?lasd6](#)、[?lasd7](#)、[?lasd8](#)、[?lasd9](#)、[?lasda](#)、[?lasdq](#)、[?lasdt](#)。

入力パラメーター

`uplo` CHARACTER*1。'U' または 'L' でなければならない。
`uplo = 'U'` の場合、 B は上二重対角行列である。
`uplo = 'L'` の場合、 B は下二重対角行列である。

<i>compq</i>	CHARACTER*1. 'N'、'P'、または 'I' でなければならない。 <i>compq</i> = 'N' の場合、特異値のみを計算する。 <i>compq</i> = 'P' の場合、特異値とコンパクト形式の特異ベクトルを計算する。 <i>compq</i> = 'I' の場合、特異値と特異ベクトルを計算する。
<i>n</i>	INTEGER。行列 <i>B</i> の次数 ($n \geq 0$)。
<i>d</i> , <i>e</i> , <i>work</i>	REAL (sbdsdc の場合) DOUBLE PRECISION (dbdsdc の場合)。 配列: <i>d</i> (*) には、二重対角行列 <i>B</i> の <i>n</i> 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、二重対角行列 <i>B</i> の非対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、 $\max(1, 4*n)$ 以上 (<i>compq</i> = 'N' の場合) または $\max(1, 6*n)$ 以上 (<i>compq</i> = 'P' の場合) または $\max(1, 3*n^2+4*n)$ 以上 (<i>compq</i> = 'I' の場合) でなければならない。
<i>ldu</i>	INTEGER。出力配列 <i>u</i> の第 1 次元。 $ldu \geq 1$ 。特異ベクトルも要求する場合、 $ldu \geq \max(1, n)$ 。
<i>ldvt</i>	INTEGER。出力配列 <i>vt</i> の第 1 次元。 $ldvt \geq 1$ 。特異ベクトルも要求する場合、 $ldvt \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。ワークスペース配列。サイズは $\max(1, 8*n)$ 以上。

出力パラメーター

<i>d</i>	<i>info</i> = 0 の場合、 <i>B</i> の特異値によって上書きされる。
<i>e</i>	終了時に、 <i>e</i> は上書きされる。
<i>u</i> , <i>vt</i> , <i>q</i>	REAL (sbdsdc の場合) DOUBLE PRECISION (dbdsdc の場合)。 配列 : <i>u</i> (<i>ldu</i> , *), <i>vt</i> (<i>ldvt</i> , *), <i>q</i> (*)。 <i>compq</i> = 'I' の場合、終了時に、二重対角行列 <i>B</i> の左特異ベクトルが <i>u</i> に格納される (ただし、 <i>info</i> ≠ 0 でない場合 (<i>info</i> を参照))。 <i>compq</i> が他の値の場合、 <i>u</i> は参照されない。 <i>u</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>compq</i> = 'I' の場合、終了時に、二重対角行列 <i>B</i> の右特異ベクトルが <i>vt</i> に格納される (ただし、 <i>info</i> ≠ 0 でない場合 (<i>info</i> を参照))。 <i>compq</i> が他の値の場合、 <i>vt</i> は参照されない。 <i>vt</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>compq</i> = 'P' の場合、終了時に、 <i>info</i> = 0 の場合、 <i>q</i> と <i>iq</i> に左と右の特異ベクトルがコンパクト形式で格納される。 <i>q</i> には、特異ベクトルのすべての REAL データ (sbdsdc の場合) または DOUBLE PRECISION データ (dbdsdc の場合) が格納される。 <i>compq</i> が他の値の場合、 <i>q</i> は参照されない。詳細は、「アプリケーション・ノート」を参照。

<i>iq</i>	INTEGER。 配列: <i>iq</i> (*)。 <i>compq</i> ='P' の場合、終了時に、 <i>info</i> =0 の場合、 <i>q</i> と <i>iq</i> に左と右の特異ベクトルがコンパクト形式で格納される。 <i>iq</i> には、特異ベクトルのすべての INTEGER データが格納される。 <i>compq</i> が他の値の場合、 <i>iq</i> は参照されない。詳細は、「アプリケーション・ノート」を参照。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムによる特異値計算に失敗したことを示す。分割統治法の更新プロセスに失敗した。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *bdsdc* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>u</i>	サイズ (<i>n,n</i>) の行列 <i>U</i> を格納する。
<i>vt</i>	サイズ (<i>n,n</i>) の行列 <i>VT</i> を格納する。
<i>q</i>	長さ (<i>ldq</i>) のベクトルを格納する。ここで $ldq \geq n * (11 + 2 * smlsiz + 8 * \text{int}(\log_2(n/(smlsiz + 1))))$ および <i>smlsiz</i> は <i>laenv</i> によって返された値で、計算ツリーの一番下の部分問題の最大サイズ (通常は約 25) と等しい。
<i>compq</i>	引数 <i>u</i> 、 <i>vt</i> 、 <i>q</i> 、および <i>iq</i> の存在に基づいて以下のように復元される。 <i>compq</i> ='N' (<i>u</i> 、 <i>vt</i> 、 <i>q</i> 、および <i>iq</i> がすべて存在しない場合)、 <i>compq</i> ='I' (<i>u</i> と <i>vt</i> の両方が存在する場合)。引数 <i>u</i> と <i>vt</i> は、両方存在するか、両方省略されるかのいずれかでなければならない。 <i>compq</i> ='P' (<i>q</i> と <i>iq</i> の両方が存在する場合)。引数 <i>q</i> と <i>iq</i> は、両方存在するか、両方省略されるかのいずれかでなければならない。 引数 <i>u</i> 、 <i>vt</i> 、 <i>q</i> 、および <i>iq</i> がすべて同時に存在する場合、エラー条件がセットされる。

対称固有値問題

対称固有値問題は、「 $n \times n$ の実対称行列または複素エルミート行列 A が与えられたときに、次の方程式を満たす固有値 λ と、それに対応する固有ベクトル z を見つけること」である。

$$Az = \lambda z \quad (\text{または } z^H A = \lambda z^H)$$

このような固有値問題の場合、実対称行列だけでなく複素エルミート行列 A に関しても、 n 個の固有値はすべて実数型になる。そして、 n 個の固有ベクトルから構成される正規直交系が 1 つ存在する。 A が対称 / エルミート正定値行列である場合は、どの固有値も正になる。

LAPACK を使って対称固有値問題を解く場合は、通常、対応する行列をまず三重対角形式に縮退した後、得られた三重対角行列を使って固有値問題を解く必要がある。

LAPACK には、直交 (またはユニタリー) の相似変換 $A = QTQ^H$ によって行列を三重対角形式に縮退させるためのルーチンだけでなく、三重対角の対称固有値問題を解くためのルーチンも含まれている。それらのルーチン (Fortran-77 インターフェイス) を、[表 4-3](#) に示す。

Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない ([「ルーチン命名規則」](#) を参照)。

対称固有値問題を解くためのルーチンには、固有ベクトルの全部あるいは一部が必要な場合は、固有値のみが必要な場合、行列 A が正定値であるかどうかなどに応じて、さまざまなものが用意されている。

これらのルーチンは、対称問題で固有値と固有ベクトルを計算する 3 つの主要アルゴリズムに基づいている。3 つの主要アルゴリズムとは、分割統治アルゴリズム、QR アルゴリズム、二分法と逆反復法の組み合わせである。一般に、分割統治アルゴリズムが最も効率が良いので、固有値と固有ベクトルをすべて計算するときには、このアルゴリズムが推奨される。

また、分割統治アルゴリズムを使用して固有値問題を解く場合は、ルーチンを 1 つだけ呼び出せばよい。一般に、QR アルゴリズム、または二分法と逆反復法の組み合わせを使用すると、複数のルーチン呼び出す必要がある。

[図 4-2](#) のデシジョンツリーを参考にすれば、実対称行列の固有値問題を解くための正しいルーチン (1 つまたは複数個) を簡単に選べる。[図 4-3](#) は、複素エルミート行列の場合のデシジョンツリーである。

図 4-2 デシジョンツリー：実対称行列の固有値問題

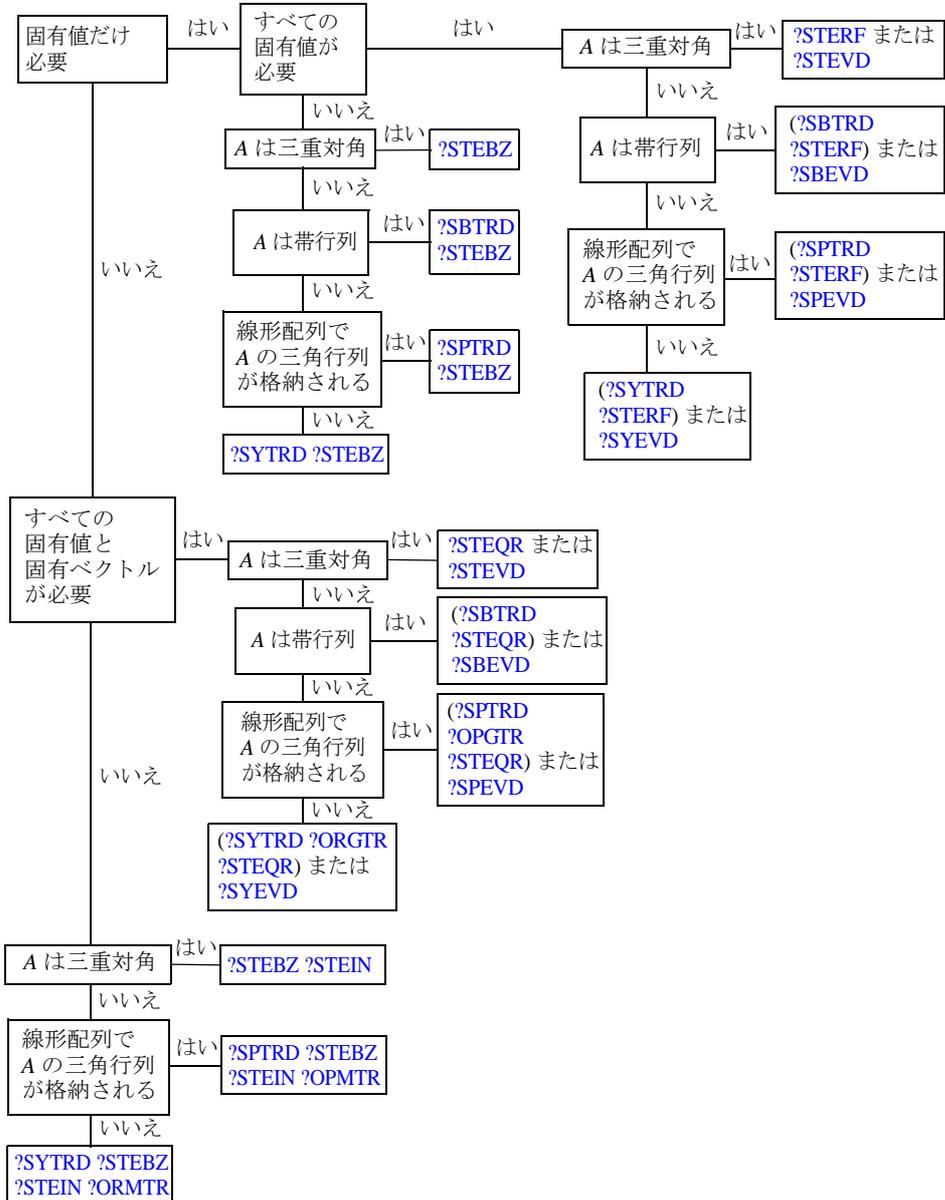


図 4-3 デシジョンツリー：複素エルミート行列の固有値問題

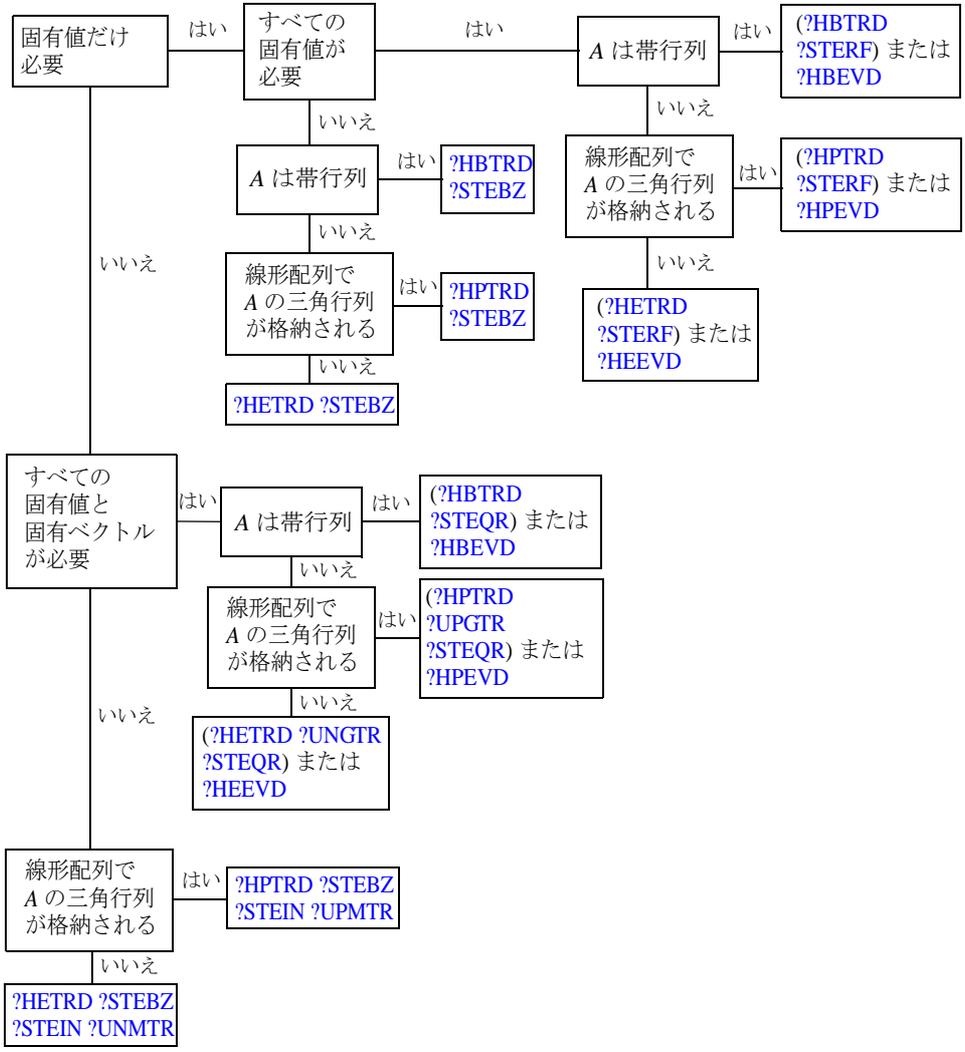


表 4-3 対称固有値問題用の計算ルーチン

演算	実対称行列		複素エルミート行列	
三重対角形式に縮退させる $A = QTQ^H$ (フル格納)	?sytrd		?hetrd	
三重対角形式に縮退させる $A = QTQ^H$ (圧縮格納)	?sptrd		?hptrd	
三重対角形式に縮退させる $A = QTQ^H$ (帯格納)	?sbtrd		?hbtrd	
行列 Q を生成する (フル格納)	?orqtr		?unqtr	
行列 Q を生成する (圧縮格納)	?opqtr		?upqtr	
行列 Q を適用する (フル格納)	?ormtr		?unmtr	
行列 Q を適用する (圧縮格納)	?opmtr		?upmtr	
三重対角行列 T のすべての固有値を見つける	?sterf			
三重対角行列 T のすべての固有値と固有ベクトルを見つける	?stegr	?stedc	?stegr	?stedc
三重対角正定値行列 T のすべての固有値と固有ベクトルを見つける	?ptegr		?ptegr	
	?stebz		?stegr	
三重対角行列 T の指定された固有値を求める	?stegr			
三重対角行列 T の指定された固有ベクトルを求める	?stein		?stein	
	?stegr		?stegr	
固有ベクトルの相互条件数を計算する	?disna		?disna	

?sytrd

実対称行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call ssytrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
call dsytrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
```

Fortran 95:

```
call sytrd(a, tau [,uplo] [,info])
```

説明

このルーチンは、直交相似変換 $A = QTQ^T$ によって、実対称行列 A を対称三重対角形式 T に縮退させる。直交行列 Q は、明示的な形式では表現されず、 $(n-1)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する (このセクションで後述)。

このルーチン [?latrd](#) で、直交相似変換によって、実対称行列を三重対角形式 T に縮退させる。

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
 uplo = 'U' の場合、a には A の上三角部分を格納する。
 uplo = 'L' の場合、a には A の下三角部分を格納する。

n INTEGER。行列 A の次数 ($n \geq 0$)。

a, work REAL (ssytrd の場合)
 DOUBLE PRECISION (dsytrd の場合)。
 a(lda,*) は、uplo の値に従って、行列 A の上三角部分または下三角部分を格納する配列である。
 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 work(lwork) は、ワークスペース配列である。

lda INTEGER。a の第 1 次元。 $\max(1, n)$ 以上。

lwork INTEGER。配列 work のサイズ ($lwork \geq n$)。
 lwork = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは work 配列の最適サイズだけを計算し、その値を work 配列の最初のエンタリーとして返し、xerbla は lwork に関するエラーメッセージを生成しない。
 lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a uplo の値に従って、三重対角行列 T と直交行列 Q の各成分によって上書きされる。

d, e, τ	REAL (ssytrd の場合) DOUBLE PRECISION (dsytrd の場合)。 配列: $d(*)$ には、行列 T の対角成分が格納される。 d の次元は、 $\max(1, n)$ 以上でなければならない。 $e(*)$ には、 T の非対角成分が格納される。 e の次元は、 $\max(1, n-1)$ 以上でなければならない。 $\tau(*)$ には、直交行列 Q の各成分が格納される。 τ の次元は、 $\max(1, n-1)$ 以上でなければならない。
$work(1)$	$info=0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sytrd` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
τ	長さ $(n-1)$ のベクトルを格納する。
d	長さ (n) のベクトルを格納する。
e	長さ $(n-1)$ のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 T は、行列 $A + E$ の近似値である。ここで、 $\|E\|_2 = c(n)\epsilon \|A\|_2$ 、 $c(n)$ は漸増する n の関数、 ϵ はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(4/3)n^3$ である。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?orgtr](#) 計算で求める行列 Q を明示的に生成する場合

[?ormtr](#) 実行列に Q を掛ける場合

このルーチンの複素数版は、[?hetrd](#) である。

?orgtr

?sytrd で求めた実直交行列 Q を生成する。

構文

Fortran 77:

```
call sorgtr(uplo, n, a, lda, tau, work, lwork, info)
call dorgtr(uplo, n, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orgtr(a, tau [,uplo] [,info])
```

説明

このルーチンは、実対称行列 A を三重対角行列の形式 $A = QTQ^T$ に縮退させるために [?sytrd](#) によって求めた $n \times n$ の直交行列 Q を明示的に生成する。このルーチンは、[?sytrd](#) を呼び出した後で使用する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 ?sytrd に指定した <code>uplo</code> と同じ値を使用する。
<code>n</code>	INTEGER。行列 Q の次数 ($n \geq 0$)。
<code>a, tau, work</code>	REAL (sorgtr の場合) DOUBLE PRECISION (dorgtr の場合)。 配列： <code>a(lda,*)</code> は、?sytrd から返された配列 <code>a</code> である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>tau(*)</code> は、?sytrd から返された配列 <code>tau</code> である。 <code>tau</code> の次元は、 $\max(1, n-1)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ ($lwork \geq n$)。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエンタリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。 <code>lwork</code> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<code>a</code>	直交行列 Q によって上書きされる。
<code>work(1)</code>	<code>info = 0</code> の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <code>lwork</code> の値が <code>work(1)</code> に出力される。これ以降の実行には、この <code>lwork</code> の値を使用する。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orgtr` のインターフェイスの詳細を以下に示す。

a サイズ (*n*,*n*) の行列 *A* を格納する。
tau 長さ (*n*-1) のベクトルを格納する。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (n-1) * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた *Q* は、正確な直交行列と行列 *E* ($\|E\|_2 = O(\epsilon)$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)n^3$ である。

このルーチンの複素数版は、[?ungtr](#) である。

?ormtr

実行列に `?sytrd` で求めた実直交行列 *Q* を掛ける。

構文

Fortran 77:

```
call sormtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork,
           info)
call dormtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork,
           info)
```

Fortran 95:

```
call ormtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

説明

このルーチンは、実行列 C に Q または Q^T を掛ける。 Q は、実対称行列 A を三重対角形式 $A = QTQ^T$ に縮退させるために `?sytrd` によって求めた直交行列 Q である。このルーチンは、`?sytrd` を呼び出した後で使用する。

このルーチンを使用すれば、パラメーター `side` と `trans` の値に従って、 QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

下記の説明で、 r は Q の次数を表す。

`side = 'L'` の場合、 $r = m$ 、`side = 'R'` の場合、 $r = n$ 。

<code>side</code>	CHARACTER*1。'L' または 'R' でなければならない。 <code>side = 'L'</code> の場合、 Q または Q^T は、 C に左側から適用される。 <code>side = 'R'</code> の場合、 Q または Q^T は、 C に右側から適用される。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>?sytrd</code> に指定した <code>uplo</code> と同じ値を使用する。
<code>trans</code>	CHARACTER*1。'N' または 'T' でなければならない。 <code>trans = 'N'</code> の場合、 C に Q を掛ける。 <code>trans = 'T'</code> の場合、 C に Q^T を掛ける。
<code>m</code>	INTEGER。行列 C の行数 ($m \geq 0$)。
<code>n</code>	INTEGER。行列 C の列数 ($n \geq 0$)。
<code>a, work, tau, c</code>	REAL (<code>sormtr</code> の場合) DOUBLE PRECISION (<code>dormtr</code> の場合)。 <code>a(lda,*)</code> と <code>tau</code> は、 <code>?sytrd</code> から返された配列である。 <code>a</code> の第 2 次元は、 $\max(1, r)$ 以上でなければならない。 <code>tau</code> の次元は、 $\max(1, r-1)$ 以上でなければならない。 <code>c(ldc,*)</code> には、行列 C を格納する。 <code>c</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $lda \geq \max(1, r)$ 。
<code>ldc</code>	INTEGER。 <code>c</code> の第 1 次元。 $ldc \geq \max(1, n)$ 。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ。次の制約がある。 $lwork \geq \max(1, n)$ (<code>side = 'L'</code> の場合)。 $lwork \geq \max(1, m)$ (<code>side = 'R'</code> の場合)。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエンタリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。 <code>lwork</code> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

`c` (`side` と `trans` の値に従って) QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの積によって上書きされる。

<code>work(1)</code>	<code>info=0</code> の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <code>lwork</code> の値が <code>work(1)</code> に出力される。これ以降の実行には、この <code>lwork</code> の値を使用する。
<code>info</code>	INTEGER。 <code>info=0</code> の場合、正常に終了したことを示す。 <code>info=-i</code> の場合、 <code>i</code> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ormtr` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (r, r) の行列 A を格納する。 $r = m$ (<code>side = 'L'</code> の場合)。 $r = n$ (<code>side = 'R'</code> の場合)。
<code>tau</code>	長さ $(r-1)$ のベクトルを格納する。
<code>c</code>	サイズ (m, n) の行列 C を格納する。
<code>side</code>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>trans</code>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ (`side = 'L'` の場合) または $lwork = m * blocksize$ (`side = 'R'` の場合) に設定する。`blocksize` は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\epsilon) \|C\|_2$) だけ異なる。

浮動小数演算のおおよその総数は、 $2 * m^2 * n$ (`side = 'L'` の場合)、または $2 * n^2 * m$ (`side = 'R'` の場合) である。

このルーチンの複素数版は、[?unmtr](#) である。

?hetrd

複素エルミート行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call chetrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
call zhetrd(uplo, n, a, lda, d, e, tau, work, lwork, info)
```

Fortran 95:

```
call hetrd(a, tau [,uplo] [,info])
```

説明

このルーチンは、ユニタリー相似変換 $A = QTQ^H$ によって、複素エルミート行列 A を対称な三重対角形式 T に縮退させる。ユニタリー行列 Q は、明示的な形式では表現されず、 $(n-1)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。(このセクションで後述)。

このルーチンは、[?latrd](#) を呼び出して、ユニタリー相似変換によって、複素エルミート行列 A をエルミート三重対角形式に縮退させる。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 <code>a</code> には A の上三角部分を格納する。 <code>uplo = 'L'</code> の場合、 <code>a</code> には A の下三角部分を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>a, work</code>	COMPLEX (chetrd の場合) DOUBLE COMPLEX (zhetrd の場合)。 <code>a(lda,*)</code> は、 <code>uplo</code> の値に従って、行列 A の上三角部分または下三角部分を格納する配列である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ ($lwork \geq n$)。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエントリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。 <code>lwork</code> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	<i>uplo</i> の値に従って、三重対角行列 <i>T</i> とユニタリー行列 <i>Q</i> の各成分によって上書きされる。
<i>d</i> , <i>e</i>	REAL (chetrd の場合) DOUBLE PRECISION (zhetrd の場合)。 配列: <i>d</i> (*) には、行列 <i>T</i> の対角成分が格納される。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<i>tau</i>	COMPLEX (chetrd の場合) DOUBLE COMPLEX (zhetrd の場合)。 配列、次元は $\max(1, n-1)$ 以上。 ユニタリー行列 <i>Q</i> の各成分が格納される。
<i>work</i> (1)	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work</i> (1) に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hetrd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>tau</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ に設定する。*blocksize* は、ブロック・アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 *T* は、行列 *A* + *E* の近似値である。ここで、 $\|E\|_2 = c(n)\epsilon \|A\|_2$ 、*c*(*n*) は漸増する *n* の関数、 ϵ はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(16/3)n^3$ である。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?ungtr](#) 計算で求める行列 Q を明示的に生成する場合

[?unmtr](#) 複素行列に Q を掛ける場合

このルーチンの実数版は、[?sytrd](#) である。

?ungtr

?hetrd で求めた複素ユニタリー行列 Q を生成する。

構文

Fortran 77:

```
call cungr( uplo, n, a, lda, tau, work, lwork, info )
```

```
call zungtr( uplo, n, a, lda, tau, work, lwork, info )
```

Fortran 95:

```
call ungr( a, tau [, uplo] [, info] )
```

このルーチンは、複素エルミート行列 A を三重対角形式 $A = QTQ^H$ に縮退するために [?hetrd](#) によって求めた $n \times n$ のユニタリー行列 Q を明示的に生成する。このルーチンは、?hetrd を呼び出した後で使用する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 ?hetrd に指定した <code>uplo</code> と同じ値を使用する。
<code>n</code>	INTEGER。行列 Q の次数 ($n \geq 0$)。
<code>a, tau, work</code>	COMPLEX (cungr の場合) DOUBLE COMPLEX (zungtr の場合)。 配列: <code>a(lda,*)</code> は、?hetrd から返された配列 <code>a</code> である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>tau(*)</code> は、?hetrd から返された配列 <code>tau</code> である。 <code>tau</code> の次元は、 $\max(1, n-1)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>lda</code>	INTEGER。 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上。
<code>lwork</code>	INTEGER。配列 <code>work</code> のサイズ ($lwork \geq n$)。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエンタリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。 <code>lwork</code> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	ユニタリ行列 Q によって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ungtr` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>tau</i>	長さ $(n-1)$ のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (n-1) * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた Q は、正確なユニタリ行列と行列 E ($\|E\|_2 = O(\epsilon)$, ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(16/3)n^3$ である。

このルーチンの実数版は、[?orgtr](#) である。

?unmtr

複素行列に ?hetrd で求めた複素ユニタリー行列 Q を掛ける。

構文

Fortran 77:

```
call cummtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork,
            info)
call zummtr(side, uplo, trans, m, n, a, lda, tau, c, ldc, work, lwork,
            info)
```

Fortran 95:

```
call unmtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

説明

このルーチンは、複素行列 C に Q または Q^H を掛ける。 Q は、複素エルミート行列 A を三重対角形式 $A = QTQ^H$ に縮退するために [?hetrd](#) によって求めたユニタリー行列 Q である。このルーチンは、?hetrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 Q^HC 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

下記の説明で、 r は Q の次数を表す。

$side = 'L'$ の場合、 $r = m$ 、 $side = 'R'$ の場合、 $r = n$ 。

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^H は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^H は、 C に右側から適用される。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 ?hetrd に指定した $uplo$ と同じ値を使用する。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'T'$ の場合、 C に Q^H を掛ける。
m	INTEGER。行列 C の行数 ($m \geq 0$)。
n	INTEGER。行列 C の列数 ($n \geq 0$)。
$a, work, tau, c$	COMPLEX (cummtr の場合) DOUBLE COMPLEX (zummtr の場合)。 $a(lda,*)$ と tau は、?hetrd から返された配列である。 a の第 2 次元は、 $\max(1, r)$ 以上でなければならない。 tau の次元は、 $\max(1, r-1)$ 以上でなければならない。 $c(ldc,*)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda	INTEGER。 a の第 1 次元。 $lda \geq \max(1, r)$ 。
ldc	INTEGER。 c の第 1 次元。 $ldc \geq \max(1, n)$ 。
$lwork$	INTEGER。 配列 $work$ のサイズ。 次の制約がある。 $lwork \geq \max(1, n)$ ($side = 'L'$ の場合)。 $lwork \geq \max(1, m)$ ($side = 'R'$ の場合)。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c	($side$ と $trans$ の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmtr` のインターフェイスの詳細を以下に示す。

a	サイズ (r, r) の行列 A を格納する。 $r = m$ ($side = 'L'$ の場合)。 $r = n$ ($side = 'R'$ の場合)。
tau	長さ $(r-1)$ のベクトルを格納する。
c	サイズ (m, n) の行列 C を格納する。
$side$	'L' または 'R' でなければならない。デフォルト値は 'L'。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$trans$	'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = n * blocksize$ ($side = 'L'$ の場合) または $lwork = m * blocksize$ ($side = 'R'$ の場合) に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\varepsilon) \|C\|_2$, ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $8 \cdot m^2 \cdot n$ ($side = 'L'$ の場合)、または $8 \cdot n^2 \cdot m$ ($side = 'R'$ の場合) である。

このルーチンの実数版は、[?ormtr](#) である。

?sptrd

圧縮格納形式の実対称行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call ssptdr(uplo, n, ap, d, e, tau, info)
call dsptdr(uplo, n, ap, d, e, tau, info)
```

Fortran 95:

```
call sptrd(a, tau [,uplo] [,info])
```

説明

このルーチンは、直交相似変換 $A = QTQ^T$ によって、圧縮格納形式の実対称行列 A を対称三重対角形式 T に縮退させる。直交行列 Q は、明示的な形式では表現されず、 $(n-1)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する (このセクションで後述)。

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
uplo='U' の場合、**ap** に A の上三角行列 (圧縮形式) を格納する。
uplo='L' の場合、**ap** に A の下三角行列 (圧縮形式) を格納する。

n INTEGER。行列 A の次数 ($n \geq 0$)。

ap REAL (ssptdr の場合)
 DOUBLE PRECISION (dsptdr の場合)。
 配列、次元は $\max(1, n(n+1)/2)$ 以上。
 (**uplo** の値に従って) 圧縮形式の A の上三角行列または下三角行列を格納する。

出力パラメーター

ap **uplo** の値に従って、三重対角行列 T と直交行列 Q の各成分によって上書きされる。

d, e, tau REAL (ssptdr の場合)
 DOUBLE PRECISION (dsptdr の場合)。
 配列:
d(*) には、行列 T の対角成分が格納される。
d の次元は、 $\max(1, n)$ 以上でなければならない。

$e(*)$ には、 T の非対角成分が格納される。
 e の次元は、 $\max(1, n-1)$ 以上でなければならない。

$\tau(*)$ には、行列 Q の各成分が格納される。
 τ の次元は、 $\max(1, n-1)$ 以上でなければならない。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sptrd` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 a_p を意味する。
 サイズ $(n*(n+1)/2)$ の配列 A を格納する。

tau 長さ $(n-1)$ のベクトルを格納する。

d 長さ (n) のベクトルを格納する。

e 長さ $(n-1)$ のベクトルを格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた行列 T は、行列 $A + E$ の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、 $c(n)$ は漸増する n の関数、 ε はマシンによって異なる値である。浮動小数演算のおおよその総数は、 $(4/3)n^3$ である。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?opqtr](#) 計算で求める行列 Q を明示的に生成する場合

[?opmtr](#) 実行列に Q を掛ける場合

このルーチンの複素数版は、[?hpqtrd](#) である。

?opqtr

`?sptrd` で求めた実直交行列 Q を生成する。

構文

Fortran 77:

```
call sopqtr(uplo, n, ap, tau, q, ldq, work, info)
call dopqtr(uplo, n, ap, tau, q, ldq, work, info)
```

Fortran 95:

```
call opqtr(a, tau, q [,uplo] [,info])
```

説明

このルーチンは、圧縮形式の実対称行列 A を三重対角行列の形式 $A = QTQ^T$ に縮退させるために [?sptprd](#) によって求めた $n \times n$ の直交行列 Q を明示的に生成する。このルーチンは、[?sptprd](#) を呼び出した後で使用する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 ?sptprd に指定した <code>uplo</code> と同じ値を使用する。
<code>n</code>	INTEGER。行列 Q の次数 ($n \geq 0$)。
<code>ap, tau</code>	REAL (sopgtr の場合) DOUBLE PRECISION (dopgtr の場合)。 配列 <code>ap</code> と <code>tau</code> は、 ?sptprd から返された配列である。 <code>ap</code> の次元は、 $\max(1, n(n+1)/2)$ 以上でなければならない。 <code>tau</code> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<code>ldq</code>	INTEGER。出力配列 q の第 1 次元は、 $\max(1, n)$ 以上でなければならない。
<code>work</code>	REAL (sopgtr の場合) DOUBLE PRECISION (dopgtr の場合)。 ワークスペース配列、次元は $\max(1, n-1)$ 以上。

出力パラメーター

<code>q</code>	REAL (sopgtr の場合) DOUBLE PRECISION (dopgtr の場合)。 配列、次元は (<code>ldq</code> , *)。 計算で求めた行列 Q が格納される。 <code>q</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `opgtr` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>q</code>	サイズ (n, n) の行列 Q を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\varepsilon)$ 、 ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)n^3$ である。

このルーチンの複素数版は、[?upgtr](#) である。

?opmtr

実行列に `?sptrd` で求めた実直交行列 Q を掛ける。

構文

Fortran 77:

```
call sopmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call dopmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

Fortran 95:

```
call opmtr(a, tau, c [,side] [,uplo] [,trans] [,info])
```

説明

このルーチンは、実行列 C に Q または Q^T を掛ける。 Q は、圧縮形式の実対称行列 A を三重対角形式 $A = QTQ^T$ に縮退させるために [?sptrd](#) によって求めた直交行列 Q である。このルーチンは、`?sptrd` を呼び出した後で使用される。

このルーチンを使用すれば、パラメーター `side` と `trans` の値に従って、 QC 、 Q^TC 、 CQ 、または CQ^T のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

下記の説明で、 r は Q の次数を表す。

`side = 'L'` の場合、 $r = m$ 、`side = 'R'` の場合、 $r = n$ 。

<code>side</code>	CHARACTER*1。'L' または 'R' でなければならない。 <code>side = 'L'</code> の場合、 Q または Q^T は、 C に左側から適用される。 <code>side = 'R'</code> の場合、 Q または Q^T は、 C に右側から適用される。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>?sptrd</code> に指定した <code>uplo</code> と同じ値を使用する。
<code>trans</code>	CHARACTER*1。'N' または 'T' でなければならない。 <code>trans = 'N'</code> の場合、 C に Q を掛ける。 <code>trans = 'T'</code> の場合、 C に Q^T を掛ける。
<code>m</code>	INTEGER。行列 C の行数 ($m \geq 0$)。
<code>n</code>	INTEGER。行列 C の列数 ($n \geq 0$)。
<code>ap, work, tau, c</code>	REAL (<code>sopmtr</code> の場合) DOUBLE PRECISION (<code>dopmtr</code> の場合)。 <code>ap</code> と <code>tau</code> は、 <code>?sptrd</code> から返された配列である。 <code>ap</code> の次元は、 $\max(1, r(r+1)/2)$ 以上でなければならない。 <code>tau</code> の次元は、 $\max(1, r-1)$ 以上でなければならない。

$c(ldc,*)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列である。

$work$ の次元は、 $\max(1, n)$ 以上 ($side = 'L'$ の場合) または $\max(1, m)$ 以上 ($side = 'R'$ の場合) でなければならない。

ldc INTEGER。 c の第 1 次元。 $ldc \geq \max(1, n)$ 。

出力パラメーター

c ($side$ と $trans$ の値に従って) QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの積によって上書きされる。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `opmtr` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 ap を意味する。
 サイズ $(r*(r+1)/2)$ の配列 A を格納する。ここで
 $r = m$ ($side = 'L'$ の場合)、
 $r = n$ ($side = 'R'$ の場合)。

tau 長さ $(r-1)$ のベクトルを格納する。

c サイズ (m, n) の行列 C を格納する。

$side$ 'L' または 'R' でなければならない。デフォルト値は 'L'。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

$trans$ 'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\epsilon) \|C\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $2*m^2*n$ ($side = 'L'$ の場合)、または $2*n^2*m$ ($side = 'R'$ の場合) である。

このルーチンの複素数版は、[?upmtr](#) である。

?hptrd

圧縮格納形式の複素エルミート行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call chptrd(uplo, n, ap, d, e, tau, info)
call zhptrd(uplo, n, ap, d, e, tau, info)
```

Fortran 95:

```
call hptrd(a, tau [,uplo] [,info])
```

説明

このルーチンは、ユニタリー相似変換 $A = QTQ^H$ によって、圧縮形式の複素エルミート行列 A を対称三重対角形式 T に縮退させる。ユニタリー行列 Q は、明示的な形式では表現されず、 $(n-1)$ 個の基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する (このセクションで後述)。

入力パラメーター

uplo CHARACTER*1。'U' または 'L' でなければならない。
 uplo='U' の場合、ap に A の上三角行列 (圧縮形式) を格納する。
 uplo='L' の場合、ap に A の下三角行列 (圧縮形式) を格納する。

n INTEGER。行列 A の次数 ($n \geq 0$)。

ap COMPLEX (chptrd の場合)
 DOUBLE COMPLEX (zhptrd の場合)。
 配列、次元は $\max(1, n(n+1)/2)$ 以上。
 (uplo の値に従って) 圧縮形式の A の上三角行列または下三角行列を格納する。

出力パラメーター

ap uplo の値に従って、三重対角行列 T と直交行列 Q の各成分によって上書きされる。

d, e REAL (chptrd の場合)
 DOUBLE PRECISION (zhptrd の場合)。
 配列:
 d(*) には、行列 T の対角成分が格納される。
 d の次元は、 $\max(1, n)$ 以上でなければならない。
 e(*) には、 T の非対角成分が格納される。
 e の次元は、 $\max(1, n-1)$ 以上でなければならない。

tau COMPLEX (chptrd の場合)
 DOUBLE COMPLEX (zhptrd の場合)。
 配列、次元は $\max(1, n-1)$ 以上。
 直交行列 Q の各成分が格納される。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpstrd` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ap* を意味する。
 サイズ $(n*(n+1)/2)$ の配列 *A* を格納する。

tau 長さ $(n-1)$ のベクトルを格納する。

d 長さ (n) のベクトルを格納する。

e 長さ $(n-1)$ のベクトルを格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた行列 *T* は、行列 $A + E$ の近似値である。ここで、 $\|E\|_2 = c(n)\varepsilon \|A\|_2$ 、 $c(n)$ は漸増する n の関数、 ε はマシンによって異なる値である。

浮動小数演算のおおよその総数は、 $(16/3)n^3$ である。

このルーチンを呼び出した後、以下のルーチンを呼び出せる。

[?upgtr](#) 計算で求める行列 *Q* を明示的に生成する場合

[?upmtr](#) 複素行列に *Q* を掛ける場合

このルーチンの実数版は、[?spstrd](#) である。

?upgtr

?hpstrd で求めた複素ユニタリー行列 *Q* を生成する。

構文

Fortran 77:

```
call cupgtr(uplo, n, ap, tau, q, ldq, work, info)
call zupgtr(uplo, n, ap, tau, q, ldq, work, info)
```

Fortran 95:

```
call upgtr(a, tau, q [,uplo] [,info])
```

説明

このルーチンは、圧縮形式の複素エルミート行列 A を三重対角形式 $A = QTQ^H$ に縮退するために [?hptrd](#) によって求めた $n \times n$ のユニタリー行列 Q を明示的に生成する。このルーチンは、`?hptrd` を呼び出した後で使用する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>?sptrd</code> に指定した <code>uplo</code> と同じ値を使用する。
<code>n</code>	INTEGER。行列 Q の次数 ($n \geq 0$)。
<code>ap, tau</code>	COMPLEX (<code>cupgtr</code> の場合) DOUBLE COMPLEX (<code>zupgtr</code> の場合)。 配列 <code>ap</code> と <code>tau</code> は、 <code>?hptrd</code> から返された配列である。 <code>ap</code> の次元は、 $\max(1, n(n+1)/2)$ 以上でなければならない。 <code>tau</code> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<code>ldq</code>	INTEGER。出力配列 q の第 1 次元は、 $\max(1, n)$ 以上でなければならない。
<code>work</code>	COMPLEX (<code>cupgtr</code> の場合) DOUBLE COMPLEX (<code>zupgtr</code> の場合)。 ワークスペース配列、次元は $\max(1, n-1)$ 以上。

出力パラメーター

<code>q</code>	COMPLEX (<code>cupgtr</code> の場合) DOUBLE COMPLEX (<code>zupgtr</code> の場合)。 配列、次元は (<code>ldq, *</code>)。 計算で求めた行列 Q が格納される。 <code>q</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>info</code>	INTEGER。 <code>info = 0</code> の場合、正常に終了したことを示す。 <code>info = -i</code> の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `upgtr` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>q</code>	サイズ (n, n) の行列 Q を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\varepsilon)$ 、 ε はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(16/3)n^3$ である。

このルーチンの実数版は、[?opgtr](#) である。

?upmtr

複素行列に ?hptrd で求めたユニタリ行列 Q を掛ける。

構文

Fortran 77:

```
call cupmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
call zupmtr(side, uplo, trans, m, n, ap, tau, c, ldc, work, info)
```

Fortran 95:

```
call upmtr(a, tau, c [,side] [,uplo] [,trans][,info])
```

説明

このルーチンは、圧縮形式の複素行列 C と Q または Q^H を掛ける。 Q は、複素エルミート行列 A を三重対角形式 $A = QTQ^H$ に縮退するために [?hptrd](#) によって求めたユニタリ行列 Q である。このルーチンは、?hptrd を呼び出した後で使用する。

このルーチンを使用すれば、パラメーター $side$ と $trans$ の値に従って、 QC 、 Q^HC 、 CQ 、または CQ^H のいずれかの行列積を求めることができる (C の値は上書きされる)。

入力パラメーター

下記の説明で、 r は Q の次数を表す。

$side = 'L'$ の場合、 $r = m$ 、 $side = 'R'$ の場合、 $r = n$ 。

$side$	CHARACTER*1。'L' または 'R' でなければならない。 $side = 'L'$ の場合、 Q または Q^H は、 C に左側から適用される。 $side = 'R'$ の場合、 Q または Q^H は、 C に右側から適用される。
$uplo$	CHARACTER*1。'U' または 'L' でなければならない。 ?hptrd に指定した $uplo$ と同じ値を使用する。
$trans$	CHARACTER*1。'N' または 'T' でなければならない。 $trans = 'N'$ の場合、 C に Q を掛ける。 $trans = 'T'$ の場合、 C に Q^H を掛ける。
m	INTEGER。行列 C の行数 ($m \geq 0$)。
n	INTEGER。行列 C の列数 ($n \geq 0$)。
$ap, tau, c, work$	COMPLEX (cupmtr の場合) DOUBLE COMPLEX (zupmtr の場合)。 ap と tau は、?hptrd から返された配列である。 ap の次元は、 $\max(1, r(r+1)/2)$ 以上でなければならない。 tau の次元は、 $\max(1, r-1)$ 以上でなければならない。

$c(ldc,*)$ には、行列 C を格納する。 c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列である。

$work$ の次元は、 $\max(1, n)$ 以上 ($side = 'L'$ の場合) または $\max(1, m)$ 以上 ($side = 'R'$ の場合) でなければならない。

ldc INTEGER。 c の第 1 次元。 $ldc \geq \max(1, n)$ 。

出力パラメーター

c ($side$ と $trans$ の値に従って) QC 、 $Q^H C$ 、 CQ 、または CQ^H のいずれかの積によって上書きされる。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `upmtr` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 a_p を意味する。
 サイズ $(r*(r+1)/2)$ の配列 A を格納する。ここで
 $r = m$ ($side = 'L'$ の場合)、
 $r = n$ ($side = 'R'$ の場合)。

tau 長さ $(r-1)$ のベクトルを格納する。

c サイズ (m, n) の行列 C を格納する。

$side$ 'L' または 'R' でなければならない。デフォルト値は 'L'。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

$trans$ 'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

計算で求めた積は、正確な積と行列 E ($\|E\|_2 = O(\epsilon) \|C\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $8*m^2*n$ ($side = 'L'$ の場合)、または $8*n^2*m$ ($side = 'R'$ の場合) である。

このルーチンの実数版は、[?opmtr](#) である。

?sbtrd

実対称帯行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call ssbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call dsbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

Fortran 95:

```
call sbtrd(a [,q] [,vect] [,uplo] [,info])
```

説明

このルーチンは、直交相似変換 $A = QTQ^T$ によって、実対称帯行列 A を対称三重対角形式 T に縮退させる。直交行列 Q は、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使用して、行列 Q を明示的に求めることができる。

入力パラメーター

<i>vect</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>vect</i> = 'V' の場合、行列 Q が明示的に返される。 <i>vect</i> = 'N' の場合、 Q は返されない。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	REAL (ssbtrd の場合) DOUBLE PRECISION (dsbtrd の場合)。 <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 行列 A の上三角部分または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $kd+1$ 以上。
<i>ldq</i>	INTEGER。 <i>q</i> の第 1 次元。 次の制約がある。 $ldq \geq \max(1, n)$ (<i>vect</i> = 'V' の場合)。 $ldq \geq 1$ (<i>vect</i> = 'N' の場合)。

出力パラメーター

<i>ab</i>	終了時に、 <i>ab</i> は上書きされる。
-----------	--------------------------

d, e, q	REAL (ssbtrd の場合) DOUBLE PRECISION (dsbtrd の場合)。 配列: $d(*)$ には、行列 T の対角成分が格納される。 d の次元は、 $\max(1, n)$ 以上でなければならない。 $e(*)$ には、 T の非対角成分が格納される。 e の次元は、 $\max(1, n-1)$ 以上でなければならない。 $vect = 'N'$ の場合、 $q(ldq, *)$ は参照されない。 $vect = 'V'$ の場合、 q には $n \times n$ の行列 Q が格納される。 q の第 2 次元は、 $\max(1, n)$ 以上 ($vect = 'V'$ の場合) または 1 以上 ($vect = 'N'$ の場合) でなければならない。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン sbtrd のインターフェイスの詳細を以下に示す。

a	Fortran 77 インターフェイスでの引数 ab を意味する。 サイズ $(kd+1, n)$ の配列 A を格納する。
q	サイズ (n, n) の行列 Q を格納する。
d	長さ (n) のベクトルを格納する。
e	長さ $(n-1)$ のベクトルを格納する。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$vect$	省略された場合、この引数は、引数 q の存在に基づいて以下のように復元される。 $vect = 'V'$ (q が存在する場合)、 $vect = 'N'$ (q が省略された場合)。 存在する場合、 $vect$ は 'V' または 'U' と等しくなければならない、引数 q も存在しなければならない。 $vect$ が存在し、 q が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

計算で求めた行列 T は、行列 $A + E$ の近似値である。ここで、 $\|E\|_2 = c(n)\epsilon \|A\|_2$ 、 $c(n)$ は漸増する n の関数、 ϵ はマシンによって異なる値である。計算で求めた行列 Q は、正確な直交行列と行列 E ($\|E\|_2 = O(\epsilon)$) だけ異なる。

浮動小数演算のおおよその総数は、 $vect = 'N'$ の場合、 $6n^2 * kd$ である。 $vect = 'V'$ の場合、 $3n^3 * (kd-1)/kd$ だけ余分な演算が必要になる。

このルーチンの複素数版は、[?hbtrd](#) である。

?hbtrd

複素エルミート帯行列を三重対角形式に縮退させる。

構文

Fortran 77:

```
call chbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
call zhbtrd(vect, uplo, n, kd, ab, ldab, d, e, q, ldq, work, info)
```

Fortran 95:

```
call hbtrd(a [,q] [,vect] [,uplo] [,info])
```

説明

このルーチンは、ユニタリー相似変換 $A = QTQ^H$ によって、複素エルミート帯行列 A を対称三重対角形式 T に縮退させる。ユニタリー行列 Q は、Givens 回転行列の積として得られる。そのため、必要に応じて、このルーチンを使用して、行列 Q を明示的に求めることができる。

入力パラメーター

<i>vect</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>vect</i> = 'V' の場合、行列 Q が明示的に返される。 <i>vect</i> = 'N' の場合、 Q は返されない。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	COMPLEX (chbtrd の場合) DOUBLE COMPLEX (zhbtrd の場合)。 <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 行列 A の上三角部分または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> の第 1 次元。 $kd+1$ 以上。
<i>ldq</i>	INTEGER。 <i>q</i> の第 1 次元。 次の制約がある。 $ldq \geq \max(1, n)$ (<i>vect</i> = 'V' の場合)。 $ldq \geq 1$ (<i>vect</i> = 'N' の場合)。

出力パラメーター

ab 終了時に、*ab* は上書きされる。

<i>d, e</i>	REAL (chbtrd の場合) DOUBLE PRECISION (zhbtrd の場合)。 配列: <i>d</i> (*) には、行列 <i>T</i> の対角成分が格納される。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。
<i>q</i>	COMPLEX (chbtrd の場合) DOUBLE COMPLEX (zhbtrd の場合)。 配列、次元は (<i>ldq</i> ,*)。 <i>vect</i> ='N' の場合、 <i>q</i> は参照されない。 <i>vect</i> ='V' の場合、 <i>q</i> には $n \times n$ の行列 <i>Q</i> が格納される。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上 (<i>vect</i> ='V' の場合) または 1 以上 (<i>vect</i> ='N' の場合) でなければならない。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbtrd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ (<i>kd</i> +1, <i>n</i>) の配列 <i>A</i> を格納する。
<i>q</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Q</i> を格納する。
<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>vect</i>	省略された場合、この引数は、引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>vect</i> ='V' (<i>q</i> が存在する場合)、 <i>vect</i> ='N' (<i>q</i> が省略された場合)。 存在する場合、 <i>vect</i> は 'V' または 'U' と等しくなければならない、引数 <i>q</i> も存在しなければならない。 <i>vect</i> が存在し、 <i>q</i> が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

計算で求めた行列 *T* は、行列 *A* + *E* の近似値である。ここで、 $\|E\|_2 = c(n)\epsilon \|A\|_2$ 、*c*(*n*) は漸増する *n* の関数、 ϵ はマシンによって異なる値である。計算で求めた行列 *Q* は、正確な直交行列と行列 *E* ($\|E\|_2 = O(\epsilon)$) だけ異なる。

浮動小数演算のおおよその総数は、*vect*='N' の場合、 $20n^2 * kd$ である。*vect*='V' の場合、 $10n^3 * (kd-1)/kd$ だけ余分な演算が必要になる。

このルーチンの実数版は、[?sbtrd](#) である。

?sterf

QR アルゴリズムを使用して、実対称三重対角行列の固有値をすべて計算する。

構文

Fortran 77:

```
call ssterf(n, d, e, info)
call dsterf(n, d, e, info)
```

Fortran 95:

```
call sterf(d, e [,info])
```

説明

このルーチンは、実対称三重対角行列 T (対称 / エルミート行列を対角形式に縮退させて得られる) の固有値をすべて計算する。このルーチンは、平方根なしの *QR* アルゴリズムを使用する。

固有値だけでなく固有ベクトルも必要な場合は、[?stegr](#) を使用する。

入力パラメーター

n INTEGER。行列 T の次数 ($n \geq 0$)。

d, e REAL (ssterf の場合)
 DOUBLE PRECISION (dsterf の場合)。
 配列:
 $d(*)$ には、 T の対角成分を格納する。
 d の次元は、 $\max(1, n)$ 以上でなければならない。
 $e(*)$ には、 T の非対角成分が格納される。
 e の次元は、 $\max(1, n-1)$ 以上でなければならない。

出力パラメーター

d n 個の固有値が、昇順に格納される ($info > 0$ でない場合)。
 $info$ も参照のこと。

e 終了時に上書きされる。 $info$ を参照。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = i$ の場合、このアルゴリズムで $30n$ 回の処理を繰り返した結果、すべての固有値を見つけられなかった (i 個の非対角成分がゼロに収束していない) ことを示す。終了時に、 d と e にそれぞれ、三重対角行列 (直交性が T に近似している) の対角成分と非対角成分が格納される。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sterf` のインターフェイスの詳細を以下に示す。

- `d` 長さ (n) のベクトルを格納する。
`e` 長さ ($n-1$) のベクトルを格納する。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

λ_i が正確な固有値で、 μ_i が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\epsilon \|T\|_2$$

$c(n)$ は、 n の漸増関数である。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。一般に、約 $14n^2$ である。

?steqr

三重対角形式に縮退させた対称/エルミート行列の固有値と固有ベクトルをすべて計算する (QR アルゴリズム)。

構文

Fortran 77:

```
call ssteqr(compz, n, d, e, z, ldz, work, info)
call dsteqr(compz, n, d, e, z, ldz, work, info)
call csteqr(compz, n, d, e, z, ldz, work, info)
call zsteqr(compz, n, d, e, z, ldz, work, info)
```

Fortran 95:

```
call rsteqr(d, e [,z] [,compz] [,info])
call steqr(d, e [,z] [,compz] [,info])
```

説明

このルーチンは、実対称三重対角行列 T の固有値と (オプションで) 固有ベクトルをすべて計算する。すなわち、スペクトル因子分解 $T = Z\Lambda Z^T$ を計算できる。 Λ は、対角成分が固有値 λ_i である対角行列で、 Z は列が固有ベクトルである直交行列である。したがって、

$$Tz_i = \lambda_i z_i \text{ for } i = 1, 2, \dots, n$$

このルーチンでは、 $\|z\|_2 = 1$ となるように、固有ベクトルを正規化する。

このルーチンを使用すれば、三重対角形式 $T (A = QTQ^H)$ に縮退させた任意の実対称 (または複素エルミート) 行列 A の固有値と固有ベクトルも計算できる。その場合、スペクトル因子分解は、 $A = QTQ^H = (QZ)\Lambda(QZ)^H$ になる。`?steqr` を呼び出す場合は、事前に A を三重対角形式に縮退させ、次のルーチンを呼び出すことによって行列 Q を明示的に生成しなければならない。

	実行列の場合	複素行列の場合
フル格納	?sytrd, ?orgtr	?hetrd, ?ungtr
圧縮格納	?sptrd, ?opgtr	?hptrd, ?upgtr
帯格納	?sbtrd (vect='V')	?hbtrd (vect='V')

固有値のみ必要な場合は、`?sterf` を呼び出した方が効率は良くなる。 T が正定値の場合、`?steqr` よりも `?pteqr` の方が、小さな固有値を正確に計算できる。

1 つのルーチン呼び出しでこの問題を解くには、分割統治ルーチンを使用する。すなわち、実対称行列の場合、`?stevd`、`?syevd`、`?spevd`、または `?sbevd` のいずれか、複素エルミート行列の場合、`?heevd`、`?hpevd`、または `?hbevd` のいずれかを使用する。

入力パラメーター

<code>compz</code>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <code>compz = 'N'</code> の場合、固有値のみを計算する。 <code>compz = 'I'</code> の場合、三重対角行列 T の固有値と固有ベクトルを計算する。 <code>compz = 'V'</code> の場合、 A の固有値と固有ベクトルを計算する (呼び出し時に、配列 z には、あらかじめ行列 Q を格納しておかなければならない)。
<code>n</code>	INTEGER。行列 T の次数 ($n \geq 0$)。
<code>d, e, work</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: $d(*)$ には、 T の対角成分を格納する。 d の次元は、 $\max(1, n)$ 以上でなければならない。 $e(*)$ には、 T の非対角成分が格納される。 e の次元は、 $\max(1, n-1)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列。 $work$ の次元は、1 以上 (<code>compz = 'N'</code> の場合) または $\max(1, 2*n-2)$ 以上 (<code>compz = 'V'</code> または 'I' の場合) でなければならない。
<code>z</code>	REAL (<code>ssteqr</code> の場合) DOUBLE PRECISION (<code>dsteqr</code> の場合) COMPLEX (<code>csteqr</code> の場合) DOUBLE COMPLEX (<code>zsteqr</code> の場合)。 配列、次元は ($ldz, *$)。 <code>compz = 'N'</code> または 'I' の場合、 z を設定する必要はない。 <code>vect = 'V'</code> の場合、 z には $n \times n$ の行列 Q を格納しなければならない。

ない。

z の第 2 次元は、1 以上 ($compz = 'N'$ の場合) または $\max(1, n)$ 以上 ($compz = 'V'$ または $'I'$ の場合) でなければならない。

$work(lwork)$ は、ワークスペース配列である。

ldz INTEGER。 z の第 1 次元。次の制約がある。
 $ldz \geq 1$ ($compz = 'N'$ の場合)。
 $ldz \geq \max(1, n)$ ($compz = 'V'$ または $'I'$ の場合)。

出力パラメーター

d n 個の固有値が、昇順に格納される ($info > 0$ でない場合)。
 $info$ も参照のこと。

e 終了時に上書きされる。 $info$ を参照。

z $info = 0$ の場合、 n 個の正規直交固有ベクトルが、列ごとに格納される (i 番目の列は、 i 番目の固有値に対応する)。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = i$ の場合、このアルゴリズムで $30n$ 回の処理を繰り返した結果、すべての固有値を見つけられなかった (i 個の非対角成分がゼロに収束していない) ことを示す。終了時に、 d と e にそれぞれ、三重対角行列 (直交性が T に近似している) の対角成分と非対角成分が格納される。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `steqr` のインターフェイスの詳細を以下に示す。

d 長さ (n) のベクトルを格納する。

e 長さ ($n-1$) のベクトルを格納する。

z サイズ (n, n) の行列 Z を格納する。

$compz$ 省略された場合、この引数は、引数 z の存在に基づいて以下のように復元される。
 $compz = 'I'$ (z が存在する場合)、
 $compz = 'N'$ (z が省略された場合)。
存在する場合、 $compz$ は $'I'$ または $'V'$ と等しくなければならず、引数 z も存在しなければならない。
 $compz$ が存在し、 z が省略された場合、エラー条件がセットされる。

z が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`steqr` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rsteqr`、複素数の場合 (単精度または倍精度) は `steqr` を使用する。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\varepsilon) \|T\|_2$ のような行列 $T + E$ (ε はマシン精度) のものである。

λ_i が正確な固有値で、 μ_i が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\varepsilon \|T\|_2$$

$c(n)$ は、 n の漸増関数である。

z_i が対応する正確な固有ベクトルで、 w_i が対応する計算ベクトルの場合、それらの間の角度 $\theta(z_i, w_i)$ は次のようになる。

$$\theta(z_i, w_i) \leq c(n)\varepsilon \|T\|_2 / \min_{i \neq j} |\lambda_i - \lambda_j|$$

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$24n^2$ ($compz = 'N'$ の場合)。

$7n^3$ (複素数型の場合、 $14n^3$) ($compz = 'V'$ または $'I'$ の場合)。

?stedc

分割統治法を使用して、対称三重行列の固有値と固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call dstedc(compz, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call cstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork, iwork,
            liwork, info)
call zstedc(compz, n, d, e, z, ldz, work, lwork, rwork, lrwork, iwork,
            liwork, info)
```

Fortran 95:

```
call rstedc(d, e [,z] [,compz] [,info])
call stedc(d, e [,z] [,compz] [,info])
```

説明

このルーチンは、分割統治法を使用して、対称三重行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

フル形式または帯形式の実対称行列または複素エルミート行列の固有ベクトルは、[?sytrd/?hetrd](#)、[?sptrd/?hptrd](#)、[?sbtrd/?hbtrd](#) のいずれかを使用して、この行列を三重対角形式に縮退させて得ることもできる。

参照

[?laed0](#)、[?laed1](#)、[?laed2](#)、[?laed3](#)、[?laed4](#)、[?laed5](#)、[?laed6](#)、[?laed7](#)、[?laed8](#)、[?laed9](#)、[?laeda](#)。

入力パラメーター

<i>compz</i>	CHARACTER*1。'N'、'I'、または'V'のいずれかでなければならない。 <i>compz</i> = 'N' の場合、固有値のみを計算する。 <i>compz</i> = 'I' の場合、三重対角行列の固有値と固有ベクトルを計算する。 <i>compz</i> = 'V' の場合、元の対称/エルミート行列の固有値と固有ベクトルを計算する。呼び出し時に、配列 <i>z</i> には、元の行列を三重対角形式に縮退させるのに使用した直交/ユニタリ行列が格納されていなければならない。
<i>n</i>	INTEGER。対称三重対角行列の次数 ($n \geq 0$)。
<i>d</i> , <i>e</i> , <i>rwork</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: <i>d</i> (*) には、三重対角行列の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、三重対角行列の劣対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。 <i>rwork</i> (<i>lrwork</i>) は、複素数型でのみ使用されるワークスペース配列である。
<i>z</i> , <i>work</i>	REAL (<i>sstedc</i> の場合) DOUBLE PRECISION (<i>dstedc</i> の場合) COMPLEX (<i>cstedc</i> の場合) DOUBLE COMPLEX (<i>zstedc</i> の場合)。 配列: <i>z</i> (<i>ldz</i> , *), <i>work</i> (*)。 <i>compz</i> = 'V' の場合、呼び出し時に、配列 <i>z</i> には、元の行列を三重対角形式に縮退させるのに使用した直交/ユニタリ行列が格納されていなければならない。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。次の制約がある。 $ldz \geq 1$ (<i>compz</i> = 'N' の場合)。 $ldz \geq \max(1, n)$ (<i>compz</i> = 'V' または 'I' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> に必要な値は、「アプリケーション・ノート」を参照。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元 (複素数型でのみ使用される)。 <i>lrwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を

	<i>rwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>lrwork</i> に関するエラーメッセージを生成しない。 <i>lrwork</i> に必要な値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。 <i>liwork</i> に必要な値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>d</i>	<i>n</i> 個の固有値が、昇順に格納される (<i>info</i> ≠ 0 でない場合)。 <i>info</i> も参照のこと。
<i>e</i>	終了時に上書きされる。 <i>info</i> を参照。
<i>z</i>	<i>info</i> = 0 の場合、 <i>compz</i> = 'V' であれば、元の対称/エルミート行列の正規直交固有ベクトルが <i>z</i> に格納され、 <i>compz</i> = 'I' であれば、対称三重対角行列の正規直交固有ベクトルが <i>z</i> に格納される。 <i>compz</i> = 'N' であれば、 <i>z</i> は参照されない。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> に最適な <i>lwork</i> 値が返される。
<i>rwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>rwork(1)</i> に最適な <i>lrwork</i> 値が返される (複素数型の場合のみ)。
<i>iwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>iwork(1)</i> に最適な <i>liwork</i> 値が返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが、 <i>i</i> /(<i>n</i> +1) から <i>mod</i> (<i>i</i> , <i>n</i> +1) までの行と列からなる部分行列の操作中に、固有値の計算に失敗したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stedc* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>compz</i>	省略された場合、この引数は、引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>compz</i> = 'I' (<i>z</i> が存在する場合)、

$compz = 'N'$ (z が省略された場合)。

存在する場合、 $compz$ は ' I ' または ' V ' と等しくなければならず、引数 z も存在しなければならない。

$compz$ が存在し、 z が省略された場合、エラー条件がセットされる。

z と $work$ が省略されると、実数および複素数の場合に曖昧な選択が行われるため、 $stedc$ ルーチンには2つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は $rstedc$ 、複素数の場合 (単精度または倍精度) は $stedc$ を使用する。

アプリケーション・ノート

ワークスペース配列に必要なサイズは次のようになる。

$sstedc/dstedc$ の場合

$compz = 'N'$ または $n \leq 1$ の場合、 $lwork$ は 1 以上でなければならない。

$compz = 'V'$ および $n > 1$ の場合、 $lwork$ は $(1 + 3n + 2n \cdot \lg n + 3n^2)$ 以上でなければならない。ここで、 $\lg(n)$ は $2^k \geq n$ となる最小の整数 k である。

$compz = 'I'$ および $n > 1$ の場合、 $lwork$ は $(1 + 4n + n^2)$ 以上でなければならない。

$compz = 'N'$ または $n \leq 1$ の場合、 $liwork$ は 1 以上でなければならない。

$compz = 'V'$ および $n > 1$ の場合、 $liwork$ は $(6 + 6n + 5n \cdot \lg n)$ 以上でなければならない。

$compz = 'I'$ および $n > 1$ の場合、 $liwork$ は $(3 + 5n)$ 以上でなければならない。

$cstedc/zstedc$ の場合

$compz = 'N'$ または ' I ' の場合、または $n \leq 1$ の場合、 $lwork$ は 1 以上でなければならない。

$compz = 'V'$ および $n > 1$ の場合、 $lwork$ は n^2 以上でなければならない。

$compz = 'N'$ または $n \leq 1$ の場合、 $lwork$ は 1 以上でなければならない。

$compz = 'V'$ および $n > 1$ の場合、 $lwork$ は $(1 + 3n + 2n \cdot \lg n + 3n^2)$ 以上でなければならない。ここで、 $\lg(n)$ は $2^k \geq n$ となるような最小の整数 k である。

$compz = 'I'$ および $n > 1$ の場合、 $lwork$ は $(1 + 4n + 2n^2)$ 以上でなければならない。

複素数型の場合に $liwork$ に必要な値は、実数型の場合と同じである。

?stegr

実対称三重対角行列の固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call sstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
           ldz, isuppz, work, lwork, iwork, liwork, info)
call dstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
           ldz, isuppz, work, lwork, iwork, liwork, info)
call cstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
           ldz, isuppz, work, lwork, iwork, liwork, info)
call zstegr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z,
           ldz, isuppz, work, lwork, iwork, liwork, info)
```

Fortran 95:

```
call rstegr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]
           [,info])
call stegr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]
           [,info])
```

説明

このルーチンは、実対称三重対角行列 T の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。固有値は *dqds* アルゴリズムで計算するが、直交固有ベクトルは種々の「良好な」 LDL^T 表現 (比較的安定した表現とも呼ばれる) で計算する。グラム - シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 T のまだ縮退されていない i 番目のブロックに対して、

- $L_i D_i L_i^T$ が比較的安定な表現になるように、 $T - \sigma_i = L_i D_i L_i^T$ を計算する。
- dqds* アルゴリズムによって、 $L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。
- 値が近い固有値からなるクラスターが存在する場合は、そのクラスターに近い σ_i を「選択」し、ステップ (a) に戻る。
- $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメーター *abstol* で指定できる。

参照

[?lasq2](#)、[?lasq5](#)、[?lasq6](#)。

入力パラメーター

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>n</i>	INTEGER。行列 <i>T</i> の次数 ($n \geq 0$)。
<i>d</i> , <i>e</i> , <i>work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: <i>d</i> (*) には、 <i>T</i> の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) の 1 ~ <i>n</i> -1 の成分に、 <i>T</i> の劣対角成分を格納する。 <i>e</i> (<i>n</i>) に値を設定する必要はない。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>v1</i> , <i>vu</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>range</i> ='V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> ='A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il</i> , <i>iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n=0$ の場合)。 <i>range</i> ='A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 各固有値と固有ベクトルに許される絶対許容値。 <i>jobz</i> ='V' の場合、出力される固有値と固有ベクトルの誤差ノルムが <i>abstol</i> 以内になる。また、異なる固有ベクトル間のドット積も <i>abstol</i> 以内になる。 <i>abstol</i> $< n\epsilon\ T\ _1$ の場合、 $n\epsilon\ T\ _1$ が代わりに使用される (ϵ はマシン精度)。固有値は、 <i>abstol</i> とは無関係に、 $\epsilon\ T\ _1$ の精度で計算される。高い相対精度が必要な場合、 <i>abstol</i> を ?lamch (「安全な最小値」) に設定する。

<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ (<i>jobz</i> = 'N' の場合)。 $ldz \geq \max(1, n)$ (<i>jobz</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 18n)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq \max(1, 10n)$ <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>d</i> , <i>e</i>	終了時に、 <i>d</i> と <i>e</i> は上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 選択された固有値が昇順に $w(1) \sim w(m)$ に格納される。
<i>z</i>	REAL (<i>sstegr</i> の場合) DOUBLE PRECISION (<i>dstegr</i> の場合) COMPLEX (<i>cstegr</i> の場合) DOUBLE COMPLEX (<i>zstegr</i> の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第2次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。 <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 <i>isuppz</i> ($2i-1$) から <i>isuppz</i> ($2i$) までの成分のみ非ゼロである。

<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、 <i>slarre</i> で内部エラーが発生したことを示す。 <i>info</i> = 2 の場合、 <i>?larrv</i> で内部エラーが発生したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stegr* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n,m</i>) の行列 <i>Z</i> を格納する。
<i>isuppz</i>	長さ (<i>2*m</i>) のベクトルを格納する。
<i>v1</i>	この引数のデフォルト値は、 <i>v1</i> = -HUGE (<i>v1</i>) であり、ここで HUGE(<i>a</i>) は引数 <i>a</i> と同じ精度のマシン最大値を意味する。
<i>vu</i>	この引数のデフォルト値は、 <i>vu</i> = HUGE (<i>v1</i>) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この引数のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

z が省略されると、実数および複素数の場合に曖昧な選択が行われるため、*stegr* ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は *rstegr*、複素数の場合 (単精度または倍精度) は *stegr* を使用する。

アプリケーション・ノート

現在、?stegr は、 $O(n^2)$ 時間内で T のすべての固有値 n と固有ベクトルを見つけ出すように設定されている。すなわち、range='A' のみサポートしている。

現在、上記のステップ (c) で適切な σ_i が選択できない場合、ルーチン ?stein が呼び出される。固有値同士が近い値であれば、?stein が変形グラム-シュミットを呼び出す。

?stegr は、無限大と NaN の処理方法が、IEEE-754 の浮動小数点標準に準拠しているマシンでのみ正しく機能する。通常の ?stegr 実行で無限大と NaN が発生するため、IEEE-754 標準に準拠しない環境で実行すると、浮動小数点例外によって異常終了する可能性がある。

?pteqr

実対称正定値三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call spteqr(compz, n, d, e, z, ldz, work, info)
call dpteqr(compz, n, d, e, z, ldz, work, info)
call cpteqr(compz, n, d, e, z, ldz, work, info)
call zpteqr(compz, n, d, e, z, ldz, work, info)
```

Fortran 95:

```
call rpteqr(d, e [,z] [,compz] [,info])
call pteqr(d, e [,z] [,compz] [,info])
```

説明

このルーチンは、実対称正定値三重対角行列 T の固有値と (オプションで) 固有ベクトルをすべて計算する。すなわち、スペクトル因子分解 $T = Z\Lambda Z^T$ を計算できる。 Λ は、対角成分が固有値 λ_i である対角行列で、 Z は列が固有ベクトルである直交行列である。したがって、

$$Tz_i = \lambda_i z_i \quad \text{for } i = 1, 2, \dots, n$$

(このルーチンでは、 $\|z_i\|_2 = 1$ となるように、固有ベクトルを正規化する。)

このルーチンを使用すれば、三重対角形式 $T (A = QTQ^H)$ に縮退させた実対称 (または複素エルミート) 正定値行列 A の固有値と固有ベクトルも計算できる。その場合、スペクトル因子分解は、 $A = QTQ^H = (QZ)\Lambda(QZ)^H$ になる。?pteqr を呼び出す場合は、事前に A を三重対角形式に縮退させ、次のルーチンを呼び出すことによって行列 Q を明示的に生成しなければならない。

	実行列の場合	複素行列の場合
フル格納	?sytrd, ?orgtr	?hetrd, ?ungtr
圧縮格納	?sptrd, ?opgtr	?hptrd, ?upgtr
帯格納	?sbtrd (vect='V')	?hbtrd (vect='V')

このルーチンは、最初に、 T を LDL^H として因子分解する(L は下側の単位二重対角行列で、 D は対角行列である)。次に、二重対角行列 $B = LD^{1/2}$ を求めた後、?bdsqrを呼び出して B の特異値(T の固有値と同じ)を求める。

入力パラメーター

<i>compz</i>	CHARACTER*1. 'N'、'I'、または'V'のいずれかでなければならない。 <i>compz</i> = 'N'の場合、固有値のみを計算する。 <i>compz</i> = 'I'の場合、三重対角行列 T の固有値と固有ベクトルを計算する。 <i>compz</i> = 'V'の場合、 A の固有値と固有ベクトルを計算する(呼び出し時に、配列 z には、あらかじめ行列 Q を格納しておかなければならない)。
<i>n</i>	INTEGER。行列 T の次数($n \geq 0$)。
<i>d, e, work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: $d(*)$ には、 T の対角成分を格納する。 d の次元は、 $\max(1, n)$ 以上でなければならない。 $e(*)$ には、 T の非対角成分が格納される。 e の次元は、 $\max(1, n-1)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列。 $work$ の次元は、1以上(<i>compz</i> ='N'の場合)または $\max(1, 4*n-4)$ 以上(<i>compz</i> ='V'または'I'の場合)でなければならない。
<i>z</i>	REAL (spteqrの場合) DOUBLE PRECISION (dpteqrの場合) COMPLEX (cpteqrの場合) DOUBLE COMPLEX (zpteqrの場合)。 配列、次元は($ldz, *$)。 <i>compz</i> ='N'または'I'の場合、 z を設定する必要はない。 <i>vect</i> ='V'の場合、 z には $n \times n$ の行列 Q を格納しなければならない。 z の第2次元は、1以上(<i>compz</i> ='N'の場合)または $\max(1, n)$ 以上(<i>compz</i> ='V'または'I'の場合)でなければならない。
<i>ldz</i>	INTEGER。 z の第1次元。 次の制約がある。 $ldz \geq 1$ (<i>compz</i> ='N'の場合)。 $ldz \geq \max(1, n)$ (<i>compz</i> ='V'または'I'の場合)。

出力パラメーター

<i>d</i>	<i>n</i> 個の固有値が、降順に格納される (<i>info</i> > 0 でない場合)。 <i>info</i> も参照のこと。
<i>e</i>	終了時に上書きされる。
<i>z</i>	<i>info</i> = 0 の場合、 <i>n</i> 個の正規直交固有ベクトルが、列ごとに格納される (<i>i</i> 番目の列は、 <i>i</i> 番目の固有値に対応する)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、先頭から次数 <i>i</i> の小行列 (したがって <i>T</i> そのもの) は正定値ではないことを示す。 <i>info</i> = <i>n</i> + <i>i</i> の場合、特異値を計算するためのアルゴリズムが収束していない (<i>i</i> 個の非対角成分がゼロに収束していない) ことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `pteqr` のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>compz</i>	省略された場合、この引数は、引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>compz</i> = 'I' (<i>z</i> が存在する場合)、 <i>compz</i> = 'N' (<i>z</i> が省略された場合)。 存在する場合、 <i>compz</i> は 'I' または 'V' と等しくなければならず、引数 <i>z</i> も存在しなければならない。 <i>compz</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。

z が省略されると、実数および複素数の場合に曖昧な選択が行われるため、`ptegr` ルーチンには 2 つの Fortran 95 インターフェイスが必要な点に注意する。したがって、実数の場合 (単精度または倍精度) は `rptegr`、複素数の場合 (単精度または倍精度) は `ptegr` を使用する。

アプリケーション・ノート

λ_i が正確な固有値で、 μ_i が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\varepsilon K \lambda_i$$

$c(n)$ は n の漸増する関数、 ε はマシン精度、 $K = \|DTD\|_2 \|(DTD)^{-1}\|_2$ 、および D は $d_{ii} = t_{ii}^{-1/2}$ の対角成分である。

z_i が対応する正確な固有ベクトルで、 w_i が対応する計算で求めたベクトルの場合、それらの間の角度 $\theta(z_i, w_i)$ は次のようになる。

$$\theta(u_i, w_i) \leq c(n)\epsilon K / \min_{i \neq j} (|\lambda_i - \lambda_j| / |\lambda_i + \lambda_j|)$$

$\min_{i \neq j} (|\lambda_i - \lambda_j| / |\lambda_i + \lambda_j|)$ は、 λ_i ともう一方の固有値との間の相対誤差である。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まる。通常は、ほぼ次の値になる。

$30n^2$ ($compz = 'N'$ の場合)。

$6n^3$ (複素数型の場合、 $12n^3$) ($compz = 'V'$ または $'I'$ の場合)。

?stebz

実対称三重対角行列の選択された固有値を二分法で計算する。

構文

Fortran 77:

```
call sstebz (range, order, n, vl, vu, il, iu, abstol, d, e, m, nsplit, w,
            iblock, isplit, work, iwork, info)
call dstebz (range, order, n, vl, vu, il, iu, abstol, d, e, m, nsplit, w,
            iblock, isplit, work, iwork, info)
```

Fortran 95:

```
call stebz(d, e, m, nsplit, w, iblock, isplit [,order] [,vl] [,vu] [,il]
          [,iu] [,abstol] [,info])
```

説明

このルーチンは、実対称三重対角行列 T の固有値の一部 (または全部) を、二分法によって計算する。このルーチンでは、ゼロまたは無視できる程度の非対角成分を検索し、 T をブロック - 対角形式 $T = \text{diag}(T_1, T_2, \dots)$ に分解できるかどうかを調べる。次に、 T_i の各ブロックに対して二分法を実行し、計算で求めた各固有値のブロックのインデックスを返す。そのため、「[?stein](#)」を呼び出した後に、続けてこのブロック構造を利用できる。

参照

[?laebz](#)。

入力パラメーター

<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>order</i>	CHARACTER*1。'B' または 'E' でなければならない。

	<i>order</i> = 'B' の場合、一連の固有値は、分解された各ブロック内で最小のものから昇順に配置される。
	<i>order</i> = 'E' の場合、固有値が行列全体で最小のものから昇順に配置される。
<i>n</i>	INTEGER。行列 <i>T</i> の次数 ($n \geq 0$)。
<i>vl, vu</i>	REAL (sstebz の場合) DOUBLE PRECISION (dstebz の場合)。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。次の制約がある。 $1 \leq il \leq iu \leq n$ 。 <i>range</i> = 'I' の場合、 $il \leq i \leq iu$ となるような固有値 λ_i を計算する (固有値 λ_i は昇順とする)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (sstebz の場合) DOUBLE PRECISION (dstebz の場合)。 各固有値に対する絶対許容値は、必ず指定しなければならない。固有値 (または集積値) は、幅 <i>abstol</i> の区間内に存在している場合に、収束しているものとみなされる。 <i>abstol</i> ≤ 0.0 の場合、許容値は $\epsilon \ T\ _1$ (ϵ はマシン精度) になる。
<i>d, e, work</i>	REAL (sstebz の場合) DOUBLE PRECISION (dstebz の場合)。 配列: <i>d</i> (*) には、 <i>T</i> の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、 <i>T</i> の非対角成分が格納される。 <i>e</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、 $\max(1, 4n)$ 以上でなければならない。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $\max(1, 3n)$ 以上。
出力パラメーター	
<i>m</i>	INTEGER。見つかった固有値の実際の個数。
<i>nsplit</i>	INTEGER。 <i>T</i> で見つかった対角ブロックの個数。
<i>w</i>	REAL (sstebz の場合) DOUBLE PRECISION (dstebz の場合)。 配列、次元は $\max(1, n)$ 以上。 計算で求めた固有値が、 <i>w</i> (1) ~ <i>w</i> (<i>m</i>) に格納される。
<i>iblock, isplit</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 正値 <i>iblock</i> (<i>i</i>) は、 <i>w</i> (<i>i</i>) に格納されている固有値のブロック番号である (<i>info</i> を参照)。 <i>isplit</i> の先頭から <i>nsplit</i> 個の成分には、ブロック <i>T</i> ₁ に

1 ~ $isplit(1)$ の行列を格納し、ブロック T_2 に $isplit(1)+1$ ~ $isplit(2)$ の行列を格納するというように、 T をブロック T_i に分割した点が格納される。

info

INTEGER。

info = 0 の場合、正常に終了したことを示す。

info = 1 の場合、*range* = 'A' または 'V' であれば、所定の精度で計算できなかった固有値がある。*iblock*(*i*) < 0 の場合、*w*(*i*) に格納されている固有値が収束していない。

info = 2 の場合、*range* = 'I' であれば、計算できなかった固有値がある。*range* = 'A' と指定して、このルーチンを再度呼び出すこと。

info = 3 の場合、*range* = 'A' または 'V' であれば、*info* = 1 と同じである。

range = 'I' であれば、*info* = 2 と同じである。

info = 4 の場合、固有値を計算することはできない。コンピュータ上で、浮動小数演算が期待通りに動作していない。

info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stebz* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i> -1) のベクトルを格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>iblock</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>isplit</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>order</i>	'B' または 'E' でなければならない。デフォルト値は 'B'。
<i>v1</i>	この引数のデフォルト値は、 <i>v1</i> = -HUGE(<i>v1</i>) であり、ここで HUGE(<i>a</i>) は引数 <i>a</i> と同じ精度のマシン最大値を意味する。
<i>vu</i>	この引数のデフォルト値は、 <i>vu</i> = HUGE(<i>v1</i>) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この引数のデフォルト値は、 <i>abstol</i> = 0.0_WP である。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

T の固有値が比較的高い精度で計算される場合は、その絶対値の変動が大きくても、標準の QR 法を使う場合などに比べて、小さな固有値がすべて正確に計算される。ただし、(このルーチンが呼び出される前に実行される) 三重対角形式への縮退によって、固有値の絶対値が大きく変動するような場合には、元の行列の固有値が小さいときに相対精度が低下する場合もある。

?stein

実対称三重対角行列の指定された固有値に対応する固有ベクトルを計算する。

構文

Fortran 77:

```
call sstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv,
           info)
call dstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv,
           info)
call cstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv,
           info)
call zstein(n, d, e, m, w, iblock, isplit, z, ldz, work, iwork, ifailv,
           info)
```

Fortran 95:

```
call stein(d, e, w, iblock, isplit, z [,ifailv] [,info])
```

説明

このルーチンは、実対称三重対角行列 T の指定された固有値に対応する固有ベクトルを、反転 (逆行列の計算) を繰り返して求める。このルーチンは特に、`order='B'` と指定して `?stebz` を呼び出し、指定の固有値を計算した後で使うように設計されている。ただし、このルーチンは他のルーチンを使って固有値を計算した後に使用できる。`?stebz` の後でこのルーチンを使う場合は、各 T_i ブロックごとに反転を個々に繰り返して、そのブロック構造を利用でき、行列 T の全体を使うよりも、効率が良くなる。

対称 / エルミート行列 A の全体を三重対角形式に縮退させて T を求めた場合は、`?ormtr` または `?opmtr` (実数型の場合)、`?unmtr` または `?upmtr` (複素数型の場合) を呼び出し、 T の固有ベクトルを A の固有ベクトルに変換できる。

入力パラメーター

n	INTEGER。行列 T の次数 ($n \geq 0$)。
m	INTEGER。返すべき固有ベクトルの個数。
d, e, w	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列: $d(*)$ には、 T の対角成分を格納する。 d の次元は、 $\max(1, n)$ 以上でなければならない。

$e(*)$ には、 T の非対角成分が格納される。

e の次元は、 $\max(1, n-1)$ 以上でなければならない。

$w(*)$ は、 $w(1) \sim w(m)$ に、[?stebz](#) から返された T の固有値を格納する。 T_1 の固有値、 T_2 の固有値の順番 (降順ではない) で指定する。次の制約がある。

$iblock(i) = iblock(i+1)$ の場合、 $w(i) \leq w(i+1)$ 。

w の次元は、 $\max(1, n)$ 以上でなければならない。

iblock, isplit

INTEGER。

配列、次元は $\max(1, n)$ 以上。

配列 *iblock* と *isplit* は、*order*='B' と指定して [?stebz](#) を呼び出し、その出力として返された配列である。

order='B' と指定して [?stebz](#) を呼び出さなかった場合、*iblock* の全成分に 1 を、*isplit*(1) に n を設定する。

ldz

INTEGER。出力配列 z の第 1 次元。 $ldz \geq \max(1, n)$ 。

work

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

ワークスペース配列、次元は $\max(1, 5n)$ 以上。

iwork

INTEGER。

ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

z

REAL (*sstein* の場合)

DOUBLE PRECISION (*dstein* の場合)

COMPLEX (*cstein* の場合)

DOUBLE COMPLEX (*zstein* の場合)。

配列、次元は (*ldz*,*)。

info=0 の場合、 z には、 m 個の正規直交固有ベクトルが、列ごとに格納される (i 番目の列は、 i 番目に指定した固有値に対応する)。

ifailv

INTEGER。配列、次元は $\max(1, m)$ 以上。

info = $i > 0$ の場合、*ifailv* の先頭から i 個の成分に、収束しなかったすべての固有ベクトルのインデックスが格納される。

info

INTEGER。

info = 0 の場合、正常に終了したことを示す。

info = i の場合、 i 個の固有ベクトル (パラメーター *ifailv* によって指定) が、5 回の繰り返しによっても収束しなかったことを示す。現時点の繰り返し計算の結果は、配列 z の対応する列に格納される。

info = $-i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stein* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>iblock</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>isplit</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n,m</i>) の行列 <i>Z</i> を格納する。
<i>ifailv</i>	長さ (<i>m</i>) のベクトルを格納する。

アプリケーション・ノート

計算で求めた各固有ベクトル z_i は、行列 $T + E_i$ の正確な固有ベクトルである ($\|E_i\|_2 = O(\epsilon) \|T\|_2$)。ただし、このルーチンで求める一連の固有ベクトルは、`?steqr` によって求める固有ベクトルと比較すると、直交性の面で精度が低くなる場合もある。

?disna

対称/エルミート行列の固有ベクトル、または一般行列の左または右の特異ベクトルに対して、条件数の逆数を計算する。

構文

Fortran 77:

```
call sdisna(job, m, n, d, sep, info)
call ddisna(job, m, n, d, sep, info)
```

Fortran 95:

```
call disna(d, sep [,job] [,minmn] [,info])
```

説明

このルーチンは、実対称/複素エルミート行列の固有ベクトル、または $m \times n$ の一般行列の左または右の特異ベクトルに対して、条件数の逆数を計算する。

条件数の逆数とは、最も近い固有値 (または特異値) 間の「ギャップ」である。

計算で求めた i 番目のベクトルの誤差範囲 (ラジアン単位の角度で測定) は、次のように表される。

$$\text{slamch}('E') * (\text{anorm} / \text{sep}(i))$$

ここで、 $\text{anorm} = \|A\|_2 = \max(|d(j)|)$ 。誤差範囲を限定するには、 $\text{sep}(i)$ が $\text{slamch}('E') * \text{anorm}$ より小さくなくてはならない。

`?disna` は、汎用対称固有値問題における固有ベクトルの誤差範囲の計算にも使用できる。

入力パラメーター

<i>job</i>	CHARACTER*1。'E'、'L'、または'R' でなければならない。 どの問題に対して条件数の逆数を計算するか指定する。 <i>job</i> ='E': 対称/エルミート行列の固有ベクトル。 <i>job</i> ='L': 一般行列の左特異ベクトル。 <i>job</i> ='R': 一般行列の右特異ベクトル。
<i>m</i>	INTEGER。行列の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。 <i>job</i> ='L' または 'R' の場合、行列の列数 ($n \geq 0$)。 <i>job</i> ='E' の場合は無視される。
<i>d</i>	REAL (<i>sdisna</i> の場合) DOUBLE PRECISION (<i>ddisna</i> の場合)。 配列、次元は $\max(1,m)$ 以上 (<i>job</i> ='E' の場合) または $\max(1, \min(m,n))$ 以上 (<i>job</i> ='L' または 'R' の場合)。 この配列には、行列の固有値 (<i>job</i> ='E' の場合) または特異値 (<i>job</i> ='L' または 'R' の場合) が、昇順または降順に格納されて いなければならない。特異値は、非負でなければならない。

出力パラメーター

<i>sep</i>	REAL (<i>sdisna</i> の場合) DOUBLE PRECISION (<i>ddisna</i> の場合)。 配列、次元は $\max(1,m)$ 以上 (<i>job</i> ='E' の場合) または $\max(1, \min(m,n))$ 以上 (<i>job</i> ='L' または 'R' の場合)。 ベクトルの条件数の逆数。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *disna* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ $\min(m,n)$ のベクトルを格納する。
<i>sep</i>	長さ $\min(m,n)$ のベクトルを格納する。
<i>job</i>	'E'、'L'、または'R' でなければならない。デフォルト値は'E'。
<i>minmn</i>	<i>m</i> または <i>n</i> で値の小さい方を示す。'M' または 'N' でなければならない。デフォルト値は'M'。 <i>job</i> ='E' の場合、この引数は余分である。 <i>job</i> ='L' または 'R' の場合、この引数はルーチンで使用される。

汎用対称固有値問題

汎用対称固有値問題は、「次のいずれかの方程式を満たすような固有値 λ とそれに対応する固有ベクトル z を求めること」である。

$$Az = \lambda Bz, \quad ABz = \lambda z, \quad \text{または} \quad BAz = \lambda z$$

A は $n \times n$ の対称行列またはエルミート行列、 B は $n \times n$ の対称正定値行列またはエルミート正定値行列である。

このような固有値問題では、実数型の固有値に対応する実数型の固有ベクトルが n 個存在している (エルミート行列 A と B が複素型の場合でも実数型になる)。

ここで説明する一連のルーチンを使えば、上記の一般化された問題を標準の対称固有値問題 $Cy = \lambda y$ に縮退させられる。縮退させた後の標準の対称固有値問題は、この章ですでに説明した一連の LAPACK ルーチン (「[対称固有値問題](#)」を参照) を呼び出して解くことができる。

種々のルーチンで、行列を従来の形式または圧縮形式で格納できる。縮退の前に、まず正定値行列 B を [?potrf](#) または [?pptrf](#) を使用して因子分解する必要がある。

帯行列 A と B の縮退ルーチンは、分割コレスキー因子分解を使用するが、このための専用ルーチン [?pbstf](#) が提供されている。このルーチンを使用することにより、行列 C の生成に必要な作業量が半分で済む。

表 4-4 に、汎用対称固有値問題を解くために使用できる LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 4-4 **汎用固有値問題から標準固有値問題への縮退用の計算ルーチン**

行列のタイプ	標準問題への縮退 (フル格納形式)	標準問題への縮退 (圧縮格納形式)	標準問題への縮退 (帯行列)	帯行列の 因子分解
実対称行列	?sygst	?spgst	?sbgst	?pbstf
複素エルミート行列	?hegst	?hpgst	?hbgst	?pbstf

?sygst

実対称の汎用固有値問題を標準形式に縮退させる。

構文**Fortran 77:**

```
call ssygst(itype, uplo, n, a, lda, b, ldb, info)
call dsygst(itype, uplo, n, a, lda, b, ldb, info)
```

Fortran 95:

```
call sygst(a, b [,itype] [,uplo] [,info])
```

説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, \quad ABz = \lambda z, \quad \text{または} \quad BAz = \lambda z$$

を、標準形式 $Cy = \lambda y$ に縮退させる。A は実対称行列で、B は実対称正定値行列である。このルーチンを呼び出す前に、[?potrf](#) を呼び出し、コレスキー因子分解 $B = U^T U$ または $B = LL^T$ を計算する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 <i>itype</i> = 1 の場合、汎用固有値問題は $Az = \lambda Bz$ 。 <i>uplo</i> = 'U' の場合、 $C = U^{-T}AU^{-1}$ 、 $z = U^{-1}y$ 。 <i>uplo</i> = 'L' の場合、 $C = L^{-1}AL^{-T}$ 、 $z = L^{-T}y$ 。 <i>itype</i> = 2 の場合、汎用固有値問題は $ABz = \lambda z$ 。 <i>uplo</i> = 'U' の場合、 $C = UAU^T$ 、 $z = U^{-1}y$ 。 <i>uplo</i> = 'L' の場合、 $C = L^TAL$ 、 $z = L^{-T}y$ 。 <i>itype</i> = 3 の場合、汎用固有値問題は $BAz = \lambda z$ 。 <i>uplo</i> = 'U' の場合、 $C = UAU^T$ 、 $z = U^T y$ 。 <i>uplo</i> = 'L' の場合、 $C = L^TAL$ 、 $z = Ly$ 。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> に A の上三角行列を格納する。B は、 $B = U^T U$ に因子分解した形式で指定しなければならない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> に A の下三角行列を格納する。B は、 $B = LL^T$ に因子分解した形式で指定しなければならない。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i>	REAL (ssygst の場合) DOUBLE PRECISION (dsygst の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、A の上三角行列または下三角行列を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、次のコレスキー因子分解された行列 B を格納する。 $B = U^T U$ または $B = LL^T$ (?potrf から返された行列)。 <i>b</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。a の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。b の第1次元。 $\max(1, n)$ 以上。

出力パラメーター

<i>a</i>	引数 <i>itype</i> と <i>uplo</i> の値に従って、A の上三角行列または下三角行列が、C の上三角行列または下三角行列によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sygst` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n,n) の行列 A を格納する。
<code>b</code>	サイズ (n,n) の行列 B を格納する。
<code>itype</code>	1、2、または3 でなければならない。デフォルト値は 1。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

縮退させた行列 C を得るのは、安定した処理である。ただし、 B^{-1} ($itype = 1$ の場合) または B ($itype = 2$ または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して B の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 n^3 である。

?hegst

複素エルミートの汎用固有値問題を標準形式に縮退させる。

構文

Fortran 77:

```
call chegst(itype, uplo, n, a, lda, b, ldb, info)
call zhegst(itype, uplo, n, a, lda, b, ldb, info)
```

Fortran 95:

```
call hegst(a, b [,itype] [,uplo] [,info])
```

説明

このルーチンは、次の複素エルミートの汎用固有値問題

$$Az = \lambda Bz, ABz = \lambda z, \text{ または } BAz = \lambda z$$

を、標準形式 $Cy = \lambda y$ に縮退させる。行列 A は複素エルミート行列で、 B は複素エルミート正定値行列である。このルーチン呼び出す前に、[?potrf](#) を呼び出し、コレスキー因子分解 $B = U^H U$ または $B = LL^H$ を計算する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 <i>itype</i> = 1 の場合、汎用固有値問題は $Az = \lambda Bz$ 。 <i>uplo</i> = 'U' の場合、 $C = U^{-H}AU^{-1}$ 、 $z = U^{-1}y$ 。 <i>uplo</i> = 'L' の場合、 $C = L^{-1}AL^{-H}$ 、 $z = L^{-H}y$ 。 <i>itype</i> = 2 の場合、汎用固有値問題は $ABz = \lambda z$ 。 <i>uplo</i> = 'U' の場合、 $C = UAU^H$ 、 $z = U^{-1}y$ 。 <i>uplo</i> = 'L' の場合、 $C = L^HAL$ 、 $z = L^{-H}y$ 。 <i>itype</i> = 3 の場合、汎用固有値問題は $BAz = \lambda z$ 。 <i>uplo</i> = 'U' の場合、 $C = UAU^H$ 、 $z = U^Hy$ 。 <i>uplo</i> = 'L' の場合、 $C = L^HAL$ 、 $z = Ly$ 。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> に <i>A</i> の上三角行列を格納する。 <i>B</i> は、 $B = U^HU$ に因子分解した形式で指定しなければならない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> に <i>A</i> の下三角行列を格納する。 <i>B</i> は、 $B = LL^H$ に因子分解した形式で指定しなければならない。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i>	COMPLEX (chegst の場合) DOUBLE COMPLEX (zhegst の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、 <i>A</i> の上三角行列または下三角行列を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、次のコレスキー因子分解された行列 <i>B</i> を格納する。 $B = U^HU$ または $B = LL^H$ (?potrf から返された行列)。 <i>b</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第1次元。 $\max(1, n)$ 以上。

出力パラメーター

<i>a</i>	引数 <i>itype</i> と <i>uplo</i> の値に従って、 <i>A</i> の上三角行列または下三角行列が、 <i>C</i> の上三角行列または下三角行列によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hegst* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>B</i> を格納する。

itype 1、2、または3 でなければならない。デフォルト値は1。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

縮退させた行列 C を得るのは、安定した処理である。ただし、 B^{-1} ($itype = 1$ の場合) または B ($itype = 2$ または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の1ステップとして使う場合、反転処理に関して B の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 n^3 である。

?spgst

圧縮格納形式の実対称の汎用固有値問題を標準形式に縮退させる。

構文

Fortran 77:

```
call sspgst(itype, uplo, n, ap, bp, info)
call dspgst(itype, uplo, n, ap, bp, info)
```

Fortran 95:

```
call spgst(a, b [,itype] [,uplo] [,info])
```

説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, \quad ABz = \lambda z, \quad \text{または} \quad BAz = \lambda z$$

を、標準形式 $Cy = \lambda y$ に縮退させる。 A は実対称行列で、 B は実対称正定値行列である。このルーチンを呼び出す前に、[?pptrf](#) を呼び出し、コレスキー因子分解 $B = U^T U$ または $B = LL^T$ を計算する。

入力パラメーター

itype INTEGER。1、2、または3 のいずれかでなければならない。
itype = 1 の場合、汎用固有値問題は $Az = \lambda Bz$ 。
 uplo = 'U' の場合、 $C = U^{-T} A U^{-1}$ 、 $z = U^{-1} y$ 。
 uplo = 'L' の場合、 $C = L^{-1} A L^{-T}$ 、 $z = L^{-T} y$ 。
itype = 2 の場合、汎用固有値問題は $ABz = \lambda z$ 。
 uplo = 'U' の場合、 $C = U A U^T$ 、 $z = U^{-1} y$ 。
 uplo = 'L' の場合、 $C = L^T A L$ 、 $z = L^{-T} y$ 。
itype = 3 の場合、汎用固有値問題は $BAz = \lambda z$ 。
 uplo = 'U' の場合、 $C = U A U^T$ 、 $z = U^T y$ 。
 uplo = 'L' の場合、 $C = L^T A L$ 、 $z = L y$ 。

<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ap</i> に <i>A</i> の上三角行列 (圧縮形式) を格納する。 <i>B</i> は、 $B = U^T U$ に因子分解した形式で指定しなければならない。 <i>uplo</i> = 'L' の場合、 <i>ap</i> に <i>A</i> の下三角行列 (圧縮形式) を格納する。 <i>B</i> は、 $B = LL^T$ に因子分解した形式で指定しなければならない。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>ap</i> , <i>bp</i>	REAL (sspgst の場合) DOUBLE PRECISION (dsgpst の場合)。 配列: <i>ap</i> (*) には、 <i>A</i> の上三角行列または下三角行列 (圧縮形式) を格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>B</i> (同一の <i>uplo</i> 値を指定して ?pptrf から返されたもの) のコレスキー因数 (圧縮形式) を格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

出力パラメーター

<i>ap</i>	引数 <i>itype</i> と <i>uplo</i> の値に従って、 <i>A</i> の上三角行列または下三角行列が、 <i>C</i> の上三角行列または下三角行列によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spgst* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>B</i> を格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

縮退させた行列 *C* を得るのは、安定した処理である。ただし、 B^{-1} (*itype* = 1 の場合) または *B* (*itype* = 2 または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 n^3 である。

?hpgst

圧縮格納形式の複素エルミート行列の汎用固有値問題を標準形式に縮退させる。

構文

Fortran 77:

```
call chpgst(itype, uplo, n, ap, bp, info)
call zhpst(itype, uplo, n, ap, bp, info)
```

Fortran 95

```
call hpgst(a, b [,itype] [,uplo] [,info])
```

説明

このルーチンは、汎用対称固有値問題

$$Az = \lambda Bz, \quad ABz = \lambda z, \quad \text{または} \quad BAz = \lambda z$$

を、標準形式 $Cy = \lambda y$ に縮退させる。 A は実対称行列で、 B は実対称正定値行列である。このルーチンを呼び出す前に、 [?pptrf](#) を呼び出し、コレスキー因子分解 $B = U^H U$ または $B = LL^H$ を計算する。

入力パラメーター

itype INTEGER。 1、 2、 または 3 のいずれかでなければならない。
itype = 1 の場合、汎用固有値問題は $Az = \lambda Bz$ 。
 uplo = 'U' の場合、 $C = U^{-H} A U^{-1}$ 、 $z = U^{-1} y$ 。
 uplo = 'L' の場合、 $C = L^{-1} A L^{-H}$ 、 $z = L^{-H} y$ 。
itype = 2 の場合、汎用固有値問題は $ABz = \lambda z$ 。
 uplo = 'U' の場合、 $C = U A U^H$ 、 $z = U^{-1} y$ 。
 uplo = 'L' の場合、 $C = L^H A L$ 、 $z = L^{-H} y$ 。
itype = 3 の場合、汎用固有値問題は $BAz = \lambda z$ 。
 uplo = 'U' の場合、 $C = U A U^H$ 、 $z = U^H y$ 。
 uplo = 'L' の場合、 $C = L^H A L$ 、 $z = L y$ 。

uplo CHARACTER*1。 'U' または 'L' でなければならない。
uplo = 'U' の場合、 *ap* に A の上三角行列 (圧縮形式) を格納する。
 B は、 $B = U^H U$ に因子分解した形式で指定しなければならない。
uplo = 'L' の場合、 *ap* に A の下三角行列 (圧縮形式) を格納する。
 B は、 $B = LL^H$ に因子分解した形式で指定しなければならない。

n INTEGER。 行列 A と B の次数 ($n \geq 0$)。

ap, *bp* COMPLEX (chpgst の場合)
 DOUBLE COMPLEX (zhpst の場合)。
 配列:
ap(*) には、 A の上三角行列または下三角行列 (圧縮形式) を格納する。
 a の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

$bp(*)$ には、 B (同一の $uplo$ 値を指定して `?pptrf` から返されたもの) のコレスキー因数 (圧縮形式) を格納する。
 b の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

出力パラメーター

ap 引数 $itype$ と $uplo$ の値に従って、 A の上三角行列または下三角行列が、 C の上三角行列または下三角行列によって上書きされる。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpgst` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 ap を意味する。
 サイズ $(n*(n+1)/2)$ の配列 A を格納する。

b Fortran 77 インターフェイスでの引数 bp を意味する。
 サイズ $(n*(n+1)/2)$ の配列 B を格納する。

$itype$ 1、2、または 3 でなければならない。デフォルト値は 1。

$uplo$ 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

縮退させた行列 C を得るのは、安定した処理である。ただし、 B^{-1} ($itype = 1$ の場合) または B ($itype = 2$ または 3 の場合) が、暗黙的に乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して B の条件が悪いと、精度が大きく低下するときがある。

浮動小数演算のおおよその総数は、 n^3 である。

?sbgst

?pbstf による因子分解を使用して、帯形式の実対称行列の汎用固有値問題を標準形式に縮退させる。

構文

Fortran 77:

```
call ssbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
            info)
call dsbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
            info)
```

Fortran 95:

```
call sbgst(a, b [,x] [,uplo] [,info])
```

説明

実対称汎用固有値問題 ($Az = \lambda Bz$) を標準形式 ($Cy = \lambda y$) に縮退させる (A 、 B 、および C は帯行列) には、このルーチンに先立って、正定値行列 B の分割コレスキー因子分解 ($B: B = S^T S$) を行う [spbstf](#)/[dpbstf](#) を呼び出す必要がある。分割コレスキー因子分解は、通常のコレスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、 A を $C = X^T A X$ で上書きする。ここで、 $X = S^{-1} Q$ と Q は A の帯幅を保持するために (暗黙的に) 選択された直交行列である。

また、このルーチンには、 X を蓄積するオプションがある。その場合、 z が C の固有ベクトルであれば、 Xz は元の問題の固有ベクトルとなる。

入力パラメーター

<i>vect</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>vect</i> = 'N' の場合、行列 X は返されない。 <i>vect</i> = 'V' の場合、行列 X が返される。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 B の優対角成分または劣対角成分の数 ($ka \geq kb \geq 0$)。
<i>ab, bb, work</i>	REAL (ssbgst の場合) DOUBLE PRECISION (dsbgst の場合) <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 A の上三角部分または下三角部分を帯形式で格納する配列である。配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> (<i>ldbb</i> ,*) は、(<i>uplo</i> 、 <i>n</i> 、および <i>kb</i> の値に従って) spbstf / dpbstf から返された分割コレスキー因子 B を帯形式で格納する

配列である。配列 *bb* の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

work()* は、ワークスペース配列、次元は $\max(1, 2*n)$ 以上。

<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldx</i>	出力配列 <i>x</i> の第 1 次元。次の制約がある。 <i>vect</i> = 'N' の場合、 $ldx \geq 1$ 。 <i>vect</i> = 'V' の場合、 $ldx \geq \max(1, n)$ 。

出力パラメーター

<i>ab</i>	終了時に、 <i>uplo</i> の値に従って、 <i>C</i> の上または下三角部分で上書きされる。
<i>x</i>	REAL (<i>ssbgst</i> の場合) DOUBLE PRECISION (<i>dsbgst</i> の場合) 配列： <i>vect</i> = 'V' の場合、 <i>x</i> (<i>ldx</i> ,*) に $n \times n$ の行列 $X = S^{-1}Q$ が格納される。 <i>vect</i> = 'N' の場合、 <i>x</i> は参照されない。 <i>x</i> の第 2 次元は、 $\max(1, n)$ 以上 (<i>vect</i> = 'V' の場合) または 1 以上 (<i>vect</i> = 'N' の場合) でなければならない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbgst* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ ($ka+1, n$) の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ ($kd+1, n$) の配列 <i>B</i> を格納する。
<i>x</i>	サイズ (n, n) の行列 <i>X</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>vect</i>	引数 <i>x</i> の存在に基づいて以下のように復元される。 <i>vect</i> = 'V' (<i>x</i> が存在する場合)、 <i>vect</i> = 'N' (<i>x</i> が省略された場合)。

アプリケーション・ノート

縮退した行列 *C* を生成するには、暗黙的に B^{-1} が乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して *B* の条件が悪いと、精度が大きく低下するときがある。

浮動小数点演算の総数は、 $vect = 'N'$ の場合、約 $6n^2 * kb$ になる。 $vect = 'V'$ の場合、さらに $(3/2)n^3 * (kb/ka)$ 回の演算が必要になる。ただしこれらの値は、 ka と kb がどちらも n と比べて十分に小さい場合の推定値である。

?hbgst

?pbstf による因子分解を使用して、帯形式の複素エルミート行列の汎用固有値問題を標準形式に縮退させる。

構文

Fortran 77:

```
call chbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
           rwork, info)
call zhbgst(vect, uplo, n, ka, kb, ab, ldab, bb, ldbb, x, ldx, work,
           rwork, info)
```

Fortran 95:

```
call hbgst(a, b [,x] [,uplo] [,info])
```

説明

複素エルミート行列の汎用固有値問題 ($Az = \lambda Bz$) を標準形式 ($Cy = \lambda y$) に縮退させる (A 、 B 、 C は帯行列) には、このルーチンに先立って、正定値行列 B の分割コレスキー因子分解 ($B: B = S^H S$) を行う [cpbstf/zpbstf](#) を呼び出す必要がある。分割コレスキー因子分解は、通常のコレスキー因子分解と比べて、作業量が半分で済む。

このルーチンは、 A を $C = X^H A X$ で上書きする。ここで、 $X = S^{-1} Q$ と Q は A の帯幅を保持するために (暗黙的に) 選択されたユニタリ行列である。

また、このルーチンには、 X を蓄積するオプションがある。その場合、 z が C の固有ベクトルであれば、 Xz は元の問題の固有ベクトルとなる。

入力パラメーター

<i>vect</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>vect</i> = 'N' の場合、行列 X は返されない。 <i>vect</i> = 'V' の場合、行列 X が返される。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 B の優対角成分または劣対角成分の数 ($ka \geq kb \geq 0$)。

<i>ab, bb, work</i>	COMPLEX (chbgvx の場合) DOUBLE COMPLEX (zhbgvx の場合) 配列: <i>ab(ldab,*)</i> は、(<i>uplo</i> の値に従って)エルミート行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb(lddb,*)</i> は、(<i>uplo</i> 、 <i>n</i> 、および <i>kb</i> の値に従って) cpbstf/zpbstf から返された分割コレスキー因子 <i>B</i> を帯形式で格納する配列である。配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(*)</i> は、ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 <i>ka</i> +1 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 <i>kb</i> +1 以上でなければならない。
<i>ldx</i>	出力配列 <i>x</i> の第 1 次元。次の制約がある。 <i>vect</i> ='N' の場合、 $ldx \geq 1$ 。 <i>vect</i> ='V' の場合、 $ldx \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbgst の場合) DOUBLE PRECISION (zhbgst の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>ab</i>	終了時に、 <i>uplo</i> の値に従って、 <i>C</i> の上または下三角部分で上書きされる。
<i>x</i>	COMPLEX (chbgst の場合) DOUBLE COMPLEX (zhbgst の場合) 配列: <i>vect</i> ='V' の場合、 <i>x(ldx,*)</i> に $n \times n$ の行列 $X = S^{-1}Q$ が格納される。 <i>vect</i> ='N' の場合、 <i>x</i> は参照されない。 <i>x</i> の第 2 次元は、 $\max(1, n)$ 以上 (<i>vect</i> ='V' の場合) または 1 以上 (<i>vect</i> ='N' の場合) でなければならない。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hbgst* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ (<i>ka</i> +1, <i>n</i>) の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ (<i>kd</i> +1, <i>n</i>) の配列 <i>B</i> を格納する。
<i>x</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>X</i> を格納する。

`uplo` 'U' または 'L' でなければならない。デフォルト値は 'U'。
`vect` 引数 x の存在に基づいて以下のように復元される。
`vect = 'V'` (x が存在する場合)、
`vect = 'N'` (x が省略された場合)。

アプリケーション・ノート

縮退した行列 C を生成するには、暗黙的に B^{-1} が乗算される。このルーチンを元の問題の固有値および固有ベクトルを計算する際の 1 ステップとして使う場合、反転処理に関して B の条件が悪いと、精度が大きく低下するときがある。

浮動小数点演算の総数は、`vect = 'N'` の場合、約 $20n^2 \cdot kb$ になる。`vect = 'V'` の場合、さらに $5n^3 \cdot (kb/ka)$ 回の演算が必要になる。ただしこれらの値は、 ka と kb がどちらも n と比べて十分に小さい場合の推定値である。

?pbstf

?sbgst/?hbgst で使用するために、帯形式の実対称/複素エルミート正定値行列の分割コレスキー因子分解を行う。

構文

Fortran 77:

```
call spbstf(uplo, n, kb, bb, ldbb, info)
call dpbstf(uplo, n, kb, bb, ldbb, info)
call cpbstf(uplo, n, kb, bb, ldbb, info)
call zpbstf(uplo, n, kb, bb, ldbb, info)
```

Fortran 95:

```
call pbstf(b [,uplo] [,info])
```

説明

このルーチンは、帯形式の実対称/複素エルミート正定値行列 B の分割コレスキー因子分解を行う。このルーチンは、[?sbgst](#)/[?hbgst](#) とともに使用する。

分割コレスキー因子分解は、 $B = S^T S$ (複素数型の場合、 $B = S^H S$) と表される。ここで、 S は、帯幅が B と同じ帯行列で、最初の $(n+kb)/2$ 行は上三角行列、残りの行は下三角行列である。

入力パラメーター

`uplo` CHARACTER*1。'U' または 'L' でなければならない。
`uplo = 'U'` の場合、`bb` に B の上三角部分を格納する。
`uplo = 'L'` の場合、`bb` に B の下三角部分を格納する。
`n` INTEGER。行列 B の次数 ($n \geq 0$)。
`kb` INTEGER。 B の優対角成分または劣対角成分の数 ($kb \geq 0$)。

<i>bb</i>	REAL (spbstf の場合) DOUBLE PRECISION (dpbstf の場合) COMPLEX (cpbstf の場合) DOUBLE COMPLEX (zpbstf の場合)。 <i>bb(ldbb,*)</i> は、(<i>uplo</i> の値に従って) 行列 <i>B</i> の上三角部分または下三角部分を帯形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldbb</i>	INTEGER。 <i>bb</i> の第 1 次元。 <i>kb+1</i> 以上でなければならない。

出力パラメーター

<i>bb</i>	終了時に、分割コレスキー因子 <i>S</i> の各成分で上書きされる。
<i>info</i>	INTEGER。 <i>info=0</i> の場合、正常に終了したことを示す。 <i>info=i</i> の場合、更新しようとした成分 b_{ii} が負値の平方根になるため、因子分解を完了できなかったことを示す。すなわち、行列 <i>B</i> が正定値でない。 <i>info=-i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *pbstf* のインターフェイスの詳細を以下に示す。

<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ (<i>kd+1, n</i>) の配列 <i>B</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた因子 *S* は、正確には摂動行列 $B + E$ の因子である。

$c(n)$ は n の適度な線形関数、 ε はマシン精度。

$$|E| \leq c(kb+1)\varepsilon|S^{\#}||S|, \quad |e_{ij}| \leq c(kb+1)\varepsilon\sqrt{b_{ii}b_{jj}}$$

浮動小数点演算の総数は、実数型の場合は約 $n(kb+1)^2$ である。複素数型の演算回数、この 4 倍になる。これらの値は、*kb* が n と比べて十分に小さい場合の推定値である。

このルーチンを呼び出した後、[?sbqst/?hbqst](#) を呼び出して汎用固有値問題 ($Az = \lambda Bz$ 、ただし、*A* と *B* は帯形式、*B* は正定値) を解くことができる。

非対称固有値問題

このセクションでは、非対称固有値問題の解決、一般行列の Schur 因子分解の計算、およびそれらに関連する各種の計算タスクを実行するための LAPACK ルーチンについて説明する。

*非対称固有値問題*は、「非対称 (または非エルミート) の行列 A が与えられたときに、次のいずれかの方程式を満たす固有値 λ と、それに対応する固有ベクトル z を求めること」である。

$$Az = \lambda z \text{ (右固有ベクトル } z)$$

または

$$z^H A = \lambda z^H \text{ (左固有ベクトル } z)$$

非対称固有値問題には、次の特性がある。

- 固有ベクトルの個数が、行列の次数より少ない場合がある (ただし、 A の相異なる固有値の個数以上である)。
- 実行列 A に対して、固有値が複素数の場合もある。
- 実非対称行列が、固有ベクトル z に対応する固有値として複素数 $a + bi$ を持っている場合は、 $a - bi$ も固有値である。
固有値 $a - bi$ は、 z の成分に対して複素共役になっている成分の固有ベクトルに対応する。

LAPACK を使って非対称固有値問題を解くには、通常、行列を上 Hessenberg 形式に縮退させた後、得られた Hessenberg 行列を使って固有値問題を解く必要がある。[表 4-5](#) に示されている LAPACK ルーチン (Fortran-77 インターフェイス) を利用すれば、直交 (またはユニタリー) の相似変換 $A = QHQ^H$ を使って、該当行列を上 Hessenberg 形式に縮退させることができる。また、Hessenberg 行列を使って固有値問題を解き、行列の Schur 因子分解を求め、対応する条件数を計算できる。

Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない ([「ルーチン命名規則」](#) を参照)。

[図 4-4](#) のデシジョンツリーを参照すれば、実非対称行列を使って固有値問題を解くためのルーチン (1 つまたは複数個) を簡単に選べる。

複素非エルミート行列を使って固有値問題を解く場合は、[図 4-5](#) のデシジョンツリーを参照する。

表 4-5 非対称固有値問題を解くための計算ルーチン

機能	実行列用のルーチン	複素行列用のルーチン
Hessenberg 形式 $A = QHQ^H$ に縮退させる	?gehrd	?gehrd
行列 Q を生成する	?orghr	?unghr
行列 Q を適用する	?ormhr	?unmhr
行列の平衡化	?gebal	?gebal
平衡化した行列の固有ベクトルを元の行列の固有ベクトルに変換する	?gebak	?gebak
固有値と Schur 因子分解を求める (QR アルゴリズム)	?hseqr	?hseqr
Hessenberg 形式から固有ベクトルを求める (反転繰り返し法)	?hsein	?hsein

表 4-5 非対称固有値問題を解くための計算ルーチン

機能	実行列用のルーチン	複素行列用のルーチン
Schur の因子分解から固有ベクトルを求める	?trevc	?trevc
固有値と固有ベクトルの条件数を見積もる	?trsna	?trsna
Schur の因子分解の順序を変更する	?trexc	?trexc
Schur の因子分解の順序変更、不変部分空間の検出、条件数の見積もり	?trsen	?trsen
シルベスター式を解く	?trsyl	?trsyl

図 4-4 デシジョンツリー：実非対称固有値問題

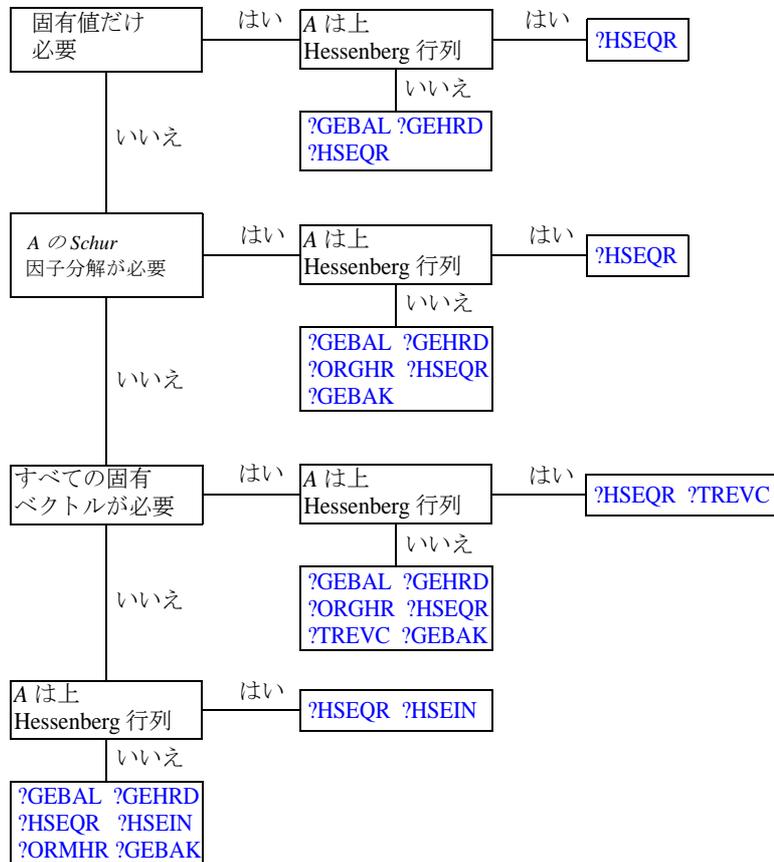
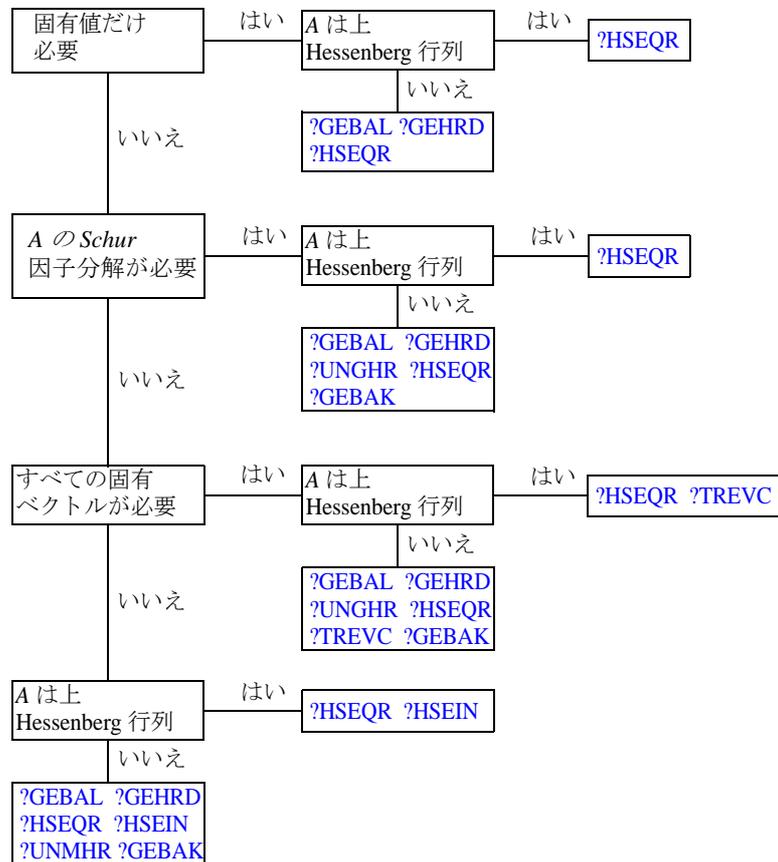


図 4-5 デシジョンツリー：複素非エルミート固有値問題



?gehrd

一般行列を上 Hessenberg 形式に縮退させる。

構文

Fortran 77:

```

call sgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call dgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call cgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)
call zgehrd(n, ilo, ihi, a, lda, tau, work, lwork, info)

```

Fortran 95:

```

call gehrd(a [,tau] [,ilo] [,ihi] [,info])

```

説明

このルーチンは、直交またはユニタリーの相似変換 $A = QHQ^H$ によって、一般行列 A を上 Hessenberg 形式 H に縮退させる。 H は、実数の対角成分を持つ。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は基本リフレクターの積として表現される。一連のルーチンでは、この形式で表現される Q を操作する。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。 A が ?gebal から出力された場合、 ilo と ihi に、そのルーチンから返された値を設定しなければならない。そうでない場合、 $ilo = 1$ かつ $ihi = n$ 。 ($n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$)。
$a, work$	REAL (sgehrd の場合) DOUBLE PRECISION (dgehrd の場合) COMPLEX (cgehrd の場合) DOUBLE COMPLEX (zgehrd の場合)。 配列： $a(lda, *)$ には、行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
$lwork$	INTEGER。配列 $work$ のサイズ。 $\max(1, n)$ 以上。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	上 Hessenberg 行列 H と行列 Q の各成分によって上書きされる。 H の劣対角成分は、実数である。
tau	REAL (sgehrd の場合) DOUBLE PRECISION (dgehrd の場合) COMPLEX (cgehrd の場合) DOUBLE COMPLEX (zgehrd の場合)。 配列、次元は $\max(1, n-1)$ 以上。 行列 Q に関するその他の情報も格納される。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 A_i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gehrd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>ilo</code>	この引数のデフォルト値は、 $ilo = 1$ である。
<code>ihi</code>	この引数のデフォルト値は、 $ihi = n$ である。

アプリケーション・ノート

パフォーマンスを最適化するには、 $lwork = n * blocksize$ に設定する。 $blocksize$ は、ブロック・アルゴリズムの最適なパフォーマンスに必要な、マシンによって異なる値 (通常は 16 ~ 64) である。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた **Hessenberg** 行列 H は、近似行列 $A + E$ の近似値である。ここで、 $\|E\|_2 < c(n)\epsilon\|A\|_2$ 、 $c(n)$ は漸増する n の関数、 ϵ はマシンによって異なる値である。

実数型の場合の浮動小数演算のおおよその総数は、 $(2/3)(ihi - ilo)^2(2ihi + 2ilo + 3n)$ である。複素数型の場合、この 4 倍になる。

?orghr

?gehrd で求めた実直交行列 Q を生成する。

構文

Fortran 77:

```
call sorghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
call dorghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call orghr(a, tau [,ilo] [,ihi] [,info])
```

説明

このルーチンは、`sgehrd/dgehrd` を呼び出して求めた直交行列 Q を明示的に生成する (ルーチン `?gehrd` は、直交相似変換 $A = QHQ^T$ によって一般実行列 A を上 Hessenberg 形式 H に縮退させ、行列 Q を $(ihi-ilo)$ 個の基本リフレクターの積として表現する。 ilo と ihi は、行列を平衡化する際に `sgebal/dgebal` で求めた値である。行列が平衡化されていない場合、 $ilo = 1$ かつ $ihi = n$ になる)。

?orghr によって生成される行列 Q は、次の構造になる。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

Q_{22} は、 $ilo \sim ihi$ の行と列を占める。

入力パラメーター

n	INTEGER。行列 Q の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。これらの値は、それぞれ ?gehrd に指定した ilo および ihi と同じでなければならない。($n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$)。
$a, tau, work$	REAL (sorghr の場合) DOUBLE PRECISION (dorghr の場合)。 配列： $a(lda, *)$ には、?gehrd から返された一連のベクトル (基本リフレクターを定義) の各成分を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $tau(*)$ には、?gehrd から返された基本リフレクターの各成分を格納する。 tau の次元は、 $\max(1, n-1)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
$lwork$	INTEGER 配列 $work$ のサイズ。 $lwork \geq \max(1, ihi-ilo)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a	$n \times n$ の直交行列 Q によって上書きされる。
$work(1)$	$info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `orghr` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>tau</code>	長さ $(n-1)$ のベクトルを格納する。
<code>ilo</code>	この引数のデフォルト値は、 $ilo = 1$ である。
<code>ihi</code>	この引数のデフォルト値は、 $ihi = n$ である。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (ihi - ilo) * blocksize$ に設定する。 $blocksize$ は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 Q は、正確な値と行列 E ($\|E\|_2 = O(\epsilon)$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、 $(4/3)(ihi - ilo)^3$ である。

このルーチンの複素数版は、[?unghr](#) である。

?ormhr

任意の実行列 C に `?gehrd` で求めた実直交行列 Q を掛ける。

構文**Fortran 77:**

```
call sormhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,
           work, lwork, info)
call dormhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,
           work, lwork, info)
```

Fortran 95:

```
call ormhr(a, tau, c [,ilo] [,ihi] [,side] [,trans] [,info])
```

説明

このルーチンは、行列 C に `sgehrd/dgehrd` を呼び出して求めた直交行列 Q を掛ける (ルーチン `?gehrd` は、直交相似変換 $A = QHQ^T$ によって一般実行列 A を上 Hessenberg 形式 H に縮退させ、行列 Q を $(ihi - ilo)$ 個の基本リフレクターの積として表現する。 ilo と ihi は、行列を平衡化する際に `sgebal/dgebal` で求めた値である。行列が平衡化されていない場合、 $ilo = 1$ かつ $ihi = n$ になる)。

?ormhr を使用すれば、 QC 、 $Q^T C$ 、 CQ 、または CQ^T のいずれかの行列積を求めることができる。その場合、 C (実矩形行列) の値が上書きされる。

?ormhr は、 H の固有ベクトルで構成される行列 V を、 A の固有ベクトルで構成される行列 QV に変換する際によく使用する。

入力パラメーター

<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 QC または $Q^T C$ を求める。 <i>side</i> = 'R' の場合、 CQ または CQ^T を求める。
<i>trans</i>	CHARACTER*1。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、 Q が C に適用される。 <i>trans</i> = 'T' の場合、 Q^T が C に適用される。
<i>m</i>	INTEGER。行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 C の列数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。これらの値は、それぞれ ?gehrd に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。 $m > 0$ かつ <i>side</i> = 'L' の場合、 $1 \leq ilo \leq ihi \leq m$ 。 $m = 0$ かつ <i>side</i> = 'L' の場合、 $ilo = 1$ かつ $ihi = 0$ 。 $n > 0$ かつ <i>side</i> = 'R' の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ かつ <i>side</i> = 'R' の場合、 $ilo = 1$ かつ $ihi = 0$ 。
<i>a, tau, c, work</i>	REAL (sormhr の場合) DOUBLE PRECISION (dormhr の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、?gehrd から返された一連のベクトル (基本リフレクターを定義) の各成分を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上 (<i>side</i> = 'L' の場合) または $\max(1, n)$ 以上 (<i>side</i> = 'R' の場合) でなければならない。 <i>tau</i> (*) には、?gehrd から返された基本リフレクターの各成分を格納する。 <i>tau</i> の次元は、 $\max(1, m-1)$ 以上 (<i>side</i> = 'L' の場合) または $\max(1, n-1)$ 以上 (<i>side</i> = 'R' の場合) でなければならない。 <i>c</i> (<i>ldc</i> ,*) には、 $m \times n$ の行列 C を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上 (<i>side</i> = 'L' の場合) または $\max(1, n)$ (<i>side</i> = 'R' の場合)。
<i>ldc</i>	INTEGER。 <i>c</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 <i>side</i> = 'L' の場合、 $lwork \geq \max(1, n)$ 。 <i>side</i> = 'R' の場合、 $lwork \geq \max(1, m)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、xerbla は <i>lwork</i> に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>c</i>	<i>C</i> は、 <i>side</i> と <i>trans</i> の値に従って、 QC 、 $Q^T C$ 、 CQ^T 、または CQ のいずれかによって上書きされる。
<i>work(1)</i>	<i>info</i> = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の <i>lwork</i> の値が <i>work(1)</i> に出力される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ormhr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>r</i> , <i>r</i>) の行列 <i>A</i> を格納する。 <i>r</i> = <i>m</i> (<i>side</i> = 'L' の場合)。 <i>r</i> = <i>n</i> (<i>side</i> = 'R' の場合)。
<i>tau</i>	長さ (<i>r</i> -1) のベクトルを格納する。
<i>c</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>C</i> を格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*side* = 'L' の場合は $n \cdot \text{blocksize}$ 以上に、*side* = 'R' の場合は $m \cdot \text{blocksize}$ 以上に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ~ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 *Q* は、正確な値と行列 *E* ($\|E\|_2 = O(\epsilon)\|C\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、次のとおりである。

$$2n(ihi-ilo)^2 \text{ (} side = 'L' \text{ の場合)},$$

$$2m(ihi-ilo)^2 \text{ (} side = 'R' \text{ の場合)}.$$

このルーチンの複素数版は、[?unmhr](#) である。

?unghr

?gehrd で求めた複素ユニタリー行列 Q を生成する。

構文

Fortran 77:

```
call cunghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
call zunghr(n, ilo, ihi, a, lda, tau, work, lwork, info)
```

Fortran 95:

```
call unghr(a, tau [,ilo] [,ihi] [,info])
```

このルーチンは、cgehrd/zgehrd (ユニタリー相似変換 $A = QHQ^H$ によって、複素行列 A を上 Hessenberg 形式 H に縮退させる) を呼び出した後で使用される。?gehrd は、行列 Q を $(ihi-ilo)$ 個の基本リフレクターの積として表現する。ilo と ihi は、行列を平衡化するために cgebal/zgebal で求めた値である。行列が平衡化されていない場合は、ilo = 1 かつ ihi = n になる。

Q を正方行列として明示的に生成するには、ルーチン ?unghr を使う。行列 Q は、次の構造になる。

$$Q = \begin{bmatrix} I & 0 & 0 \\ 0 & Q_{22} & 0 \\ 0 & 0 & I \end{bmatrix}$$

Q_{22} は、ilo ~ ihi の行と列を占める。

入力パラメーター

<i>n</i>	INTEGER。行列 Q の次数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。これらの値は、それぞれ ?gehrd に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。($n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$)。
<i>a, tau, work</i>	COMPLEX (cunghr の場合) DOUBLE COMPLEX (zunghr の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、?gehrd から返された一連のベクトル (基本リフレクターを定義) の各成分を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>tau</i> (*) には、?gehrd から返された基本リフレクターの各成分を格納する。 <i>tau</i> の次元は、 $\max(1, n-1)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。

lwork INTEGER。配列 *work* のサイズ。
 $lwork \geq \max(1, ihi-ilo)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエンタリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a $n \times n$ のユニタリ行列 Q によって上書きされる。
work(1) *info* = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work(1)* に出力される。これ以降の実行には、この *lwork* の値を使用する。
info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *unghr* のインターフェイスの詳細を以下に示す。

a サイズ (n, n) の行列 A を格納する。
tau 長さ ($n-1$) のベクトルを格納する。
ilo この引数のデフォルト値は、 $ilo = 1$ である。
ihi この引数のデフォルト値は、 $ihi = n$ である。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = (ihi-ilo) * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 Q は、正確な値と行列 E ($\|E\|_2 = O(\epsilon)$ 、 ϵ はマシン精度) だけ異なる。

実数型の場合の浮動小数演算のおおよその総数は、 $(16/3)(ihi-ilo)^3$ である。

このルーチンの実数版は、[?orghr](#) である。

?unmhr

任意の複素行列 C に ?gehrd で求めた複素ユニタリー行列 Q を掛ける。

構文

Fortran 77:

```
call cunmhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,
           work, lwork, info)
call zunmhr(side, trans, m, n, ilo, ihi, a, lda, tau, c, ldc,
           work, lwork, info)
```

Fortran 95:

```
call unmhr(a, tau, c [,ilo] [,ihi] [,side] [,trans] [,info])
```

説明

このルーチンは、行列 C に cgehrd/zgehrd を呼び出して求めたユニタリー行列 Q を掛ける (ルーチン ?gehrd は、直交相似変換 $A = QHQ^H$ によって一般実行列 A を上 Hessenberg 形式 H に縮退させ、行列 Q を $(ihi - ilo)$ 個の基本リフレクターの積として表現する。 ilo と ihi は、行列を平衡化する際に cgebal/zgebal で求めた値である。行列が平衡化されていない場合は、 $ilo = 1$ かつ $ihi = n$ になる)。

?unmhr を使用すれば、 QC 、 Q^HC 、 CQ 、または CQ^H のいずれかの行列積を求めることができる。その場合、 C (複素矩形行列) の値が上書きされる。このルーチンは、 H の固有ベクトルで構成される行列 V を、 A の固有ベクトルで構成される行列 QV に変換する際によく使われる。

入力パラメーター

<i>side</i>	CHARACTER*1. 'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、 QC または Q^HC を求める。 <i>side</i> = 'R' の場合、 CQ または CQ^H を求める。
<i>trans</i>	CHARACTER*1. 'N' または 'C' でなければならない。 <i>trans</i> = 'N' の場合、 Q が C に適用される。 <i>trans</i> = 'T' の場合、 Q^H が C に適用される。
<i>m</i>	INTEGER。 行列 C の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。 行列 C の列数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。 これらの値は、それぞれ ?gehrd に指定した <i>ilo</i> および <i>ihi</i> と同じでなければならない。 $m > 0$ かつ <i>side</i> = 'L' の場合、 $1 \leq ilo \leq ihi \leq m$ 。 $m = 0$ かつ <i>side</i> = 'L' の場合、 $ilo = 1$ かつ $ihi = 0$ 。 $n > 0$ かつ <i>side</i> = 'R' の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ かつ <i>side</i> = 'R' の場合、 $ilo = 1$ かつ $ihi = 0$ 。
<i>a, tau, c, work</i>	COMPLEX (cunmhr の場合) DOUBLE COMPLEX (zunmhr の場合)。 配列:

$a(lda,*)$ には、?gehrd から返された一連のベクトル (基本リフレクターを定義) の各成分を格納する。

a の第 2 次元は、 $\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ 以上 ($side = 'R'$ の場合) でなければならない。

$tau(*)$ には、?gehrd から返された基本リフレクターの各成分を格納する。

tau の次元は、 $\max(1, m-1)$ 以上 ($side = 'L'$ の場合) または $\max(1, n-1)$ 以上 ($side = 'R'$ の場合) でなければならない。

$c(ldc,*)$ には、 $m \times n$ の行列 C を格納する。

c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上 ($side = 'L'$ の場合) または $\max(1, n)$ ($side = 'R'$ の場合)。

ldc INTEGER。 c の第 1 次元。 $\max(1, m)$ 以上。

$lwork$ INTEGER。 配列 $work$ のサイズ。
 $side = 'L'$ の場合、 $lwork \geq \max(1, n)$ 。
 $side = 'R'$ の場合、 $lwork \geq \max(1, m)$ 。
 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

c C は、 $side$ と $trans$ の値に従って、 QC 、 $Q^H C$ 、 CQ^H 、または CQ のいずれかによって上書きされる。

$work(1)$ $info = 0$ の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の $lwork$ の値が $work(1)$ に出力される。これ以降の実行には、この $lwork$ の値を使用する。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `unmhr` のインターフェイスの詳細を以下に示す。

a サイズ (r, r) の行列 A を格納する。
 $r = m$ ($side = 'L'$ の場合)。
 $r = n$ ($side = 'R'$ の場合)。

tau 長さ $(r-1)$ のベクトルを格納する。

c サイズ (m, n) の行列 C を格納する。

<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。
<i>trans</i>	'N' または 'C' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、*lwork* を、*side* = 'L' の場合は $n \cdot \text{blocksize}$ 以上に、*side* = 'R' の場合は $m \cdot \text{blocksize}$ 以上に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる（通常は 16 ～ 64）。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

計算で求めた行列 Q は、正確な値と行列 E ($\|E\|_2 = O(\epsilon)\|C\|_2$ 、 ϵ はマシン精度) だけ異なる。

浮動小数演算のおおよその総数は、次のとおりである。

$8n(ihi-ilo)^2$ (*side* = 'L' の場合)、
 $8m(ihi-ilo)^2$ (*side* = 'R' の場合)。

このルーチンの実数版は、[?ormhr](#) である。

?gebal

一般行列を平衡化し、固有値および固有ベクトルの計算精度を改善する。

構文

Fortran 77:

```
call sgebal(job, n, a, lda, ilo, ihi, scale, info)
call dgebal(job, n, a, lda, ilo, ihi, scale, info)
call cgebal(job, n, a, lda, ilo, ihi, scale, info)
call zgebal(job, n, a, lda, ilo, ihi, scale, info)
```

Fortran 95:

```
call gebal(a [,scale] [,ilo] [,ihi] [,job] [,info])
```

説明

このルーチンは、次の 2 つの相似変換の一方または両方を実行して、行列 A を平衡化する。

(1) まず、 A をブロック上三角形式に置換してみる。

$$PAP^T = A' = \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix}$$

P は置換行列、 A'_{11} と A'_{33} は上三角行列である。 A'_{11} と A'_{33} の対角成分は、 A の固有値である。 A の残りの固有値は、中央の対角ブロック A'_{22} の固有値 ($i_{lo} \sim i_{hi}$ の行と列) である。 A の固有値 (またはその Schur 因子分解) を計算するためのこれ以降の演算は、この範囲の行と列に対して適用するだけで済む。そのため、 $i_{lo} > 1$ かつ $i_{hi} < n$ の場合、作業を大幅に節約できる。適切な置換行列が存在しない場合 (存在しない場合が多い) は、 $i_{lo} = 1$ かつ $i_{hi} = n$ と設定され、 A'_{22} が A の全体になる。

(2) 対角相似変換を A' に適用し、 A'_{22} の行と列をノルム内でできるだけ近づける。

$$A'' = DA'D^{-1} = \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22} & 0 \\ 0 & 0 & I \end{bmatrix} \times \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix} \times \begin{bmatrix} I & 0 & 0 \\ 0 & D_{22}^{-1} & 0 \\ 0 & 0 & I \end{bmatrix}$$

このスケーリングによって、この行列のノルムが縮退される (つまり、 $\|A''_{22}\| < \|A'_{22}\|$)。その結果、固有値と固有ベクトルの計算精度に対する丸め誤差の影響が減る。

入力パラメーター

<i>job</i>	CHARACTER*1. 'N'、'P'、'S'、または 'B' のいずれかでなければならない。 <i>job</i> ='N' の場合、 A の置換およびスケーリングは実行されない (ただし、 <i>ilo</i> 、 <i>ihi</i> 、 <i>scale</i> は、入力された値を受け取る)。 <i>job</i> ='P' の場合、 A の置換が実行され、スケーリングは実行されない。 <i>job</i> ='S' の場合、 A のスケーリングが実行され、置換は実行されない。 <i>job</i> ='B' の場合、 A のスケーリングと置換が両方とも実行される。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i>	REAL (sgebal の場合) DOUBLE PRECISION (dgebal の場合) COMPLEX (cgebal の場合) DOUBLE COMPLEX (zgebal の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>job</i> ='N' の場合、 <i>a</i> は参照されない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。

出力パラメーター

<i>a</i>	平衡化された行列によって上書きされる (<i>job</i> ='N' の場合、 <i>a</i> は参照されない)。
----------	--

<i>ilo, ihi</i>	INTEGER。 <i>ilo</i> および <i>ihi</i> は、終了時に、 $a(i, j)$ が 0 となるような値 ($i > j$ かつ $1 \leq j < ilo$ あるいは $ihi < i \leq n$ の場合。 <i>job</i> ='N' または 'S' の場合、 $ilo=1$ 、 $ihi=n$ に設定される。
<i>scale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 置換とスケーリング係数の各成分が格納される。 もう少し詳しく説明すると、 p_j が行と列 j と交換された行と列のインデックスであり、 d_j が行と列 j を平衡化する際に使ったスケーリング係数である場合は、次のようになる。 $scale(j) = p_j$ ($j = 1, 2, \dots, ilo-1, ihi+1, \dots, n$)。 $scale(j) = d_j$ ($j = ilo, ilo+1, \dots, ihi$)。 交換の実行順序は、 $n \sim ihi+1$ 、次に、 $1 \sim ilo-1$ になる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gebal` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>scale</i>	長さ (n) のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 $ilo=1$ である。
<i>ihi</i>	この引数のデフォルト値は、 $ihi=n$ である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。

アプリケーション・ノート

誤差は、これ以降の計算における誤差に比べて無視できる程度である。

行列 A をこのルーチンによって平衡化した場合は、以降に計算する固有ベクトルはすべて、行列 A' の固有ベクトルになる。そのため、[?qebak](#) を呼び出して、それらを A の固有ベクトルに変換しなければならない。

A の Schur ベクトルが必要な場合は、*job*='S' または 'B' と指定してこのルーチンを呼び出してはならない。つまり、これらを指定すると、平衡化のための変換が直交 (複素数型の場合はユニタリー) にならないからである。*job*='P' と指定してこのルーチンを呼び出すと、以降に計算する Schur ベクトルはすべて、行列 A' の Schur ベクトルになる。そのため、[?qebak](#) (*side*='R' と指定) を呼び出して、それらを A の Schur ベクトルに変換し直す必要がある。

浮動小数演算の総数は、 n^2 に比例する。

?gebak

平衡化後の行列の固有ベクトルを、元の非対称行列の固有ベクトルに変換する。

構文

Fortran 77:

```
call sgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call dgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call cgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
call zgebak(job, side, n, ilo, ihi, scale, m, v, ldv, info)
```

Fortran 95:

```
call gebak(v, scale [,ilo] [,ihi] [,job] [,side] [,info])
```

説明

このルーチンは、?gebal を呼び出して行列 A を平衡化した後、平衡化後の行列 A''_{22} の固有ベクトルを計算してから使用する。

平衡化については、?gebal を参照。平衡化後の行列 A'' は、 $A'' = DPAP^T D^{-1}$ として得られる (P は置換行列、 D はスケーリング用の対角行列)。このルーチンでは、固有ベクトルを次のように変換する。

x が A'' の右固有ベクトルである場合、 $P^T D^{-1} x$ が A の右固有ベクトルになる。

x が A'' の左固有ベクトルである場合、 $P^T D y$ が A の左固有ベクトルになる。

入力パラメーター

<i>job</i>	CHARACTER*1。'N'、'P'、'S'、または 'B' のいずれかでなければならない。 ?gebal に指定した <i>job</i> と同じ値を使用する。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> = 'L' の場合、左固有ベクトルが変換される。 <i>side</i> = 'R' の場合、右固有ベクトルが変換される。
<i>n</i>	INTEGER。固有ベクトルで構成される行列の行数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。 <i>ilo</i> と <i>ihi</i> の値は、?gebal から返された値である。 $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ 。
<i>scale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 元の一般行列を平衡化する際に使った置換やスケーリング係数の各成分 (?gebal から返されたもの) を格納する。
<i>m</i>	INTEGER。固有ベクトルで構成される行列の列数 ($m \geq 0$)。

<i>v</i>	REAL (sgebak の場合) DOUBLE PRECISION (dgebak の場合) COMPLEX (cgebak の場合) DOUBLE COMPLEX (zgebak の場合)。 配列: <i>v</i> (<i>ldv</i> ,*) には、変換対象の左または右の固有ベクトルで構成される行列を格納する。 <i>v</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。
<i>ldv</i>	INTEGER。 <i>v</i> の第 1 次元。 $\max(1, n)$ 以上。

出力パラメーター

<i>v</i>	変換後の固有ベクトルによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gebak* のインターフェイスの詳細を以下に示す。

<i>v</i>	サイズ (<i>n</i> , <i>m</i>) の行列 <i>V</i> を格納する。
<i>scale</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。
<i>side</i>	'L' または 'R' でなければならない。デフォルト値は 'L'。

アプリケーション・ノート

このルーチンでの誤差は、無視できる程度である。

浮動小数演算のおおよその総数は、 $m \cdot n$ に比例する。

?hseqr

Hessenberg 形式に縮退された行列の固有値をすべて計算し、(オプションで) Schur 因子分解を行う。

構文

Fortran 77:

```
call shseqr(job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work, lwork,
           info)
call dhseqr(job, compz, n, ilo, ihi, h, ldh, wr, wi, z, ldz, work, lwork,
           info)
call chseqr(job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork,
           info)
call zhseqr(job, compz, n, ilo, ihi, h, ldh, w, z, ldz, work, lwork,
           info)
```

Fortran 95:

```
call hseqr(h, wr, wi [,ilo] [,ihi] [,z] [,job] [,compz] [,info])
call hseqr(h, w [,ilo] [,ihi] [,z] [,job] [,compz] [,info])
```

説明

このルーチンは、上 Hessenberg 行列 $H: H = ZTZ^H$ の固有値をすべて計算し、オプションで、Schur 因子分解を行う。 T は、上三角 (実数型の場合は準三角) 行列 (H の Schur 形式) である。 Z は、ユニタリ行列または直交行列で、列は Schur ベクトル z_i である。

このルーチンを使用すれば、上 Hessenberg 形式 $H: A = QHQ^H$ (Q はユニタリ行列で、実数型の場合は直交行列) に縮退されている一般行列 A の Schur 因子分解を求めることができる。 $A = (QZ)T(QZ)^H$ である。

この場合、[?gghrd](#) によって A を Hessenberg 形式に縮退させた後、[?orghr](#) を呼び出して Q を明示的に求め、`compz='V'` と指定して Q を [?hseqr](#) に渡す。

[?gebak](#) を呼び出して、[?hseqr](#) を使って Hessenberg 形式に縮退させる前に、元の行列を平衡化することもできる。Hessenberg の行列 H は、次の構造を持つ。

$$\begin{bmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{bmatrix}$$

H_{11} と H_{33} は、上三角行列である。

この場合、中央の対角ブロック H_{22} ($i_{lo} \sim i_{hi}$ の行と列) だけをさらに Schur 形式に縮退させる必要がある (H_{12} と H_{23} のブロックも影響を受ける)。したがって、 i_{lo} と i_{hi} の値を、`?hseqr` に直接与えられる。また、このルーチン呼び出し後は、`?gebal` を呼び出して、平衡化後の行列の Schur ベクトルを置換して、元の行列の Schur ベクトルにしなければならない。

ただし、`?gebal` を呼び出さなかった場合は、 i_{lo} に 1 を、 i_{hi} に n を設定しなければならない。A の Schur 因子分解が必要な場合は、`job='S'` または `'B'` を指定して `?gebal` を呼び出してはならない。つまり、これらを指定すると、平衡化のための変換がユニタリ (実数型の場合は直交) にならないからである。

`?hseqr` では、上 Hessenberg の QR アルゴリズムのマルチシフト形式を使う。Schur ベクトルは、 $\|z\|_2 = 1$ になるように正規化される。ただし、絶対値が 1 となる複素数の係数 (13.462 mm 実数型の場合は係数 ± 1) の範囲に収まる。

入力パラメーター

<code>job</code>	CHARACTER*1。'E' または 'S' でなければならない。 <code>job='E'</code> の場合、固有値だけを求める。 <code>job='S'</code> の場合、Schur 形式 T を求める。
<code>compz</code>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <code>compz='N'</code> の場合、Schur ベクトルは計算されない (配列 z は参照されない)。 <code>compz='I'</code> の場合、 H の Schur ベクトルが計算される (配列 z は、このルーチンによって初期化される)。 <code>compz='V'</code> の場合、A の Schur ベクトルが計算される (呼び出し時に、配列 z には、あらかじめ行列 Q を格納しておかなければならない)。
<code>n</code>	INTEGER。行列 H の次数 ($n \geq 0$)。
<code>i_{lo}, i_{hi}</code>	INTEGER。A の平衡化を <code>?gebal</code> によって実行した場合は、 i_{lo} と i_{hi} に、 <code>?gebal</code> から返された値を設定しなければならない。そうでない場合は、 i_{lo} に 1 を、 i_{hi} に n を設定しなければならない。
<code>h, z, work</code>	REAL (<code>shseqr</code> の場合) DOUBLE PRECISION (<code>dhseqr</code> の場合) COMPLEX (<code>chseqr</code> の場合) DOUBLE COMPLEX (<code>zhseqr</code> の場合)。 配列: <code>h(ldh,*)</code> $n \times n$ の上 Hessenberg 行列 H 。 h の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <code>z(ldz,*)</code> <code>compz='V'</code> の場合、 z には、Hessenberg 形式に縮退させた行列 Q を設定しなければならない。 <code>compz='I'</code> の場合、 z を設定する必要はない。 <code>compz='N'</code> の場合、 z は参照されない。 z の第 2 次元は、 $\max(1, n)$ 以上 (<code>compz='V'</code> または <code>'I'</code> の場合) または 1 以上 (<code>compz='N'</code> の場合) でなければならない。

work(lwork) は、ワークスペース配列である。
work の次元は、 $\max(1, n)$ 以上でなければならない。

<i>ldh</i>	INTEGER。 <i>h</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。 <i>compz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>compz</i> = 'V' または 'I' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 $lwork \geq \max(1, n)$ <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w</i>	COMPLEX (<i>chseqr</i> の場合) DOUBLE COMPLEX (<i>zhseqr</i> の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> > 0 でない場合、計算で求めた固有値が格納される。一連の固有値は、Schur 形式 <i>T</i> (求めるように指定した場合) の対角成分と同じ順序で格納される。
<i>wr, wi</i>	REAL (<i>shseqr</i> の場合) DOUBLE PRECISION (<i>dhseqr</i> の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 <i>info</i> > 0 でない場合、計算で求めた固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。一連の固有値は、Schur 形式 <i>T</i> (求めるように指定した場合) の対角成分と同じ順序で格納される。
<i>z</i>	<i>compz</i> = 'V' または 'I' の場合、 <i>info</i> > 0 でない限り、Schur ベクトルを求めるように指定したときには、Schur ベクトルのユニタリ (直交) 行列が <i>z</i> に格納される。 <i>compz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> に最適な <i>lwork</i> 値が返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> > 0 の場合、このアルゴリズムで 30 (<i>ihi</i> - <i>ilo</i> +1) 回の処理を繰り返した結果、すべての固有値を見つけられなかった。 <i>info</i> = <i>i</i> の場合、見つかった固有値の実数部と虚数部が、 <i>wr</i> と <i>wi</i> の 1, 2, ..., <i>ilo</i> -1 および <i>i</i> +1, <i>i</i> +2, ..., <i>n</i> の成分に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hseqr` のインターフェイスの詳細を以下に示す。

<code>h</code>	サイズ (n, n) の行列 H を格納する。
<code>wr</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>z</code>	サイズ (n, n) の行列 Z を格納する。
<code>job</code>	'E' または 'S' でなければならない。デフォルト値は 'E'。
<code>compz</code>	省略された場合、この引数は、引数 z の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> (z が存在する場合)、 <code>compz = 'N'</code> (z が省略された場合)。 存在する場合、 <code>compz</code> は 'I' または 'V' と等しくなければならない、引数 z も存在しなければならない。 <code>compz</code> が存在し、 z が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

計算で求めた Schur 因子分解は、近似行列 $H + E$ の正確な因子分解である。ここで、 $\|E\|_2 < O(\epsilon) \|H\|_2 / s_i$ 、 ϵ はマシン精度である。

λ_i が正確な固有値で、 μ_i が対応する計算で求めた値の場合、 $|\lambda_i - \mu_i| \leq c(n)\epsilon \|H\|_2 / s_i$ になる。 $c(n)$ は、 n の漸増する関数である。 s_i は、 λ_i の条件数の逆数である。条件数 s_i は、[?trsna](#) を呼び出して、求められる。

浮動小数演算の総数は、このアルゴリズムの収束の速さによって決まり、通常は、次の数値になる。

固有値のみを計算する場合：	$7n^3$ (実数型の場合) $25n^3$ (複素数型の場合)
Schur 形式を計算する場合：	$10n^3$ (実数型の場合) $35n^3$ (複素数型の場合)
Schur の因子分解をすべて計算する場合：	$20n^3$ (実数型の場合) $70n^3$ (複素数型の場合)

?hsein

指定された固有値に対応する上 Hessenberg 行列の固有ベクトルを選択して計算する。

構文

Fortran 77:

```
call shsein(job, eigsrc, initv, select, n, h, ldh, wr, wi, vl, ldvl, vr,
            ldvr, mm, m, work, ifaill, ifailr, info)
call dhsein(job, eigsrc, initv, select, n, h, ldh, wr, wi, vl, ldvl, vr,
            ldvr, mm, m, work, ifaill, ifailr, info)
call chsein(job, eigsrc, initv, select, n, h, ldh, w, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, ifaill, ifailr, info)
call zhsein(job, eigsrc, initv, select, n, h, ldh, w, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, ifaill, ifailr, info)
```

Fortran 95:

```
call hsein(h, wr, wi, select [,vl] [,vr] [,ifaill] [,ifailr] [,initv]
           [,eigsrc] [,m] [,info])
call hsein(h, w, select [,vl] [,vr] [,ifaill] [,ifailr] [,initv]
           [,eigsrc] [,m] [,info])
```

説明

このルーチンは、選択された固有値に対応する、上 Hessenberg 行列 H の左または右の固有ベクトルを計算する。

右の固有ベクトル x と左の固有ベクトル y (固有値 λ に対応) は、 $Hx = \lambda x$ と $y^H H = \lambda y^H$ (または $H^H y = \lambda^* y$) として定義される。
 λ^* は、 λ の共役を表す。

一連の固有ベクトルは、反転を繰り返すことによって計算される。そして、実数型の固有ベクトル x の場合は $\max|x_i| = 1$ で、複素数型の固有ベクトルの場合は $\max(|\operatorname{Re}x_i| + |\operatorname{Im}x_i|) = 1$ になるようにスケーリングされる。

一般行列 A を上 Hessenberg 形式に縮退させて H を求めた場合は、[?ormhr](#) または [?unmhr](#) によって、 H の固有ベクトルを A の固有ベクトルに変換できる。

入力パラメーター

job CHARACTER*1. 'R'、'L'、または 'B' でなければならない。
 $job = 'R'$ の場合、右の固有ベクトルだけが計算される。
 $job = 'L'$ の場合、左の固有ベクトルだけが計算される。
 $job = 'B'$ の場合、すべての固有ベクトルが計算される。

eigsrc CHARACTER*1. 'Q' または 'N' でなければならない。
 $eigsrc = 'Q'$ の場合、[?hseqr](#) を使って、 H の固有値が求められている。そのため、 H の劣対角成分の中にゼロのものがある (つまり、ブロック三角である) 場合、 j 番目の固有値を j 番目の行と列が格納されているブロックの固有値とみなせる。このルー

チンでは、この特性により、1つの対角ブロックに関してだけ反転の繰り返し処理を実行できる。

`eigsrc='N'` の場合、そのような仮定を行わず、このルーチンでは行列全体を使って反転の繰り返し処理を実行する。

<code>initv</code>	<p>CHARACTER*1. 'N' または 'U' でなければならない。 <code>initv='N'</code> の場合、選択した固有ベクトルに対して初期の見積もりを指定しない。 <code>initv='U'</code> の場合、選択した固有ベクトルに対して初期の見積もりを <code>v1</code> や <code>vr</code> で指定する。</p>
<code>select</code>	<p>LOGICAL。 配列、次元は $\max(1, n)$ 以上。 計算の対象となる固有ベクトルを指定する。 <i>実数型の場合:</i> 実数の固有値 $wr(j)$ に対応する実数型の固有ベクトルを求めるには、<code>select(j)</code> に <code>.TRUE.</code> を設定する。 複素数の固有値 $(wr(j), wi(j))$ (複素共役は $(wr(j+1), wi(j+1))$) に対応する複素数型の固有ベクトルを選択するには、<code>select(j)</code> や <code>select(j+1)</code> に <code>.TRUE.</code> を設定する。ペアの最初の固有値に対応する固有ベクトルが計算される。 <i>複素数型の場合:</i> 固有値 $w(j)$ に対応する固有ベクトルを選択するには、<code>select(j)</code> に <code>.TRUE.</code> を設定する。</p>
<code>n</code>	INTEGER。行列 H の次数 ($n \geq 0$)。
<code>h, v1, vr, work</code>	<p>REAL (<code>shsein</code> の場合) DOUBLE PRECISION (<code>dhsein</code> の場合) COMPLEX (<code>chsein</code> の場合) DOUBLE COMPLEX (<code>zhsein</code> の場合)。 配列: <code>h(ldh,*)</code> $n \times n$ の上 Hessenberg 行列 H。 <code>h</code> の第2次元は、$\max(1, n)$ 以上でなければならない。 <code>v1(ldv1,*)</code> <code>initv='V'</code> かつ <code>job='L'</code> または 'B' の場合、左固有ベクトルの反転の繰り返し処理用の開始ベクトルを <code>v1</code> に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1つまたは複数個) に格納しなければならない。 <code>initv='N'</code> の場合、<code>v1</code> を設定する必要はない。 <code>v1</code> の第2次元は、$\max(1, mm)$ 以上 (<code>job='L'</code> または 'B' の場合) または 1 以上 (<code>job='R'</code> の場合) でなければならない。 <code>job='R'</code> の場合、配列 <code>v1</code> は参照されない。 <code>vr(ldvr,*)</code> <code>initv='V'</code> かつ <code>job='R'</code> または 'B' の場合、右固有ベクトルの反転の繰り返し処理用の開始ベクトルを <code>vr</code> に格納しなければならない。それぞれの開始ベクトルは、対応する固有ベクトルを格納する際に使うのと同じ列 (1つまたは複数個) に格納しなければならない。 <code>initv='N'</code> の場合、<code>vr</code> を設定する必要はない。</p>

	<p>vr の第 2 次元は、$\max(1, mm)$ 以上 ($job='R'$ または $'B'$ の場合) または 1 以上 ($job='L'$ の場合) でなければならない。 $job='L'$ の場合、配列 vr は参照されない。</p> <p>$work(*)$ は、ワークスペース配列。 次元は $\max(1, n*(n+2))$ 以上 (実数型の場合) または $\max(1, n*n)$ 以上 (複素数型の場合)。</p>
ldh	INTEGER。 h の第 1 次元。 $\max(1, n)$ 以上。
w	COMPLEX (chsein の場合) DOUBLE COMPLEX (zhsein の場合)。 配列、次元は $\max(1, n)$ 以上。 行列 H の固有値を格納する。 $eigsrc='Q'$ の場合、この配列は ?hseqr から返されたものと同じでなければならない。
wr, wi	REAL (shsein の場合) DOUBLE PRECISION (dhsein の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 行列 H の固有値の実数部と虚数部をそれぞれ格納する。複素共役ペアの値は、この配列の連続する成分に格納しなければならない。 $eigsrc='Q'$ の場合、この配列は、?hseqr から返されたものと同じでなければならない。
$ldvl$	INTEGER。 vl の第 1 次元。 $job='L'$ または $'B'$ の場合、 $ldvl \geq \max(1, n)$ 。 $job='R'$ の場合、 $ldvl \geq 1$ 。
$ldvr$	INTEGER。 vr の第 1 次元。 $job='R'$ または $'B'$ の場合、 $ldvr \geq \max(1, n)$ 。 $job='L'$ の場合、 $ldvr \geq 1$ 。
mm	INTEGER。 vl や vr の列数。 m (実際に必要な列数。下記の出力パラメーターを参照) 以上でなければならない。 実数型の場合、選択された実数の固有値ごとに 1 を、選択された複素数の固有値ごとに 2 をカウントすれば、 m が得られる (select を参照)。 複素数型の場合、選択された固有ベクトルの個数が m になる (select を参照)。次の制約がある。 $0 \leq mm \leq n$ 。
$rwork$	REAL (chsein の場合) DOUBLE PRECISION (zhsein の場合)。 配列、次元は $\max(1, n)$ 以上。

出力パラメーター

$select$	実数型の場合だけ、上書きされる。上記のように複素数の固有値が選択された場合は、 $select(j)$ に $.TRUE.$ が、 $select(j+1)$ に $.FALSE.$ が設定される。
w	w の一部の成分の実数部が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに振動するからである。

<i>wr</i>	<i>w</i> の一部の成分が変動する場合がある。これは、独立した固有ベクトルを検索する際に、近似した固有値がわずかに摂動するからである。
<i>v1, vr</i>	<p><i>job</i>='L' または 'B' の場合、<i>v1</i> には、(<i>select</i> の値に従って) 計算で求めた左の固有ベクトルが格納される。</p> <p><i>job</i>='R' または 'B' の場合、<i>vr</i> には、(<i>select</i> の値に従って) 計算で求めた右の固有ベクトルが格納される。</p> <p>一連の固有ベクトルは、対応する固有値と同じ順序で、この配列の列に連続して格納される。</p> <p>実数型の場合: 選択された実数の固有値に対応する実数型の固有ベクトルによって列を 1 つ占める。選択された固有値に対応する複素数型の固有ベクトルによって列を 2 つ占める。最初の列には実数部が、2 番目の列には虚数部が格納される。</p>
<i>m</i>	<p>INTEGER。</p> <p>実数型の場合: 選択された固有ベクトルの格納に必要な <i>v1</i> や <i>vr</i> の列数。</p> <p>複素数型の場合: 選択した固有ベクトルの個数。</p>
<i>ifaill, ifailr</i>	<p>INTEGER。</p> <p>配列、次元はそれぞれ $\max(1, mm)$ 以上。</p> <p><i>ifaill</i>(<i>i</i>) = 0 (<i>v1</i> の <i>i</i> 番目の列が収束した場合)。</p> <p><i>ifaill</i>(<i>i</i>) = <i>j</i> > 0 (<i>v1</i> の <i>i</i> 番目の列に格納されている固有ベクトル (<i>j</i> 番目の固有値に対応) が収束しなかった場合)。</p> <p><i>ifailr</i>(<i>i</i>) = 0 (<i>vr</i> の <i>i</i> 番目の列が収束した場合)。</p> <p><i>ifailr</i>(<i>i</i>) = <i>j</i> > 0 (<i>vr</i> の <i>i</i> 番目の列に格納されている固有ベクトル (<i>j</i> 番目の固有値に対応) が収束しなかった場合)。</p> <p>実数型の場合: <i>v1</i> の <i>i</i> 番目と (<i>i</i>+1) 番目の列に選択した複素数型の固有ベクトルが格納されている場合は、<i>ifaill</i>(<i>i</i>) と <i>ifaill</i>(<i>i</i>+1) に同じ値が設定される。<i>vr</i> と <i>ifailr</i> にも、これと同様の規則が適用される。</p> <p><i>job</i>='R' の場合、配列 <i>ifaill</i> は参照されない。 <i>job</i>='L' の場合、配列 <i>ifailr</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。</p> <p><i>info</i> > 0 の場合、<i>i</i> 個の固有ベクトル (上記のパラメーター <i>ifaill</i> や <i>ifailr</i> で指定) が、収束していない。<i>v1</i> や <i>vr</i> の対応する列に、有用な情報が格納されていない。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hsein` のインターフェイスの詳細を以下に示す。

<code>h</code>	サイズ (n, n) の行列 H を格納する。
<code>wr</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>select</code>	長さ (n) のベクトルを格納する。
<code>v1</code>	サイズ (n, mm) の行列 VL を格納する。
<code>vr</code>	サイズ (n, mm) の行列 VR を格納する。
<code>ifaill</code>	長さ (mm) のベクトルを格納する。 <code>ifaill</code> が存在し、 <code>v1</code> が省略された場合、エラー条件がセットされる。
<code>ifailr</code>	長さ (mm) のベクトルを格納する。 <code>ifailr</code> が存在し、 <code>vr</code> が省略された場合、エラー条件がセットされる。
<code>initv</code>	'N' または 'U' でなければならない。デフォルト値は 'N'。
<code>eigsrc</code>	'N' または 'Q' でなければならない。デフォルト値は 'N'。
<code>job</code>	引数 <code>v1</code> および <code>vr</code> の存在に基づいて以下のように復元される。 <code>job = 'B'</code> (<code>v1</code> と <code>vr</code> の両方が存在する場合)、 <code>job = 'L'</code> (<code>v1</code> が存在し、 <code>vr</code> が省略された場合)、 <code>job = 'R'</code> (<code>v1</code> が省略され、 <code>vr</code> が存在する場合)。 <code>v1</code> と <code>vr</code> の両方が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

計算で求めた右固有ベクトル x_i はそれぞれ、近似行列 $A + E_i$ の正確な固有ベクトル ($\|E_i\| < O(\epsilon)\|A\|$) である。そのため、誤差は小さいものになる ($\|Ax_i - \lambda_i x_i\| = O(\epsilon)\|A\|$)。

ただし、近似したあるいは一致した固有値に対応する固有ベクトルが、関連する部分空間に正確に張られていない場合がある。

計算で求めた左固有ベクトルについても、これと同様の注意が適用される。

?trevc

?hseqr で求めた上(準)三角行列の固有ベクトルを選択して計算する。

構文

Fortran 77:

```
call strevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, info)
call dtrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, info)
call ctrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, info)
call ztrevc(side, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            mm, m, work, rwork, info)
```

Fortran 95:

```
call trevc(t [,howmny] [,select] [,vl] [,vr] [,m] [,info])
```

説明

このルーチンは、上三角行列 T (実数型の場合、上準三角行列 T) の左あるいは右の一部またはすべての固有ベクトルを計算する。このタイプの行列は、一般行列の Schur 因子分解 ($A = QTQ^H$) で生成される ([?hseqr](#) で計算される)。

T の固有値 w に対応する右固有ベクトル x と左固有ベクトル y は、次のように表される。

$$Tx = wx, \quad y^H T = wy^H$$
 y^H は y の共役転置を表す。

固有値は、このルーチンに入力されず、 T の対角ブロックから直接読み込まれる。

このルーチンは、 T の右と左固有ベクトルが入った行列 X と Y を返すか、積 QX と積 QY を返す (Q は入力行列)。

Q が、行列 A を Schur 形式 T に縮退する直交/ユニタリ行列の場合、 QX と QY は、 A の右と左の固有ベクトルからなる行列である。

入力パラメーター

<i>side</i>	CHARACTER*1. 'R'、'L'、または 'B' でなければならない。 <i>side</i> ='R' の場合、右の固有ベクトルだけが計算される。 <i>side</i> ='L' の場合、左の固有ベクトルだけが計算される。 <i>side</i> ='B' の場合、すべての固有ベクトルが計算される。
<i>howmny</i>	CHARACTER*1. 'A'、'B'、または 'S' のいずれかでなければならない。 <i>howmny</i> ='A' の場合、(<i>side</i> の値に従って)すべての固有ベクトルが計算される。 <i>howmny</i> ='B' の場合、(<i>side</i> の値に従って)すべての固有ベクトル

が計算され、 $v1$ と vr で与えられる行列によって逆変換される。
 $howmny='S'$ の場合、($side$ と $select$ の値に従って) 選択した固有ベクトルが計算される。

select LOGICAL。
 配列、次元は $\max(1, n)$ 以上。
 $howmny='S'$ の場合、*select* は計算する固有ベクトルを指定する。
 $howmny='A'$ または $'B'$ の場合、*select* は参照されない。
 実数型の場合:
 ω_j が実固有値の場合、 $select(j)$ が $.TRUE.$ であれば、それに対応する実固有ベクトルが計算される。
 ω_j と ω_{j+1} が複素固有値の実部と虚部の場合、 $select(j)$ または $select(j+1)$ が $.TRUE.$ であれば、それに対応する複素固有ベクトルが計算され、終了時に、 $select(j)$ は $.TRUE.$ 、 $select(j+1)$ は $.FALSE.$ に設定される。
 複素数型の場合:
 $select(j)$ が $.TRUE.$ であれば、 j 番目の固有値に対応する固有ベクトルが計算される。

n INTEGER。行列 T の次数 ($n \geq 0$)。

t, v1, vr, work REAL (*strevc* の場合)
 DOUBLE PRECISION (*dtrevc* の場合)
 COMPLEX (*ctrevc* の場合)
 DOUBLE COMPLEX (*ztrevc* の場合)。
 配列:
 $t(ldt, *)$ には、 $n \times n$ の行列 T を標準の Schur 形式で格納する。
 t の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$v1(ldv1, *)$
 $howmny='B'$ かつ $side='L'$ または $'B'$ の場合、 $v1$ には、 $n \times n$ の行列 Q (通常は、*?hseqr* から返された Schur ベクトルから成る行列) を格納しなければならない。
 $howmny='A'$ または $'S'$ の場合、 $v1$ を設定する必要はない。
 $v1$ の第 2 次元は、 $\max(1, mm)$ 以上 ($side='L'$ または $'B'$ の場合) または 1 以上 ($side='R'$ の場合) でなければならない。
 $side='R'$ の場合、配列 $v1$ は参照されない。

$vr(ldvr, *)$
 $howmny='B'$ かつ $side='R'$ または $'B'$ の場合、 vr には、 $n \times n$ の行列 Q (通常は、*?hseqr* から返された Schur ベクトルから成る行列) を格納しなければならない。
 $howmny='A'$ または $'S'$ の場合、 vr を設定する必要はない。
 vr の第 2 次元は、 $\max(1, mm)$ 以上 ($side='R'$ または $'B'$ の場合) または 1 以上 ($side='L'$ の場合) でなければならない。
 $side='L'$ の場合、配列 vr は参照されない。

$work(*)$ は、ワークスペース配列。
 次元は、 $\max(1, 3*n)$ 以上 (実数型の場合) または $\max(1, 2*n)$ 以上 (複素数型の場合)。

ldt INTEGER。 t の第 1 次元。 $\max(1, n)$ 以上。

<i>ldvl</i>	INTEGER。 <i>v1</i> の第 1 次元。 <i>side</i> ='L' または 'B' の場合、 $ldvl \geq \max(1, n)$ 。 <i>side</i> ='R' の場合、 $ldvl \geq 1$ 。
<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>side</i> ='R' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>side</i> ='L' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 配列 <i>v1</i> や <i>vr</i> の列数。 <i>m</i> (必要な列の正確な個数) 以上でなければならない。 <i>howmny</i> ='A' または 'B' の場合、 $m = n_0$ 。 <i>howmny</i> ='S' の場合： 実数型の場合、選択された実数の固有ベクトルごとに 1 を、選択された複素数の固有ベクトルごとに 2 をカウントすれば、 <i>m</i> が得られる。 複素数型の場合、選択された固有ベクトルの個数が <i>m</i> になる (<i>select</i> を参照)。 次の制約がある。 $0 \leq m \leq n_0$ 。
<i>rwork</i>	REAL (<i>ctrevc</i> の場合) DOUBLE PRECISION (<i>ztrvec</i> の場合)。 ワークスペース配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>select</i>	上記のように実行列の複素数型の固有ベクトルを選択した場合は、 <i>select(j)</i> に .TRUE. が、 <i>select(j+1)</i> に .FALSE. が設定される。
<i>v1, vr</i>	<i>side</i> ='L' または 'B' の場合、 <i>v1</i> に、(<i>howmny</i> と <i>select</i> の値に従って) 計算で求めた左固有ベクトルが格納される。 <i>side</i> ='R' または 'B' の場合、 <i>vr</i> に、(<i>howmny</i> と <i>select</i> の値に従って) 計算で求めた右固有ベクトルが格納される。 一連の固有ベクトルは、対応する固有値と同じ順序で、この配列の列に連続して格納される。 実数型の場合： 実数の固有値のそれぞれに実数の固有ベクトルが 1 つ対応し、列を 1 つ占める。固有値の複素共役ペアのそれぞれに複素数の固有ベクトルが 1 つ対応し、列を 2 つ占める。最初の列には実数部、2 番目の列には虚数部が格納される。
<i>m</i>	INTEGER。 複素数型の場合： 選択した固有ベクトルの個数。 <i>howmny</i> ='A' または 'B' の場合、 <i>m</i> には <i>n</i> が設定される。 実数型の場合： 選択した固有ベクトルの格納に実際に使用される <i>v1</i> や <i>vr</i> の列数。 <i>howmny</i> ='A' または 'B' の場合、 <i>m</i> には <i>n</i> が設定される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trevc` のインターフェイスの詳細を以下に示す。

<code>t</code>	サイズ (n, n) の行列 T を格納する。
<code>select</code>	長さ (n) のベクトルを格納する。
<code>v1</code>	サイズ (n, mm) の行列 VL を格納する。
<code>vr</code>	サイズ (n, mm) の行列 VR を格納する。
<code>side</code>	省略された場合、この引数は、引数 <code>v1</code> と <code>vr</code> の存在に基づいて以下のように復元される。 <code>side = 'B'</code> (<code>v1</code> と <code>vr</code> の両方が存在する場合)、 <code>side = 'L'</code> (<code>vr</code> が省略された場合)、 <code>side = 'R'</code> (<code>v1</code> が省略された場合)。 <code>v1</code> と <code>vr</code> の両方が省略された場合、エラー条件がセットされる。
<code>howmny</code>	省略された場合、この引数は、引数 <code>select</code> の存在に基づいて以下のように復元される。 <code>howmny = 'V'</code> (<code>q</code> が存在する場合)、 <code>howmny = 'N'</code> (<code>q</code> が省略された場合)。 存在する場合、 <code>vect = 'V'</code> または <code>'U'</code> で、引数 <code>select</code> も存在しなければならない。 <code>select</code> と <code>howmny</code> の両方が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

x_i が正確な固有ベクトルで、 y_i が対応する計算で求めた固有ベクトルの場合、それらの間の角 $\theta(y_i, x_i)$ は次のようになる。 $\theta(y_i, x_i) \leq (c(n)\epsilon \|T\|_2) / \text{sep}_i$ (sep_i は x_i の条件数の逆数)。条件数 sep_i は、`?trсна` を呼び出すことによって求められる。

?trsna

上(準)三角行列の指定された固有値および右固有ベクトルに関して条件数を見積もる。

構文

Fortran 77:

```
call strсна(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            s, sep, mm, m, work, ldwork, iwork, info)
call dtrsна(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            s, sep, mm, m, work, ldwork, iwork, info)
call ctrсна(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            s, sep, mm, m, work, ldwork, rwork, info)
call ztrsна(job, howmny, select, n, t, ldt, vl, ldvl, vr, ldvr,
            s, sep, mm, m, work, ldwork, rwork, info)
```

Fortran 95:

```
call trсна(t [,s] [,sep] [,vl] [,vr] [,select] [,m] [,info])
```

説明

このルーチンは、上三角行列 T (実数型の場合、標準形の Schur 形式の上準三角行列 T) の指定された固有値や右固有ベクトルに関して、条件数を見積もる。これらは、元の行列 $A = ZTZ^H$ (Z はユニタリー行列。ただし、実数型の場合は直交行列) の固有値と右固有ベクトルの条件数と同じである。ここから T を導き出すことができる。

このルーチンでは、固有値 λ_i の条件数の逆数を $s_i = |v^H u| / (\|u\|_E \|v\|_E)$ として計算する。 u と v はそれぞれ、 λ_i に対応する T の右と左の固有ベクトルである。この条件数の逆数は、常に、ゼロ (悪い条件の場合) と 1 (良い条件の場合) の間になる。

計算で求めた固有値 λ_i のおおよその誤差は、 $\varepsilon \|T\| / s_i$ (ε はマシン精度) と見積もれる。

λ_i に対応する右固有ベクトルの条件数の逆数を見積もるために、このルーチンではまず [?trexc](#) を呼び出して一連の固有値の順序を変更し、 λ_i が先頭の位置にする。

$$T = Q \begin{bmatrix} \lambda_i & C^H \\ 0 & T_{22} \end{bmatrix} Q^H$$

次に、この固有ベクトルの条件数の逆数を、 sep_i (行列 T_{22} の最小の特異値 $-\lambda_i I$) として見積もる。この値は、ゼロ (悪い条件の場合) から非常に大きな値 (良い条件の場合) の間になる。

λ_i に対応して計算で求めた右固有ベクトル u のおおよその誤差は、 $\varepsilon \|T\| / sep_i$ と見積もれる。

入力パラメーター

<i>job</i>	CHARACTER*1。'E'、'V'、または 'B' のいずれかでなければならない。 <i>job</i> ='E' の場合、固有値だけの条件数が計算される。 <i>job</i> ='V' の場合、固有ベクトルだけの条件数が計算される。 <i>job</i> ='B' の場合、固有値と固有ベクトルの両方の条件数が計算される。
<i>howmny</i>	CHARACTER*1。'A' または 'S' でなければならない。 <i>howmny</i> ='A' の場合、すべての固有ペアの条件数が計算される。 <i>howmny</i> ='S' の場合、選択した固有ペア (<i>select</i> で指定) の条件数が計算される。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上 (<i>howmny</i> ='S' の場合) または 1 以上 (それ以外の場合)。 <i>howmny</i> ='S' の場合、条件数を計算する固有ペアを指定する。 <i>実数型の場合:</i> 実数の固有値 λ_j に対応する固有ペアの条件数を選択するには、 <i>select</i> (<i>j</i>) に .TRUE. を設定しなければならない。 固有値 λ_j と λ_{j+1} の複素共役ペアに対応する固有ペアの条件数を選択するには、 <i>select</i> (<i>j</i>) や <i>select</i> (<i>j</i> +1) に .TRUE. を設定しなければならない。 <i>複素数の場合:</i> 固有値 λ_j に対応する固有ペアの条件数を選択するには、 <i>select</i> (<i>j</i>) に .TRUE. を設定しなければならない。 <i>howmny</i> ='A' の場合、 <i>select</i> は参照されない。
<i>n</i>	INTEGER。行列 <i>T</i> の次数 ($n \geq 0$)。
<i>t, v1, vr, work</i>	REAL (<i>strsna</i> の場合) DOUBLE PRECISION (<i>dtrsna</i> の場合) COMPLEX (<i>ctrsna</i> の場合) DOUBLE COMPLEX (<i>ztrsna</i> の場合)。 配列: <i>t</i> (<i>ldt</i> ,*) には、 $n \times n$ の行列 <i>T</i> を格納する。 <i>t</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>v1</i> (<i>ldv1</i> ,*) <i>job</i> ='E' または 'B' の場合、 <i>v1</i> には、 <i>howmny</i> と <i>select</i> で指定した固有ペアに対応する <i>T</i> (または <i>Q</i> がユニタリ行列か直交行列のような任意の行列 QTQ^H) の左固有ベクトルを格納しなければならない。この固有ベクトルは、 ?trevc や ?hsein で返される通りに、 <i>v1</i> の連続した列に格納されていなければならない。 <i>v1</i> の第 2 次元は、 $\max(1, mm)$ 以上 (<i>job</i> ='E' または 'B' の場合) または 1 以上 (<i>job</i> ='V' の場合) でなければならない。 <i>job</i> ='V' の場合、配列 <i>v1</i> は参照されない。 <i>vr</i> (<i>ldvr</i> ,*) <i>job</i> ='E' または 'B' の場合、 <i>vr</i> には、 <i>howmny</i> と <i>select</i> で指定した固有ペアに対応する <i>T</i> (または <i>Q</i> がユニタリ行列か直交行列のような任意の行列 QTQ^H) の右固有ベクトルを格納しなけれ

ばならない。この固有ベクトルは、[?trevc](#) や [?hsein](#) で返される通りに、*vr* の連続した列に格納されていなければならない。*vr* の第 2 次元は、 $\max(1, mm)$ 以上 (*job*='E' または 'B' の場合) または 1 以上 (*job*='V' の場合) でなければならない。*job*='V' の場合、配列 *vr* は参照されない。

work(ldwork,)* は、ワークスペース配列である。*work* の第 2 次元は、 $\max(1, n+1)$ 以上 (複素数型の場合) または $\max(1, n+6)$ 以上 (実数型の場合) (*job*='V' または 'B' の場合) または 1 以上 (*job*='E' の場合) でなければならない。*job*='E' の場合、配列 *work* は参照されない。

<i>ldt</i>	INTEGER。 <i>t</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldv1</i>	INTEGER。 <i>v1</i> の第 1 次元。 <i>job</i> ='E' または 'B' の場合、 $ldv1 \geq \max(1, n)$ 。 <i>job</i> ='V' の場合、 $ldv1 \geq 1$ 。
<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>job</i> ='E' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>job</i> ='R' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 配列 <i>s</i> と <i>sep</i> の成分の個数、または <i>v1</i> と <i>vr</i> の列数 (使う場合)。 <i>m</i> (必要とする正確な個数) 以上でなければならない。 <i>howmny</i> ='A' の場合、 $m = n$ 。 <i>howmny</i> ='S' の場合、実数型の場合、選択された実数の固有値ごとに 1 を、固有値の選択された複素共役ペアごとに 2 をカウントすれば、 <i>m</i> が得られる。 複素数型の場合、選択された固有ペアの個数が <i>m</i> になる (<i>select</i> を参照)。次の制約がある。 $0 \leq m \leq n$ 。
<i>ldwork</i>	INTEGER。 <i>work</i> の第 1 次元。 <i>job</i> ='V' または 'B' の場合、 $ldwork \geq \max(1, n)$ 。 <i>job</i> ='E' の場合、 $ldwork \geq 1$ 。
<i>rwork</i>	REAL (<i>ctrnsa</i> 、 <i>ztrsna</i> の場合)。 配列、次元は $\max(1, n)$ 以上。
<i>iwork</i>	INTEGER (<i>strnsa</i> 、 <i>dtrsna</i> の場合)。 配列、次元は $\max(1, n)$ 以上。

出力パラメーター

<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, mm)$ 以上 (<i>job</i> ='E' または 'B' の場合) または 1 以上 (<i>job</i> ='V' の場合)。 <i>job</i> ='E' または 'B' の場合、選択した固有値の条件数の逆数が、この配列の連続する成分に格納される。そのため、 <i>s(j)</i> 、 <i>sep(j)</i> 、および <i>v1</i> と <i>vr</i> の <i>j</i> 番目の列はすべて、同じ固有ペアに対応している (ただし、選択しなかった固有ペアが存在する場合は、一般に、 <i>j</i> 番目の固有ペアにはならない)。
----------	---

	<p><i>実数型の場合</i>： 固有値の複素共役ペアでは、S の 2 つの連続する成分に同じ値が設定される。 $job='V'$ の場合、配列 s は参照されない。</p>
<i>sep</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, mm)$ 以上 ($job='V'$ または 'B' の場合) または 1 以上 ($job='E'$ の場合)。 $job='V'$ または 'B' の場合、選択した右固有ベクトルについての見積りの条件数の逆数が、この配列の連続する成分に格納される。 <i>実数型の場合</i>： 複素数の固有ベクトルのときは、sep の 2 つの連続する成分に同じ値が設定される。$sep(j)$ を計算するために一連の固有値の順序を変更できない場合は、$sep(j)$ にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。 $job='E'$ の場合、配列 sep は参照されない。</p>
<i>m</i>	<p>INTEGER。 <i>複素数型の場合</i>： 選択した固有ペアの個数。$howmny='A'$ の場合、m は n に設定される。 <i>実数型の場合</i>： 条件数の見積り値の格納に実際に使用される s や sep の成分の個数。$howmny='A'$ の場合、m は n に設定される。</p>
<i>info</i>	<p>INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、i 番目のパラメーターの値が不正である。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trsna` のインターフェイスの詳細を以下に示す。

<i>t</i>	サイズ (n, n) の行列 T を格納する。
<i>s</i>	長さ (mm) のベクトルを格納する。
<i>sep</i>	長さ (mm) のベクトルを格納する。
<i>vl</i>	サイズ (n, mm) の行列 VL を格納する。
<i>vr</i>	サイズ (n, mm) の行列 VR を格納する。
<i>select</i>	長さ (n) のベクトルを格納する。
<i>job</i>	<p>引数 s と sep の存在に基づいて以下のように復元される。 $job='B'$ (s と sep の両方が存在する場合)、 $job='E'$ (s が存在し、sep が省略された場合)、 $job='V'$ (s が省略され、sep が存在する場合)。 s と sep の両方が省略された場合、エラー条件がセットされる。</p>

`howmny` 引数 `select` の存在に基づいて以下のように復元される。
`howmny = 'S'` (`select` が存在する場合)、
`howmny = 'A'` (`select` が省略された場合)。

引数 `s`、`v1`、および `vr` は、すべて存在するか、すべて省略されなければならない。そうでない場合、エラー条件がセットされる。

アプリケーション・ノート

計算で求めた値 sep_i は、真の値を過大見積もりする場合もあるが、3 を超える係数になるのはほとんどない。

?trexc

一般行列の Schur 因子分解の順序を変更する。

構文

Fortran 77:

```
call strexc(compq, n, t, ldt, q, ldq, ifst, ilst, work, info)
call dtrexc(compq, n, t, ldt, q, ldq, ifst, ilst, work, info)
call ctrexc(compq, n, t, ldt, q, ldq, ifst, ilst, info)
call ztrexc(compq, n, t, ldt, q, ldq, ifst, ilst, info)
```

Fortran 95:

```
call trexc(t, ifst, ilst [,q] [,info])
```

説明

このルーチンは、一般行列 $A = QTQ^H$ の Schur 因子分解の順序を変更し、 T の行インデックス `ifst` の対角成分または対角ブロックを行 `ilst` に移動する。

順序を変更した後の Schur 形式 S は、ユニタリー (実数型の場合は直交) 相似変換 $S = Z^H T Z$ によって計算される。また、Schur ベクトルの更新後の行列 P を、 $P = QZ$ ($A = PSP^H$) として計算できる。

入力パラメーター

`compq` CHARACTER*1。'V' または 'N' でなければならない。
`compq = 'V'` の場合、Schur ベクトル (Q) が更新される。
`compq = 'N'` の場合、Schur ベクトルは更新されない。

`n` INTEGER。行列 T の次数 ($n \geq 0$)。

`t, q` REAL (strexc の場合)
DOUBLE PRECISION (dtrexc の場合)
COMPLEX (ctrexc の場合)
DOUBLE COMPLEX (ztrexc の場合)。
配列:
`t(ldt,*)` には、 $n \times n$ の行列 T を格納する。
`t` の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

	$q(ldq, *)$ $compq='V'$ の場合、 q には Q (Schur ベクトル) を格納しなければならない。 $compq='N'$ の場合、 q は参照されない。
	q の第 2 次元は、 $\max(1, n)$ 以上 ($compq='V'$ の場合) または 1 以上 ($compq='N'$ の場合) でなければならない。
ldt	INTEGER。 t の第 1 次元。 $\max(1, n)$ 以上。
ldq	INTEGER。 q の第 1 次元。 $compq='N'$ の場合、 $ldq \geq 1$ 。 $compq='V'$ の場合、 $ldq \geq \max(1, n)$ 。
$ifst, ilst$	INTEGER。 $1 \leq ifst \leq n; 1 \leq ilst \leq n$ 。 行列 T の対角成分 (実数型の場合は対角ブロック) の順序の変更を指定する。行インデックスが $ifst$ の成分 (またはブロック) は、隣接する成分 (またはブロック) 間での一連の交換処理により、行 $ilst$ に移される。
$work$	REAL (strex) の場合) DOUBLE PRECISION (dtrex) の場合)。 配列、次元は $\max(1, n)$ 以上。

出力パラメーター

t	更新後の行列 S によって上書きされる。
q	$compq='V'$ の場合、 q に、Schur ベクトルで構成される更新後の行列が格納される。
$ifst, ilst$	実数型の場合にのみ上書きされる。 呼び出し時に、 $ifst$ が 2×2 のブロックの 2 行目を指している場合は、先頭行を指すように変更される。 $ilst$ は常に、最終位置 (入力値から ± 1 だけ変動している場合もある) のブロックの先頭行を指す。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trexc` のインターフェイスの詳細を以下に示す。

t	サイズ (n, n) の行列 T を格納する。
q	サイズ (n, n) の行列 Q を格納する。
$compq$	引数 q の存在に基づいて以下のように復元される。 $compq='V'$ (q が存在する場合)、 $compq='N'$ (q が省略された場合)。

アプリケーション・ノート

計算で求めた行列 S は、行列 $T + E$ の近似値である。ここで、 $\|E\|_2 = O(\varepsilon) \|T\|_2$ 、 ε はマシン精度である。

2×2 の対角ブロックが順序変更に関与する場合は、一般に、その非対角成分が変更される。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合もある。 2×2 のブロックは、2つの 1×1 のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。

ただし、固有値の値が、順序の変更によって変更されたりはしない。

浮動小数演算のおおよその総数は、次のようになる。

実数型の場合： $6n(\text{ifst}-\text{ilst})$ ($\text{compq} = 'N'$ の場合)
 $12n(\text{ifst}-\text{ilst})$ ($\text{compq} = 'V'$ の場合)

複素数型の場合： $20n(\text{ifst}-\text{ilst})$ ($\text{compq} = 'N'$ の場合)
 $40n(\text{ifst}-\text{ilst})$ ($\text{compq} = 'V'$ の場合)

?trsen

行列の Schur 因子分解の順序を変更し、(オプションで) 固有値の選択した束についての条件数の逆数と不変部分空間を計算する。

構文

Fortran 77:

```
call strsen(job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s,
           sep, work, lwork, iwork, liwork, info)
call dtrsen(job, compq, select, n, t, ldt, q, ldq, wr, wi, m, s,
           sep, work, lwork, iwork, liwork, info)
call ctrsen(job, compq, select, n, t, ldt, q, ldq, w, m, s,
           sep, work, lwork, info)
call ztrsen(job, compq, select, n, t, ldt, q, ldq, w, m, s,
           sep, work, lwork, info)
```

Fortran 95:

```
call trsen(t, select [,wr] [,wi] [,m] [,s] [,sep] [,q] [,info])
call trsen(t, select [,w] [,m] [,s] [,sep] [,q] [,info])
```

説明

このルーチンは、一般行列 $A = QTQ^H$ の Schur 因子分解の順序を変更して、固有値の選択した束が Schur 形式の主対角成分 (実数型の場合は対角ブロック) にくるようにする。順序を変更した後の Schur 形式 R は、ユニタリー (直交) 相似変換 $R = Z^H T Z$ によって計算される。また、Schur ベクトルの更新後の行列 P を、 $P = QZ$ ($A = PRP^H$) として計算できる。

次のように仮定し、

$$R = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{13} \end{bmatrix}$$

選択した固有値が、厳密な意味で $m \times m$ の先頭の部分行列 T_{11} の固有値であるとする。 P を $(Q_1 \ Q_2)$ のように対応させて分割し、 Q_1 が Q の先頭から m 列を構成させる。これで、 $AQ_1 = Q_1T_{11}$ になり、 Q_1 の m 列は固有値の選択した束に対応する不変部分空間の直交基になる。

また、このルーチンでは、固有値の束と不変部分空間の平均の条件数の逆数を見積もれる。

入力パラメーター

<i>job</i>	CHARACTER*1。'N'、'E'、'V'、または 'B' のいずれかでなければならない。 <i>job</i> ='N' の場合、条件数は計算されない。 <i>job</i> ='E' の場合、固有値の束の条件数だけが計算される。 <i>job</i> ='V' の場合、不変部分空間の条件数だけが計算される。 <i>job</i> ='B' の場合、束と不変部分空間の条件数が計算される。
<i>compq</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>compq</i> ='V' の場合、Schur ベクトルの Q が更新される。 <i>compq</i> ='N' の場合、Schur ベクトルは更新されない。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 選択するクラスターに属する固有値を指定する。 固有値 λ_j を選択するには、 <i>select</i> (<i>j</i>) が .TRUE. でなければならない。 実数型の場合： 固有値 λ_j と λ_{j+1} (対応する 2×2 の対角ブロック) の複素共役ペアを選択するには、 <i>select</i> (<i>j</i>) や <i>select</i> (<i>j</i> +1) が .TRUE. でなければならない。複素共役 λ_j と λ_{j+1} は、束に両方とも含まれているか、あるいは両方とも束から除外されていなければならない。
<i>n</i>	INTEGER。行列 T の次数 ($n \geq 0$)。
<i>t, q, work</i>	REAL (strsen の場合) DOUBLE PRECISION (dtrsen の場合) COMPLEX (ctrsen の場合) DOUBLE COMPLEX (ztrsen の場合)。 配列： <i>t</i> (<i>ldt</i> ,*)。 $n \times n$ の T 。 <i>t</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>q</i> (<i>ldq</i> ,*) <i>compq</i> ='V' の場合、 <i>q</i> に Schur ベクトルの Q を格納しなければならない。 <i>compq</i> ='N' の場合、 <i>q</i> は参照されない。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上 (<i>compq</i> ='V' の場合)、あるいは 1 以上 (<i>compq</i> ='N' の場合) でなければならない。

$work(lwork)$ は、ワークスペース配列である。

複素数の場合:

$job='N'$ の場合、配列 $work$ は参照されない。

必要なワークスペースの実際の量は、 $n^2/4$ ($job='E'$ の場合)、あるいは $n^2/2$ ($job='V'$ または $'B'$ の場合) を超えることはできない。

ldt	INTEGER。 t の第 1 次元。 $\max(1, n)$ 以上。
ldq	INTEGER。 q の第 1 次元。 $compq='N'$ の場合、 $ldq \geq 1$ 。 $compq='V'$ の場合、 $ldq \geq \max(1, n)$ 。
$lwork$	INTEGER。 配列 $work$ の次元。 $job='V'$ または $'B'$ の場合、 $lwork \geq \max(1, 2m(n-m))$ 。 $job='E'$ の場合、 $lwork \geq \max(1, m(n-m))$ $job='N'$ の場合、 $lwork \geq 1$ (複素数型の場合) または $lwork \geq \max(1, n)$ (実数型の場合)。 $lwork=-1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。
$iwork$	INTEGER。 $iwork$ ($liwork$) は、ワークスペース配列である。 $job='N'$ または $'E'$ の場合、配列 $iwork$ は参照されない。 $job='V'$ または $'B'$ の場合、必要なワークスペースの実際の量は $n^2/2$ を超えることはできない。
$liwork$	INTEGER。 配列 $iwork$ の次元。 $job='V'$ または $'B'$ の場合、 $lwork \geq \max(1, 2m(n-m))$ 。 $job='E'$ または $'E'$ の場合、 $lwork \geq 1$ 。 $liwork=-1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $iwork$ 配列の最適サイズだけを計算し、その値を $iwork$ 配列の最初のエンタリーとして返す。 $xerbla$ は $liwork$ に関するエラーメッセージを生成しない。

出力パラメーター

t	更新後の行列 R によって上書きされる。
q	$compq='V'$ の場合、 q には、Schur ベクトルで構成される更新後の行列が格納される。 Q の先頭から m 個の列は、指定した不変部分空間の直交基を形成する。
w	COMPLEX ($ctrsen$ の場合) DOUBLE COMPLEX ($ztrsen$ の場合)。 配列、次元は $\max(1, n)$ 以上。 格納される R の固有値。一連の固有値は、 R の対角成分と同じ順序で格納される。

<i>wr, wi</i>	REAL (strsen の場合) DOUBLE PRECISION (dtrsen の場合) 配列、次元は $\max(1, n)$ 以上。 R の順序変更後の固有値の実数部と虚数部が、それぞれ格納される。一連の固有値は、 R の対角成分と同じ順序で格納される。ただし、複素数の固有値の条件が非常に悪い場合には、その値が順序変更前から大幅に変更される場合がある。
<i>m</i>	INTEGER。 <i>複素数型の場合:</i> 指定した不変部分空間の個数であり、選択した固有値の個数と同じである (<i>select</i> を参照)。 <i>実数型の場合:</i> 指定した不変部分空間の次元。選択した実数の固有値ごとに 1 を、固有値の選択した複素共役ペアごとに 2 をカウントすれば、 <i>m</i> の値が得られる (<i>select</i> を参照)。 次の制約がある。 $0 \leq m \leq n$ 。
<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>job='E' または 'B' の場合</i> 、 <i>s</i> は、固有値の選択した束の平均の条件数の逆数の下限になる。 $m=0$ または n の場合、 $s=1$ 。 <i>実数型の場合:</i> <i>info=1</i> の場合、 <i>s</i> はゼロに設定される。 <i>job='N' または 'V' の場合</i> 、 <i>s</i> は参照されない。
<i>sep</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>job='V' または 'B' の場合</i> 、 <i>sep</i> は、指定した不変部分空間の条件数の逆数の見積もりである。 $m=0$ または n の場合、 $sep = \ T\ $ 。 <i>実数型の場合:</i> <i>info=1</i> の場合、 <i>sep</i> はゼロに設定される。 <i>job='N' または 'E' の場合</i> 、 <i>sep</i> は参照されない。
<i>work(1)</i>	終了時に、 <i>info=0</i> の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 <i>info=0</i> の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info=0</i> の場合、正常に終了したことを示す。 <i>info=-i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *trsen* のインターフェイスの詳細を以下に示す。

t サイズ (n, n) の行列 T を格納する。

<code>select</code>	長さ (n) のベクトルを格納する。
<code>wr</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>q</code>	サイズ (n, n) の行列 Q を格納する。
<code>compq</code>	引数 q の存在に基づいて以下のように復元される。 <code>compq = 'V'</code> (q が存在する場合)、 <code>compq = 'N'</code> (q が省略された場合)。
<code>job</code>	引数 s と <code>sep</code> の存在に基づいて以下のように復元される。 <code>job = 'B'</code> (s と <code>sep</code> の両方が存在する場合)、 <code>job = 'E'</code> (s が存在し、 <code>sep</code> が省略された場合)、 <code>job = 'V'</code> (s が省略され、 <code>sep</code> が存在する場合)、 <code>job = 'N'</code> (s と <code>sep</code> の両方が省略された場合)。

アプリケーション・ノート

計算で求めた行列 R は、行列 $T + E$ の近似値である。ここで、 $\|E\|_2 = O(\epsilon)\|T\|_2$ 、 ϵ はマシン精度である。

計算で求めた s は、 $(\min(m, n-m))^{1/2}$ を超える係数で条件数の真の逆数を過小評価できない。`sep` は、真の値から $(m*n-m^2)^{1/2}$ だけ異なっている場合がある。計算で求めた不変部分空間と真の部分空間の間の角は、 $O(\epsilon)\|A\|_2/\text{sep}$ である。

2×2 の対角ブロックが順序変更に関与する場合は、一般に、その非対角成分が変更される。そのブロックの条件が非常に悪い場合を除き、そのブロックの対角成分と固有値は変更されない。条件が非常に悪い場合は、それらが明確に変更される場合もある。 2×2 のブロックは、2つの 1×1 のブロックに分割できる。つまり、複素数の固有値ペアを、純粋な実数にできる。ただし、固有値の値が、順序の変更によって変更されたりはしない。

?trsyl

実準三角行列または複素三角行列に関するシルベスター式を解く。

構文

Fortran 77:

```
call strsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
           info)
call dtrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
           info)
call ctrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
           info)
call ztrsyl(trana, tranb, isgn, m, n, a, lda, b, ldb, c, ldc, scale,
           info)
```

Fortran 95:

```
call trsyl(a, b, c, scale [,trana] [,tranb] [,isgn] [,info])
```

説明

このルーチンは、シルベスターの行列方程式 $\text{op}(A)X \pm X\text{op}(B) = \alpha C$ を解く。 $\text{op}(A) = A$ または A^H であり、行列 A と B は上三角行列 (実数型の場合は標準形の Schur 形式の上準三角行列) である。 $\alpha \leq 1$ は、 X のオーバーフローを防止するためにこのルーチンで求めるスケール係数である。 A は $m \times m$ 、 B は $n \times n$ 、 C と X はどちらも $m \times n$ である。行列 X は、後方置換によって簡単に求められる。

この方程式は、 $\alpha_i \pm \beta_i \neq 0$ の場合にだけ、一意な解を持つ。 $\{\alpha_i\}$ と $\{\beta_i\}$ はそれぞれ、 A と B の固有値である。符号 (+ または -) は、解くべき方程式の中で使われているものと同じである。

入力パラメーター

<i>trana</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 <i>trana</i> = 'N' の場合、 $\text{op}(A) = A$ 。 <i>trana</i> = 'T' の場合、 $\text{op}(A) = A^T$ (実数型の場合のみ)。 <i>trana</i> = 'C' の場合、 $\text{op}(A) = A^H$ 。
<i>tranb</i>	CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 <i>tranb</i> = 'N' の場合、 $\text{op}(B) = B$ 。 <i>tranb</i> = 'T' の場合、 $\text{op}(B) = B^T$ (実数型の場合のみ)。 <i>tranb</i> = 'C' の場合、 $\text{op}(B) = B^H$ 。
<i>isgn</i>	INTEGER。シルベスター式の形式を指定する。 <i>isgn</i> = +1 の場合、 $\text{op}(A)X + X\text{op}(B) = \alpha C$ 。 <i>isgn</i> = -1 の場合、 $\text{op}(A)X - X\text{op}(B) = \alpha C$ 。
<i>m</i>	INTEGER。A の次数、および X と C ($m \geq 0$) の行数。
<i>n</i>	INTEGER。B の次数、および X と C ($n \geq 0$) の行数。
<i>a, b, c</i>	REAL (strsyl の場合) DOUBLE PRECISION (dtrsyl の場合) COMPLEX (ctrsyl の場合) DOUBLE COMPLEX (ztrsyl の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、行列 B を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>c</i> (<i>ldc</i> ,*) には、行列 C を格納する。 <i>c</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。a の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。b の第 1 次元。 $\max(1, n)$ 以上。
<i>ldc</i>	INTEGER。c の第 1 次元。 $\max(1, n)$ 以上でなければならない。

出力パラメーター

<i>c</i>	解行列 X によって上書きされる。
<i>scale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 スケール係数 a の値。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - i の場合、 i 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、 A と B は共通または近似の固有値を持ち、その摂動値を使ってこの方程式が解かれる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `trsy1` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, m) の行列 A を格納する。
<i>b</i>	サイズ (n, n) の行列 B を格納する。
<i>c</i>	サイズ (m, n) の行列 C を格納する。
<i>trana</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>tranb</i>	'N'、'C'、または 'T' でなければならない。デフォルト値は 'N'。
<i>isgn</i>	+1 または -1 でなければならない。デフォルト値は +1 である。

アプリケーション・ノート

X が正確な解、 Y が対応する計算で求めた解、 R が誤差行列であり、 $R = C - (AY \pm YB)$ とする。その場合の誤差は、常に小さくなる。

$$\|R\|_F = O(\epsilon) (\|A\|_F + \|B\|_F) \|Y\|_F$$

ただし、 Y は、わずかに摂動した方程式の場合、正確な解とならない場合もある。つまり、この解は後方安定ではない。

前方誤差に関しては、次の制限が成立する。

$$\|Y - X\|_F \leq \|R\|_F / \text{sep}(A, B)$$

ただし、これは、大幅な過大評価である場合もある。 $\text{sep}(A, B)$ の定義については、[\[Golub96\]](#) を参照。

実数型の場合の浮動小数演算のおおよその総数は、 $m * n * (m + n)$ である。複素数型の場合、この 4 倍になる。

汎用非対称固有値問題

このセクションでは、汎用非対称固有値問題を解いたり、行列ペアの Schur 因子分解を順序変更したり、関連する種々の計算タスクを実行するための LAPACK ルーチンについて説明する。

汎用非対称固有値問題は、次のように説明できる。 $n \times n$ の非対称行列 (または非エルミート行列) のペア A と B があるとき、次の 2 つの式を満たす汎用固有値 λ と、それに対応する汎用固有ベクトル x と y を見つけ出す。

$$Ax = \lambda Bx \quad (\text{右汎用固有ベクトル } x)$$

および

$$y^H A = \lambda y^H B \quad (\text{左汎用固有ベクトル } y)$$

表 4-6 に、汎用非対称固有値問題と汎用シルベスター式を解くための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。

Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 4-6 汎用非対称固有値問題を解く計算ルーチン

ルーチン名	機能
?gghrd	直交 / ユニタリー変換を使用して、行列ペアを汎用上 Hessenberg 形式に縮退する。
?ggbal	一般の実 / 複素行列ペアを平衡化する。
?ggbak	汎用固有値問題の右または左固有ベクトルを求める。
?hgeqz	QZ 法により行列ペア (H, T) の汎用固有値を求める。
?tgevc	上三角行列ペアから右または左汎用固有ベクトルの一部または全部を求める。
?tgexc	行列ペア (A, B) において、ある対角ブロックが別の行インデックスに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
?tgsen	選択した固有値クラスターが行列ペア (A, B) の主対角ブロックに移動するように、(A, B) の汎用 Schur 分解の順序を変更する。
?tgsyl	汎用シルベスター式を解く。
?tgsna	汎用実 Schur 標準形の行列ペアに関して、指定した固有値と固有ベクトルの条件数の逆数を見積もる。

?gghrd

直交/ユニタリー変換を使用して、行列ペアを汎用上 Hessenberg 形式に縮退する。

構文

Fortran 77:

```
call sgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz,
            info)
call dgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz,
            info)
call cgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz,
            info)
call zgghrd(compq, compz, n, ilo, ihi, a, lda, b, ldb, q, ldq, z, ldz,
            info)
```

Fortran 95:

```
call gghrd(a, b [,ilo] [,ihi] [,q] [,z] [,compq] [,compz] [,info])
```

説明

このルーチンは、直交/ユニタリー変換を使用して、実/複素行列ペア (A, B) を、汎用上 Hessenberg 形式に縮退する。ここで、A は一般行列、B は上三角行列である。汎用固有値問題は、 $Ax = \lambda Bx$ で表される。一般に、B は、QR 因子分解を計算し、直交行列 Q を式の左辺に移動すると、上三角行列になる。

このルーチンは、A を次のように Hessenberg 行列 H に縮退し、

$$Q^H A Z = H$$

同時に B を次のように別の上三角行列 T に変換すると、

$$Q^H B Z = T$$

元の問題を標準形式 $Hy = \lambda Ty$ (ただし、 $y = Z^H x$) に縮退する。

直交/ユニタリー行列 Q と Z は、Givens 回転の積で求められる。この 2 つの行列は、明示的に求められ、次のように入力行列 Q_1 と Z_1 に後からも掛けられる。

$$Q_1 A Z_1^H = (Q_1 Q) H (Z_1 Z)^H$$

$$Q_1 B Z_1^H = (Q_1 Q) T (Z_1 Z)^H$$

Q_1 が、元の式 $Ax = \lambda Bx$ における B の QR 因子分解で得た直交行列であれば、?gghrd は元の問題を汎用 Hessenberg 形式に縮退する。

入力パラメーター

compq CHARACTER*1. 'N'、'I'、または 'V' のいずれかでなければならない。

compq = 'N' の場合、行列 Q は計算されない。

compq = 'I' の場合、Q は単位行列に初期化され、直交/ユニタリー行列 Q が返される。

	<i>compq</i> = 'V' の場合、呼び出し時に、 Q には、直交 / ユニタリー行列 Q_1 が格納されていなければならない。終了時に、積 $Q_1 Q$ が返される。
<i>compz</i>	CHARACTER*1。'N'、'I'、または 'V' のいずれかでなければならない。 <i>compz</i> = 'N' の場合、行列 Z は計算されない。 <i>compz</i> = 'I' の場合、 Z は単位行列に初期化され、直交 / ユニタリー行列 Z が返される。 <i>compz</i> = 'V' の場合、 Z には、呼び出し時に、直交 / ユニタリー行列 Z_1 が格納されていなければならない。終了時に、積 $Z_1 Z$ が返される。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。 <i>ilo</i> と <i>ihi</i> は、 A のどの行と列を縮退するかをマークする。 A において、行と列が $1:iilo-1$ と $ihhi+1:n$ の部分は、すでに上三角形式になっているものとする。一般に、 <i>ilo</i> と <i>ihi</i> の値は、先に呼び出した zgqbal によって設定されるが、そうでない場合はそれぞれ 1 と n に設定する。次の制約がある。 $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihhi = 0$ 。
<i>a, b, q, z</i>	REAL (sgghrd の場合) DOUBLE PRECISION (dgghrd の場合) COMPLEX (cgghrd の場合) DOUBLE COMPLEX (zgghrd の場合)。 配列： <i>a(lda,*)</i> $n \times n$ の一般行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b(l db,*)</i> $n \times n$ の上三角行列 B を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>q(ldq,*)</i> <i>compq</i> = 'N' の場合、 <i>q</i> は参照されない。 <i>compq</i> = 'I' の場合、呼び出し時に、 <i>q</i> を設定する必要はない。 <i>compq</i> = 'V' の場合、 <i>q</i> には、直交 / ユニタリー行列 Q_1 (通常は B の QR 因子分解) が格納されていなければならない。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>z(ldz,*)</i> <i>compq</i> = 'N' の場合、 <i>z</i> は参照されない。 <i>compq</i> = 'I' の場合、呼び出し時に、 <i>z</i> を設定する必要はない。 <i>compq</i> = 'V' の場合、 <i>z</i> には、直交 / ユニタリー行列 Z_1 が格納されていなければならない。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldq</i>	INTEGER。 <i>q</i> の第 1 次元。 <i>compq</i> = 'N' の場合、 $ldq \geq 1$ 。 <i>compq</i> = 'I' または 'V' の場合、 $ldq \geq \max(1, n)$ 。

ldz INTEGER。 *z* の第 1 次元。
compq = 'N' の場合、 $ldz \geq 1$ 。
compq = 'I' または 'V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

a 終了時に、*A* の上三角部分と最初の劣対角成分が上 Hessenberg 行列 *H* で上書きされ、残りの成分はゼロに設定される。

b 終了時に、上三角行列 $T = Q^H B Z$ で上書きされる。対角成分より下の各成分はゼロに設定される。

q *compq* = 'I' の場合、*q* に、直交 / ユニタリー行列 *Q* (Q^H は、*A* と *B* の左側に適用される Givens 変換の積) が格納される。
compq = 'V' の場合、*q* は積 $Q_1 Q$ で上書きされる。

z *compq* = 'I' の場合、*z* に、直交 / ユニタリー行列 *Z* (*A* と *B* の右側に適用される Givens 変換の積) が格納される。
compq = 'V' の場合、*z* は積 $Z_1 Z$ で上書きされる。

info INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gghrd* のインターフェイスの詳細を以下に示す。

a サイズ (*n, n*) の行列 *A* を格納する。

b サイズ (*n, n*) の行列 *B* を格納する。

q サイズ (*n, n*) の行列 *Q* を格納する。

z サイズ (*n, n*) の行列 *Z* を格納する。

ilo この引数のデフォルト値は、*ilo* = 1 である。

ihi この引数のデフォルト値は、*ihi* = *n* である。

compq 省略された場合、この引数は、引数 *q* の存在に基づいて以下のように復元される。
compq = 'I' (*q* が存在する場合)、
compq = 'N' (*q* が省略された場合)。
存在する場合、*compz* は 'I' または 'V' と等しくなければならない、引数 *q* も存在しなければならない。
compz が存在し、*q* が省略された場合、エラー条件がセットされる。

compz 省略された場合、この引数は、引数 *z* の存在に基づいて以下のように復元される。
compz = 'I' (*z* が存在する場合)、
compz = 'N' (*z* が省略された場合)。

存在する場合、*compz* は 'I' または 'V' と等しくなければならず、引数 *z* も存在しなければならない。
compz が存在し、*z* が省略された場合、エラー条件がセットされる。

?ggbal

一般の実/複素行列ペアを平衡化する。

構文

Fortran 77:

```
call sggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call dggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call cggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
call zggbal(job, n, a, lda, b, ldb, ilo, ihi, lscale, rscale, work, info)
```

Fortran 95:

```
call ggbal(a, b [,ilo] [,ihi] [,lscale] [,rscale] [,job] [,info])
```

説明

このルーチンは、一般の実/複素行列ペア (A, B) を平衡化する。すなわち、まず相似変換によって A と B を置換し、固有値を対角線上の最初の $1 \sim ilo-1$ と最後の $ihi+1 \sim n$ の成分に分離する。次に、 $ilo \sim ihi$ の行と列に対角相似変換を適用して、これらの行と列のノルムができる限り近くなるようにする。この2つのステップはオプションである。

平衡化により、行列の 1- ノルムが縮退され、汎用固有値問題 $Ax = \lambda Bx$ における固有値と固有ベクトルの計算精度が向上する。

入力パラメーター

<i>job</i>	CHARACTER*1。 A と B に対して実行する演算を指定する。 'N'、'P'、'S'、または 'B' のいずれかでなければならない。 <i>job</i> ='N' の場合、演算は行わず、 $ilo=1$ 、 $ihi=n$ 、 $lscale(i)=1.0$ 、 $rscale(i)=1.0$ (ただし、 $i=1, \dots, n$) に設定する。 <i>job</i> ='P' の場合、置換のみ。 <i>job</i> ='S' の場合、スケーリングのみ。 <i>job</i> ='B' の場合、置換とスケーリング。
<i>n</i>	INTEGER。 行列 A と B の次数 ($n \geq 0$)。
<i>a, b</i>	REAL (sggbal の場合) DOUBLE PRECISION (dggbal の場合) COMPLEX (cggbal の場合) DOUBLE COMPLEX (zggbal の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、行列 A を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。

	$b(ldb,*)$ には、行列 B を格納する。 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。
$work$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 ワークスペース配列、次元は $\max(1, 6n)$ 以上。

出力パラメーター

a, b	平衡化した行列 A と B でそれぞれが上書きされる。 $job='N'$ の場合、 a と b は参照されない。
ilo, ihi	INTEGER。 終了時に、 ilo と ihi は、 $i > j$ で $j=1, \dots, ilo-1$ または $i=ihi+1, \dots, n$ であれば、 $a(i, j)=0$ と $b(i, j)=0$ が満たされるような整数に設定される。 $job='N'$ または ' S ' の場合、 $ilo=1$ 、 $ihi=n$ に設定される。
$lscale, rscale$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 $lscale$ には、 A と B の左側に適用される置換およびスケール係数の各成分が格納される。 行 j と交換する行のインデックスを P_j とし、行 j に適用されるスケール係数を D_j とすると、 $lscale(j) = P_j (j = 1, \dots, ilo-1 \text{ の場合})$ $= D_j (j = ilo, \dots, ihi \text{ の場合})$ $= P_j (j = ihi+1, \dots, n \text{ の場合})。$ $rscale$ には、 A と B の右側に適用される置換およびスケール係数の各成分が格納される。 列 j と交換する列のインデックスを P_j とし、列 j に適用されるスケール係数を D_j とすると、 $rscale(j) = P_j (j = 1, \dots, ilo-1 \text{ の場合})$ $= D_j (j = ilo, \dots, ihi \text{ の場合})$ $= P_j (j = ihi+1, \dots, n \text{ の場合})。$ 交換を実行する順序は、 $n \sim ihi+1$ 、次に $1 \sim ilo-1$ である。
$info$	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggbal` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
-----	-----------------------------

<i>b</i>	サイズ (n, n) の行列 B を格納する。
<i>lscale</i>	長さ (n) のベクトルを格納する。
<i>rscale</i>	長さ (n) のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 $ilo = 1$ である。
<i>ihi</i>	この引数のデフォルト値は、 $ihi = n$ である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。

?ggbak

汎用固有値問題の右または左固有ベクトルを
求める。

構文

Fortran 77:

```
call sggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call dggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call cggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
call zggbak(job, side, n, ilo, ihi, lscale, rscale, m, v, ldv, info)
```

Fortran 95:

```
call ggbak(v [,ilo] [,ihi] [,lscale] [,rscale] [,job] [,info])
```

説明

このルーチンは、平衡化した行列ペア ([?ggbal](#) から返されたもの) で計算した固有ベクトルを逆変換して、実数 / 複素数の汎用固有値問題 $Ax = \lambda Bx$ における右または左固有ベクトルを求める。

入力パラメーター

<i>job</i>	CHARACTER*1。必要な逆変換のタイプを指定する、'N'、'P'、'S'、または 'B' でなければならない。 <i>job</i> ='N' の場合、演算を行わずにリターンする。 <i>job</i> ='P' の場合、置換に対する逆変換のみ行う。 <i>job</i> ='S' の場合、スケーリングに対する逆変換のみ行う。 <i>job</i> ='B' の場合、置換とスケーリングに対する逆変換を行う。 この引数は、 ?ggbal に指定した引数 <i>job</i> と同じ値でなければならない。
<i>side</i>	CHARACTER*1。'L' または 'R' でなければならない。 <i>side</i> ='L' の場合、 <i>v</i> に左固有ベクトルが格納される。 <i>side</i> ='R' の場合、 <i>v</i> に右固有ベクトルが格納される。
<i>n</i>	INTEGER。行列 V の行数 ($n \geq 0$)。

<i>ilo, ihi</i>	INTEGER。?gebal で決定された整数 <i>ilo</i> と <i>ihi</i> 。次の制約がある。 <i>n</i> > 0 の場合、 $1 \leq ilo \leq ihi \leq n$ 。 <i>n</i> = 0 の場合、 <i>ilo</i> = 1 かつ <i>ihi</i> = 0。
<i>lscale, rscale</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 配列 <i>lscale</i> には、 <i>A</i> と <i>B</i> の左側に適用する置換およびスカラー係数の詳細 (?ggbal から返された値) を格納する。 配列 <i>rscale</i> には、 <i>A</i> と <i>B</i> の右側に適用する置換およびスカラー係数の詳細 (?ggbal から返された値) を格納する。
<i>m</i>	INTEGER。行列 <i>V</i> の列数 ($m \geq 0$)。
<i>v</i>	REAL (sggbak の場合) DOUBLE PRECISION (dggbak の場合) COMPLEX (cggbak の場合) DOUBLE COMPLEX (zggbak の場合)。 配列 <i>v</i> (<i>ldv</i> , *)。変換する右または左固有ベクトルが入った行列 (?tqevc から返された値) を格納する。 <i>v</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。
<i>ldv</i>	INTEGER。 <i>v</i> の第 1 次元。 $\max(1, n)$ 以上。

出力パラメーター

<i>v</i>	変換後の固有ベクトルによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン ggbak のインターフェイスの詳細を以下に示す。

<i>v</i>	サイズ (<i>n</i> , <i>m</i>) の行列 <i>V</i> を格納する。
<i>lscale</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>rscale</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>ilo</i>	この引数のデフォルト値は、 <i>ilo</i> = 1 である。
<i>ihi</i>	この引数のデフォルト値は、 <i>ihi</i> = <i>n</i> である。
<i>job</i>	'B'、'S'、'P'、または 'N' でなければならない。デフォルト値は 'B'。
<i>side</i>	省略された場合、この引数は、引数 <i>lscale</i> と <i>rscale</i> の存在に基づいて以下のように復元される。 <i>side</i> = 'L' (<i>lscale</i> が存在し、 <i>rscale</i> が省略された場合)、

`side = 'R'` (`lscale` が省略され、`rscale` が存在する場合)。
`lscale` と `rscale` の両方が存在するか、両方が省略された場合、エラー条件がセットされる。

?hgeqz

QZ 法により行列ペア (H, T) の汎用固有値を求める。

構文

Fortran 77:

```
call shgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphas,
            alphai, beta, q, ldq, z, ldz, work, lwork, info)
call dhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alphas,
            alphai, beta, q, ldq, z, ldz, work, lwork, info)
call chgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info)
call zhgeqz(job, compq, compz, n, ilo, ihi, h, ldh, t, ldt, alpha,
            beta, q, ldq, z, ldz, work, lwork, rwork, info)
```

Fortran 95:

```
call hgeqz(h, t [,ilo] [,ihi] [,alphar] [,alphai] [,beta] [,q] [,z]
           [,job] [,compq] [,compz] [,info])
call hgeqz(h, t [,ilo] [,ihi] [,alpha] [,beta] [,q] [,z] [,job] [,compq]
           [,compz] [,info])
```

説明

このルーチンは、ダブルシフト版 (実数型の場合) またはシングルシフト版 (複素数型の場合) の *QZ* 法を使用して、実/複素行列ペア (H, T) の固有値を計算する。ただし、 H は上 Hessenberg 行列、 T は上三角行列である。

このタイプの行列ペアは、実/複素行列ペア (A, B) の汎用上 Hessenberg 形式への縮退で得られる。

$$A = Q_1 H Z_1^H, \quad B = Q_1 T Z_1^H$$

?gghrd によって計算される。

実数型の場合:

`job = 'S'` の場合、Hessenberg- 三角行列ペア (H, T) も、次のような汎用 Schur 形式に縮退される。

$$H = Q S Z^T, \quad T = Q P Z^T,$$

Q と Z は直交行列、 P は上三角行列、 S は、 1×1 と 2×2 の対角ブロックを持つ準三角行列である。

1×1 のブロックは、行列ペア (H, T) の実固有値に対応し、 2×2 のブロックは固有値の複素共役ペアに対応する。

また、 S の 2×2 のブロックに対応する P の 2×2 の上三角対角ブロックは、正値対角形式に縮退される。すなわち、 $S(j+1, j)$ がゼロでなければ、 $P(j+1, j) = P(j, j+1) = 0$ 、 $P(j, j) > 0$ 、 $P(j+1, j+1) > 0$ となる。

複素数型の場合:

$job = 'S'$ の場合、Hessenberg- 三角行列ペア (H, T) も、次のような汎用 Schur 形式に縮退される。

$$H = Q S Z^H, \quad T = Q P Z^H$$

Q と Z はユニタリー行列、 S と P は上三角行列である。

すべての場合:

オプションで、汎用 Schur 因子分解から得た直交 / ユニタリー行列 Q を入力行列 Q_1 の後に掛け、直交 / ユニタリー行列 Z を入力行列 Z_1 の後に掛けられる。 Q_1 と Z_1 が、`?ggghrd` により行列ペア (A, B) を汎用上 Hessenberg 形式に縮退した直交 / ユニタリー行列であれば、出力行列 $Q_1 Q$ と $Z_1 Z$ は、 (A, B) の汎用 Schur 因子分解で得た直交 / ユニタリー因子である。

$$A = (Q_1 Q) S (Z_1 Z)^H, \quad B = (Q_1 Q) P (Z_1 Z)^H$$

オーバーフローを避けるために、行列ペア (H, T) の固有値 (すなわち、 (A, B) の固有値) は、値のペア (α, β) として計算される。`chgeqz / zhgeqz` の場合、 α と β は複素数である。`shgeqz / dhgeqz` の場合、 α は複素数、 β は実数である。 β がゼロでなければ、 $\lambda = \alpha / \beta$ は、次の汎用非対称固有値問題 (GNEP) の固有値である。

$$Ax = \lambda Bx$$

また、 α がゼロでなければ、 $\mu = \beta / \alpha$ は、次のように別の形式で表された GNEP の固有値である。

$$\mu Ay = By$$

実固有値 (実数型の場合)、または i 番目の固有値を表す α と β (複素数型の場合) は、次のように汎用 Schur 形式から直接読み込める。

$$\alpha = S(i, i), \quad \beta = P(i, i).$$

入力パラメーター

<code>job</code>	CHARACTER*1. 実行する演算を指定する。'E' または 'S' でなければならない。 <code>job = 'E'</code> の場合、固有値のみを計算する。 <code>job = 'S'</code> の場合、固有値と Schur 形式を計算する。
<code>compq</code>	CHARACTER*1. 'N'、'I'、または 'V' のいずれかでなければならない。 <code>compq = 'N'</code> の場合、左 Schur ベクトル (q) は計算されない。 <code>compq = 'I'</code> の場合、 q は単位行列に初期化され、 (H, T) の左 Schur ベクトルからなる行列が返される。 <code>compq = 'V'</code> の場合、 q には、呼び出し時に、直交 / ユニタリー行列 Q_1 が格納されていないなければならない。終了時に、積 $Q_1 Q$ が返される。
<code>compz</code>	CHARACTER*1. 'N'、'I'、または 'V' のいずれかでなければならない。

	<i>compz</i> = 'N' の場合、左 Schur ベクトル (<i>q</i>) は計算されない。 <i>compz</i> = 'I' の場合、 <i>z</i> は単位行列に初期化され、(<i>H</i> , <i>T</i>) の右 Schur ベクトルからなる行列が返される。
	<i>compz</i> = 'V' の場合、 <i>z</i> には、呼び出し時に、直交 / ユニタリー行列 Z_1 が格納されていなければならない。終了時に、積 $Z_1 Z$ が返される。
<i>n</i>	INTEGER。行列 <i>H</i> , <i>T</i> , <i>Q</i> , および <i>Z</i> の次数 ($n \geq 0$)。
<i>ilo</i> , <i>ihi</i>	INTEGER。 <i>ilo</i> と <i>ihi</i> は、 <i>H</i> のどの行と列が Hessenberg 形式かをマークする。 <i>H</i> において、行と列が $1:iilo-1$ と $ihi+1:n$ の範囲は、すでに上三角形式になっているものとする。次の制約がある。 $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。 $n = 0$ の場合、 $ilo = 1$ かつ $ihi = 0$ 。
<i>h</i> , <i>t</i> , <i>q</i> , <i>z</i> , <i>work</i>	REAL (<i>shgeqz</i> の場合) DOUBLE PRECISION (<i>dhgeqz</i> の場合) COMPLEX (<i>chgeqz</i> の場合) DOUBLE COMPLEX (<i>zhgeqz</i> の場合)。 配列: 呼び出し時に、 <i>h</i> (<i>ldh</i> , *) には、 $n \times n$ の上 Hessenberg 行列 <i>H</i> を格納する。 <i>h</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 呼び出し時に、 <i>t</i> (<i>ldt</i> , *) には、 $n \times n$ の上三角行列 <i>T</i> を格納する。 <i>t</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>q</i> (<i>ldq</i> , *): 呼び出し時に、 <i>compq</i> = 'V' の場合、(<i>A</i> , <i>B</i>) を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリー行列 Q_1 を格納する。 <i>compq</i> = 'N' の場合、 <i>q</i> は参照されない。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>z</i> (<i>ldz</i> , *): 呼び出し時に、 <i>compz</i> = 'V' の場合、(<i>A</i> , <i>B</i>) を汎用 Hessenberg 形式に縮退するのに使用した直交 / ユニタリー行列 Z_1 を格納する。 <i>compz</i> = 'N' の場合、 <i>z</i> は参照されない。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>ldh</i>	INTEGER。 <i>h</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldt</i>	INTEGER。 <i>t</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldq</i>	INTEGER。 <i>q</i> の第 1 次元。 <i>compq</i> = 'N' の場合、 $ldq \geq 1$ 。 <i>compq</i> = 'I' または 'V' の場合、 $ldq \geq \max(1, n)$ 。
<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。 <i>compz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>compz</i> = 'I' または 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 $lwork \geq \max(1, n)$ $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルー

チンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。

rwork

REAL (chgeqz の場合)

DOUBLE PRECISION (zhgeqz の場合)。

ワークスペース配列、次元は $\max(1, n)$ 以上。複素数型でのみ使用される。

出力パラメーター

h

実数型の場合:

job='S' の場合、終了時に、汎用 Schur 因子分解で得た上準三角行列 *S* が *h* に格納される。2×2 の対角ブロック (固有値の複素共役ペアに対応) が標準形で返され、 $h(i, i) = h(i+1, i+1)$ および $h(i+1, i) * h(i, i+1) < 0$ となる。

job='E' の場合、終了時に、*h* の対角ブロックは *S* と同じになるが、*h* のその他の部分は設定されない。

複素数型の場合:

job='S' の場合、終了時に、汎用 Schur 因子分解で得た上三角行列 *S* が *h* に格納される。

job='E' の場合、終了時に、*h* の対角成分は *S* と同じになるが、*h* のその他の部分は設定されない。

t

job='S' の場合、終了時に、汎用 Schur 因子分解で得た上三角行列 *P* が *t* に格納される。

実数型の場合:

S の 2×2 のブロックに対応する *P* の 2×2 の対角ブロックが、正値対角形式に縮退される。すなわち、 $h(j+1, j)$ がゼロでない場合、 $t(j+1, j) = t(j, j+1) = 0$ 、 $t(j, j)$ と $t(j+1, j+1)$ が正になる。

job='E' の場合、終了時に、*t* の対角ブロックは *P* と同じになるが、*t* のその他の部分は設定されない。

複素数型の場合:

job='E' の場合、終了時に、*t* の対角成分は *P* と同じになるが、*t* のその他の部分は設定されない。

alphar, alphai

REAL (shgeqz の場合)

DOUBLE PRECISION (dhgeqz の場合)。

配列、次元は $\max(1, n)$ 以上。

汎用非対称固有値問題において固有値を定義する各スカラー *alpha* の実部と虚部。

alphai(*j*) がゼロの場合、*j* 番目の固有値は実数である。正の場合、*j* 番目と (*j*+1) 番目の固有値は、 $\text{alphai}(j+1) = -\text{alphai}(j)$ であるような複素共役ペアである。

alpha

COMPLEX (chgeqz の場合)

DOUBLE COMPLEX (zhgeqz の場合)。

配列、次元は $\max(1, n)$ 以上。

汎用非対称固有値問題で固有値を定義する複素スカラー *alpha*。汎用 Schur 因子分解で、 $\text{alphai}(i) = S(i, i)$

<i>beta</i>	<p>REAL (shgeqz の場合) DOUBLE PRECISION (dhgeqz の場合) COMPLEX (chgeqz の場合) DOUBLE COMPLEX (zhgeqz の場合)。 配列、次元は $\max(1, n)$ 以上。 実数型の場合: 汎用非対称固有値問題で固有値を定義するスカラー <i>beta</i>。 $\alpha = (\alpha_{\text{phar}}(j), \alpha_{\text{hai}}(j))$ と $\beta = \beta(j)$ とで、行列ペア (A, B) の j 番目の固有値が、$\lambda = \alpha/\beta$、$\mu = \beta/\alpha$ のどちらかの形式で表される。λ または μ がオーバーフローする可能性があるので、一般にその計算は行わない。</p> <p>複素数型の場合: 汎用非対称固有値問題で固有値を定義する、非負の実スカラー <i>beta</i>。汎用 Schur 因子分解で、$\beta(i) = P(i, i)$。 $\alpha = \alpha(j)$ と $\beta = \beta(j)$ とで、行列ペア (A, B) の j 番目の固有値が、$\lambda = \alpha/\beta$、$\mu = \beta/\alpha$ のどちらかの形式で表される。λ または μ がオーバーフローする可能性があるので、一般にその計算は行わない。</p>
<i>q</i>	<p>終了時に、$\text{compq} = 'I'$ の場合、<i>q</i> は、行列ペア (H, T) の左 Schur ベクトルからなる直交/ユニタリー行列で上書きされる。 $\text{compq} = 'V'$ の場合、行列ペア (A, B) の左 Schur ベクトルからなる直交/ユニタリー行列で <i>q</i> は上書きされる。</p>
<i>z</i>	<p>終了時に、$\text{compz} = 'I'$ の場合、<i>z</i> は、行列ペア (H, T) の右 Schur ベクトルからなる直交/ユニタリー行列で上書きされる。 $\text{compz} = 'V'$ の場合、行列ペア (A, B) の右 Schur ベクトルからなる直交/ユニタリー行列で <i>z</i> は上書きされる。</p>
<i>work(1)</i>	<p>$\text{info} \geq 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。</p>
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1, ..., <i>n</i> の場合、QZ 反復で収束しなかった。 (H, T) は Schur 形式ではないが、$\alpha_{\text{phar}}(i)$、$\alpha_{\text{hai}}(i)$ (実数型の場合)、$\alpha(i)$ (複素数型の場合)、$\beta(i)$ は正しい値である (ただし、$i = \text{info} + 1, \dots, n$) <i>info</i> = $n + 1, \dots, 2n$ の場合、シフト計算に失敗した。 (H, T) は Schur 形式ではないが、$\alpha_{\text{phar}}(i)$、$\alpha_{\text{hai}}(i)$ (実数型の場合)、$\alpha(i)$ (複素数型の場合)、$\beta(i)$ は正しい値である (ただし、$i = \text{info} - n + 1, \dots, n$)。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hgeqz` のインターフェイスの詳細を以下に示す。

<code>h</code>	サイズ (n,n) の行列 H を格納する。
<code>t</code>	サイズ (n,n) の行列 T を格納する。
<code>alphar</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>alphai</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>alpha</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>beta</code>	長さ (n) のベクトルを格納する。
<code>q</code>	サイズ (n,n) の行列 Q を格納する。
<code>z</code>	サイズ (n,n) の行列 Z を格納する。
<code>ilo</code>	この引数のデフォルト値は、 $ilo = 1$ である。
<code>ihi</code>	この引数のデフォルト値は、 $ihi = n$ である。
<code>job</code>	'E' または 'S' でなければならない。デフォルト値は 'E'。
<code>compq</code>	省略された場合、この引数は、引数 q の存在に基づいて以下のように復元される。 <code>compq = 'I'</code> (q が存在する場合)、 <code>compq = 'N'</code> (q が省略された場合)。 存在する場合、 <code>compz</code> は 'I' または 'V' と等しくなければならず、引数 q も存在しなければならない。 <code>compz</code> が存在し、 q が省略された場合、エラー条件がセットされる。
<code>compz</code>	省略された場合、この引数は、引数 z の存在に基づいて以下のように復元される。 <code>compz = 'I'</code> (z が存在する場合)、 <code>compz = 'N'</code> (z が省略された場合)。 存在する場合、 <code>compz</code> は 'I' または 'V' と等しくなければならず、引数 z も存在しなければならない。 <code>compz</code> が存在し、 z が省略された場合、エラー条件がセットされる。

?tgevc

上三角行列ペアの右と左汎用固有ベクトルの一部または全部を求める。

構文

Fortran 77:

```
call stgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, info)
call dtgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, info)
call ctgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, rwork, info)
call ztgevc(side, howmny, select, n, s, lds, p, ldp, vl, ldvl, vr,
            ldvr, mm, m, work, rwork, info)
```

Fortran 95:

```
call tgevc(s, p [,howmny] [,select] [,vl] [,vr] [,m] [,info])
```

説明

このルーチンは、実/複素行列ペア (S, P) の右と左固有ベクトルの一部または全部を求める。 S は準三角行列 (実数型の場合) または上三角行列 (複素数型の場合)、 P は上三角行列である。

このタイプの行列ペアは、実/複素行列ペア (A, B) の汎用 Schur 因子分解で得られる。

$$A = Q S Z^H, \quad B = Q P Z^H$$

?gghrd と ?hgeqz によって計算される。

固有値 w に対応する、 (S, P) の右固有ベクトル x と左固有ベクトル y は次のように定義される。

$$Sx = wPx, \quad y^H S = w y^H P$$

固有値は、このルーチンに入力されず、 S と P の対角ブロックまたは対角成分から直接計算される。

このルーチンは、 (S, P) の右と左の固有ベクトルからなる行列 X と Y 、または積 ZX と QY (Z と Q は入力行列) を返す。

Q と Z が行列ペア (A, B) の汎用 Schur 因子分解で得た直交/ユニタリー因子であれば、 ZX と QY は (A, B) の右と左の固有ベクトルからなる行列である。

入力パラメーター

side CHARACTER*1。'R'、'L'、または 'B' でなければならない。
side='R' の場合、右固有ベクトルのみを計算する。
side='L' の場合、左固有ベクトルのみを計算する。
side='B' の場合、右と左の固有ベクトルを両方計算する。

<i>howmny</i>	<p>CHARACTER*1. 'A'、'B'、または 'S' でなければならない。 <i>howmny</i>='A' の場合、右と左の固有ベクトルをすべて計算する。 <i>howmny</i>='B' の場合、<i>vr</i> と <i>v1</i> 内の行列を逆変換して、右と左の固有ベクトルをすべて計算する。 <i>howmny</i>='S' の場合、論理配列 <i>select</i> で指定した右と左の固有ベクトルのみを計算する。</p>
<i>select</i>	<p>LOGICAL。 配列、次元は $\max(1, n)$ 以上。 <i>howmny</i>='S' の場合、<i>select</i> は、計算する固有値を指定する。 <i>howmny</i>='A' または 'B' の場合、<i>select</i> は参照されない。 実数型の場合: ω_j が実固有値の場合、<i>select</i>(<i>j</i>) が .TRUE. であれば、それに対応する実固有ベクトルが計算される。 ω_j と ω_{j+1} が複素固有値の実部と虚部の場合、<i>select</i>(<i>j</i>) または <i>select</i>(<i>j</i>+1) が .TRUE. であれば、それに対応する複素固有ベクトルが計算され、終了時に、<i>select</i>(<i>j</i>) は .TRUE.、<i>select</i>(<i>j</i>+1) は .FALSE. に設定される。 複素数型の場合: <i>select</i>(<i>j</i>) が .TRUE. であれば、<i>j</i> 番目の固有値に対応する固有ベクトルが計算される。</p>
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>s, p, v1, vr, work</i>	<p>REAL (<i>stgevc</i> の場合) DOUBLE PRECISION (<i>dtgevc</i> の場合) COMPLEX (<i>ctgevc</i> の場合) DOUBLE COMPLEX (<i>ztgevc</i> の場合)。 配列:</p> <p><i>s</i>(<i>lds</i>,*) には、?hgeqz による汎用 Schur 因子分解で得た行列 <i>S</i> を格納する。この行列は、上準三角行列 (実数型の場合) または三角行列 (複素数型の場合) である。 <i>s</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p><i>p</i>(<i>ldp</i>,*) には、?hgeqz による汎用 Schur 因子分解で得た行列 <i>P</i> を格納する。実数型の場合、<i>S</i> の 2×2 のブロックに対応する <i>P</i> の 2×2 の対角ブロックが、正値対角形式でなければならない。複素数型の場合、<i>P</i> は実対角成分を持たなければならない。 <i>p</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p><i>side</i>='L' または 'B' で <i>howmny</i>='B' の場合、 <i>v1</i>(<i>ldv1</i>,*) には、$n \times n$ の行列 <i>Q</i> (通常は、?hgeqz から返された、左 Schur ベクトルからなる直交/ユニタリー行列 <i>Q</i>) が格納されていなければならない。<i>v1</i> の第 2 次元は、$\max(1, mm)$ 以上でなければならない。<i>side</i>='R' の場合、<i>v1</i> は参照されない。</p> <p><i>side</i>='R' または 'B' で <i>howmny</i>='B' の場合、 <i>vr</i>(<i>ldvr</i>,*) には、$n \times n$ の行列 <i>Z</i> (通常は、?hgeqz から返された、右 Schur ベクトルからなる直交/ユニタリー行列 <i>Z</i>) が格納されていなければならない。<i>vr</i> の第 2 次元は、$\max(1, mm)$ 以上でなければならない。<i>side</i>='L' の場合、<i>vr</i> は参照されない。</p>

work(*) は、ワークスペース配列。
次元は $\max(1, 6*n)$ 以上 (実数型の場合) または $\max(1, 2*n)$ 以上 (複素数型の場合)。

<i>lds</i>	INTEGER。 <i>s</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldp</i>	INTEGER。 <i>p</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldvl</i>	INTEGER。 <i>v1</i> の第 1 次元。 <i>side</i> = 'L' または 'B' の場合、 $ldvl \geq \max(1, n)$ 。 <i>side</i> = 'R' の場合、 $ldvl \geq 1$ 。
<i>ldvr</i>	INTEGER。 <i>vr</i> の第 1 次元。 <i>side</i> = 'R' または 'B' の場合、 $ldvr \geq \max(1, n)$ 。 <i>side</i> = 'L' の場合、 $ldvr \geq 1$ 。
<i>mm</i>	INTEGER。 配列 <i>v1</i> と <i>vr</i> の列数 ($mm \geq m$)。
<i>rwork</i>	REAL (ctgevc の場合) DOUBLE PRECISION (ztgevc の場合)。 ワークスペース配列、次元は $\max(1, 2*n)$ 以上。複素数型でのみ 使用される。

出力パラメーター

<i>v1</i>	終了時に、 <i>side</i> = 'L' または 'B' の場合、 <i>v1</i> に次の値が格納される。 <i>howmny</i> = 'A' の場合、(<i>S</i> , <i>P</i>) の左固有ベクトルからなる行列 <i>Y</i> 。 <i>howmny</i> = 'B' の場合、行列 <i>QY</i> 。 <i>howmny</i> = 'S' の場合、 <i>select</i> で指定した (<i>S</i> , <i>P</i>) の左固有ベクトルが、固有値と同じ順に、 <i>v1</i> の連続した列に格納される。 実数型の場合： 複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。
<i>vr</i>	終了時に、 <i>side</i> = 'R' または 'B' の場合、 <i>vr</i> に次の値が格納される。 <i>howmny</i> = 'A' の場合、(<i>S</i> , <i>P</i>) の右固有ベクトルからなる行列 <i>X</i> 。 <i>howmny</i> = 'B' の場合、行列 <i>ZX</i> 。 <i>howmny</i> = 'S' の場合、 <i>select</i> で指定した (<i>S</i> , <i>P</i>) の右固有ベクトルが、固有値と同じ順に、 <i>vr</i> の連続した列に格納される。 実数型の場合： 複素固有値に対応する複素固有ベクトルが、連続した 2 つの列に格納される (最初の列に実部、2 番目の列に虚部)。
<i>m</i>	INTEGER。実際に固有値の格納に使用された配列 <i>v1</i> と <i>vr</i> の列数。 <i>howmny</i> = 'A' または 'B' の場合、 <i>m</i> には <i>n</i> が設定される。 実数型の場合： 選択された実固有ベクトル 1 つに対して 1 列、選択された複素固有ベクトル 1 つに対して 2 列が必要である。 複素数型の場合： 選択された固有ベクトル 1 つに対して 1 列必要である。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
 実数型の場合：
info = *i* > 0 の場合、2 × 2 のブロック (*i*:*i*+1) に複素固有値が格納されていない。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgevc* のインターフェイスの詳細を以下に示す。

s サイズ (*n*,*n*) の行列 *S* を格納する。

p サイズ (*n*,*n*) の行列 *P* を格納する。

select 長さ (*n*) のベクトルを格納する。

v1 サイズ (*n*,*mm*) の行列 *VL* を格納する。

vr サイズ (*n*,*mm*) の行列 *VR* を格納する。

side 引数 *v1* および *vr* の存在に基づいて以下のように復元される。
side = 'B' (*v1* と *vr* の両方が存在する場合)、
side = 'L' (*v1* が存在し、*vr* が省略された場合)、
side = 'R' (*v1* が省略され、*vr* が存在する場合)。
v1 と *vr* の両方が省略された場合、エラー条件がセットされる。

howmny 省略された場合、この引数は、引数 *select* の存在に基づいて以下のように復元される。
howmny = 'S' (*select* が存在する場合)、
howmny = 'A' (*select* が省略された場合)。
 存在する場合、*howmny* は 'A' または 'B' と等しくなければならず、引数 *select* も存在しなければならない。
howmny と *select* の両方が存在する場合、エラー条件がセットされる。

?tgexc

行列ペア (A, B) において、ある対角ブロックが別の行インデックスに移動するように、 (A, B) の汎用 Schur 分解の順序を変更する。

構文

Fortran 77:

```
call stgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,
           ifst, ilst, work, lwork, info)
call dtgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,
           ifst, ilst, work, lwork, info)
call ctgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,
           ifst, ilst, info)
call ztgexc(wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz,
           ifst, ilst, info)
```

Fortran 95:

```
call tgexc(a, b [,ifst] [,ilst] [,z] [,q] [,info])
```

説明

このルーチンは、直交 / ユニタリー等価変換を使用して、実 / 複素行列ペア (A, B) の汎用実 Schur/Schur 分解の順序を変更する。

$$(A, B) = Q (A, B) Z^H$$

このとき、 (A, B) において、行インデックス $ifst$ の対角ブロックが行 $ilst$ に移動するように変更する。

行列ペア (A, B) は、汎用実 Schur/Schur 標準形 ([?qges](#) から返される形式)、すなわち、 A は 1×1 と 2×2 の対角ブロックを持つブロック上三角行列、 B は上三角行列でなければならない。

オプションで、汎用 Schur ベクトルからなる行列 Q と Z を更新できる。

$$Q(\text{in}) * A(\text{in}) * Z(\text{in})' = Q(\text{out}) * A(\text{out}) * Z(\text{out})'$$

$$Q(\text{in}) * B(\text{in}) * Z(\text{in})' = Q(\text{out}) * B(\text{out}) * Z(\text{out})'$$

入力パラメーター

<code>wantq, wantz</code>	LOGICAL。 <code>wantq=.TRUE.</code> の場合、左側の変換行列 Q を更新する。 <code>wantq=.FALSE.</code> の場合、 Q を更新しない。 <code>wantz=.TRUE.</code> の場合、右側の変換行列 Z を更新する。 <code>wantz=.FALSE.</code> の場合、 Z を更新しない。
<code>n</code>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<code>a, b, q, z</code>	REAL (stgexc の場合) DOUBLE PRECISION (dtgexc の場合) COMPLEX (ctgexc の場合)

DOUBLE COMPLEX (ztgexc の場合)。

配列:

$a(lda,*)$ には、行列 A を格納する。

a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(l db,*)$ には、行列 B を格納する。

b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$q(ldq,*)$

$wantq = .FALSE.$ の場合、 q は参照されない。

$wantq = .TRUE.$ の場合、 q に直交 / ユニタリー行列 Q が格納されていなければならない。

q の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$z(ldz,*)$

$wantz = .FALSE.$ の場合、 z は参照されない。

$wantz = .TRUE.$ の場合、 z に直交 / ユニタリー行列 Z が格納されていなければならない。

z の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。
ldq	INTEGER。 q の第 1 次元。 $wantq = .FALSE.$ の場合、 $ldq \geq 1$ $wantq = .TRUE.$ の場合、 $ldq \geq \max(1, n)$ 。
ldz	INTEGER。 z の第 1 次元。 $wantz = .FALSE.$ の場合、 $ldz \geq 1$ 。 $wantz = .TRUE.$ の場合、 $ldz \geq \max(1, n)$ 。
$ifst, ilst$	INTEGER。 (A, B) の対角ブロックの順序変更を指定する。行インデックス $ifst$ のブロックが、隣接ブロックの連続的交換により、行 $ilst$ に移動される。次の制約がある。 $1 \leq ifst, ilst \leq n$ 。
$work$	REAL (stgexc の場合) DOUBLE PRECISION (dtgexc の場合)。 ワークスペース配列、次元は ($lwork$)。実数型でのみ使用される。
$lwork$	INTEGER。 $work$ の次元。 $4n+16$ 以上でなければならない $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、xerbla は $lwork$ に関するエラーメッセージを生成しない。

出力パラメーター

a, b	更新された行列 A と B で上書きされる。
$ifst, ilst$	実数型の場合にのみ上書きされる。 呼び出し時に、 $ifst$ が 2×2 のブロックの 2 行目を指している場合は、先頭行を指すように変更される。 $ilst$ は常に、最終位置 (入力値から ± 1 だけ変動している場合もある) のブロックの先頭行を指す。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = 1 の場合、変換後の行列ペア (*A*, *B*) が汎用 Schur 形式から遠すぎる。すなわち、この問題は悪条件である。*(A, B)* は部分的に順序変更されている可能性があり、*ilst* は移動中のブロックの現在位置の第 1 行を指している。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tgexc` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n, n</i>) の行列 <i>B</i> を格納する。
<i>z</i>	サイズ (<i>n, n</i>) の行列 <i>Z</i> を格納する。
<i>q</i>	サイズ (<i>n, n</i>) の行列 <i>Q</i> を格納する。
<i>wantq</i>	引数 <i>q</i> の存在に基づいて以下のように復元される。 <i>wantq</i> = .TRUE (<i>q</i> が存在する場合)、 <i>wantq</i> = .FALSE (<i>q</i> が省略された場合)。
<i>wantz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>wantz</i> = .TRUE (<i>z</i> が存在する場合)、 <i>wantz</i> = .FALSE (<i>z</i> が省略された場合)。

?tgsen

選択した固有値クラスターが行列ペア (A, B) の主対角ブロックに移動するように、 (A, B) の汎用 Schur 分解の順序を変更する。

構文

Fortran 77:

```
call stgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alphas,
           alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork, iwork,
           liwork, info)
call dtgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alphas,
           alphai, beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork, iwork,
           liwork, info)
call ctgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,
           beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork, iwork, liwork,
           info)
call ztgsen(ijob, wantq, wantz, select, n, a, lda, b, ldb, alpha,
           beta, q, ldq, z, ldz, m, pl, pr, dif, work, lwork, iwork, liwork,
           info)
```

Fortran 95:

```
call tgsen(a, b, select [,alphar] [,alphai] [,beta] [,ijob] [,q] [,z]
           [,pl] [,pr] [,dif] [,m] [,info])
call tgsen(a, b, select [,alpha] [,beta] [,ijob] [,q] [,z] [,pl] [,pr]
           [,dif] [,m] [,info])
```

説明

このルーチンは、実 / 複素行列ペア (A, B) の汎用実 Schur/Schur 分解の順序を変更し (直交 / ユニタリー等価変換 $Q' * (A, B) * Z$ を使用)、選択した固有値クラスターが、行列ペア (A, B) の主対角ブロックに現れるようにする。

Q と Z の先頭列は、対応する左と右の固有空間 (収縮部分空間) の正規直交 / ユニタリー基底を形成する。

(A, B) は、汎用実 Schur/Schur 標準形 ([?qges](#) から返される形式)、すなわち A と B がともに上三角行列でなければならない。

?tgsen は、順序変更後の行列ペア (A, B)

$\omega_j = (\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$ (実数型の場合)

$\omega_j = \text{alpha}(j)/\text{beta}(j)$ (複素数型の場合)

の汎用固有値も計算する。

このルーチンは、オプションで、固有値と固有空間の条件数の逆数を見積もる。すなわち、 $\text{Difu}[(A_{11}, B_{11}), (A_{22}, B_{22})]$ と $\text{Difl}[(A_{11}, B_{11}), (A_{22}, B_{22})]$ を計算する。これは、行列ペア (A_{11}, B_{11}) と (A_{22}, B_{22}) (選択したクラスターに対応するものとそれ以外の固有値に対応するもの) の差と、(1,1) ブロック内の選択したクラスターに対する左と右固有空間への「投影」のノルムの差である。

入力パラメーター

<i>ijob</i>	INTEGER。固有値クラスターに関する条件数 ($p1$ と pr)、または収縮部分空間 $Difu$ と $Difl$ を求めるかどうか指定する。 <i>ijob</i> = 0 の場合、 <i>select</i> に対する順序変更のみ。 <i>ijob</i> = 1 の場合、選択したクラスターに関する左と右の固有空間への「投影」のノルムの逆数 ($p1$ と pr)。 <i>ijob</i> = 2 の場合、F- ノルムベースの推定値を使用して、 $Difu$ と $Difl$ の上限値を計算する (<i>dif</i> (1:2))。 <i>ijob</i> = 3 の場合、1- ノルムベースの推定値を使用して、 $Difu$ と $Difl$ の推定値を計算する (<i>dif</i> (1:2))。このオプションは、 <i>ijob</i> = 2 の 5 倍の負荷がかかる。 <i>ijob</i> = 4 の場合、 $p1$ 、 pr 、 <i>dif</i> を計算する (上記のオプション 0、1、2 の組み合わせ)。すべてを計算する効率のよいオプション。 <i>ijob</i> = 5 の場合、 $p1$ 、 pr 、 <i>dif</i> を計算する (上記のオプション 0、1、3 の組み合わせ)。
<i>wantq</i> , <i>wantz</i>	LOGICAL。 <i>wantq</i> = .TRUE. の場合、左側の変換行列 Q を更新する。 <i>wantq</i> = .FALSE. の場合、 Q を更新しない。 <i>wantz</i> = .TRUE. の場合、右側の変換行列 Z を更新する。 <i>wantz</i> = .FALSE. の場合、 Z を更新しない。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 選択するクラスターに属する固有値を指定する。 固有値 ω_j を選択するには、 <i>select</i> (j) を .TRUE. に設定する。 実数型の場合: 複素共役固有値のペア ω_j と ω_{j+1} (2×2 の対角ブロックに対応) を選択するには、 <i>select</i> (j) と <i>select</i> ($j+1$) のどちらかを .TRUE. に設定する。複素共役固有値 ω_j と ω_{j+1} は、ともにクラスターに含めるか、ともにクラスターから除外する必要がある。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>q</i> , <i>z</i> , <i>work</i>	REAL (<i>stgsen</i> の場合) DOUBLE PRECISION (<i>dtgsen</i> の場合) COMPLEX (<i>ctgsen</i> の場合) DOUBLE COMPLEX (<i>ztgsen</i> の場合)。 配列: <i>a</i> (<i>lda</i> , *) には、行列 A を格納する。 実数型の場合: A は上準三角行列で、 (A, B) は汎用実 Schur 標準形。 複素数型の場合: A は上三角行列で、汎用 Schur 標準形。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> , *) には、行列 B を格納する。 実数型の場合: B は上三角行列で、 (A, B) は汎用実 Schur 標準形。 複素数型の場合: B は上三角行列で、汎用 Schur 標準形。 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

	<p>$q(ldq, *)$ $wantq = .TRUE.$ の場合、q は $n \times n$ の行列。 $wantq = .FALSE.$ の場合、q は参照されない。 q の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p>$z(ldz, *)$ $wantz = .TRUE.$ の場合、z は $n \times n$ の行列。 $wantz = .FALSE.$ の場合、z は参照されない。 z の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p>$work(lwork)$ は、ワークスペース配列である。 $ijob = 0$ の場合、$work$ は参照されない。</p>
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。
ldq	INTEGER。 q の第 1 次元。 $ldq \geq 1$ 。 $wantq = .TRUE.$ の場合、 $ldq \geq \max(1, n)$ 。
ldz	INTEGER。 z の第 1 次元。 $ldz \geq 1$ 。 $wantz = .TRUE.$ の場合、 $ldz \geq \max(1, n)$ 。
$lwork$	<p>INTEGER。 配列 $work$ の次元。 実数型の場合： $ijob = 1, 2, 4$ の場合、$lwork \geq \max(4n+16, 2m(n-m))$。 $ijob = 3$ または 5 の場合、$lwork \geq \max(4n+16, 4m(n-m))$。 複素数型の場合： $ijob = 1, 2, 4$ の場合、$lwork \geq \max(1, 2m(n-m))$。 $ijob = 3$ または 5 の場合、$lwork \geq \max(1, 4m(n-m))$。</p> <p>$lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、$xerbla$ は $lwork$ に関するエラーメッセージを生成しない。</p>
$iwork$	<p>INTEGER。 ワークスペース配列、次元は $(liwork)$。 $ijob = 0$ の場合、$iwork$ は参照されない。</p>
$liwork$	<p>INTEGER。 配列 $iwork$ の次元。 実数型の場合： $ijob = 1, 2, 4$ の場合、$liwork \geq n+6$。 $ijob = 3$ または 5 の場合、$liwork \geq \max(n+6, 2m(n-m))$。 複素数型の場合： $ijob = 1, 2, 4$ の場合、$liwork \geq n+2$。 $ijob = 3$ または 5 の場合、$liwork \geq \max(n+2, 2m(n-m))$。</p> <p>$liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $iwork$ 配列の最適サイズだけを計算し、その値を $iwork$ 配列の最初のエンタリーとして返す。$xerbla$ は $liwork$ に関するエラーメッセージを生成しない。</p>

出力パラメーター

a, b 順序変更された行列 A と B でそれぞれ上書きされる。

<i>alphar, alphai</i>	REAL (stgsen の場合) DOUBLE PRECISION (dtgsen の場合)。 配列、次元は $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>alpha</i>	COMPLEX (ctgsen の場合) DOUBLE COMPLEX (ztgsen の場合)。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>beta</i>	REAL (stgsen の場合) DOUBLE PRECISION (dtgsen の場合) COMPLEX (ctgsen の場合) DOUBLE COMPLEX (ztgsen の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>実数型の場合:</i> 終了時に、 $(\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$ 、 $j=1, \dots, n$ は、汎用固有値になる。 $\text{alphar}(j) + \text{alphai}(j)*i$ と $\text{beta}(j)$ 、 $j=1, \dots, n$ は、 (A, B) の実数汎用 Schur 形式の 2×2 の対角ブロックを複素数のユニタリー変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 (S, T) の対角である。 $\text{alphai}(j)$ がゼロの場合、 j 番目の固有値が実数であり、正の場合、 j 番目と $(j+1)$ 番目の固有値が複素共役ペアである ($\text{alphai}(j+1)$ は負)。 <i>複素数型の場合:</i> (A, B) を汎用 Schur 形式に縮退させたときの A と B の対角成分。 $\text{alpha}(i)/\text{beta}(i)$ 、 $i=1, \dots, n$ は、汎用固有値。
<i>q</i>	<i>wantq</i> = .TRUE. の場合、終了時に、 (A, B) の順序を変更する左直交変換行列が Q に掛けられる。 Q の先頭 m 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。
<i>z</i>	<i>wantz</i> = .TRUE. の場合、終了時に、 (A, B) の順序を変更する左直交変換行列が Z に掛けられる。 Z の先頭 m 列は、指定された左固有空間 (収縮部分空間) のペアに対する正規直交基底を形成する。
<i>m</i>	INTEGER。左と右固有空間 (収縮部分空間) のペアの次数、 $0 \leq m \leq n$ 。
<i>p1, pr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $i\text{job} = 1, 4, 5$ の場合、 $p1$ と pr は、選択したクラスターに関する左と右固有空間への「投影」のノルムの逆数の下限値。 $0 < p1, pr \leq 1$ 。 $m = 0$ または $m = n$ の場合、 $p1 = pr = 1$ 。 $i\text{job} = 0, 2, 3$ の場合、 $p1$ と pr は参照されない。
<i>dif</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は (2)。 $i\text{job} \geq 2$ の場合、 <i>dif</i> (1:2) に Difu と Difl の推定値が格納される。 $i\text{job} = 2$ または 4 の場合、 <i>dif</i> (1:2) は、F-ノルムベースの Difu と Difl の上限値。

	$ijob=3$ または 5 の場合、 $dif(1:2)$ は、1-ノルムベースの $Difu$ と $Difl$ の推定値。 $m=0$ または n の場合、 $dif(1:2)=F\text{-norm}([A, B])$ 。 $ijob=0$ または 1 の場合、 dif は参照されない。
$work(1)$	$ijob \neq 0$ かつ $info=0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$iwork(1)$	$ijob \neq 0$ かつ $info=0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $liwork$ の最小値が $iwork(1)$ に格納される。以降の実行では、この $liwork$ 値を使用する。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。 $info=1$ の場合、変換後の行列ペア (A, B) が汎用 Schur 形式から遠すぎるため、 (A, B) の順序変更で失敗した。すなわち、この問題は悪条件である。 (A, B) は部分的に順序変更されている可能性がある。 $dif(*)$ 、 $p1$ 、 pr が要求されている場合、 0 が返される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tgsen` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
b	サイズ (n, n) の行列 B を格納する。
$select$	長さ (n) のベクトルを格納する。
$alphar$	長さ (n) のベクトルを格納する。実数型でのみ使用される。
$alphai$	長さ (n) のベクトルを格納する。実数型でのみ使用される。
$alpha$	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
$beta$	長さ (n) のベクトルを格納する。
q	サイズ (n, n) の行列 Q を格納する。
z	サイズ (n, n) の行列 Z を格納する。
dif	長さ (2) のベクトルを格納する。
$ijob$	0 、 1 、 2 、 3 、 4 、または 5 でなければならない。デフォルト値は 0 。
$wantq$	引数 q の存在に基づいて以下のように復元される。 $wantq = .TRUE$ (q が存在する場合)、 $wantq = .FALSE$ (q が省略された場合)。
$wantz$	引数 z の存在に基づいて以下のように復元される。 $wantz = .TRUE$ (z が存在する場合)、 $wantz = .FALSE$ (z が省略された場合)。

?tgsyl

汎用シルベスター式を解く。

構文

Fortran 77:

```

call stgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
           lde, f, ldf, scale, dif, work, lwork, iwork, info)
call dtgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
           lde, f, ldf, scale, dif, work, lwork, iwork, info)
call ctgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
           lde, f, ldf, scale, dif, work, lwork, iwork, info)
call ztgsyl(trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e,
           lde, f, ldf, scale, dif, work, lwork, iwork, info)

```

Fortran 95:

```
call tgsyl(a, b, c, d, e, f [,ijob] [,trans] [,scale] [,dif] [,info])
```

説明

このルーチンは、次の汎用シルベスター式を解く。

$$A R - L B = scale * C$$

$$D R - L E = scale * F$$

R と L は未知の $m \times n$ の行列、 (A, D) 、 (B, E) 、 (C, F) はそれぞれ $m \times m$ 、 $n \times n$ 、 $m \times n$ の与えられた行列で、実/複素成分を持っている。 (A, D) と (B, E) は、汎用実 Schur/Schur 標準形でなければならない。すなわち、 A と B は上準三角行列/上三角行列、 D と E は上三角行列でなければならない。

(C, F) は、解 (R, L) で上書きされる。 $scale$ ($0 \leq scale \leq 1$) は、オーバーフローを避けるために選択された出力スケール係数である。

上の式を行列表記すると次のようになる。
 $Zx = scale * b$ を解く。ただし、 Z は

$$Z = \begin{pmatrix} \text{kron}(I_n, A) - \text{kron}(B', I_m) \\ \text{kron}(I_n, D) - \text{kron}(E', I_m) \end{pmatrix}$$

I_k はサイズが k の単位行列、 X' は X の転置/共役転置行列である。 $\text{kron}(X, Y)$ は、行列 X と Y の Kronecker 積である。

$trans = 'T'$ (実数型) または $trans = 'C'$ (複素数型) の場合、ルーチン ?tgsyl は、転置/共役転置式 $Z'y = scale * b$ を解く。これは、次の R と L を解くのと等価である。

$$A' R + D' L = scale * C$$

$$R B' + L E' = scale * (-F)$$

このケース (stgsyl/dtgsyl の場合は $trans = 'T'$ 、ctgsyl/ztgsyl の場合は $trans = 'C'$) を使用して、 $Dif[(A, D), (B, E)]$ の 1- ノルムベースの推定値、すなわち行列ペア (A, D) と (B, E) の隔たりを [slacon/clacon](#) を使用して計算する。

$ijob \geq 1$ の場合、?tgsyl は、 $Dif[(A, D), (B, E)]$ の Frobenius ノルムベースの推定値を計算する。すなわち、 Z の最小特異値の逆数に対する下限値の逆数である。これはレベル 3 BLAS アルゴリズムである。

入力パラメーター

<i>trans</i>	CHARACTER*1. 'N'、'T'、または 'C' でなければならない。 <i>trans</i> = 'N' の場合、汎用シルベスター式を解く。 <i>trans</i> = 'T' の場合、「転置」式を解く (実数型の場合のみ)。 <i>trans</i> = 'C' の場合、「共役転置」式を解く (複素数型の場合のみ)。
<i>ijob</i>	INTEGER。実行する機能を指定する。 <i>ijob</i> = 0 の場合、汎用シルベスター式のみ解く。 <i>ijob</i> = 1 の場合、 <i>ijob</i> = 0 と <i>ijob</i> = 3 を組み合わせた機能を実行する。 <i>ijob</i> = 2 の場合、 <i>ijob</i> = 0 と <i>ijob</i> = 4 を組み合わせた機能を実行する。 <i>ijob</i> = 3 の場合、 $Dif[(A, D), (B, E)]$ の推定値のみ求める (先読み法を使用する)。 <i>ijob</i> = 4 の場合、 $Dif[(A, D), (B, E)]$ の推定値のみ求める (部分式に対して ?gecon を使用する)。 <i>trans</i> = 'T' または 'C' の場合、 <i>ijob</i> は参照されない。
<i>m</i>	INTEGER。 行列 <i>A</i> と <i>D</i> の次数、行列 <i>C</i> 、 <i>F</i> 、 <i>R</i> 、および <i>L</i> の行のサイズ。
<i>n</i>	INTEGER。 行列 <i>B</i> と <i>E</i> の次数、行列 <i>C</i> 、 <i>F</i> 、 <i>R</i> 、および <i>L</i> の列のサイズ。
<i>a, b, c, d, e, f, work</i>	REAL (stgsyl の場合) DOUBLE PRECISION (dtgsyl の場合) COMPLEX (ctgsyl の場合) DOUBLE COMPLEX (ztgsyl の場合)。

配列:

a(*lda*,*) 行列 *A* (実数型の場合は上準三角行列、複素数型の場合は上三角行列) を格納する。

a の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

b(*ldb*,*) 行列 *B* (実数型の場合は上準三角行列、複素数型の場合は上三角行列) を格納する。

b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

c(*ldc*,*) 汎用シルベスター式の最初の行列式の右辺を格納する (*trans* で指定)。

c の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

d(*ldd*,*) 上三角行列 *D* を格納する。

d の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$e(lde, *)$ 上三角行列 E を格納する。
 e の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$f(ldf, *)$ 汎用シルベスター式の 2 番目の行列式の右辺を格納する ($trans$ で指定)。
 f の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。 $ijob = 0$ の場合、 $work$ は参照されない。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

ldb INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。

ldc INTEGER。 c の第 1 次元。 $\max(1, m)$ 以上。

ldd INTEGER。 d の第 1 次元。 $\max(1, m)$ 以上。

lde INTEGER。 e の第 1 次元。 $\max(1, n)$ 以上。

ldf INTEGER。 f の第 1 次元。 $\max(1, m)$ 以上。

$lwork$ INTEGER。 配列 $work$ の次元。 $lwork \geq 1$ 。 $ijob = 1$ または 2 で、 $trans = 'N'$ の場合、 $lwork \geq 2mn$ 。
 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$iwork$ INTEGER。 ワークスペース配列、次元は $(m+n+6)$ 以上 (実数型の場合) または $(m+n+2)$ 以上 (複素数型の場合)。
 $ijob = 0$ の場合、 $iwork$ は参照されない。

出力パラメーター

c $ijob = 0, 1, 2$ の場合、解 R で上書きされる。
 $ijob = 3$ または 4 で $trans = 'N'$ の場合、 c に R (Dif 推定値の計算中に求められた解) が格納される。

f $ijob = 0, 1, 2$ の場合、解 L で上書きされる。
 $ijob = 3$ または 4 および $trans = 'N'$ の場合、 f に L (Dif 推定値の計算中に求められた解) が格納される。

dif REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)。
終了時に、 dif は、Dif 関数の逆数の下限の逆数に設定される。
つまり、 dif は、 $Dif[(A,D), (B,E)] = \sigma_{\min}(Z)$ の上限に設定される。
ここで、 Z は (2)。
 $ijob = 0$ 、または $trans = 'T'$ (実数型) または $trans = 'C'$ (複素数型) の場合、 dif は設定されない。

$scale$ REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)。
終了時に、 $scale$ には、汎用シルベスター式のスケール係数が格納される。 $0 < scale < 1$ の場合、 c と f にはそれぞれ、わずかに摂動する系の解 R と L が入るが、入力行列 A, B, D 、および E

は変更されない。 $scale = 0$ の場合、 c と f にはそれぞれ、 $C = F = 0$ の均一な系の解 R と L が入る。通常は $scale = 1$ である。

<code>work(1)</code>	$ijob \neq 0$ かつ $info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が <code>work(1)</code> に格納される。これ以降の実行には、この <code>lwork</code> の値を使用する。
<code>info</code>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info > 0$ の場合、 (A, D) と (B, E) は、同じか近い固有値を持つ。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `tgsyl` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (m, m) の行列 A を格納する。
<code>b</code>	サイズ (n, n) の行列 B を格納する。
<code>c</code>	サイズ (m, n) の行列 C を格納する。
<code>d</code>	サイズ (m, m) の行列 D を格納する。
<code>e</code>	サイズ (n, n) の行列 E を格納する。
<code>f</code>	サイズ (m, n) の行列 F を格納する。
<code>ijob</code>	0、1、2、3、または 4 でなければならない。デフォルト値は 0。
<code>trans</code>	'N' または 'T' でなければならない。デフォルト値は 'N'。

?tgsna

汎用実 Schur 標準形の行列ペアに関して、指定した固有値と固有ベクトルの条件数の逆数を見積もる。

構文

Fortran 77:

```
call stgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
           ldvr, s, dif, mm, m, work, lwork, iwork, info)
call dtgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
           ldvr, s, dif, mm, m, work, lwork, iwork, info)
call ctgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
           ldvr, s, dif, mm, m, work, lwork, iwork, info)
call ztgsna(job, howmny, select, n, a, lda, b, ldb, vl, ldvl, vr,
           ldvr, s, dif, mm, m, work, lwork, iwork, info)
```

Fortran 95:

```
call tgsna(a, b [,s] [,dif] [,vl] [,vr] [,select] [,m] [,info])
```

説明

実数型の場合 (stgsna/dtgsna の場合) は、汎用実 Schur 標準形の行列ペア (A, B) 、または直交行列 Q と Z を持つ任意の行列ペア $(QA Z^T, QB Z^T)$ において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。

(A, B) は、汎用実 Schur 形式 (sgges/dgges から返される形式) でなければならない。すなわち、 A は 1×1 と 2×2 の対角ブロックを持つブロック上三角行列、 B は上三角である。

複素数型の場合 (ctgsna/ztgsna の場合) は、行列ペア (A, B) において、指定された固有値と固有ベクトルの条件数の逆数を見積もる。 (A, B) は汎用 Schur 標準形でなければならない。すなわち、 A と B はどちらも上三角行列でなければならない。

入力パラメーター

<i>job</i>	CHARACTER*1。固有値と固有ベクトルに対して条件数の逆数を計算するかどうかを指定する。 'E'、'V'、または 'B' のいずれかでなければならない。 <i>job</i> ='E' の場合、固有値のみ (<i>s</i> を計算する)。 <i>job</i> ='V' の場合、固有ベクトルのみ (<i>dif</i> を計算する)。 <i>job</i> ='B' の場合、固有値と固有ベクトルの両方 (<i>s</i> と <i>dif</i> を計算する)。
<i>howmny</i>	CHARACTER*1。'A' または 'S' でなければならない。 <i>howmny</i> ='A' の場合、すべての固有ペアに対して条件数を計算する。 <i>howmny</i> ='S' の場合、論理配列 <i>select</i> で指定した固有ペアに対してのみ条件数を計算する。
<i>select</i>	LOGICAL。 配列、次元は $\max(1, n)$ 以上。 <i>howmny</i> ='S' の場合、 <i>select</i> は、条件数を計算する固有ペアを指定する。 <i>howmny</i> ='A' の場合、 <i>select</i> は参照されない。 実数型の場合: 実固有値 ω_j に対応する固有ペアの条件数を選択するには、 <i>select</i> (<i>j</i>) を .TRUE. に設定しなければならない。複素共役固有値ペア ω_j と ω_{j+1} の条件数を選択するには、 <i>select</i> (<i>j</i>) と <i>select</i> (<i>j+1</i>) のどちらかを .TRUE. に設定しなければならない。 複素数型の場合: <i>j</i> 番目の固有値と固有ベクトルの条件数を選択するには、 <i>select</i> (<i>j</i>) を .TRUE. に設定しなければならない。
<i>n</i>	INTEGER。正方行列ペア (A, B) の次数 ($n \geq 0$)。
<i>a, b, vl, vr, work</i>	REAL (stgsna の場合) DOUBLE PRECISION (dtgsna の場合) COMPLEX (ctgsna の場合) DOUBLE COMPLEX (ztgsna の場合)。

配列:

$a(lda,*)$ 行列ペア (A, B) における上準三角 (実数型の場合) または上三角 (複素数型の場合) 行列 A を格納する。

a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(l db,*)$ 行列ペア (A, B) における上三角行列 B を格納する。

b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$job='E'$ または $'B'$ の場合、 $v1(ldv1,*)$ には、 $howmny$ と $select$ で指定した固有ペアに対応する、 (A, B) の左固有ベクトルが格納されていなければならない。この固有ベクトルは、 $?tgevc$ で返される通りに、 $v1$ の連続した列に格納されていなければならない。

$job='V'$ の場合、 $v1$ は参照されない。

$v1$ の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$job='E'$ または $'B'$ の場合、 $vr(ldvr,*)$ には、 $howmny$ と $select$ で指定した固有ペアに対応する、 (A, B) の右固有ベクトルが格納されていなければならない。

この固有ベクトルは、 $?tgevc$ で返される通りに、 vr の連続した列に格納されていなければならない。

$job='V'$ の場合、 vr は参照されない。

vr の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。 $job='E'$ の場合、 $work$ は参照されない。

lda INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。

ldb INTEGER。 b の第 1 次元。 $\max(1, n)$ 以上。

$ldv1$ INTEGER。 $v1$ の第 1 次元。 $ldv1 \geq 1$ 。
 $job='E'$ または $'B'$ の場合、 $ldv1 \geq \max(1, n)$ 。

$ldvr$ INTEGER。 vr の第 1 次元。 $ldvr \geq 1$ 。
 $job='E'$ または $'B'$ の場合、 $ldvr \geq \max(1, n)$ 。

mm INTEGER。 配列 s と dif の成分の個数 ($mm \geq m$)。

$lwork$ INTEGER。 配列 $work$ の次元。

実数型の場合:

$lwork \geq n$ 。

$job='V'$ または $'B'$ の場合、 $lwork \geq 2n(n+2)+16$ 。

複素数型の場合:

$lwork \geq 1$ 。

$job='V'$ または $'B'$ の場合、 $lwork \geq 2n^2$ 。

$lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$iwork$ INTEGER。 ワークスペース配列、次元は $(n+6)$ 以上 (実数型の場合) または $(n+2)$ 以上 (複素数型の場合)。

$ijob='E'$ の場合、 $iwork$ は参照されない。

出力パラメーター

<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は (<i>mm</i>)。 <i>job</i> ='E' または 'B' の場合、選択した固有値の条件数の逆数が、配列の連続成分に格納される。 <i>job</i> ='V' の場合、 <i>s</i> は参照されない。 実数型の場合： 固有値の複素共役ペアに対して、 <i>s</i> の 2 つの連続成分が同じ値に設定される。したがって、 <i>s</i> (<i>j</i>)、 <i>dif</i> (<i>j</i>)、 <i>vl</i> と <i>vr</i> の <i>j</i> 番目の列が、同じ固有ペアに対応することになる (ただし、すべての固有ペアが選択される場合を除き、 <i>j</i> 番目の固有ペアに対応することにはならない)。
<i>dif</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は (<i>mm</i>)。 <i>job</i> ='V' または 'B' の場合、選択した固有ベクトルの条件数の逆数の推定値が、配列の連続成分に格納される。 <i>dif</i> (<i>j</i>) を計算するための固有値の順序変更ができなければ、 <i>dif</i> (<i>j</i>) が 0 に設定される。これは、True 値が非常に小さい場合にのみ起こる。 <i>job</i> ='E' の場合、 <i>dif</i> は参照されない。 実数型の場合： 1 つの複素固有ベクトルに対して、 <i>dif</i> の 2 つの連続成分が同じ値に設定される。 複素数型の場合： <i>select</i> で指定された 1 つの固有値 / 固有ベクトルに対して、 <i>Difl</i> の Frobenius ノルムベースの推定値が <i>dif</i> に格納される。
<i>m</i>	INTEGER。配列 <i>s</i> と <i>dif</i> において、指定された条件数を格納するのに使用した成分の個数。選択された 1 つの固有値に対して、1 つの成分が使用される。 <i>howmny</i> ='A' の場合、 <i>m</i> は <i>n</i> に設定される。
<i>work</i> (1)	<i>job</i> ≠ 'E' かつ <i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work</i> (1) に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgsna* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n, n</i>) の行列 <i>B</i> を格納する。
<i>s</i>	長さ (<i>mm</i>) のベクトルを格納する。

<i>dif</i>	長さ (<i>mm</i>) のベクトルを格納する。
<i>vl</i>	サイズ (<i>n,mm</i>) の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ (<i>n,mm</i>) の行列 <i>VR</i> を格納する。
<i>select</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>howmny</i>	引数 <i>select</i> の存在に基づいて以下のように復元される。 <i>howmny</i> = 'S' (<i>select</i> が存在する場合)、 <i>howmny</i> = 'A' (<i>select</i> が省略された場合)。
<i>job</i>	引数 <i>s</i> および <i>dif</i> の存在に基づいて以下のように復元される。 <i>job</i> = 'B' (<i>s</i> と <i>dif</i> の両方が存在する場合)、 <i>job</i> = 'L' (<i>s</i> が存在し、 <i>dif</i> が省略された場合)、 <i>job</i> = 'R' (<i>s</i> が省略され、 <i>dif</i> が存在する場合)。 <i>s</i> と <i>dif</i> の両方が省略された場合、エラー条件がセットされる。

汎用特異値分解

このセクションでは、2つの行列 A と B に対して次のような汎用特異値分解 (GSVD) を行う LAPACK 計算ルーチンについて説明する。

$$U^H A Q = D_1 * (0 \ R),$$

$$V^H B Q = D_2 * (0 \ R),$$

U 、 V 、および Q は直交 / ユニタリー行列、 R は非特異上三角行列、 D_1 、 D_2 は「対角」行列 (詳しい構造については、各ルーチンの説明を参照) である。

表 4-7 に、行列の特異値分解を実行するための LAPACK ルーチン (Fortran-77 インターフェイス) を示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない (「[ルーチン命名規則](#)」を参照)。

表 4-7 汎用特異値分解のための計算ルーチン

ルーチン名	機能
?qgsvp	汎用 SVD のための予備的な分解を行う。
?tgsja	2つの上三角行列または台形行列の汎用 SVD を求める。

一般三角行列ペアの GSVD を得るには、上記のルーチンを使用することも、ドライバールーチン [?qgsvd](#) を使用することもできる。

?qgsvp

汎用 SVD のための予備的な分解を行う。

構文

Fortran 77:

```
call sggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info)
call dggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, tau, work, info)
call cggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work, info)
call zggsvp(jobu, jobv, jobq, m, p, n, a, lda, b, ldb, tola, tolb,
            k, l, u, ldu, v, ldv, q, ldq, iwork, rwork, tau, work, info)
```

Fortran 95:

```
call ggsvp(a, b, tola, tolb [,k] [,l] [,u] [,v] [,q] [,info])
```

説明

このルーチンは、次のような直交行列 U 、 V 、および Q を求める。

$$U^H A Q = \begin{matrix} & n-k-1 & k & 1 \\ & k & & \\ & 1 & & \\ & m-k-1 & & \end{matrix} \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \\ 0 & 0 & 0 \end{pmatrix}, \quad (m-k-1 \geq 0 \text{ の場合})$$

$$= \begin{matrix} & n-k-1 & k & 1 \\ & k & & \\ & m-k & & \end{matrix} \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix}, \quad (m-k-1 < 0 \text{ の場合})$$

$$V^H B Q = \begin{matrix} & n-k-1 & k & 1 \\ & 1 & & \\ & p-1 & & \end{matrix} \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix}$$

$k \times k$ の行列 A_{12} と 1×1 の行列 B_{13} は、非特異上三角行列である。 A_{23} は、 $m-k-1 \geq 0$ の場合は 1×1 の上三角行列、それ以外の場合、 A_{23} は $(m-k) \times 1$ の上台形行列である。 $k+1$ の和は、 $(m+p) \times n$ の行列 $(A^H, B^H)^H$ の有効階数に等しい。

この分解は、汎用特異値分解 (GSVD) の予備ステップとして行われる。サブルーチン [?sggsvd](#) を参照。

入力パラメーター

<i>jobu</i>	CHARACTER*1. 'U' または 'N' でなければならない。 <i>jobu</i> = 'U' の場合、直交 / ユニタリー行列 U が計算される。 <i>jobu</i> = 'N' の場合、 U は計算されない。
<i>jobv</i>	CHARACTER*1. 'V' または 'N' でなければならない。 <i>jobv</i> = 'V' の場合、直交 / ユニタリー行列 V が計算される。 <i>jobv</i> = 'N' の場合、 V は計算されない。
<i>jobq</i>	CHARACTER*1. 'Q' または 'N' でなければならない。 <i>jobq</i> = 'Q' の場合、直交 / ユニタリー行列 Q が計算される。 <i>jobq</i> = 'N' の場合、 Q は計算されない。
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>p</i>	INTEGER。行列 B の行数 ($p \geq 0$)。
<i>n</i>	INTEGER。行列 A と B の列数 ($n \geq 0$)。
<i>a, b, tau, work</i>	REAL (sggsvp の場合) DOUBLE PRECISION (dggsvp の場合) COMPLEX (cggsvp の場合) DOUBLE COMPLEX (zggsvp の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、 $m \times n$ の行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(ldb, *)$ には、 $p \times n$ の行列 B を格納する。
 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$tau(*)$ は、ワークスペース配列である。 tau の次元は、 $\max(1, n)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(1, 3n, m, p)$ 以上でなければならない。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

ldb INTEGER。 b の第 1 次元。 $\max(1, p)$ 以上。

$tola, tolb$ REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)。
 $tola$ と $tolb$ は、行列 B と部分ブロック A の有効階数を決定するためのしきい値である。一般に、次のように設定される。
 $tola = \max(m, n) * \|A\| * \text{MACHEPS}$
 $tolb = \max(p, n) * \|B\| * \text{MACHEPS}$
 $tola$ と $tolb$ のサイズが、分解の後退誤差のサイズに影響する可能性がある。

ldu INTEGER。出力配列 u の第 1 次元。 $jobu = 'U'$ の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。

ldv INTEGER。出力配列 v の第 1 次元。 $jobv = 'V'$ の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。

ldq INTEGER。出力配列 q の第 1 次元。 $jobq = 'Q'$ の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。

$iwork$ INTEGER。ワークスペース配列、次元は $\max(1, n)$ 以上。

$rwork$ REAL (cggsvp の場合)
 DOUBLE PRECISION (zggsvp の場合)。
 ワークスペース配列、次元は $\max(1, 2n)$ 以上。複素数型でのみ使用される。

出力パラメーター

a 説明で示した三角 (台形) 行列で上書きされる。

b 説明で示した三角行列で上書きされる。

k, l INTEGER。
 終了時に、 k と l に部分ブロックのサイズが格納される。
 $k+1$ は、 $(A^H, B^H)^H$ の有効階数に一致する。

u, v, q REAL (sggsvp の場合)
 DOUBLE PRECISION (dggsvp の場合)
 COMPLEX (cggsvp の場合)
 DOUBLE COMPLEX (zggsvp の場合)。
 配列：
 $jobu = 'U'$ の場合、 $u(ldu, *)$ に、直交 / ユニタリー行列 U が格納される。
 u の第 2 次元は、 $\max(1, m)$ 以上でなければならない。
 $jobu = 'N'$ の場合、 u は参照されない。

$jobv = 'V'$ の場合、 $v(ldv, *)$ に、直交 / ユニタリー行列 V が格納される。

v の第 2 次元は、 $\max(1, m)$ 以上でなければならない。

$jobv = 'N'$ の場合、 v は参照されない。

$jobq = 'Q'$ の場合、 $q(ldq, *)$ に、直交 / ユニタリー行列 Q が格納される。

q の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$jobq = 'N'$ の場合、 q は参照されない。

info

INTEGER.

$info = 0$ の場合、実行は正常に終了したことを示す。

$info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggsvp` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>b</i>	サイズ (p, n) の行列 B を格納する。
<i>u</i>	サイズ (m, m) の行列 U を格納する。
<i>v</i>	サイズ (p, m) の行列 V を格納する。
<i>q</i>	サイズ (n, n) の行列 Q を格納する。
<i>jobu</i>	引数 u の存在に基づいて以下のように復元される。 $jobu = 'U'$ (u が存在する場合)、 $jobu = 'N'$ (u が省略された場合)。
<i>jobv</i>	引数 v の存在に基づいて以下のように復元される。 $jobv = 'V'$ (v が存在する場合)、 $jobv = 'N'$ (v が省略された場合)。
<i>jobq</i>	引数 q の存在に基づいて以下のように復元される。 $jobq = 'Q'$ (q が存在する場合)、 $jobq = 'N'$ (q が省略された場合)。

?tgsja

2 つの上三角行列または台形行列の汎用 SVD を計算する。

構文

Fortran 77:

```
call stgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call dtgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call ctgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
call ztgsja(jobu, jobv, jobq, m, p, n, k, l, a, lda, b, ldb, tola,
            tolb, alpha, beta, u, ldu, v, ldv, q, ldq, work, ncycle, info)
```

Fortran 95:

```
call tgsja(a, b, tola, tolb, k, l [,u] [,v] [,q] [,jobu] [,jobv] [,jobq]
          [,alpha] [,beta] [,ncycle] [,info])
```

説明

このルーチンは、2 つの実 / 複素上三角行列 (または台形行列) A と B に対して、汎用特異値分解 (GSVD) を計算する。呼び出し時に、行列 A と B は次の形式であるとみなされる。この形式は、 $m \times n$ の一般行列 A と $p \times n$ の一般行列 B を、予備的サブルーチン [?qgsvp](#) で前処理して得られる。

$$A = \begin{matrix} & n-k-l & k & l \\ & k & & \\ & l & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} & \\ m-k-l & m-k-l & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} & \end{matrix}, \quad (m-k-l \geq 0 \text{ の場合})$$

$$= \begin{matrix} & n-k-l & k & l \\ & k & & \\ = & m-k & \begin{pmatrix} 0 & A_{12} & A_{13} \\ 0 & 0 & A_{23} \end{pmatrix} & \end{matrix}, \quad (m-k-l < 0 \text{ の場合})$$

$$B = \begin{matrix} & n-k-l & k & l \\ & l & \begin{pmatrix} 0 & 0 & B_{13} \\ 0 & 0 & 0 \end{pmatrix} & \\ p-l & p-l & \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} & \end{matrix}$$

$k \times k$ の行列 A_{12} と $l \times l$ の行列 B_{13} は、非特異上三角行列である。 A_{23} は、 $m-k-l \geq 0$ の場合は $l \times l$ の上三角行列、それ以外の場合、 A_{23} は $(m-k) \times l$ の上台形行列である。

終了時は、

$$U^H A Q = D_1 * (0 \ R), \quad V^H B Q = D_2 * (0 \ R)$$

U 、 V 、および Q は直交/ユニタリー行列、 R は非特異上三角行列、 D_1 と D_2 は次の構造を持つ「対角」行列である。

$m-k-l \geq 0$ の場合、

$$D_1 = \begin{matrix} & k & l \\ & I & 0 \\ l & 0 & C \\ m-k-l & 0 & 0 \end{matrix}$$

$$D_2 = \begin{matrix} & k & l \\ l & 0 & S \\ p-l & 0 & 0 \end{matrix}$$

$$(0 \ R) = \begin{matrix} & n-k-l & k & l \\ k & 0 & R_{11} & R_{12} \\ l & 0 & 0 & R_{22} \end{matrix}$$

ここで、

$$C = \text{diag}(\alpha(k+1), \dots, \alpha(k+l))$$

$$S = \text{diag}(\beta(k+1), \dots, \beta(k+l))$$

$$C^2 + S^2 = I$$

終了時に、 R は $a(1:k+l, n-k-l+1:n)$ に格納される。

$m-k-l < 0$ の場合、

$$D_1 = \begin{matrix} & k & m-k & k+l-m \\ & I & 0 & 0 \\ m-k & 0 & C & 0 \end{matrix}$$

$$D_2 = \begin{matrix} & k & m-k & k+l-m \\ m-k & 0 & S & 0 \\ k+l-m & 0 & 0 & I \\ p-l & 0 & 0 & 0 \end{matrix}$$

$$(0 \ R) = \begin{matrix} & n-k-l & k & m-k & k+l-m \\ & k & \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \\ & m-k & \\ & k+l-m & \end{matrix}$$

ここで、

$$\begin{aligned} C &= \text{diag}(\alpha(k+1), \dots, \alpha(m)), \\ S &= \text{diag}(\beta(k+1), \dots, \beta(m)), \\ C^2 + S^2 &= I \end{aligned}$$

終了時に、 $\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$ は $a(1:m, n-k-l+1:n)$ に格納され、 R_{33} は

$b(m-k+1:l, n+m-k-l+1:n)$ に格納される。

オプションで、直交/ユニタリー変換行列 U 、 V 、および Q を計算できる。これらの行列は明示的に生成することができる。また、入力行列 U_1 、 V_1 、および Q_1 に後から掛けることもできる。

入力パラメーター

<i>jobu</i>	CHARACTER*1。'U'、'I'、または 'N' でなければならない。 <i>jobu</i> ='U' の場合、 <i>u</i> には、呼び出し時に、直交/ユニタリー行列 U_1 が格納されていないなければならない。 <i>jobu</i> ='I' の場合、 <i>u</i> は単位行列に初期化される。 <i>jobu</i> ='N' の場合、 <i>u</i> は計算されない。
<i>jobv</i>	CHARACTER*1。'V'、'I'、または 'N' でなければならない。 <i>jobv</i> ='V' の場合、 <i>v</i> には、呼び出し時に、直交/ユニタリー行列 V_1 が格納されていないなければならない。 <i>jobv</i> ='I' の場合、 <i>v</i> は単位行列に初期化される。 <i>jobv</i> ='N' の場合、 <i>v</i> は計算されない。
<i>jobq</i>	CHARACTER*1。'Q'、'I'、または 'N' でなければならない。 <i>jobq</i> ='Q' の場合、 <i>q</i> には、呼び出し時に、直交/ユニタリー行列 Q_1 が格納されていないなければならない。 <i>jobq</i> ='I' の場合、 <i>q</i> は単位行列に初期化される。 <i>jobq</i> ='N' の場合、 <i>q</i> は計算されない。
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>p</i>	INTEGER。行列 B の行数 ($p \geq 0$)。
<i>n</i>	INTEGER。行列 A と B の列数 ($n \geq 0$)。
<i>k, l</i>	INTEGER。入力行列 A と B の部分ブロックを指定する。?tgsja は、これから GSVD を計算する

$a, b, u, v, q, work$	<p>REAL (stgsja の場合) DOUBLE PRECISION (dtgsja の場合) COMPLEX (ctgsja の場合) DOUBLE COMPLEX (ztgsja の場合)。 配列: $a(lda, *)$ には、$m \times n$ の行列 A を格納する。 a の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $b(lb, *)$ には、$p \times n$ の行列 B を格納する。 b の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $jobu = 'U'$ の場合、$u(ldu, *)$ には、行列 U_I (通常は、?ggsvp から返された直交/ユニタリー行列) が格納されていなければならない。 u の第 2 次元は、$\max(1, m)$ 以上でなければならない。 $jobv = 'V'$ の場合、$v(ldv, *)$ には、行列 V_I (通常は、?ggsvp から返された直交/ユニタリー行列) が格納されていなければならない。 v の第 2 次元は、$\max(1, p)$ 以上でなければならない。 $jobq = 'Q'$ の場合、$q(ldq, *)$ には、行列 Q_I (通常は、?ggsvp から返された直交/ユニタリー行列) が格納されていなければならない。 q の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $work(*)$ は、ワークスペース配列である。$work$ の次元は、$\max(1, 2n)$ 以上でなければならない。</p>
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, p)$ 以上。
ldu	INTEGER。 配列 u の第 1 次元。 $jobu = 'U'$ の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。
ldv	INTEGER。 配列 v の第 1 次元。 $jobv = 'V'$ の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。
ldq	INTEGER。 配列 q の第 1 次元。 $jobq = 'Q'$ の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。
$tola, tolb$	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $tola$ と $tolb$ は、Jacobi-Kogbetliantz 反復法における収束基準である。一般に、?ggsvp で使用した値と同じ値にする。 $tola = \max(m, n) * \ A\ * \text{MACHEPS}$ $tolb = \max(p, n) * \ B\ * \text{MACHEPS}$</p>

出力パラメーター

a	終了時に、 $a(n-k+1:n, 1:\min(k+1, m))$ に三角行列 R または R の一部が格納される。
b	終了時に、必要な場合、 $b(m-k+1:1, n+m-k-1+1:n)$ に R の一部が格納される。

<i>alpha, beta</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>A</i> と <i>B</i> の汎用特異値ペアが、次のように格納される。</p> <p>$\alpha(1:k) = 1$ $\beta(1:k) = 0$</p> <p>$m-k-1 \geq 0$ の場合、 $\alpha(k+1:k+1) = \text{diag}(C)$, $\beta(k+1:k+1) = \text{diag}(S)$</p> <p>$m-k-1 < 0$ の場合、 $\alpha(k+1:m) = C$, $\alpha(m+1:k+1) = 0$ $\beta(k+1:m) = S$, $\beta(m+1:k+1) = 1$</p> <p>$k+1 < n$ の場合、 $\alpha(k+1+1:n) = 0$ $\beta(k+1+1:n) = 0$</p>
<i>u</i>	<p><i>jobu</i> = 'I' の場合、<i>u</i> には、直交 / ユニタリ行列 <i>U</i> が格納される。 <i>jobu</i> = 'U' の場合、<i>u</i> には、積 $U_1 U$ が格納される。 <i>jobu</i> = 'N' の場合、<i>u</i> は参照されない。</p>
<i>v</i>	<p><i>jobv</i> = 'I' の場合、<i>v</i> には、直交 / ユニタリ行列 <i>U</i> が格納される。 <i>jobv</i> = 'V' の場合、<i>v</i> には、積 $V_1 V$ が格納される。 <i>jobv</i> = 'N' の場合、<i>v</i> は参照されない。</p>
<i>q</i>	<p><i>jobq</i> = 'I' の場合、<i>q</i> には、直交 / ユニタリ行列 <i>U</i> が格納される。 <i>jobq</i> = 'Q' の場合、<i>q</i> には、積 $Q_1 Q$ が格納される。 <i>jobq</i> = 'N' の場合、<i>q</i> は参照されない。</p>
<i>ncycle</i>	INTEGER。収束するのに要したサイクル数。
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、この方法では MAXIT サイクル以内に収束しなかったことを示す。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *tgsja* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>p, n</i>) の行列 <i>B</i> を格納する。
<i>u</i>	サイズ (<i>m, m</i>) の行列 <i>U</i> を格納する。
<i>v</i>	サイズ (<i>p, p</i>) の行列 <i>V</i> を格納する。

<i>q</i>	サイズ (n, n) の行列 Q を格納する。
<i>alpha</i>	長さ (n) のベクトルを格納する。
<i>beta</i>	長さ (n) のベクトルを格納する。
<i>jobu</i>	省略された場合、この引数は、引数 u の存在に基づいて以下のように復元される。 <i>jobu</i> = 'U' (u が存在する場合)、 <i>jobu</i> = 'N' (u が省略された場合)。 存在する場合、 <i>jobu</i> は 'I' または 'U' と等しくなければならず、引数 u も存在しなければならない。 <i>jobu</i> が存在し、 u が省略された場合、エラー条件がセットされる。
<i>jobv</i>	省略された場合、この引数は、引数 v の存在に基づいて以下のように復元される。 <i>jobv</i> = 'V' (v が存在する場合)、 <i>jobv</i> = 'N' (v が省略された場合)。 存在する場合、 <i>jobv</i> は 'I' または 'V' と等しくなければならず、引数 v も存在しなければならない。 <i>jobv</i> が存在し、 v が省略された場合、エラー条件がセットされる。
<i>jobq</i>	省略された場合、この引数は、引数 q の存在に基づいて以下のように復元される。 <i>jobq</i> = 'Q' (q が存在する場合)、 <i>jobq</i> = 'N' (q が省略された場合)。 存在する場合、 <i>jobq</i> は 'I' または 'Q' と等しくなければならず、引数 q も存在しなければならない。 <i>jobq</i> が存在し、 q が省略された場合、エラー条件がセットされる。

ドライバールーチン

LAPACK ドライバールーチンを使用すると、ある問題を完全に解くのに、1つのルーチン呼び出すだけで済む。

各ドライバールーチンは一般に、一連の[計算ルーチン](#)を適切に組み合わせて呼び出す。以下の各セクションで、ドライバールーチンについて説明する。

[線形最小二乗 \(LLS\) 問題](#)

[汎用 LLS 問題](#)

[対称固有値問題](#)

[非対称固有値問題](#)

[特異値分解](#)

[汎用対称固有値問題](#)

[汎用非対称固有値問題](#)

線形最小二乗 (LLS) 問題

このセクションでは、線形最小二乗問題を解くための LAPACK ドライバールーチンについて説明する。表 4-8 に、Fortran-77 インターフェイスのルーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-8 **LLS 問題を解くためのドライバールーチン**

ルーチン名	機能
?gels	QR または LQ 因子分解を使用して、最大階数の行列で表される、優決定または劣決定の線形問題を解く。
?gelsy	A の完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を求める。
?gelss	A の特異値分解を使用して、線形最小二乗問題の最小ノルム解を求める。
?gelsd	A の特異値分解と分割統治法を使用して、線形最小二乗問題の最小ノルム解を求める。

?gels

QR または LQ 因子分解を使用して、最大階数の行列で表される、優決定または劣決定の線形問題を解く。

構文

Fortran 77:

```
call sgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call dgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call cgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
call zgels(trans, m, n, nrhs, a, lda, b, ldb, work, lwork, info)
```

Fortran 95:

```
call gels(a, b [,trans] [,info])
```

説明

このルーチンは、 A の QR または LQ 因子分解を使用して、 $m \times n$ の行列 A (またはその転置 / 共役転置行列) で表される、実数 / 複素数の優決定 / 劣決定の線形問題を解く。ただし、 A は最大階数の行列とする。

次のオプションが提供されている。

1. $trans = 'N'$ で、 $m \geq n$ の場合 : 優決定の最小二乗解を求める。

すなわち次の最小二乗問題を解く。

$$\text{minimize } \|b - Ax\|_2$$

2. $trans = 'N'$ で、 $m < n$ の場合 : 劣決定の問題 $AX = B$ の最小ノルム解を求める。

3. $trans = 'T'$ または $'C'$ で、 $m \geq n$ の場合 : 劣決定の問題 $A^H X = B$ の最小ノルム解を求める。

4. $trans = 'T'$ または $'C'$ で、 $m < n$ の場合 : 優決定の最小二乗解を求める。

すなわち次の最小二乗問題を解く。

$$\text{minimize } \|b - A^H x\|_2$$

複数の右辺のベクトル b と解ベクトル x を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$ の右辺の行列 B と $n \times nrh$ の解行列 X の各列に格納される。

入力パラメーター

<i>trans</i>	CHARACTER*1. 'N'、'T'、または 'C' でなければならない。 <i>trans</i> = 'N' の場合、行列 A で表される線形問題。 <i>trans</i> = 'T' の場合、転置行列 A^T で表される線形問題 (実数型の場合のみ)。 <i>trans</i> = 'C' の場合、共役転置行列 A^H で表される線形問題 (複素数型の場合のみ)。
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 A の列数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sgels の場合) DOUBLE PRECISION (dgels の場合) COMPLEX (cgels の場合) DOUBLE COMPLEX (zgels の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、 $m \times n$ の行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、右辺のベクトルが各列に入った行列 B を格納する。 B は、 <i>trans</i> = 'N' の場合は $m \times nrhs$ 、 <i>trans</i> = 'T' または 'C' の場合は $n \times nrhs$ 。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。

lwork INTEGER。配列 *work* のサイズ。 $\min(m, n) + \max(1, m, n, nrhs)$ 以上でなければならない。
lwork = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。
lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

a 終了時に、因子分解データで次のように上書きされる。
 $m \geq n$ の場合、行列 *A* の *QR* 因子分解の詳細 ([?qgeqrf](#) から返された値) が、配列 *a* に格納される。 $m < n$ の場合、行列 *A* の *LQ* 因子分解の詳細 ([?qgelqf](#) から返された値) が、配列 *a* に格納される。

b 各列が解ベクトルで上書きされる。
 $trans = 'N'$ で $m \geq n$ の場合、*b* の 1 ~ *n* 行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の *n*+1 ~ *m* の成分の二乗和で与えられる。
 $trans = 'N'$ で $m < n$ の場合、*b* の 1 ~ *n* 行に、最小ノルム解のベクトルが格納される。
 $trans = 'T'$ または '*C*' で $m \geq n$ の場合、*b* の 1 ~ *m* 行に、最小ノルム解のベクトルが格納される。
 $trans = 'T'$ または '*C*' で $m < n$ の場合、*b* の 1 ~ *m* 行に、最小ノルム解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の *m*+1 ~ *n* の成分の二乗和で与えられる。

work(1) *info* = 0 の場合、ルーチンの終了時に、最適なパフォーマンスに必要な最小限の *lwork* の値が *work*(1) に出力される。これ以降の実行には、この *lwork* の値を使用する。

info INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gels* のインターフェイスの詳細を以下に示す。

a サイズ (*m*, *n*) の行列 *A* を格納する。

b サイズ $\max(m, n) \times nrhs$ の行列を格納する。
 $trans = 'N'$ の場合、呼び出し時に、*b* のサイズは $m \times nrhs$ である。
 $trans = 'T'$ の場合、呼び出し時に、*b* のサイズは $n \times nrhs$ である。

trans 'N' または 'T' でなければならない。デフォルト値は 'N'。

アプリケーション・ノート

パフォーマンスを改善するには、 $lwork = \min(m, n) + \max(1, m, n, nrhs) * blocksize$ に設定する。*blocksize* は、ブロック化アルゴリズムの最適なパフォーマンスを得るために必要な値で、マシンによって異なる (通常は 16 ~ 64)。

必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

?gelsy

A の完全直交因子分解を使用して、線形最小二乗問題の最小ノルム解を求める。

構文

Fortran 77:

```
call sgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
           lwork, info)
call dgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
           lwork, info)
call cgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
           lwork, rwork, info)
call zgelsy(m, n, nrhs, a, lda, b, ldb, jpvt, rcond, rank, work,
           lwork, rwork, info)
```

Fortran 95:

```
call gelsy(a, b [,rank] [,jpvt] [,rcond] [,info])
```

説明

このルーチンは、実数 / 複素数の線形最小二乗問題、すなわち

$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、*A* の完全直交因子分解を使用して求める。*A* は $m \times n$ の行列で、階数不足の可能性がある。

複数の右辺のベクトル *b* と解ベクトル *x* を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$ の右辺の行列 *B* と $n \times nrhs$ の解行列 *X* の各列に格納される。

このルーチンは、まず列ピボット演算を用いた *QR* 因子分解を行う。

$$AP = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

R_{11} は最大の先頭部分行列で、条件数は $1/rcond$ 以下と見積もられる。 R_{11} の次数 (*rank*) が、 A の有効階数である。

次に、 R_{22} は無視できるとみなし、 R_{12} を右からの直交/ユニタリー変換でゼロにして、次のような完全直交因子分解を得る。

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z$$

最小ノルム解は次のようになる。

$$x = PZ^H \begin{pmatrix} T_{11}^{-1} Q_1^H b \\ 0 \end{pmatrix}$$

Q_1 は、 Q の先頭 *rank* 列である。このルーチンは、元の `?gelsx` と基本的に同じであるが、次の点が異なる。

- サブルーチン `?gqpf` ではなく、サブルーチン `?gqpf3` を呼び出している。このサブルーチンは、ピボット演算を用いた QR 因子分解の BLAS-3 版である。
- 行列 B (右辺) が BLAS-3 で更新される。
- 行列 B (右辺) の置換が、より高速かつ単純になっている。

入力パラメーター

<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 A の列数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
<i>a, b, work</i>	REAL (sgelsy の場合) DOUBLE PRECISION (dgelsy の場合) COMPLEX (cgelsy の場合) DOUBLE COMPLEX (zgelsy の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、 $m \times n$ の行列 A を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、 $m \times nrhs$ の右辺の行列 B を格納する。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
<i>jpvt</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 呼び出し時に、 <i>jpvt</i> (<i>i</i>) $\neq 0$ の場合、 A の <i>i</i> 番目の列が AP の先頭に置換される。それ以外の場合、 A の <i>i</i> 番目の列がフリー列になる。
<i>rcond</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。

rcond は、*A* の有効階数を決定するのに使用する。
有効階数は、*A* のピボット演算を用いた *QR* 因子分解で得られた、先頭の最大三角部分行列 R_{11} の次数で定義される。その条件数の推定値 $< 1/rcond$ である。

<i>lwork</i>	INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (<i>cgelsy</i> の場合) DOUBLE PRECISION (<i>zgelsy</i> の場合)。 ワークスペース配列、次元は $\max(1, 2n)$ 以上。 複素数型でのみ使用される。

出力パラメーター

<i>a</i>	終了時に、 <i>A</i> の完全直交因子分解の詳細で上書きされる。
<i>b</i>	$n \times nrhs$ の解行列 <i>X</i> で上書きされる。
<i>jpvt</i>	終了時に、 <i>jpvt</i> (<i>i</i>) = <i>k</i> の場合、 <i>AP</i> の <i>i</i> 番目の列が <i>A</i> の <i>k</i> 番目の列である。
<i>rank</i>	INTEGER。 <i>A</i> の有効階数、すなわち部分行列 R_{11} の次数。これは、 <i>A</i> の完全直交因子分解における部分行列 T_{11} の次数と同じである。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gelsy* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $\max(m, n) \times nrhs$ の行列を格納する。 呼び出し時に、 $m \times nrhs$ の右辺の行列 <i>B</i> を格納する。 終了時に、 $n \times nrhs$ の解の行列 <i>X</i> によって上書きされる。
<i>jpvt</i>	長さ (<i>n</i>) のベクトルを格納する。この成分のデフォルト値は、 <i>jpvt</i> (<i>i</i>) である。
<i>rcond</i>	この成分のデフォルト値は $rcond = 100 * EPSILON(1.0_WP)$ である。

アプリケーション・ノート

実数型の場合：

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$$lwork \geq \max(mn+3n+1, 2*mn + nrhs)$$

ただし、 $mn = \min(m, n)$ である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$$lwork \geq \max(mn+2n+nb*(n+1), 2*mn+nb*nrhs),$$

ここで、 nb は、[ilaenv](#) がルーチン `sgeqp3/dgeqp3`、`stzrzf/dtzrzf`、`stzrqf/dtzrqf`、`sormqr/dormqr`、`sormrz/dormrz` に対して返したブロックサイズの上限值である。

複素数型の場合：

ブロック化アルゴリズムを使用しない場合は、次のワークスペースが必要である。

$$lwork \geq mn + \max(2*mn, n+1, mn + nrhs)$$

ただし、 $mn = \min(m, n)$ である。

ブロック化アルゴリズムを使用する場合は、次のようになる。

$$lwork \geq mn + \max(2*mn, nb*(n+1), mn+mn*nb, mn+nb*nrhs)$$

ここで、 nb は、[ilaenv](#) がルーチン `cgeqp3/zgeqp3`、`ctzrzf/ztzrzf`、`ctzrqf/ztzrqf`、`cunmqr/zunmqr`、`cunmrz/zunmrz` に対して返したブロックサイズの上限值である。

?gelss

A の特異値分解を使用して、線形最小乗問題の最小ノルム解を求める。

構文

Fortran 77:

```
call sgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, info)
call dgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, info)
call cgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, rwork, info)
call zgelss(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, rwork, info)
```

Fortran 95:

```
call gelss(a, b [,rank] [,s] [,rcond] [,info])
```

説明

このルーチンは、実数の線形最小二乗問題、すなわち

$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、 A の特異値分解 (SVD) を使用して求める。 A は $m \times n$ の行列で、階数不足の可能性はある。

複数の右辺のベクトル b と解ベクトル x を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$ の右辺の行列 B と $n \times nrhs$ の解行列 X の各列に格納される。

A の有効階数は、最大特異値の $rcond$ 倍より小さい特異値をゼロとして扱うことで決定される。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。行列 A の列数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。
$a, b, work$	REAL (sgelss の場合) DOUBLE PRECISION (dgelss の場合) COMPLEX (cgelss の場合) DOUBLE COMPLEX (zgelss の場合)。 配列: $a(lda, *)$ には、 $m \times n$ の行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b ldb, *)$ には、 $m \times nrhs$ の右辺の行列 B を格納する。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
$rcond$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $rcond$ は、 A の有効階数を決定するのに使用する。 特異値 $s(i) \leq rcond * s(1)$ をゼロとして扱う。 $rcond < 0$ の場合、代わりにマシン精度を使用する。
$lwork$	INTEGER。配列 $work$ のサイズ、 $lwork \geq 1$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。
$rwork$	REAL (cgelss の場合) DOUBLE PRECISION (zgelss の場合)。 複素数型でのみ使用されるワークスペース配列である。 次元は $\max(1, 5 * \min(m, n))$ 以上。

出力パラメーター

a	終了時に、 A の先頭から $\min(m, n)$ 個の各行に、右特異ベクトルが格納される。
b	$n \times nrhs$ の解行列 X で上書きされる。

$m \geq n$ で $rank = n$ の場合、 i 番目の列の解の二乗和の誤差は、その列の $n+1:m$ の成分の二乗和で与えられる。

s	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 A の特異値が降順で格納される。 A の 2-ノルムの条件数は、 $k_2(A) = s(1) / s(\min(m, n))$ 。
$rank$	INTEGER。 A の有効階数、すなわち、 $rcond * s(1)$ より大きい特異値の個数。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、SVD の計算アルゴリズムが収束に失敗したことを示す。 i は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gelss` のインターフェイスの詳細を以下に示す。

a	サイズ (m, n) の行列 A を格納する。
b	サイズ $\max(m, n) \times nrhs$ の行列を格納する。 呼び出し時に、 $m \times nrhs$ の右辺の行列 B を格納する。 終了時に、 $n \times nrhs$ の解の行列 X によって上書きされる。
s	長さ $\min(m, n)$ のベクトルを格納する。
$rcond$	この成分のデフォルト値は $rcond = 100 * \text{EPSILON}(1.0_WP)$ である。

アプリケーション・ノート

実数型の場合:

$$lwork \geq 3 * \min(m, n) + \max(2 * \min(m, n), \max(m, n), nrhs)$$

複素数型の場合:

$$lwork \geq 2 * \min(m, n) + \max(m, n, nrhs)$$

パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

?gelsd

A の特異値分解と分割統治法を使用して、線形最小二乗問題の最小ノルム解を求める。

構文

Fortran 77:

```
call sgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, iwork, info)
call dgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, iwork, info)
call cgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, rwork, iwork, info)
call zgelsd(m, n, nrhs, a, lda, b, ldb, s, rcond, rank, work,
           lwork, rwork, iwork, info)
```

Fortran 95:

```
call gelsd(a, b [,rank] [,s] [,rcond] [,info])
```

説明

このルーチンは、実数の線形最小二乗問題、すなわち

$$\text{minimize } \|b - Ax\|_2$$

の最小ノルム解を、 A の特異値分解 (SVD) を使用して求める。 A は $m \times n$ の行列で、階数不足の可能性はある。

複数の右辺のベクトル b と解ベクトル x を、一度の呼び出しで処理できる。これらのベクトルは、 $m \times nrhs$ の右辺の行列 B と $n \times nrhs$ の解行列 X の各列に格納される。

この問題は、次の 3 ステップで解く。

1. Householder 変換により係数行列 A を二重対角形式に縮退し、元の問題を「二重対角形式の最小二乗問題 (BLS)」に縮退する。
2. 分割統治法を用いて、BLS を解く。
3. すべての Householder 変換を逆に適用して、元の最小二乗問題を解く。

A の有効階数は、最大特異値の $rcond$ 倍より小さい特異値をゼロとして扱うことで決定される。

このルーチンは、補助ルーチン [?lals0](#) および [?lalsa](#) を使用する。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。行列 A の列数 ($n \geq 0$)。
$nrhs$	INTEGER。右辺の数。 B の列数 ($nrhs \geq 0$)。

<i>a, b, work</i>	<p>REAL (sgelsd の場合) DOUBLE PRECISION (dgelsd の場合) COMPLEX (cgelsd の場合) DOUBLE COMPLEX (zgelsd の場合)。 配列： <i>a</i>(<i>lda</i>,*) には、$m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>b</i>(<i>ldb</i>,*) には、$m \times nrhs$ の右辺の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, m, n)$ 以上でなければならない。
<i>rcond</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 <i>rcond</i> は、<i>A</i> の有効階数を決定するのに使用する。 特異値 $s(i) \leq rcond * s(1)$ をゼロとして扱う。 <i>rcond</i> ≤ 0 の場合、代わりにマシン精度を使用する。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> のサイズ、$lwork \geq 1$。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、<i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>
<i>iwork</i>	<p>INTEGER。ワークスペース配列。 <i>iwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>
<i>rwork</i>	<p>REAL (cgelsd の場合) DOUBLE PRECISION (zgelsd の場合) 複素数型の場合にのみ使用されるワークスペース配列。 <i>rwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>
出力パラメーター	
<i>a</i>	終了時に、 <i>A</i> が上書きされる。
<i>b</i>	<p>$n \times nrhs$ の解行列 <i>X</i> で上書きされる。 $m \geq n$ で $rank = n$ の場合、<i>i</i> 番目の列の解の二乗和の誤差は、その列の $n+1:m$ の成分の二乗和で与えられる。</p>
<i>s</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。<i>A</i> の特異値が降順で格納される。<i>A</i> の 2-ノルムの条件数は、$k_2(A) = s(1) / s(\min(m, n))$。</p>
<i>rank</i>	<p>INTEGER。 <i>A</i> の有効階数、すなわち、$rcond * s(1)$ より大きい特異値の個数。</p>

<i>work(1)</i>	<i>info</i> = 0 の場合、終了時に、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が <i>work(1)</i> に格納される。これ以降の実行には、この <i>lwork</i> の値を使用する。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、SVD の計算アルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の二重対角形式の非対角成分で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gelsd` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ $\max(m, n) \times nrhs$ の行列を格納する。 呼び出し時に、 $m \times nrhs$ の右辺の行列 <i>B</i> を格納する。 終了時に、 $n \times nrhs$ の解の行列 <i>X</i> によって上書きされる。
<i>s</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>rcond</i>	この成分のデフォルト値は $rcond = 100 * \text{EPSILON}(1.0_wp)$ である。

アプリケーション・ノート

分割統治法は、浮動小数点演算において、非常に緩い仮定を行う。すなわち、加減算においてガード桁を持つマシンで機能する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

必要なワークスペースの最小値は、*m*、*n*、*nrhs* によって決まる。ワークスペース配列 *work* のサイズ *lwork* は、次に示す値以上でなければならない。

実数型の場合:

$$m \geq n \text{ の場合、} \\ lwork \geq 12n + 2n * smlsiz + 8n * nlvl + n * nrhs + (smlsiz + 1)^2$$

$$m < n \text{ の場合、} \\ lwork \geq 12m + 2m * smlsiz + 8m * nlvl + m * nrhs + (smlsiz + 1)^2$$

複素数型の場合:

$$m \geq n \text{ の場合、} \\ lwork \geq 2n + n * nrhs$$

$$m < n \text{ の場合、} \\ lwork \geq 2m + m * nrhs$$

smlsiz は `ilaenv` に返された値で、計算ツリーが一番下の部分問題の最大サイズ (通常は約 25) で、 $nlvl = \text{INT}(\log_2(\min(m, n)/(smlsiz + 1))) + 1$ 。

パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。必要なワークスペースの大きさがわからない場合は、最初の実行では $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

ワークスペース配列 $iwork$ のサイズは、 $3 * \min(m, n) * nvl + 11 * \min(m, n)$ 以上でなければならない。

ワークスペース配列 $rwork$ (複素数型の場合) のサイズ $lrwork$ は、次の値でなければならない。

$m \geq n$ の場合、

$$lrwork \geq 10n + 2n * smlsiz + 8n * nvl + 3 * smlsiz * nrhs + (smlsiz + 1)^2$$

$m < n$ の場合、

$$lrwork \geq 10m + 2m * smlsiz + 8m * nvl + 3 * smlsiz * nrhs + (smlsiz + 1)^2$$

汎用 LLS 問題

このセクションでは、汎用線形最小二乗問題を解くための LAPACK ドライバールーチンについて説明する。表 4-9 に、Fortran-77 インターフェイスのルーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-9 汎用 LLS 問題を解くためのドライバールーチン

ルーチン名	機能
?qglse	汎用 RQ 因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。
?ggglm	汎用 QR 因子分解を使用して、Gauss-Markov 線形モデル問題を解く。

?gglse

汎用 RQ 因子分解を使用して、均衡制約を持つ線形最小二乗問題を解く。

構文

Fortran 77:

```
call sgglse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call dgglse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call cgglse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
call zgglse(m, n, p, a, lda, b, ldb, c, d, x, work, lwork, info)
```

Fortran 95:

```
call gglse(a, b, c, d, x [,info])
```

説明

このルーチンは、次のような均衡制約を持つ線形最小二乗 (LSE) 問題を解く。

$$\text{minimize } \|c - Ax\|_2 \quad \text{ただし、} Bx = d$$

A は $m \times n$ の行列、 B は $p \times n$ の行列、 c は与えられた m -ベクトル、 d は与えられた p -ベクトルである。ただし、 $p \leq n \leq m+p$ 、

$$\text{rank}(B) = p, \text{rank} \begin{pmatrix} A \\ B \end{pmatrix} = n \text{ と仮定する。}$$

これらの条件により、この LSE 問題は、行列 B と A の汎用 RQ 因子分解により、一意の解が得られる問題になる。

入力パラメーター

m INTEGER。行列 A の行数 ($m \geq 0$)。

n INTEGER。行列 A と B の列数 ($n \geq 0$)。

p INTEGER。行列 B の行数 ($0 \leq p \leq n \leq m+p$)。

$a, b, c, d, work$	REAL (sgglse の場合) DOUBLE PRECISION (dgglsse の場合) COMPLEX (cgglse の場合) DOUBLE COMPLEX (zgglse の場合) 配列: $a(lda, *)$ には、 $m \times n$ の行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b(l db, *)$ には、 $p \times n$ の行列 B を格納する。 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $c(*)$ は、サイズが $\max(1, m)$ 以上の配列で、LSE 問題の最小二乗部分に関する右辺のベクトルを格納する。 $d(*)$ は、サイズが $\max(1, p)$ 以上の配列で、制約式の右辺のベクトルを格納する。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, p)$ 以上。
$lwork$	INTEGER。 配列 $work$ のサイズ。 $lwork \geq \max(1, m+n+p)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

x	REAL (sgglse の場合) DOUBLE PRECISION (dgglsse の場合) COMPLEX (cgglse の場合) DOUBLE COMPLEX (zgglse の場合)。 配列、次元は $\max(1, n)$ 以上。 終了時に、LSE 問題の解が格納される。
a, b, d	終了時に、これらの配列が上書きされる。
c	終了時に、解の二乗和の誤差が、ベクトル c の $n-p+1 \sim m$ の成分の二乗和で与えられる。
$work(1)$	$info = 0$ の場合、終了時に、最適なパフォーマンスを得るために必要な $lwork$ の最小値が $work(1)$ に格納される。これ以降の実行には、この $lwork$ の値を使用する。
$info$	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gglse` のインターフェイスの詳細を以下に示す。

- `a` サイズ (m, n) の行列 A を格納する。
- `b` サイズ (p, n) の行列 B を格納する。
- `c` 長さ (m) のベクトルを格納する。
- `d` 長さ (p) のベクトルを格納する。
- `x` 長さ (n) のベクトルを格納する。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq p + \min(m, n) + \max(m, n) * nb$$

ここで、`nb` は、`?geqrf`、`?gerqf`、`?ormqr`/`?unmqr`、および `?ormrq`/`?unmrq` に対する最適ブロックサイズの上限值である。

?ggglm

汎用 *QR* 因子分解を使用して、*Gauss-Markov* 線形モデル問題を解く。

構文

Fortran 77:

```
call sggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call dggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call cggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
call zggglm(n, m, p, a, lda, b, ldb, d, x, y, work, lwork, info)
```

Fortran 95:

```
call ggglm(a, b, d, x, y [,info])
```

説明

このルーチンは、次のような一般 *Gauss-Markov* 線形モデル (GLM) 問題を解く。

$$\text{minimize}_x \|y\|_2 \quad \text{ただし、} d = Ax + By$$

ここで、 A は $n \times m$ の行列、 B は $n \times p$ の行列、 d は与えられた n -ベクトルである。

$m \leq n \leq m+p$ 、 $\text{rank}(A) = m$ 、 $\text{rank}(AB) = n$ と仮定する。

これらの仮定により、制約式が常に成立し、 A と B の汎用 *QR* 因子分解により、一意の解 x と最小 2-ノルム解 y が得られる。

特に、行列 B が正方非特異行列の場合、この GLM 問題は、次のような重み付け線形最小二乗問題と等価になる。

$$\text{minimize}_x \|B^{-1}(d - Ax)\|_2$$

入力パラメーター

`n` INTEGER。行列 A と B の行数 ($n \geq 0$)。

<i>m</i>	INTEGER。行列 <i>A</i> の列数 ($m \geq 0$)。
<i>p</i>	INTEGER。行列 <i>B</i> の列数 ($p \geq n - m$)。
<i>a, b, d, work</i>	REAL (sggg1m の場合) DOUBLE PRECISION (dggg1m の場合) COMPLEX (cggg1m の場合) DOUBLE COMPLEX (zggg1m の場合) 配列: <i>a(lda,*)</i> には、 $n \times m$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>b(l db,*)</i> には、 $n \times p$ の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は、 $\max(1, p)$ 以上でなければならない。 <i>d(*)</i> は、サイズが $\max(1, n)$ 以上の配列で、GLM 式の左辺を格納する。 <i>work(lwork)</i> は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>l db</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>l work</i>	INTEGER。配列 <i>work</i> のサイズ。 $l work \geq \max(1, n+m+p)$ 。 <i>l work = -1</i> の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>l work</i> に関するエラーメッセージを生成しない。 <i>l work</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>x, y</i>	REAL (sggg1m の場合) DOUBLE PRECISION (dggg1m の場合) COMPLEX (cggg1m の場合) DOUBLE COMPLEX (zggg1m の場合)。 配列 <i>x(*)</i> 、 <i>y(*)</i> 。次元は $\max(1, m)$ 以上 (<i>x</i> の場合) または $\max(1, p)$ 以上 (<i>y</i> の場合)。 終了時に、 <i>x</i> と <i>y</i> が GLM 問題の解になる。
<i>a, b, d</i>	終了時に、これらの配列が上書きされる。
<i>work(1)</i>	<i>info = 0</i> の場合、終了時に、最適なパフォーマンスを得るために必要な <i>l work</i> の最小値が <i>work(1)</i> に格納される。
<i>info</i>	INTEGER。 <i>info = 0</i> の場合、実行は正常に終了したことを示す。 <i>info = -i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggg1m` のインターフェイスの詳細を以下に示す。

- `a` サイズ (n,m) の行列 A を格納する。
- `b` サイズ (n,p) の行列 B を格納する。
- `d` 長さ (n) のベクトルを格納する。
- `x` 長さ (m) のベクトルを格納する。
- `y` 長さ (p) のベクトルを格納する。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq m + \min(n, p) + \max(n, p) * nb$$

ここで、 nb は [?gegrf](#)、[?gerqf](#)、[?ormqr/?unmqr](#)、および [?ormrqr/?unmrqr](#) に対する最適ブロックサイズの上限值である。

対称固有値問題

このセクションでは、対称固有値問題を解くための LAPACK ドライバールーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

表 4-10 に Fortran-77 インターフェイスのすべてのドライバールーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-10 対称固有値問題を解くためのドライバールーチン

ルーチン名	機能
?syev / ?heev	実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?syevd / ?heevd	分割統治アルゴリズムを使用して、実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?syevx / ?heevx	対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
?syevr / ?heevr	「比較的安定な表現」を使用して、実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
?spev / ?hpev	圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?spevd / ?hpevd	分割統治アルゴリズムを使用して、圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?spevx / ?hpevx	圧縮形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
?sbev / ?hbев	帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?sbevd / ?hbевd	分割統治アルゴリズムを使用して、帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?sbevх / ?hbевх	帯形式の実対称 / エルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。
?stev	実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?stevd	分割統治アルゴリズムを使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。
?stevx	実対称三重対角行列の固有値と固有ベクトルを選択的に計算する。
?stevr	「比較的安定な表現」を使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

?syev

実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssyev(jobz, uplo, n, a, lda, w, work, lwork, info)
call dsyev(jobz, uplo, n, a, lda, w, work, lwork, info)
```

Fortran 95:

```
call syev(a, w [,jobz] [,uplo] [,info])
```

説明

このルーチンは、実対称行列 A のすべての固有値と (オプションで) 固有ベクトルをすべて計算する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syevr](#) 関数を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 a には A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 a には A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	REAL (ssyev の場合) DOUBLE PRECISION (dsyev の場合) 配列: $a(lda,*)$ は、 <i>uplo</i> の値に従って、対称行列 A の上または下三角部分を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>lwork</i>	INTEGER。配列 $work$ の次元。 次の制約がある。 $lwork \geq \max(1, 3n-1)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	<i>jobz</i> ='V' の場合、終了時に、 <i>info</i> =0 であれば、行列 <i>A</i> の正規直交固有ベクトルが、配列 <i>a</i> に格納される。 <i>jobz</i> ='N' の場合、終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> ='L' の場合) または上三角部分 (<i>uplo</i> ='U' の場合) が、対角成分を含めて上書きされる。
<i>w</i>	REAL (<i>ssyev</i> の場合) DOUBLE PRECISION (<i>dsyev</i> の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> =0 の場合、行列 <i>A</i> の固有値が昇順に格納される。
<i>work(1)</i>	終了時に、 <i>lwork</i> >0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> =0 の場合、正常に終了したことを示す。 <i>info</i> =- <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *syev* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>job</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+2)*n$$

ここで、*nb* は、*ilaenv* が *?sytrd* に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

?heev

エルミート行列の固有値と (オプションで)
固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call cheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
call zheev(jobz, uplo, n, a, lda, w, work, lwork, rwork, info)
```

Fortran 95:

```
call heev(a, w [,jobz] [,uplo] [,info])
```

説明

このルーチンは、エルミート行列 A のすべての固有値と (オプションで) 固有ベクトルをすべて計算する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?heevr](#) 関数を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 a には A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 a には A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	COMPLEX (cheev の場合) DOUBLE COMPLEX (zheev の場合) 配列: $a(lda,*)$ は、 <i>uplo</i> の値に従って、エルミート行列 A の上または下三角部分を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>lwork</i>	INTEGER。配列 $work$ の次元。 次の制約がある。 $lwork \geq \max(1, 2n-1)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

rwork REAL (cheev の場合)
DOUBLE PRECISION (zheev の場合)。
ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

出力パラメーター

a *jobz*='V' の場合、終了時に、*info*=0 であれば、行列 *A* の正規直交固有ベクトルが、配列 *a* に格納される。
jobz='N' の場合、終了時に、行列 *A* の下三角部分 (*uplo*='L' の場合) または上三角部分 (*uplo*='U' の場合) が、対角成分を含めて上書きされる。

w REAL (cheev の場合)
DOUBLE PRECISION (zheev の場合)
配列、次元は $\max(1, n)$ 以上。
info=0 の場合、行列 *A* の固有値が昇順に格納される。

work(1) 終了時に、*lwork*>0 の場合、*work(1)* は *lwork* の必要最小サイズを返す。

info INTEGER。
info=0 の場合、正常に終了したことを示す。
info=-*i* の場合、*i* 番目のパラメーターの値が不正である。
info=*i* の場合、このアルゴリズムが収束に失敗したことを示す。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *heev* のインターフェイスの詳細を以下に示す。

a サイズ (*n*,*n*) の行列 *A* を格納する。
w 長さ (*n*) のベクトルを格納する。
job 'N' または 'V' でなければならない。デフォルト値は 'N'。
uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、*nb* は、*ilaenv* が *?hetrd* に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

?syevd

分割統治アルゴリズムを使用して、実対称行列の固有値と(オプションで)固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
call dsyevd(job, uplo, n, a, lda, w, work, lwork, iwork, liwork, info)
```

Fortran 95:

```
call syevd(a, w [,jobz] [,uplo] [,info])
```

説明

このルーチンは、分割統治アルゴリズムを使用して、実対称行列 A の固有値と(オプションで)固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = Z\Lambda Z^T$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような対角行列、 Z は各列が固有ベクトル z_i であるような直交行列である。したがって、

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syevr](#) 関数を使用する。[?syevd](#) は、より多くのワークスペースが必要となるが、特定のケース(特に大きな行列の場合)ではより高速である。

入力パラメーター

<i>job</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>job</i> = 'N' の場合、固有値のみを計算する。 <i>job</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i>	REAL (ssyevd の場合) DOUBLE PRECISION (dsyevd の場合) 配列、次元は (<i>lda</i> ,*)。 <i>a</i> (<i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 A の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>work</i>	REAL (<i>ssyevd</i> の場合) DOUBLE PRECISION (<i>dsyevd</i> の場合)。 ワークスペース配列、次元は <i>lwork</i> 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n+1$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 3n^2 + (5+2k) * n + 1$ (ここで、 k は、 $2^k \geq n$ を満たす最小の整数)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $liwork \geq 1$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n+2$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w</i>	REAL (<i>ssyevd</i> の場合) DOUBLE PRECISION (<i>dsyevd</i> の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>a</i>	<i>job</i> ='V' の場合、終了時に、 <i>A</i> の固有ベクトルが入った直交行列 <i>Z</i> で上書きされる。
<i>work(1)</i>	終了時に、 <i>lwork</i> > 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `syevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>job</code>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

このルーチンの複素数版は、[?heevd](#) である。

?heevd

分割統治アルゴリズムを使用して、複素エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call cheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,
            iwork, liwork, info)
call zheevd(job, uplo, n, a, lda, w, work, lwork, rwork, lrwork,
            iwork, liwork, info)
```

Fortran 95:

```
call heevd(a, w [,job] [,uplo] [,info])
```

説明

このルーチンは、分割統治アルゴリズムを使用して、複素エルミート行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = Z\Lambda Z^H$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような実対角行列、 Z は各列が固有ベクトル z_i であるような (複素) ユニタリー行列である。したがって、

$$Az_i = \lambda_i z_i \quad \text{ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで `?heevr` 関数を使用する。`?heevd` は、より多くのワークスペースが必要となるが、特定のケース（特に大きな行列の場合）ではより高速である。

入力パラメーター

<code>job</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>job = 'N'</code> の場合、固有値のみを計算する。 <code>job = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 <code>a</code> には A の上三角部分を格納する。 <code>uplo = 'L'</code> の場合、 <code>a</code> には A の下三角部分を格納する。
<code>n</code>	INTEGER。行列 A の次数 ($n \geq 0$)。
<code>a</code>	COMPLEX (<code>cheevd</code> の場合) DOUBLE COMPLEX (<code>zheevd</code> の場合) 配列、次元は $(lda, *)$ 。 <code>a(lda, *)</code> は、 <code>uplo</code> の値に従って、エルミート行列 A の上三角部分または下三角部分を格納する配列である。 <code>a</code> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<code>lda</code>	INTEGER。配列 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<code>work</code>	COMPLEX (<code>cheevd</code> の場合) DOUBLE COMPLEX (<code>zheevd</code> の場合)。 ワークスペース配列、次元は <code>lwork</code> 以上。
<code>lwork</code>	INTEGER。配列 <code>work</code> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <code>job = 'N'</code> で $n > 1$ の場合、 $lwork \geq n+1$ 。 <code>job = 'V'</code> で $n > 1$ の場合、 $lwork \geq n^2+2n$ 。 <code>lwork = -1</code> の場合はワークスペースのクエリーとみなされ、ルーチンは <code>work</code> 配列の最適サイズだけを計算し、その値を <code>work</code> 配列の最初のエントリーとして返し、 <code>xerbla</code> は <code>lwork</code> に関するエラーメッセージを生成しない。
<code>rwork</code>	REAL (<code>cheevd</code> の場合) DOUBLE PRECISION (<code>zheevd</code> の場合) ワークスペース配列、次元は <code>lrwork</code> 以上。
<code>lrwork</code>	INTEGER。配列 <code>rwork</code> の次元。 次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 <code>job = 'N'</code> で $n > 1$ の場合、 $lrwork \geq n$ 。 <code>job = 'V'</code> で $n > 1$ の場合、 $lrwork \geq 3n^2 + (4+2k) * n + 1$ (ここで、 k は、 $2^k \geq n$ を満たす最小の整数)。 <code>lrwork = -1</code> の場合、ワークスペースのクエリーとみなされ、ルーチンは <code>rwork</code> 配列の最適サイズだけを計算し、その値を <code>rwork</code> 配列の最初のエントリーとして返す。 <code>xerbla</code> は <code>lrwork</code> に関するエラーメッセージを生成しない。

<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 $job = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。 $job = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n+2$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。xerbla は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w</i>	REAL (cheevd の場合) DOUBLE PRECISION (zheevd の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>a</i>	$job = 'V'$ の場合、終了時に、 <i>A</i> の固有ベクトルが入ったユニタリ行列 <i>Z</i> で上書きされる。
<i>work(1)</i>	終了時に、 $lwork > 0$ の場合、 <i>work(1)</i> の実数部は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 $lrwork > 0$ の場合、 <i>rwork(1)</i> は <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $liwork > 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *heevd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>job</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|A\|_2$ のような行列 $A + E$ (ϵ はマシン精度) のものである。

このルーチンの実数版は、[?syeval](#) である。
圧縮形式の行列に対する [?hpeval](#) と、帯形式の行列に関する [?hbeval](#) も参照のこと。

?syevx

対称行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call ssyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,
           m, w, z, ldz, work, lwork, iwork, ifail, info)
call dsyevx(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,
           m, w, z, ldz, work, lwork, iwork, ifail, info)
```

Fortran 95:

```
call syevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
           [,abstol] [,info])
```

説明

このルーチンは、実対称行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?syeval](#) 関数を使用する。[?syevx](#) は、選択する固有値が少ない場合、より高速である。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 (<i>vl</i> , <i>vu</i>) の固有値をすべて計算する。 <i>range</i> = 'I' の場合、インデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には A の下三角部分を格納する。

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合)。 配列: <i>a</i> (<i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>v1</i> , <i>vu</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値 ($v1 \leq vu$)。 <i>range</i> = 'A' または 'I' の場合は参照されない。
<i>il</i> , <i>iu</i>	INTEGER。 <i>range</i> = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n = 0$ の場合)。 <i>range</i> = 'A' または 'V' の場合は参照されない。
<i>abstol</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> の第 1 次元。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 8n)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $\max(1, 5n)$ 以上。
出力パラメーター	
<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> = 'L' の場合) または上三角部分 (<i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。

<i>w</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合) 配列、次元は $\max(1, n)$ 以上。 先頭の m 個の成分に、行列 A の選択された固有値が昇順に格納される。
<i>z</i>	REAL (ssyevx の場合) DOUBLE PRECISION (dsyevx の場合)。 配列 $z(ldz, *)$ には、固有ベクトルが格納される。 z の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 z の先頭の m 列に、行列 A の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 z の i 番目の列に格納される。固有ベクトルが収束できない場合、 z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 z は参照されない。 注：配列 z には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 m の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>work(1)</i>	終了時に、 $lwork > 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から m 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'V' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - i の場合、 i 番目のパラメーターの値が不正である。 <i>info</i> = i の場合、 i 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *syevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>ifail</i>	長さ (n) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は $vu = HUGE(v1)$ である。

<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+3)*n$$

ここで、*nb* は、*ilaenv* が ?sytrd と ?ormtr に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

abstol + ϵ * max(|*a*|, |*b*|) 以下の幅の区間 [*a*, *b*] 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロまたは負の場合、代わりに $\epsilon * |T|$ を使用する。*|T|* は、*A* を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の 2 倍、*2*slamch*('S') に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、*abstol* を *2*slamch*('S') に設定して、やり直してみる。

?heevx

エルミート行列の固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call cheevx(jobz, range, uplo, n, a, lda, v1, vu, i1, iu, abstol,
           m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
call zheevx(jobz, range, uplo, n, a, lda, v1, vu, i1, iu, abstol,
           m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
```

Fortran 95:

```
call heevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
          [,abstol] [,info])
```

説明

このルーチンは、複素エルミート行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、デフォルトで [?heevr](#) 関数を使用する。[?heevx](#) は、選択する固有値が少ない場合、より高速である。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' でなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 (<i>vl</i> , <i>vu</i>) の固有値をすべて計算する。 <i>range</i> = 'I' の場合、インデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	COMPLEX (cheevx の場合) DOUBLE COMPLEX (zheevx の場合)。 配列: <i>a</i> (<i>lda</i> ,*) は、 <i>uplo</i> の値に従って、エルミート行列 A の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>vl</i> , <i>vu</i>	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値 ($vl \leq vu$)。 <i>range</i> = 'A' または 'I' の場合は参照されない。
<i>il</i> , <i>iu</i>	INTEGER。 <i>range</i> = 'I' の場合、求める固有値の最小インデックスと最大インデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n = 0$ の場合)。 <i>range</i> = 'A' または 'V' の場合は参照されない。

<i>abstol</i>	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> の第 1 次元。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 2n-1)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合)。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> = 'L' の場合) または上三角部分 (<i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (cheevx の場合) DOUBLE PRECISION (zheevx の場合) 配列、次元は $\max(1, n)$ 以上。 先頭の <i>m</i> 個の成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。
<i>z</i>	COMPLEX (cheevx の場合) DOUBLE COMPLEX (zheevx の場合)。 配列 <i>z</i> (<i>ldz</i> , *) には、固有ベクトルが格納される。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>work</i> (1)	終了時に、 <i>lwork</i> > 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。

<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'V' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *heevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE(<i>v1</i>) である。
<i>vu</i>	この成分のデフォルト値は <i>vu</i> = HUGE(<i>v1</i>) である。
<i>il</i>	この引数のデフォルト値は、 <i>il</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、*nb* は、*ilaenv* が ?hetrd と ?unmtr に対して返したブロックサイズである。

必要なワークスペースの大きさがわからない場合は、最初の実行では `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。`abstol` がゼロまたは負の場合、代わりに $\epsilon * |T|$ を使用する。 $|T|$ は、 A を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。

固有値が最も正確に計算されるのは、`abstol` がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * \text{slamch}('S')$ に設定されたときである。このルーチンで `info > 0` が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、`abstol` を $2 * \text{slamch}('S')$ に設定して、やり直してみる。

?syevr

「比較的安定な表現」を使用して、実対称行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call ssyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,
           m, w, z, ldz, isuppz, work, lwork, iwork, liwork, info)
call dsyevr(jobz, range, uplo, n, a, lda, vl, vu, il, iu, abstol,
           m, w, z, ldz, isuppz, work, lwork, iwork, liwork, info)
```

Fortran 95:

```
call syevr(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz]
           [,abstol] [,info])
```

説明

このルーチンは、実対称行列 T の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、`?syevr` は [sstegr/dstegr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。`?stegr` は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。

T のまだ縮退されていない i 番目のブロックに対して、

- (a) $L_i D_i L_i^T$ が比較的安定な表現になるように、 $T - \sigma_i = L_i D_i L_i^T$ を計算する。
- (b) *dqds* アルゴリズムによって、 $L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスターが存在する場合は、そのクラスターに近い σ_i を「選択」し、ステップ (a) に戻る。
- (d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメーター *abstol* で指定できる。

ルーチン `?syevr` は、IEEE-754 の浮動小数点標準に準拠しているマシンでフルスペクトルが要求された場合は、[sstegr/dstegr](#) を呼び出す。`?syevr` は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合に、[sstebz/dstebz](#) と [sstein/dstein](#) を呼び出す。

ほとんどの実対称固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、`?syevr` を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半开区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。 <i>range</i> = 'V' または 'I' で、 $iu - il < n - 1$ の場合、 sstebz/dstebz と sstein/dstein を呼び出す。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>a</i> には <i>A</i> の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>a</i> には <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (<code>ssyevr</code> の場合) DOUBLE PRECISION (<code>dsyevr</code> の場合)。 配列： <i>a</i> (<i>lda</i> ,*) は、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。

<i>v1, vu</i>	REAL (ssyevr の場合) DOUBLE PRECISION (dsyevr の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (ssyevr の場合) DOUBLE PRECISION (dsyevr の場合)。 各固有値および固有ベクトルに許される絶対誤差許容値。 <i>jobz</i> = 'V' の場合、出力される固有値と固有ベクトルの誤差ノルムが <i>abstol</i> 以内になる。また、異なる固有ベクトル間のドット積も <i>abstol</i> 以内になる。 $abstol < n\epsilon\ T\ _1$ の場合、 $n\epsilon\ T\ _1$ が代わりに使用される (ϵ はマシン精度)。固有値は、 <i>abstol</i> とは無関係に、 $\epsilon\ T\ _1$ の精度で計算される。高い相対精度が必要な場合は、 <i>abstol</i> を ?lamch('S') に設定する。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ (<i>jobz</i> = 'N' の場合)。 $ldz \geq \max(1, n)$ (<i>jobz</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 次の制約がある。 $lwork \geq \max(1, 26n)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq \max(1, 10n)$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエンタリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> = 'L' の場合) または上三角部分 (<i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
----------	--

<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i> , <i>z</i>	REAL (ssyevr の場合) DOUBLE PRECISION (dsyevr の場合)。 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。選択された固有値が昇順に <i>w</i> (1) ~ <i>w</i> (<i>m</i>) に格納される。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。 <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 <i>isuppz</i> (2 <i>i</i> -1) から <i>isuppz</i> (2 <i>i</i>) までの成分のみ非ゼロである。 現在は実装されていない。また、参照されない。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメータの値が不正である。 <i>info</i> = <i>i</i> の場合、内部エラーが発生したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *syevr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>isuppz</i>	長さ (2* <i>m</i>) のベクトルを格納する。ここで、値 (2* <i>m</i>) は有意である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE(<i>v1</i>) である。

<i>vu</i>	この成分のデフォルト値は $vu = \text{HUGE}(v1)$ である。
<i>i1</i>	この引数のデフォルト値は、 $i1 = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>isuppz</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+6)*n$$

ここで、*nb* は、*ilaenv* が *?sytrd* と *?ormtr* に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

通常 *?stegr* の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

?heevr

「比較的安定な表現」を使用して、エルミート行列の固有値と(オプションで)固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call cheevr(jobz, range, uplo, n, a, lda, v1, vu, i1, iu, abstol, m, w,
           z, ldz, isuppz, work, lwork, rwork, lrwork, iwork, liwork, info)
call zheevr(jobz, range, uplo, n, a, lda, v1, vu, i1, iu, abstol, m, w,
           z, ldz, isuppz, work, lwork, rwork, lrwork, iwork, liwork, info)
```

Fortran 95:

```
call heevr(a, w [,uplo] [,z] [,v1] [,vu] [,i1] [,iu] [,m] [,isuppz]
           [,abstol] [,info])
```

説明

このルーチンは、複素エルミート行列 T の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、`?heevr` は [csteqr/zsteqr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。`?stegr` は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 T のまだ縮退されていない i 番目のブロックに対して、

- (a) $L_i D_i L_i^T$ が比較的安定な表現になるように、 $T - \sigma_i = L_i D_i L_i^T$ を計算する。
- (b) *dqds* アルゴリズムによって、 $L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスターが存在する場合は、そのクラスターに近い σ_i を「選択」し、ステップ (a) に戻る。
- (d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメーター *abstol* で指定できる。

ルーチン `?heevr` は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフルスペクトルが要求された場合は、[csteqr/zsteqr](#) を呼び出す。`?heevr` は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合に、[sstebz/dstebz](#) と [cstein/zstein](#) を呼び出す。

ほとんどの複素エルミート固有値問題では、基本となるアルゴリズムがより高速でワークスペースの使用が少なくなるように、`?heevr` を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半开区間 $v_l < \lambda_j \leq v_u$ の固有値 λ_j を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。 <i>range</i> ='V' または 'I' の場合、 sstebz/dstebz と cstein/zstein を呼び出す。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>a</i> には A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>a</i> には A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。

<i>a, work</i>	<p>COMPLEX (cheevr の場合) DOUBLE COMPLEX (zheevr の場合)。 配列: $a(lda,*)$ は、<i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角部分または下三角部分を格納する配列である。 <i>a</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>work(lwork)</i> は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。</p>
<i>v1, vu</i>	<p>REAL (cheevr の場合) DOUBLE PRECISION (zheevr の場合)。 <i>range = 'V'</i> の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$。 <i>range = 'A'</i> または <i>'I'</i> の場合、<i>v1</i> と <i>vu</i> は参照されない。</p>
<i>il, iu</i>	<p>INTEGER。 <i>range = 'I'</i> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range = 'A'</i> または <i>'V'</i> の場合、<i>il</i> と <i>iu</i> は参照されない。</p>
<i>abstol</i>	<p>REAL (cheevr の場合) DOUBLE PRECISION (zheevr の場合)。 各固有値および固有ベクトルに許される絶対誤差許容値。 <i>jobz = 'V'</i> の場合、出力される固有値と固有ベクトルの誤差ノルムが <i>abstol</i> 以内になる。また、異なる固有ベクトル間のドット積も <i>abstol</i> 以内になる。$abstol < n\epsilon\ T\ _1$ の場合、$n\epsilon\ T\ _1$ が代わりに使用される (ϵ はマシン精度)。固有値は、<i>abstol</i> とは無関係に、$\epsilon\ T\ _1$ の精度で計算される。高い相対精度が必要な場合は、<i>abstol</i> を <code>?lamch('S')</code> に設定する。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ (<i>jobz = 'N'</i> の場合)。 $ldz \geq \max(1, n)$ (<i>jobz = 'V'</i> の場合)。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。次の制約がある。 $lwork \geq \max(1, 2n)$。 <i>lwork = -1</i> の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、<i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。</p>
<i>rwork</i>	<p>REAL (cheevr の場合) DOUBLE PRECISION (zheevr の場合)。 ワークスペース配列、次元は (<i>lwork</i>)。</p>

<i>lwork</i>	INTEGER。配列 <i>rwork</i> の次元。 $lwork \geq \max(1, 24n)$ 。 <i>lwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリーとして返す。xerbla は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq \max(1, 10n)$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ A ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。xerbla は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> = 'L' の場合) または上三角部分 (<i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (<i>cheevr</i> の場合) DOUBLE PRECISION (<i>zheevr</i> の場合)。 配列、次元は $\max(1, n)$ 以上。選択された固有値が昇順に $w(1) \sim w(m)$ に格納される。
<i>z</i>	COMPLEX (<i>cheevr</i> の場合) DOUBLE COMPLEX (<i>zheevr</i> の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。 <i>z</i> に格納されている固有ベクトルのサポート情報、すなわち <i>z</i> に格納されている非ゼロの成分を示すインデックス。 <i>i</i> 番目の固有ベクトルは、 <i>isuppz</i> ($2i - 1$) から <i>isuppz</i> ($2i$) までの成分のみ非ゼロである。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>rwork(1)</i> は、 <i>lwork</i> の必要最小サイズを返す。

<i>iwork(1)</i>	終了時に、 <i>info=0</i> の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info=0</i> の場合、正常に終了したことを示す。 <i>info=-i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info=i</i> の場合、内部エラーが発生したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *heevr* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n,n</i>) の行列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n,m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>isuppz</i>	長さ (<i>2*m</i>) のベクトルを格納する。ここで、値 (<i>2*m</i>) は有意である。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は <i>v1 = -HUGE(v1)</i> である。
<i>vu</i>	この成分のデフォルト値は <i>vu = HUGE(v1)</i> である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1 = 1</i> である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu = n</i> である。
<i>abstol</i>	この成分のデフォルト値は <i>abstol = 0.0_WP</i> である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz = 'V'</i> (<i>z</i> が存在する場合)、 <i>jobz = 'N'</i> (<i>z</i> が省略された場合)。 <i>isuppz</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range = 'V'</i> (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range = 'I'</i> (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range = 'A'</i> (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、*nb* は、[ilaenv](#) が *hetrd* と *zunmtr* に対して返したブロックサイズである。必要なワークスペースの大きさがわからない場合は、最初の実行では *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

通常の `?stegr` の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

?spev

圧縮形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sspev(jobz, uplo, n, ap, w, z, ldz, work, info)
call dspev(jobz, uplo, n, ap, w, z, ldz, work, info)
```

Fortran 95:

```
call spev(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、圧縮形式の実対称行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	REAL (sspev の場合) DOUBLE PRECISION (dspev の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 A の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 3n)$ 以上。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> ='N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

<i>w</i> , <i>z</i>	REAL (sspev の場合) DOUBLE PRECISION (dspev の場合) 配列: <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、 <i>w</i> に、行列 <i>A</i> の固有値が昇順に格納される。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spev` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?hpev

圧縮形式のエルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chpev(jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
call zhpev(jobz, uplo, n, ap, w, z, ldz, work, rwork, info)
```

Fortran 95:

```
call hpev(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、圧縮形式のエルミート行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	COMPLEX (chpev の場合) DOUBLE COMPLEX (zhpev の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n-1)$ 以上。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> ='N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chpev の場合) DOUBLE PRECISION (zhpev の場合)。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

出力パラメーター

<i>w</i>	REAL (chpev の場合) DOUBLE PRECISION (zhpev の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、 <i>w</i> に、行列 <i>A</i> の固有値が昇順に格納される。
<i>z</i>	COMPLEX (chpev の場合) DOUBLE COMPLEX (zhpev の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpev* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?spevd

分割統治アルゴリズムを使用して、圧縮形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sspevd(job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork,
            info)
call dspevd(job, uplo, n, ap, w, z, ldz, work, lwork, iwork, liwork,
            info)
```

Fortran 95:

```
call spevd(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、圧縮形式の実対称行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = Z\Lambda Z^T$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような対角行列、 Z は各列が固有ベクトル z_i であるような直交行列である。したがって、

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメーター

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	REAL (sspevd の場合) DOUBLE PRECISION (dspevd の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 A の上または下三角部分を圧縮形式で格納する。 <i>ap</i> のサイズは、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は <i>lwork</i> 以上。

<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2 + (5+2k) * n + 1$ (ここで、 <i>k</i> は、 $2^k \geq n$ を満たす最小の整数)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>liwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $liwork \geq 1$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n + 3$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w, z</i>	REAL (sspevd の場合) DOUBLE PRECISION (dspevd の場合) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、1 以上 (<i>job</i> ='N' の場合) または $\max(1, n)$ 以上 (<i>job</i> ='V' の場合) でなければならない。 <i>job</i> ='V' の場合、この配列は、 <i>A</i> の固有ベクトルの入った直交行列 <i>Z</i> で上書きされる。 <i>job</i> ='N' の場合、 <i>z</i> は参照されない。
<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) に最適な <i>lwork</i> 値が返される。
<i>iwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>iwork</i> (1) に最適な <i>liwork</i> 値が返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示

す。 i は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>z</code>	サイズ (n, n) の行列 Z を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

このルーチンの複素数版は、[?hpevd](#) である。

フル形式の行列に対する [?syevd](#) と、帯形式の行列に関する [?sbevd](#) も参照のこと。

?hpevd

分割統治アルゴリズムを使用して、圧縮形式の複素エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chpevd(job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork,
            iwork, liwork, info)
call zhpevd(job, uplo, n, ap, w, z, ldz, work, lwork, rwork, lrwork,
            iwork, liwork, info)
```

Fortran 95:

```
call hpevd(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、圧縮形式の複素エルミート行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = Z\Lambda Z^H$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような実対角行列、 Z は各列が固有ベクトル z_i であるような (複素) ユニタリー行列である。したがって、

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメーター

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap,work</i>	COMPLEX (chpevd の場合) DOUBLE COMPLEX (zhpevd の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上または下三角部分を圧縮形式で格納する。 <i>ap</i> のサイズは、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は <i>lwork</i> 以上。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 $ldz \geq 1$ 。 <i>job</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lwork \geq n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (chpevd の場合) DOUBLE PRECISION (zhpevd の場合) ワークスペース配列、次元は <i>lrwork</i> 以上。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 <i>job</i> ='N' で $n > 1$ の場合、 $lrwork \geq n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lrwork \geq 3n^2 + (4+2k) * n + 1$ (ここで、 k は、 $2^k \geq n$ を満たす最小の整数)。

$lwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $rwork$ 配列の最適サイズだけを計算し、その値を $rwork$ 配列の最初のエントリーとして返す。xerbla は $lwork$ に関するエラーメッセージを生成しない。

iwork INTEGER。
ワークスペース配列、次元は *liwork* 以上。

liwork INTEGER。配列 *iwork* の次元。次の制約がある。
 $n \leq 1$ の場合、 $liwork \geq 1$ 。
 $job = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。
 $job = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n + 2$ 。
 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリーとして返す。xerbla は *liwork* に関するエラーメッセージを生成しない。

出力パラメーター

w REAL (chpevd の場合)
DOUBLE PRECISION (zhpevd の場合)
配列、次元は $\max(1, n)$ 以上。
 $info = 0$ の場合、行列 *A* の固有値が昇順に格納される。
info も参照のこと。

z COMPLEX (chpevd の場合)
DOUBLE COMPLEX (zhpevd の場合)
配列、次元は (*ldz*, *)。z の第 2 次元は、1 以上 ($job = 'N'$ の場合) または $\max(1, n)$ 以上 ($job = 'V'$ の場合) でなければならない。
 $job = 'V'$ の場合、この配列は、*A* の固有ベクトルの入ったユニタリ行列 *Z* で上書きされる。 $job = 'N'$ の場合、*z* は参照されない。

ap 終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、*A* の対応する成分が上書きされる。

work(1) 終了時に、 $lwork > 0$ の場合、*work(1)* の実数部は *lwork* の必要最小サイズを返す。

rwork(1) 終了時に、 $lwork > 0$ の場合、*rwork(1)* は *lwork* の必要最小サイズを返す。

iwork(1) 終了時に、 $liwork > 0$ の場合、*iwork(1)* は *liwork* の必要最小サイズを返す。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = i$ の場合、このアルゴリズムが収束に失敗したことを示す。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。
 $info = -i$ の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>z</code>	サイズ (n,n) の行列 <code>Z</code> を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (<code>z</code> が存在する場合)、 <code>jobz = 'N'</code> (<code>z</code> が省略された場合)。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

このルーチンの実数版は、[?spevd](#) である。

フル形式の行列に対する [?heevd](#) と、帯形式の行列に関する [?hbevd](#) も参照のこと。

?spevx

圧縮形式の実対称行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call sspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z,
           ldz, work, iwork, ifail, info)
call dspevx(jobz, range, uplo, n, ap, vl, vu, il, iu, abstol, m, w, z,
           ldz, work, iwork, ifail, info)
```

Fortran 95:

```
call spevx(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
           [,abstol] [,info])
```

説明

圧縮形式の実対称行列 `A` の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半开区間 $v_l < \lambda_i \leq v_u$ の固有値 λ_i を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の <i>A</i> の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の <i>A</i> の下三角部分を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>ap, work</i>	REAL (sspevx の場合) DOUBLE PRECISION (dspevx の場合) 配列： <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 8n)$ 以上。
<i>vl, vu</i>	REAL (sspevx の場合) DOUBLE PRECISION (dspevx の場合) <i>range</i> ='V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v_l < v_u$ 。 <i>range</i> ='A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq i_l \leq i_u \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n=0$ の場合)。 <i>range</i> ='A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (sspevx の場合) DOUBLE PRECISION (dspevx の場合) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> ='N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = i_u - i_l + 1$ 。
<i>w, z</i>	REAL (sspevx の場合) DOUBLE PRECISION (dspevx の場合) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、選択された <i>A</i> の固有値が昇順に格納される。 <i>z</i> (ldz, *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *spevx* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n, m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

<i>v1</i>	この成分のデフォルト値は $v1 = -\text{HUGE}(v1)$ である。
<i>vu</i>	この成分のデフォルト値は $vu = \text{HUGE}(v1)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。*abstol* がゼロ以下の場合、 $\varepsilon * \|T\|_1$ は所定の場所で使用される (*T* は、*A* を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、*abstol* がゼロでなく、アンダーフローのしきい値の2倍、 $2 * \lambda_{\text{mach}}('S')$ に設定されたときである。このルーチンで *info* > 0 が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、*abstol* を $2 * \lambda_{\text{mach}}('S')$ に設定して、やり直してみる。

?hpevx

圧縮形式のエルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call chpevx(jobz, range, uplo, n, ap, v1, vu, il, iu, abstol, m, w, z,
           ldz, work, rwork, iwork, ifail, info)
call zhpevx(jobz, range, uplo, n, ap, v1, vu, il, iu, abstol, m, w, z,
           ldz, work, rwork, iwork, ifail, info)
```

Fortran 95:

```
call hpevx(a, w [,uplo] [,z] [,v1] [,vu] [,il] [,iu] [,m] [,ifail]
           [,abstol] [,info])
```

説明

このルーチンは、圧縮形式の複素エルミート行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ap</i> に圧縮形式の A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ap</i> に圧縮形式の A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>ap, work</i>	COMPLEX (chpevx の場合) DOUBLE COMPLEX (zhpevx の場合) 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上または下三角部分を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n)$ 以上。
<i>v1, vu</i>	REAL (chpevx の場合) DOUBLE PRECISION (zhpevx の場合) <i>range</i> ='V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> ='A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> ='I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n=0$ の場合)。 <i>range</i> ='A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chpevx の場合) DOUBLE PRECISION (zhpevx の場合) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。

<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (<i>chpevx</i> の場合) DOUBLE PRECISION (<i>zhpevx</i> の場合) ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>ap</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。すなわち、生成された三重対角行列の対角成分と非対角成分によって、 <i>A</i> の対応する成分が上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (<i>chpevx</i> の場合) DOUBLE PRECISION (<i>zhpevx</i> の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、選択された <i>A</i> の固有値が昇順に格納される。
<i>z</i>	COMPLEX (<i>chpevx</i> の場合) DOUBLE COMPLEX (<i>zhpevx</i> の場合) 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>ifail</i>	INTEGER。配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束に失敗した固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hpevx` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ap</code> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <code>A</code> を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>z</code>	サイズ (n,m) の行列 <code>Z</code> を格納する。ここで、値 n と m は有意である。
<code>ifail</code>	長さ (n) のベクトルを格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は <code>v1 = -HUGE(v1)</code> である。
<code>vu</code>	この成分のデフォルト値は <code>vu = HUGE(v1)</code> である。
<code>i1</code>	この引数のデフォルト値は、 <code>i1 = 1</code> である。
<code>iu</code>	この引数のデフォルト値は、 <code>iu = n</code> である。
<code>abstol</code>	この成分のデフォルト値は <code>abstol = 0.0_WP</code> である。
<code>jobz</code>	引数 <code>z</code> の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (<code>z</code> が存在する場合)、 <code>jobz = 'N'</code> (<code>z</code> が省略された場合)。 <code>ifail</code> が存在し、 <code>z</code> が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 <code>v1</code> 、 <code>vu</code> 、 <code>i1</code> 、および <code>iu</code> の存在に基づいて以下のように復元される。 <code>range = 'V'</code> (<code>v1</code> と <code>vu</code> のいずれかまたは両方が存在する場合)、 <code>range = 'I'</code> (<code>i1</code> と <code>iu</code> のいずれかまたは両方が存在する場合)、 <code>range = 'A'</code> (<code>v1</code> 、 <code>vu</code> 、 <code>i1</code> 、および <code>iu</code> がすべて存在しない場合)。 <code>v1</code> と <code>vu</code> のいずれかまたは両方が存在し、同時に <code>i1</code> と <code>iu</code> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。`abstol` がゼロ以下の場合、 $\epsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。固有値が最も正確に計算されるのは、`abstol` がゼロでなく、アンダーフローのしきい値の2倍、`2*?lamch('S')` に設定されたときである。このルーチンで `info > 0` が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、`abstol` を `2*?lamch('S')` に設定して、やり直してみる。

?sbev

帯形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
call dsbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, info)
```

Fortran 95:

```
call sbev(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、帯形式の実対称行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	REAL (ssbev の場合) DOUBLE PRECISION (dsbev の場合)。 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 A の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 3n-2)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

<i>w, z</i>	REAL (ssbev の場合) DOUBLE PRECISION (dsbev の場合) 配列: <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 (<i>kd</i> と <i>kd</i> +1) に返される。 <i>uplo</i> = 'L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbev* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ (<i>kd</i> +1, <i>n</i>) の配列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?hbev

帯形式のエルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork, info)
call zhbev(jobz, uplo, n, kd, ab, ldab, w, z, ldz, work, rwork, info)
```

Fortran 95:

```
call hbev(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、帯形式の複素エルミート行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	COMPLEX (chbev の場合) DOUBLE COMPLEX (zhbev の場合)。 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 A の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>jobz</i> = 'N' の場合、 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbev の場合) DOUBLE PRECISION (zhbev の場合) ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

出力パラメーター

<i>w</i>	REAL (chbev の場合) DOUBLE PRECISION (zhbev の場合) 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chbev の場合) DOUBLE COMPLEX (zhbev の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> に、行列 <i>A</i> の正規直交固有ベクトルが格納される。そのとき、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 (<i>kd</i> と <i>kd</i> +1) に返される。 <i>uplo</i> = 'L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hbev* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ (<i>kd</i> +1, <i>n</i>) の配列 <i>A</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?sbevd

分割統治アルゴリズムを使用して、帯形式の実対称行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork, iwork,
            liwork, info)
call dsbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork, iwork,
            liwork, info)
```

Fortran 95:

```
call sbevd(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、帯形式の実対称行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = \mathbf{Z}\mathbf{\Lambda}\mathbf{Z}^T$ を求めることができる。 $\mathbf{\Lambda}$ は対角成分が固有値 λ_i であるような対角行列、 \mathbf{Z} は各列が固有ベクトル z_i であるような直交行列である。したがって、

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメーター

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	REAL (ssbevd の場合) DOUBLE PRECISION (dsbevd の場合)。 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 A の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 <i>lwork</i> 以上でなければならない。

<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 <i>kd+1</i> 以上でなければならない。
<i>ldz</i>	INTEGER。 出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 <i>ldz</i> ≥ 1。 <i>job</i> ='V' の場合、 <i>ldz</i> ≥ max(1, <i>n</i>)。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 次の制約がある。 <i>n</i> ≤ 1 の場合、 <i>lwork</i> ≥ 1。 <i>job</i> ='N' で <i>n</i> > 1 の場合、 <i>lwork</i> ≥ 2 <i>n</i> 。 <i>job</i> ='V' で <i>n</i> > 1 の場合、 <i>lwork</i> ≥ 3 <i>n</i> ² +(4+2 <i>k</i>)* <i>n</i> +1 (ここで、 <i>k</i> は、 $2^k \geq n$ を満たす最小の整数)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列 \tilde{A} 最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。 配列 <i>iwork</i> の次元。 次の制約がある。 <i>n</i> ≤ 1 の場合、 <i>liwork</i> ≥ 1。 <i>job</i> ='N' で <i>n</i> > 1 の場合、 <i>liwork</i> ≥ 1。 <i>job</i> ='V' で <i>n</i> > 1 の場合、 <i>liwork</i> ≥ 5 <i>n</i> +2。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w, z</i>	REAL (ssbevd の場合) DOUBLE PRECISION (dsbevd の場合) 配列: <i>w</i> (*)、次元は max(1, <i>n</i>) 以上。 <i>info</i> = 0 の場合、行列 <i>A</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、1 以上 (<i>job</i> ='N' の場合) または max(1, <i>n</i>) 以上 (<i>job</i> ='V' の場合) でなければならない。 <i>job</i> ='V' の場合、この配列は、 <i>A</i> の固有ベクトルの入った直交行列 <i>Z</i> で上書きされる。すなわち、固有値 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>Z</i> の <i>i</i> 番目の列に格納される。 <i>job</i> ='N' の場合、 <i>z</i> は参照されない。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。
<i>work</i> (1)	終了時に、 <i>lwork</i> > 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = *i* の場合、このアルゴリズムが収束に失敗したことを示す。*i* は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbevd` のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。
 サイズ $(kd+1, n)$ の配列 *A* を格納する。

w 長さ (n) のベクトルを格納する。

z サイズ (n, n) の行列 *Z* を格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

jobz 引数 *z* の存在に基づいて以下のように復元される。
jobz = 'V' (*z* が存在する場合)、
jobz = 'N' (*z* が省略された場合)。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

このルーチンの複素数版は、[?hbevd](#) である。

フル形式の行列に対する [?syevd](#) と、圧縮形式の行列に関する [?spevd](#) も参照のこと。

?hbevd

分割統治アルゴリズムを使用して、帯形式の複素エルミート行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
           rwork, lrwork, iwork, liwork, info)
call zhbevd(job, uplo, n, kd, ab, ldab, w, z, ldz, work, lwork,
           rwork, lrwork, iwork, liwork, info)
```

Fortran 95:

```
call hbevd(a, w [,uplo] [,z] [,info])
```

説明

このルーチンは、帯形式の複素エルミート行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 A のスペクトル因子分解 $A = Z\Lambda Z^H$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような実対角行列、 Z は各列が固有ベクトル z_i であるような (複素) ユニタリー行列である。したがって、

$$Az_i = \lambda_i z_i \text{ ただし } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

入力パラメーター

<i>job</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER. 行列 A の次数 ($n \geq 0$).
<i>kd</i>	INTEGER. A の優対角成分または劣対角成分の数 ($kd \geq 0$).
<i>ab, work</i>	COMPLEX (chbevd の場合) DOUBLE COMPLEX (zhbevd の場合). 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 A の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 <i>lwork</i> 以上でなければならない。
<i>ldab</i>	INTEGER. <i>ab</i> のリーディング・ディメンジョンのサイズ。 <i>kd</i> +1 以上でなければならない。
<i>ldz</i>	INTEGER. 出力配列 z のリーディング・ディメンジョン。 次の制約がある。 <i>job</i> ='N' の場合、 <i>ldz</i> ≥ 1 。 <i>job</i> ='V' の場合、 <i>ldz</i> $\geq \max(1, n)$ 。
<i>lwork</i>	INTEGER. 配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 <i>lwork</i> ≥ 1 。 <i>job</i> ='N' で $n > 1$ の場合、 <i>lwork</i> $\geq n$ 。 <i>job</i> ='V' で $n > 1$ の場合、 <i>lwork</i> $\geq 2n^2$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (chbevd の場合) DOUBLE PRECISION (zhbevd の場合) ワークスペース配列、次元は <i>lrwork</i> 以上。

<i>lwork</i>	INTEGER。配列 <i>rwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 $job='N'$ で $n > 1$ の場合、 $lwork \geq n$ 。 $job='V'$ で $n > 1$ の場合、 $lwork \geq 3n^2 + (4+2k) * n + 1$ (ここで、 k は、 $2^k \geq n$ を満たす最小の整数)。 $lwork = -1$ の場合、ワークスペースのクエリーとみなされ、 ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリーとして返す。xerbla は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。次の制約がある。 $job='N'$ または $n \leq 1$ の場合、 $liwork \geq 1$ 。 $job='V'$ で $n > 1$ の場合、 $liwork \geq 5n + 2$ 。 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、 ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。xerbla は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>w</i>	REAL (chbevd の場合) DOUBLE PRECISION (zhbevd の場合) 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、行列 <i>A</i> の固有値が昇順に格納される。 $info$ も参照のこと。
<i>z</i>	COMPLEX (chbevd の場合) DOUBLE COMPLEX (zhbevd の場合) 配列、次元は $(ldz, *)$ 。 <i>z</i> の第 2 次元は、1 以上 ($job='N'$ の場 合) または $\max(1, n)$ 以上 ($job='V'$ の場合) でなければならない。 $job='V'$ の場合、この配列は、 <i>A</i> の固有ベクトルの入ったユニ タリ行列 <i>Z</i> で上書きされる。すなわち、固有値 $w(i)$ に対応す る固有ベクトルが、 <i>Z</i> の <i>i</i> 番目の列に格納される。 $job='N'$ の場合、 <i>z</i> は参照されない。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上 書きされる。
<i>work(1)</i>	終了時に、 $lwork > 0$ の場合、 <i>work(1)</i> の実数部は <i>lwork</i> の必要 最小サイズを返す。
<i>rwork(1)</i>	終了時に、 $lwork > 0$ の場合、 <i>rwork(1)</i> は <i>lwork</i> の必要最小 サイズを返す。
<i>iwork(1)</i>	終了時に、 $liwork > 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小 サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = i$ の場合、このアルゴリズムが収束に失敗したことを示

す。 i は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。

$info = -i$ の場合、 i 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbevd` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。 サイズ $(kd+1, n)$ の配列 A を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>z</code>	サイズ (n, n) の行列 Z を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

このルーチンの実数版は、[?sbevd](#) である。

フル形式の行列に対する [?heevd](#) と、圧縮形式の行列に関する [?hpevd](#) も参照のこと。

?sbevz

帯形式の実対称行列の固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call ssbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu,
           abstol, m, w, z, ldz, work, iwork, ifail, info)
call dsbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu,
           abstol, m, w, z, ldz, work, iwork, ifail, info)
```

Fortran 95:

```
call sbevz(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]
           [,abstol] [,info])
```

説明

このルーチンは、帯形式の実対称行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> = 'L' の場合、 <i>ab</i> に A の下三角部分を格納する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	REAL (ssbevz の場合) DOUBLE PRECISION (dsbevz の場合)。 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 A の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, 7n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 <i>kd</i> +1 以上でなければならない。
<i>v1, vu</i>	REAL (ssbevz の場合) DOUBLE PRECISION (dsbevz の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。

<i>abstol</i>	REAL (chpevx の場合) DOUBLE PRECISION (zhpevx の場合) 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldq, ldz</i>	INTEGER。出力配列 <i>q</i> と <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldq \geq 1, ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldq \geq \max(1, n)$ および $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>q</i>	REAL (ssbevx の場合) DOUBLE PRECISION (dsbevx の場合)。 配列、次元は (<i>ldz</i> , <i>n</i>)。 <i>jobz</i> = 'V' の場合、 $n \times n$ の直交行列が三重対角形式への縮退に使用される。 <i>jobz</i> = 'N' の場合、配列 <i>q</i> は参照されない。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z</i>	REAL (ssbevx の場合) DOUBLE PRECISION (dsbevx の場合) 配列： <i>w</i> (*)、サイズは $\max(1, n)$ 以上。 <i>w</i> の先頭 <i>m</i> 成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>ab</i>	終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。 <i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、 <i>ab</i> の 2 つの行 (<i>kd</i> と <i>kd+1</i>) に返される。 <i>uplo</i> = 'L' の場合、 <i>T</i> の対角成分と第一劣対角成分が、 <i>ab</i> の先頭 2 行に返される。
<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の

成分はゼロになる。 $info > 0$ であれば、収束に失敗した固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。
info = *i* の場合、*i* 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 *ifail* に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbevxx* のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。
 サイズ $(kd+1, n)$ の配列 *A* を格納する。

w 長さ (n) のベクトルを格納する。

z サイズ (n, m) の行列 *Z* を格納する。ここで、値 *n* と *m* は有意である。

ifail 長さ (n) のベクトルを格納する。

q サイズ (n, n) の行列 *Q* を格納する。

uplo 'U' または 'L' でなければならない。デフォルト値は 'U'。

v1 この成分のデフォルト値は $v1 = -HUGE(v1)$ である。

vu この成分のデフォルト値は $vu = HUGE(v1)$ である。

il この引数のデフォルト値は、 $il = 1$ である。

iu この引数のデフォルト値は、 $iu = n$ である。

abstol この成分のデフォルト値は $abstol = 0.0_WP$ である。

jobz 引数 *z* の存在に基づいて以下のように復元される。
 jobz = 'V' (*z* が存在する場合)、
 jobz = 'N' (*z* が省略された場合)。
 ifail または *q* のいずれかが存在し、*z* が省略された場合、エラー条件がセットされる。

range 引数 *v1*、*vu*、*il*、および *iu* の存在に基づいて以下のように復元される。
 range = 'V' (*v1* と *vu* のいずれかまたは両方が存在する場合)、
 range = 'I' (*il* と *iu* のいずれかまたは両方が存在する場合)、
 range = 'A' (*v1*、*vu*、*il*、および *iu* がすべて存在しない場合)。
 v1 と *vu* のいずれかまたは両方が存在し、同時に *il* と *iu* のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?hbevz

帯形式のエルミート行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call chbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu,
            abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)
call zhbevz(jobz, range, uplo, n, kd, ab, ldab, q, ldq, vl, vu, il, iu,
            abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)
```

Fortran 95:

```
call hbevz(a, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,q]
           [,abstol] [,info])
```

説明

このルーチンは、帯形式の複素エルミート行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、 <i>ab</i> に A の上三角部分を格納する。 <i>uplo</i> ='L' の場合、 <i>ab</i> に A の下三角部分を格納する。

<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>kd</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($kd \geq 0$)。
<i>ab, work</i>	COMPLEX (chbevz の場合) DOUBLE COMPLEX (zhbevz の場合)。 配列： <i>ab(ldab,*)</i> は、(<i>uplo</i> の値に従って) エルミート行列 <i>A</i> の上または下三角部分を帯形式で格納する配列である。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(*)</i> は、ワークスペース配列である。 <i>work</i> の次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョンのサイズ。 $kd+1$ 以上でなければならない。
<i>v1, vu</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合)。 <i>range = 'V'</i> の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range = 'A'</i> または ' <i>I</i> ' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range = 'I'</i> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range = 'A'</i> または ' <i>V</i> ' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合)。 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。
<i>ldq, ldz</i>	INTEGER。出力配列 <i>q</i> と <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldq \geq 1, ldz \geq 1$ 。 <i>jobz = 'V'</i> の場合、 $ldq \geq \max(1, n)$ および $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbevz の場合) DOUBLE PRECISION (zhbevz の場合) ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>q</i>	COMPLEX (chbevz の場合) DOUBLE COMPLEX (zhbevz の場合)。 配列、次元は (<i>ldz</i> , <i>n</i>)。
----------	---

	<p><i>jobz</i> = 'V' の場合、$n \times n$ のユニタリ行列が三重対角形式への縮退に使用される。</p> <p><i>jobz</i> = 'N' の場合、配列 <i>q</i> は参照されない。</p>
<i>m</i>	<p>INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。</p> <p><i>range</i> = 'A' の場合、$m = n$、<i>range</i> = 'I' の場合、$m = iu - il + 1$。</p>
<i>w</i>	<p>REAL (<i>chbev</i>x の場合)</p> <p>DOUBLE PRECISION (<i>zhbev</i>x の場合)</p> <p>配列、次元は $\max(1, n)$ 以上。</p> <p>先頭の <i>m</i> 個の成分に、行列 <i>A</i> の選択された固有値が昇順に格納される。</p>
<i>z</i>	<p>COMPLEX (<i>chbev</i>x の場合)</p> <p>DOUBLE COMPLEX (<i>zhbev</i>x の場合)。</p> <p>配列 <i>z</i>(<i>ldz</i>, *)。 <i>z</i> の第 2 次元は、$\max(1, m)$ 以上でなければならない。</p> <p><i>jobz</i> = 'V' の場合、<i>info</i> = 0 であれば、<i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、<i>w</i>(<i>i</i>) に対応する固有ベクトルが、<i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルが収束できない場合、<i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。</p> <p><i>jobz</i> = 'N' の場合、<i>z</i> は参照されない。</p> <p>注：配列 <i>z</i> には、$\max(1, m)$ 以上の列が提供されなければならない。<i>range</i> = 'V' の場合、<i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。</p>
<i>ab</i>	<p>終了時に、三重対角形式に縮退させたときに生成された値で上書きされる。</p> <p><i>uplo</i> = 'U' の場合、三重対角行列 <i>T</i> の第一優対角成分と対角成分が、<i>ab</i> の 2 つの行 (<i>kd</i> と <i>kd</i>+1) に返される。<i>uplo</i> = 'L' の場合、<i>T</i> の対角成分と第一劣対角成分が、<i>ab</i> の先頭 2 行に返される。</p>
<i>ifail</i>	<p>INTEGER。</p> <p>配列、次元は $\max(1, n)$ 以上。</p> <p><i>jobz</i> = 'V' の場合、<i>info</i> = 0 であれば、<i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。<i>info</i> > 0 であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。</p> <p><i>jobz</i> = 'N' の場合、<i>ifail</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。</p> <p><i>info</i> = 0 の場合、正常に終了したことを示す。</p> <p><i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。</p> <p><i>info</i> = <i>i</i> の場合、<i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hbev*x のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(kd+1, n)$ の配列 <i>A</i> を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, m) の行列 <i>Z</i> を格納する。ここで、値 n と m は有意である。
<i>ifail</i>	長さ (n) のベクトルを格納する。
<i>q</i>	サイズ (n, n) の行列 <i>Q</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は $vu = HUGE(v1)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> または <i>q</i> のいずれかが存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>il</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>il</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?stev

実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sstev(jobz, n, d, e, z, ldz, work, info)
call dstev(jobz, n, d, e, z, ldz, work, info)
```

Fortran 95:

```
call stev(d, e [,z] [,info])
```

説明

このルーチンは、実対称三重対角行列 A の固有値と (オプションで) 固有ベクトルをすべて計算する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>d, e, work</i>	REAL (sstev の場合) DOUBLE PRECISION (dstev の場合)。 配列: <i>d</i> (*) には、三重対角行列 A の n 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、三重対角行列 A の $n-1$ 個の劣対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。この配列の n 番目の成分は、ワークスペースとして使用される。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、 $\max(1, 2n-2)$ 以上でなければならない。 <i>jobz</i> = 'N' の場合、 <i>work</i> は参照されない。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

<i>d</i>	終了時に、 <i>info</i> = 0 の場合、行列 A の固有値が昇順に格納される。
<i>z</i>	REAL (sstev の場合) DOUBLE PRECISION (dstev の場合) 配列、次元は (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、行列 A の正規直交固有ベ

	クトルが z に格納される。すなわち、 $d(i)$ に対応する固有ベクトルが、 z の i 番目の列に格納される。 $job='N'$ の場合、 z は参照されない。
e	終了時に、中間結果で上書きされる。
$info$	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。 $info=i$ の場合、このアルゴリズムが収束に失敗したことを示す。 e の i 個の成分が、ゼロに収束しなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `stev` のインターフェイスの詳細を以下に示す。

d	長さ (n) のベクトルを格納する。
e	長さ (n) のベクトルを格納する。
z	サイズ (n, n) の行列 Z を格納する。
$jobz$	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。

?stevd

分割統治アルゴリズムを使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sstevd(job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
call dstevd(job, n, d, e, z, ldz, work, lwork, iwork, liwork, info)
```

Fortran 95:

```
call stevd(d, e [,z] [,info])
```

説明

このルーチンは、実対称三重対角行列 T の固有値と (オプションで) 固有ベクトルをすべて計算する。言い換えると、 T のスペクトル因子分解 $T = \Lambda Z^T$ を求めることができる。 Λ は対角成分が固有値 λ_i であるような対角行列、 Z は各列が固有ベクトル z_i であるような直交行列である。したがって、

$$Tz_i = \lambda_i z_i \quad \text{for } i = 1, 2, \dots, n$$

このルーチンは、固有ベクトルも要求された場合は、分割統治アルゴリズムを使用して、固有値と固有ベクトルを計算する。ただし、固有値のみ要求された場合は、Pal-Walker-Kahan 変形版の QL または QR アルゴリズムを使用して、固有値を計算する。

このルーチンの複素数版はない。

入力パラメーター

<i>job</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>job</i> ='N' の場合、固有値のみを計算する。 <i>job</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>n</i>	INTEGER。行列 T の次数 ($n \geq 0$)。
<i>d, e, work</i>	REAL (sstevd の場合) DOUBLE PRECISION (dstevd の場合) 配列: <i>d</i> (*) には、三重対角行列 T の n 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>e</i> (*) には、 T の $n-1$ 個の非対角成分を格納する。 <i>e</i> の次元は、 $\max(1, n)$ 以上でなければならない。この配列の n 番目の成分は、ワークスペースとして使用される。 <i>work</i> (*) は、ワークスペース配列。 <i>work</i> の次元は、 <i>lwork</i> 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ (<i>job</i> ='N' の場合)。 $ldz \geq \max(1, n)$ (<i>job</i> ='V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 <i>job</i> ='N' または $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2 + (3+2k) * n + 1$ (ここで、 k は、 $2^k \geq n$ を満たす最小の整数)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、xerbla は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は <i>liwork</i> 以上。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 次の制約がある。 <i>job</i> ='N' または $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>job</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n + 2$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。xerbla は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>d</i>	終了時に、 <i>info</i> = 0 の場合、行列 <i>T</i> の固有値が昇順に格納される。 <i>info</i> も参照のこと。
<i>z</i>	REAL (sstevd の場合) DOUBLE PRECISION (dstevd の場合) 配列、次元は (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、1 以上 (<i>job</i> = 'N' の場合) または $\max(1, n)$ 以上 (<i>job</i> = 'V' の場合) でなければならない。 <i>job</i> = 'V' の場合、 <i>T</i> の固有ベクトルが格納された直交行列 <i>Z</i> で上書きされる。 <i>job</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>e</i>	終了時に、中間結果で上書きされる。
<i>work</i> (1)	終了時に、 <i>lwork</i> > 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>liwork</i> > 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> の場合、このアルゴリズムが収束に失敗したことを示す。 <i>i</i> は、中間の三重対角形式で、ゼロに収束しなかった成分の個数を表す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stevd* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

アプリケーション・ノート

計算で求めた固有値と固有ベクトルは、正確には、 $\|E\|_2 = O(\epsilon) \|T\|_2$ のような行列 $T + E$ (ϵ はマシン精度) のものである。

λ_i が正確な固有値で、 μ_i が対応する計算で求めた値の場合、次のようになる。

$$|\mu_i - \lambda_i| \leq c(n)\epsilon \|T\|_2,$$

$c(n)$ は、 n の漸増関数である。

z_i が対応する正確な固有ベクトルで、 w_i が対応する計算ベクトルの場合、それらの間の角度 $\theta(z_i, w_i)$ は次のようになる。

$$\theta(z_i, w_i) \leq c(n)\varepsilon \|T\|_2 / \min_{i \neq j} |\lambda_i - \lambda_j|$$

このように、計算された固有ベクトルの精度は、それが対応する固有値と、他のすべての固有値との隔たりに依存するのがわかる。

?stevx

実対称三重対角行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call sstevx(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz,
           work, iwork, ifail, info)
call dstevx(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz,
           work, iwork, ifail, info)
```

Fortran 95:

```
call stevx(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail] [,abstol]
           [,info])
```

説明

このルーチンは、実対称三重対角行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> ='I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>d, e, work</i>	REAL (sstevx の場合) DOUBLE PRECISION (dstevx の場合)。 配列: <i>d</i> (*) には、三重対角行列 A の n 個の対角成分を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。

	<p>$e(*)$ には、A の $n-1$ 個の劣対角成分を格納する。 e の次元は、$\max(1, n)$ 以上でなければならない。この配列の n 番目の成分は、ワークスペースとして使用される。</p> <p>$work(*)$ は、ワークスペース配列。 $work$ の次元は、$\max(1, 5n)$ 以上でなければならない。</p>
vl, vu	<p>REAL (sstevx の場合) DOUBLE PRECISION (dstevx の場合)。 $range = 'V'$ の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $vl < vu$。 $range = 'A'$ または $'I'$ の場合、vl と vu は参照されない。</p>
il, iu	<p>INTEGER。 $range = 'I'$ の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 $range = 'A'$ または $'V'$ の場合、il と iu は参照されない。</p>
$abstol$	<p>REAL (sstevx の場合) DOUBLE PRECISION (dstevx の場合)。 各固有値に許される絶対誤差許容値。誤差許容値の詳細は、「アプリケーション・ノート」を参照。</p>
ldz	<p>INTEGER。出力配列 z のリーディング・ディメンジョン。$ldz \geq 1$。 $jobz = 'V'$ の場合、$ldz \geq \max(1, n)$。</p>
$iwork$	<p>INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。</p>

出力パラメーター

m	<p>INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 $range = 'A'$ の場合、$m = n$、$range = 'I'$ の場合、$m = iu - il + 1$。</p>
w, z	<p>REAL (sstevx の場合) DOUBLE PRECISION (dstevx の場合)。 配列： $w(*)$、次元は $\max(1, n)$ 以上。 w の先頭 m 成分に、行列 A の選択された固有値が昇順に格納される。</p> <p>$z(ldz, *)$。z の第 2 次元は、$\max(1, m)$ 以上でなければならない。 $jobz = 'V'$ の場合、$info = 0$ であれば、z の先頭の m 列に、行列 A の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、$w(i)$ に対応する固有ベクトルが、z の i 番目の列に格納される。固有ベクトルが収束できない場合、z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは $ifail$ に返される。 $jobz = 'N'$ の場合、z は参照されない。 注：配列 z には、$\max(1, m)$ 以上の列が提供されなければならない。$range = 'V'$ の場合、m の正確な値が事前にわからないため、上限値を使用する必要がある。</p>

<i>d, e</i>	終了時に、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。
<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>ifail</i> の先頭から <i>m</i> 個の成分はゼロになる。 <i>info</i> > 0 であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> の場合、 <i>i</i> 番目の固有ベクトルが収束に失敗した。その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *stevx* のインターフェイスの詳細を以下に示す。

<i>d</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>e</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n, m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE(<i>v1</i>) である。
<i>vu</i>	この成分のデフォルト値は <i>vu</i> = HUGE(<i>v1</i>) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。
<i>iu</i>	この引数のデフォルト値は、 <i>iu</i> = <i>n</i> である。
<i>abstol</i>	この成分のデフォルト値は <i>abstol</i> = 0.0_WP である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロまたは負の場合、代わりに $\varepsilon * \|A\|_1$ を使用する。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかが収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?stevr

「比較的安定な表現」を使用して、実対称三重対角行列の固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call sstevr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz,
            isuppz, work, lwork, iwork, liwork, info)
call dstevr(jobz, range, n, d, e, vl, vu, il, iu, abstol, m, w, z, ldz,
            isuppz, work, lwork, iwork, liwork, info)
```

Fortran 95:

```
call stevr(d, e, w [,z] [,vl] [,vu] [,il] [,iu] [,m] [,isuppz] [,abstol]
           [,info])
```

説明

このルーチンは、実対称三重対角行列 T の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

可能であれば、?stevr は [sstegr/dstegr](#) を呼び出し、「比較的安定な表現」を使用して、固有スペクトルを計算する。?stegr は、固有値の計算には *dqds* アルゴリズムを使用するが、直交固有ベクトルは、種々の「良好な」 LDL^T 表現 (「比較的安定な表現」とも呼ばれる) から計算する。グラム-シュミットの直交化はできる限り避ける。アルゴリズムの各ステップを詳しく説明すると次のようになる。 T のまだ縮退されていない i 番目のブロックに対して、

- (a) $L_i D_i L_i^T$ が比較的安定な表現になるように、 $T - \sigma_i = L_i D_i L_i^T$ を計算する。
- (b) *dqds* アルゴリズムによって、 $L_i D_i L_i^T$ の固有値 λ_j を高い相対精度で計算する。
- (c) 値が近い固有値からなるクラスターが存在する場合は、そのクラスターに近い σ_i を「選択」し、ステップ (a) に戻る。
- (d) $L_i D_i L_i^T$ の近似的な固有値 λ_j に対する固有ベクトルを、階数表示のツイスト分解によって求める。

出力値に必要な精度は、入力パラメーター $abstol$ で指定できる。

ルーチン `?stevr` は、IEEE-754 の浮動小数点標準に準拠しているマシンにおいてフルスペクトルが要求された場合、[sstegr/dstegr](#) を呼び出す。`?stevr` は、IEEE-754 に準拠しないマシンの場合や、部分スペクトルが要求された場合、[sstebz/dstebz](#) と [sstein/dstein](#) を呼び出す。

入力パラメーター

<code>jobz</code>	CHARACTER*1. 'N' または 'V' でなければならない。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>range</code>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <code>range = 'A'</code> の場合、固有値をすべて計算する。 <code>range = 'V'</code> の場合、半开区間 $v_l < \lambda_i \leq v_u$ の固有値 λ_i を計算する。 <code>range = 'I'</code> の場合、インデックス i_l から i_u の固有値を計算する。 <code>range = 'V'</code> または 'I' で、 $i_u - i_l < n - 1$ の場合、 <code>sstebz/dstebz</code> と <code>sstein/dstein</code> を呼び出す。
<code>n</code>	INTEGER。行列 T の次数 ($n \geq 0$)。
<code>d, e, work</code>	REAL (<code>sstevr</code> の場合) DOUBLE PRECISION (<code>dstevr</code> の場合)。 配列: <code>d(*)</code> には、三重対角行列 T の n 個の対角成分を格納する。 <code>d</code> の次元は、 $\max(1, n)$ 以上でなければならない。 <code>e(*)</code> には、 A の $n-1$ 個の劣対角成分を格納する。 <code>e</code> の次元は、 $\max(1, n)$ 以上でなければならない。この配列の n 番目の成分は、ワークスペースとして使用される。 <code>work(lwork)</code> は、ワークスペース配列である。
<code>v1, vu</code>	REAL (<code>sstevr</code> の場合) DOUBLE PRECISION (<code>dstevr</code> の場合)。 <code>range = 'V'</code> の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v_l < v_u$ 。 <code>range = 'A'</code> または 'I' の場合、 <code>v1</code> と <code>vu</code> は参照されない。
<code>il, iu</code>	INTEGER。 <code>range = 'I'</code> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq i_l \leq i_u \leq n$ ($n > 0$ の場合)。 $i_l = 1$ かつ $i_u = 0$ ($n = 0$ の場合)。 <code>range = 'A'</code> または 'V' の場合、 <code>il</code> と <code>iu</code> は参照されない。
<code>abstol</code>	REAL (<code>ssyevr</code> の場合) DOUBLE PRECISION (<code>dsyevr</code> の場合)。 各固有値および固有ベクトルに許される絶対誤差許容値。 <code>jobz = 'V'</code> の場合、出力される固有値と固有ベクトルの誤差ノルムが <code>abstol</code> 以内になる。また、異なる固有ベクトル間のドット積も <code>abstol</code> 以内になる。 $abstol < n\epsilon \ T\ _1$ の場合、 $n\epsilon \ T\ _1$ が代わ

	りに使用される (ϵ はマシン精度)。固有値は、 <i>abstol</i> とは無関係に、 $\epsilon \ T\ _1$ の精度で計算される。高い相対精度が必要な場合は、 <i>abstol</i> を ?lamch('S') に設定する。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ (<i>jobz</i> = 'N' の場合)。 $ldz \geq \max(1, n)$ (<i>jobz</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $lwork \geq \max(1, 20n)$ <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq \max(1, 10n)$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエンタリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。
出力パラメーター	
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z</i>	REAL (<i>sstevr</i> の場合) DOUBLE PRECISION (<i>dstevr</i> の場合)。 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭 <i>m</i> 成分に、行列 <i>T</i> の選択された固有値が昇順に格納される。 <i>z</i> (<i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>T</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。 注：配列 <i>z</i> には、 $\max(1, m)$ 以上の列が提供されなければならない。 <i>range</i> = 'V' の場合、 <i>m</i> の正確な値が事前にわからないため、上限値を使用する必要がある。
<i>d, e</i>	終了時に、固有値を計算するときのオーバーフローまたはアンダーフローを避けるために、選択された定係数が掛けられることがある。
<i>isuppz</i>	INTEGER。 配列、次元は $2 * \max(1, m)$ 以上。

z に格納されている固有ベクトルのサポート情報、すなわち z に格納されている非ゼロの成分を示すインデックス。 i 番目の固有ベクトルは、 $isuppz(2i-1)$ から $isuppz(2i)$ までの成分のみ非ゼロである。

$range = 'A'$ または $range = 'I'$ で $iu - il = n - 1$ の場合にのみ有効である。

$work(1)$	終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。
$iwork(1)$	終了時に、 $info = 0$ の場合、 $iwork(1)$ は $liwork$ の必要最小サイズを返す。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ の場合、内部エラーが発生したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `stevr` のインターフェイスの詳細を以下に示す。

d	長さ (n) のベクトルを格納する。
e	長さ (n) のベクトルを格納する。
w	長さ (n) のベクトルを格納する。
z	サイズ (n, m) の行列 Z を格納する。ここで、値 n と m は有意である。
$isuppz$	長さ ($2 * m$) のベクトルを格納する。ここで、値 ($2 * m$) は有意である。
$v1$	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
vu	この成分のデフォルト値は $vu = HUGE(v1)$ である。
il	この引数のデフォルト値は、 $il = 1$ である。
iu	この引数のデフォルト値は、 $iu = n$ である。
$abstol$	この成分のデフォルト値は $abstol = 0.0_WP$ である。
$jobz$	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。 $ifail$ が存在し、 z が省略された場合、エラー条件がセットされる。
$range$	引数 $v1$ 、 vu 、 il 、および iu の存在に基づいて以下のように復元される。 $range = 'V'$ ($v1$ と vu のいずれかまたは両方が存在する場合)、 $range = 'I'$ (il と iu のいずれかまたは両方が存在する場合)、 $range = 'A'$ ($v1$ 、 vu 、 il 、および iu がすべて存在しない場合)。 $v1$ と vu のいずれかまたは両方が存在し、同時に il と iu のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

通常の実行で NaN と無限大が発生するため、NaN と無限大が IEEE 標準のデフォルト方法以外で処理される環境では、浮動小数点例外によって異常終了する可能性がある。

非対称固有値問題

このセクションでは、非対称固有値問題を解くために使用する LAPACK ドライバールーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

表 4-11 に Fortran-77 インターフェイスのすべてのドライバールーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-11 非対称固有値問題を解くためのドライバールーチン

ルーチン名	機能
?gees	一般行列の固有値と Schur 因子分解を計算し、指定された固有値が Schur 形式の左上にくるように因子分解を順序付けする。
?geesx	一般行列の固有値と Schur 因子分解を計算し、因子分解の順序付けと、条件数の逆数の計算を実行する。
?geev	一般行列の固有値と左 / 右の固有ベクトルを計算する。
?geevx	あらかじめ行列を平衡化して、一般行列の固有値と左 / 右の固有ベクトル、固有値と右固有ベクトルに対する条件数の逆数を計算する。

?gees

一般行列の固有値と Schur 因子分解を計算し、指定された固有値が Schur 形式の左上にくるように因子分解を順序付けする。

構文

Fortran 77:

```
call sgees(jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs, work,
           lwork, bwork, info)
call dgees(jobvs, sort, select, n, a, lda, sdim, wr, wi, vs, ldvs, work,
           lwork, bwork, info)
call cgees(jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work,
           lwork, rwork, bwork, info)
call zgees(jobvs, sort, select, n, a, lda, sdim, w, vs, ldvs, work,
           lwork, rwork, bwork, info)
```

Fortran 95:

```
call gees(a, wr, wi [,vs] [,select] [,sdim] [,info])
call gees(a, w [,vs] [,select] [,sdim] [,info])
```

説明

このルーチンは、 $n \times n$ の実 / 複素非対称行列 A について、固有値、実数 Schur 形式 T と (オプションで) Schur ベクトル Z の行列を計算する。これにより、Schur 因子分解 $A = TZT^H$ が提供される。

また、このルーチンは、指定された固有値が左上にくるように、実数 Schur/Schur 形式の対角上の固有値を順序付けることもできる。Z の先頭の列は、指定された固有値に対応する不変部分空間の直交基を形成する。

1×1 ブロックと 2×2 ブロックを持つ上準三角行列の場合、実数行列は、実数 Schur 形式となる。2×2 ブロックは、次の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix}$$

ここで、 $b*c < 0$ 。このようなブロックの固有値は、 $a \pm \sqrt{bc}$ となる。

複素行列は、上三角の場合は Schur 形式になる。

入力パラメーター

<i>jobvs</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvs</i> ='N' の場合、Schur ベクトルは計算されない。 <i>jobvs</i> ='V' の場合、Schur ベクトルは計算される。
<i>sort</i>	CHARACTER*1。'N' または 'S' でなければならない。 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。 <i>sort</i> ='N' の場合、固有値は順序付けされない。 <i>sort</i> ='S' の場合、固有値は順序付けされる (<i>select</i> を参照)。
<i>select</i>	実数型の場合、2つの REAL 引数の LOGICAL FUNCTION。 複素数型の場合、1つの COMPLEX 引数の LOGICAL FUNCTION。 <i>select</i> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。 <i>sort</i> ='S' の場合、 <i>select</i> は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。 <i>sort</i> ='N' の場合、 <i>select</i> は参照されない。 実数型の場合： <i>select</i> (<i>wr</i> (<i>j</i>), <i>wi</i> (<i>j</i>)) が真の場合、固有値 $wr(j) + \sqrt{-1} * wi(j)$ が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。そのため、順序付けを実行した後、選択した複素数の固有値は、 <i>select</i> (<i>wr</i> (<i>j</i>), <i>wi</i> (<i>j</i>))=.TRUE. を満たさないときがある。この場合、 <i>info</i> には、 <i>n</i> +2 が設定される (以下の <i>info</i> を参照)。 複素数型の場合： <i>select</i> (<i>w</i> (<i>j</i>)) が真の場合、固有値 <i>w</i> (<i>j</i>) が選択される。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (<i>sgees</i> の場合) DOUBLE PRECISION (<i>dgees</i> の場合) COMPLEX (<i>cgees</i> の場合) DOUBLE COMPLEX (<i>zgees</i> の場合)。 配列： <i>a</i> (<i>lda</i> ,*) は、 $n \times n$ の行列 A を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。

<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvs</i>	INTEGER。出力配列 <i>vs</i> のリーディング・ディメンジョン。 次の制約がある。 $ldvs \geq 1$ 。 $ldvs \geq \max(1, n)$ (<i>jobvs</i> = 'V' の場合)。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $lwork \geq \max(1, 3n)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (<i>cgees</i> の場合) DOUBLE PRECISION (<i>zgees</i> の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。 複素数型でのみ使用される。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 <i>sort</i> = 'N' の場合は参照されない。

出力パラメーター

<i>a</i>	終了時に、実数 Schur/Schur 形式 <i>T</i> で上書きされる。
<i>sdim</i>	INTEGER。 <i>sort</i> = 'N' の場合、 <i>sdim</i> = 0 である。 <i>sort</i> = 'S' の場合、 <i>sdim</i> は、 <i>select</i> が真の (ソート後の) 固有値の数に等しくなる。 実数型の場合、どちらかの固有値について <i>select</i> が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。
<i>wr, wi</i>	REAL (<i>sgees</i> の場合) DOUBLE PRECISION (<i>dgees</i> の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 出力実数 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で、計算された固有値の実数部と虚数部をそれぞれ格納する。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。
<i>w</i>	COMPLEX (<i>cgees</i> の場合) DOUBLE COMPLEX (<i>zgees</i> の場合)。 配列、次元は $\max(1, n)$ 以上。 計算された固有値が格納される。一連の固有値は、出力 Schur 形式 <i>T</i> の対角上に格納されるのと同じ順序で格納される。
<i>vs</i>	REAL (<i>sgees</i> の場合) DOUBLE PRECISION (<i>dgees</i> の場合) COMPLEX (<i>cgees</i> の場合)

DOUBLE COMPLEX (zgees の場合)。
 配列 $vs(ldvs, *)$ 。 vs の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$jobvs = 'V'$ の場合、 vs には、Schur ベクトルの直交 / ユニタリー行列 Z が格納される。
 $jobvs = 'N'$ の場合、 vs は参照されない。

$work(1)$ 終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。

$info$ INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。
 $info = i$ 、かつ $i \leq n$:
 QR アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、 wr と wi (実数型の場合) または w (複素数型の場合) の $1:i1o-1$ と $i+1:n$ の成分に格納される。 $jobvs = 'V'$ の場合、 vs には、 A をその部分収束 Schur 形式に縮退させる行列を格納する。
 $i = n+1$:
 いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった (非常に悪い条件が問題)。
 $i = n+2$:
 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が $select = .TRUE.$ を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gees` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
wr	長さ (n) のベクトルを格納する。実数型でのみ使用される。
wi	長さ (n) のベクトルを格納する。実数型でのみ使用される。
w	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
vs	サイズ (n, n) の行列 VS を格納する。
$jobvs$	引数 vs の存在に基づいて以下のように復元される。 $jobvs = 'V'$ (vs が存在する場合)、 $jobvs = 'N'$ (vs が省略された場合)。

`sort` 引数 `select` の存在に基づいて以下のように復元される。
`sort = 'S'` (`select` が存在する場合)、
`sort = 'N'` (`select` が省略された場合)。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

?geesx

一般行列の固有値と Schur 因子分解を計算し、因子分解の順序付けと、条件数の逆数の計算を実行する。

構文

Fortran 77:

```
call sgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs,
           ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork, info)
call dgeesx(jobvs, sort, select, sense, n, a, lda, sdim, wr, wi, vs,
           ldvs, rconde, rcondv, work, lwork, iwork, liwork, bwork, info)
call cgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs,
           rconde, rcondv, work, lwork, rwork, bwork, info)
call zgeesx(jobvs, sort, select, sense, n, a, lda, sdim, w, vs, ldvs,
           rconde, rcondv, work, lwork, rwork, bwork, info)
```

Fortran 95:

```
call geesx(a, wr, wi [,vs] [,select] [,sdim] [,rconde] [,rcondv]
           [,info])
call geesx(a, w [,vs] [,select] [,sdim] [,rconde] [,rcondv] [,info])
```

説明

このルーチンは、 $n \times n$ の実/複素非対称行列 A について、固有値、実数 Schur/Schur 形式 T と (オプションで) Schur ベクトル Z の行列を計算する。これにより、Schur 因子分解 $A = ZTZ^H$ が提供される。

このルーチンは、指定された固有値が左上にくるように、実数 Schur/Schur 形式の対角上の固有値を順序付けもできる。また、指定した固有値の平均に対する条件数の逆数 (`rconde`) を計算したり、指定した固有値に対応する右不変部分空間に対する条件数の逆数 (`rcondv`) も計算できる。 Z の先頭の列は、この不変部分空間の直交基を形成する。

条件数の逆数 `rconde`、`rcondv` の詳細は、4.10 の [\[LUG\]](#) を参照のこと (これらの大きさはそれぞれ、`s` と `sep` と呼ぶ)。

1×1ブロックと2×2ブロックを持つ上準三角行列の場合、実数行列は、実数 Schur 形式となる。2×2ブロックは、次の形式で標準化される。

$$\begin{pmatrix} a & b \\ c & a \end{pmatrix},$$

ここで、 $b*c < 0$ 。このようなブロックの固有値は、 $a \pm \sqrt{bc}$ となる。

複素行列は、上三角の場合は Schur 形式になる。

入力パラメーター

<i>jobvs</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvs</i> ='N' の場合、Schur ベクトルは計算されない。 <i>jobvs</i> ='V' の場合、Schur ベクトルは計算される。
<i>sort</i>	CHARACTER*1。'N' または 'S' でなければならない。 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。 <i>sort</i> ='N' の場合、固有値は順序付けされない。 <i>sort</i> ='S' の場合、固有値は順序付けされる (<i>select</i> を参照)。
<i>select</i>	実数型の場合、2つの REAL 引数の LOGICAL FUNCTION。 複素数型の場合、1つの COMPLEX 引数の LOGICAL FUNCTION。 <i>select</i> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。 <i>sort</i> ='S' の場合、 <i>select</i> は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。 <i>sort</i> ='N' の場合、 <i>select</i> は参照されない。 実数型の場合： <i>select</i> (<i>wr</i> (<i>j</i>), <i>wi</i> (<i>j</i>)) が真の場合、固有値 $wr(j) + \sqrt{-1} * wi(j)$ が選択される。つまり、固有値の複素共役ペアの一方を選択すると、複素数の固有値が両方選択される。そのため、順序付けを実行した後、選択した複素数の固有値は、 <i>select</i> (<i>wr</i> (<i>j</i>), <i>wi</i> (<i>j</i>))=.TRUE. を満たさないときがある。この場合、 <i>info</i> には、 <i>n</i> +2 が設定される (以下の <i>info</i> を参照)。 複素数型の場合： <i>select</i> (<i>w</i> (<i>j</i>)) が真の場合、固有値 <i>w</i> (<i>j</i>) が選択される。
<i>sense</i>	CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。 <i>sense</i> ='N' の場合、計算は実行されない。 <i>sense</i> ='E' の場合、指定された固有値の平均についてのみ計算が実行される。 <i>sense</i> ='V' の場合、指定された右不変部分空間についてのみ計算が実行される。 <i>sense</i> ='B' の場合、両方について計算が実行される。 <i>sense</i> が 'E'、'V'、または 'B' の場合、 <i>sort</i> は 'S' に等しくなければならない。
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (<i>sgeesx</i> の場合) DOUBLE PRECISION (<i>dgeesx</i> の場合) COMPLEX (<i>cgeesx</i> の場合)

	DOUBLE COMPLEX (zgeesx の場合)。 配列: $a(lda,*)$ は、 $n \times n$ の行列 A を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldvs</i>	INTEGER。出力配列 vs のリーディング・ディメンジョン。 次の制約がある。 $ldvs \geq 1$ 。 $ldvs \geq \max(1, n)$ ($jobvs = 'V'$ の場合)。
<i>lwork</i>	INTEGER。配列 $work$ の次元。次の制約がある。 $lwork \geq \max(1, 3n)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 また、 $sense = 'E'$ 、 $'V'$ 、または $'B'$ の場合、次のようになる。 $lwork \geq n+2*sdim*(n-sdim)$ (実数型の場合)。 $lwork \geq 2*sdim*(n-sdim)$ (複素数型の場合)。 ここで、 $sdim$ は、このルーチンで計算される指定された固有値の数である。また、 $2*sdim*(n-sdim) \leq n*n/2$ である。 パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は ($liwork$)。実数型でのみ使用される。 $sense = 'N'$ または $'E'$ の場合は参照されない。
<i>liwork</i>	INTEGER。配列 $iwork$ の次元。実数型でのみ使用される。 次の制約がある。 $liwork \geq 1$ 。 $sense = 'V'$ または $'B'$ の場合、 $liwork \geq sdim*(n-sdim)$ 。
<i>rwork</i>	REAL (cgeesx の場合) DOUBLE PRECISION (zgeesx の場合) ワークスペース配列、次元は $\max(1, n)$ 以上。 複素数型でのみ使用される。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 $sort = 'N'$ の場合は参照されない。

出力パラメーター

<i>a</i>	終了時に、実数 Schur/Schur 形式 T で上書きされる。
<i>sdim</i>	INTEGER。 $sort = 'N'$ の場合、 $sdim=0$ である。 $sort = 'S'$ の場合、 $sdim$ は、 $select$ が真の (ソート後の) 固有値の数に等しくなる。 実数型の場合、どちらかの固有値について $select$ が真の複素共役ペアは、2 としてカウントされるのに注意する必要がある。

<i>wr, wi</i>	<p>REAL (sgeesx の場合) DOUBLE PRECISION (dgeesx の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 出力実数 Schur 形式 T の対角上に格納されるのと同じ順序で、計算された固有値の実数部と虚数部をそれぞれ格納する。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。</p>
<i>w</i>	<p>COMPLEX (cgeesx の場合) DOUBLE COMPLEX (zgeesx の場合)。 配列、次元は $\max(1, n)$ 以上。 計算された固有値が格納される。一連の固有値は、出力 Schur 形式 T の対角上に格納されるのと同じ順序で格納される。</p>
<i>vs</i>	<p>REAL (sgeesx の場合) DOUBLE PRECISION (dgeesx の場合) COMPLEX (cgeesx の場合) DOUBLE COMPLEX (zgeesx の場合)。 配列 $vs(ldvs, *)$。 vs の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $jobvs = 'V'$ の場合、 vs には、Schur ベクトルの直交 / ユニタリ一行列 Z が格納される。 $jobvs = 'N'$ の場合、 vs は参照されない。</p>
<i>rconde, rcondv</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 $sense = 'E'$ または $'B'$ の場合、 $rconde$ には、指定した固有値の平均に対する条件数の逆数が格納される。 $sense = 'N'$ または $'V'$ の場合、 $rconde$ は参照されない。 $sense = 'V'$ または $'B'$ の場合、 $rcondv$ には、指定した右不変部分空間に対する条件数の逆数が格納される。 $sense = 'N'$ または $'E'$ の場合、 $rcondv$ は参照されない。</p>
<i>work(1)</i>	<p>終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。</p>
<i>info</i>	<p>INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$、かつ $i \leq n$: QR アルゴリズムが固有値をすべて計算できなかった。収束した固有値は、 wr と wi (実数型の場合) または w (複素数型の場合) の $1:i:lo-1$ と $i+1:n$ の成分に格納される。 $jobvs = 'V'$ の場合、 vs には、 A をその部分収束 Schur 形式に縮退させる変換を格納する。 $i = n+1$: いくつかの固有値が接近しすぎて分離できなかったため、固有値を順序変更できなかった (非常に悪い条件が問題)。 $i = n+2$:</p>

順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更されたため、Schur 形式の先頭の固有値が `select = .TRUE.` を満たさなくなった。これは、スケーリングによるアンダーフローによっても引き起こされる。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geesx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>wr</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>vs</code>	サイズ (n, n) の行列 VS を格納する。
<code>jobvs</code>	引数 <code>vs</code> の存在に基づいて以下のように復元される。 <code>jobvs = 'V'</code> (<code>vs</code> が存在する場合)、 <code>jobvs = 'N'</code> (<code>vs</code> が省略された場合)。
<code>sort</code>	引数 <code>select</code> の存在に基づいて以下のように復元される。 <code>sort = 'S'</code> (<code>select</code> が存在する場合)、 <code>sort = 'N'</code> (<code>select</code> が省略された場合)。
<code>sense</code>	引数 <code>rconde</code> と <code>rcondv</code> の存在に基づいて以下のように復元される。 <code>sense = 'B'</code> (<code>rconde</code> と <code>rcondv</code> の両方が存在する場合)、 <code>sense = 'E'</code> (<code>rconde</code> が存在し、 <code>rcondv</code> が省略された場合)、 <code>sense = 'V'</code> (<code>rconde</code> が省略され、 <code>rcondv</code> が存在する場合)、 <code>sense = 'N'</code> (<code>rconde</code> と <code>rcondv</code> の両方が省略された場合)。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

?geev

一般行列の固有値と左/右の固有ベクトルを計算する。

構文

Fortran 77:

```
call sgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work,
           lwork, info)
call dgeev(jobvl, jobvr, n, a, lda, wr, wi, vl, ldvl, vr, ldvr, work,
           lwork, info)
call cgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork,
           rwork, info)
call zgeev(jobvl, jobvr, n, a, lda, w, vl, ldvl, vr, ldvr, work, lwork,
           rwork, info)
```

Fortran 95:

```
call geev(a, wr, wi [,vl] [,vr] [,info])
call geev(a, w [,vl] [,vr] [,info])
```

説明

このルーチンは、 $n \times n$ の実/複素非対称行列 A について、固有値と (オプションで) 左/右の固有ベクトルを計算する。 A の右固有ベクトル $v(j)$ は、以下を満たす。

$$A * v(j) = \lambda(j) * v(j)$$

$\lambda(j)$ は、その固有値である。

A の左固有ベクトル $u(j)$ は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H$$

$u(j)^H$ は、 $u(j)$ の共役転置を示す。

計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

入力パラメーター

<i>jobvl</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvl</i> ='N' の場合、 A の左固有ベクトルは計算されない。 <i>jobvl</i> ='V' の場合、 A の左固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvr</i> ='N' の場合、 A の右固有ベクトルは計算されない。 <i>jobvr</i> ='V' の場合、 A の右固有ベクトルは計算される。
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	REAL (sgeev の場合) DOUBLE PRECISION (dgeev の場合) COMPLEX (cgeev の場合)

	DOUBLE COMPLEX (zggev の場合)。 配列: $a(lda,*)$ は、 $n \times n$ の行列 A を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
$ldv1, ldvr$	INTEGER。それぞれ、出力配列 $v1$ と vr のリーディング・ディメンション。次の制約がある。 $ldv1 \geq 1, ldvr \geq 1$ 。 $jobv1 = 'V'$ の場合、 $ldv1 \geq \max(1, n)$ 。 $jobvr = 'V'$ の場合、 $ldvr \geq \max(1, n)$ 。
$lwork$	INTEGER。配列 $work$ の次元。次の制約がある。 $lwork \geq \max(1, 3n)$ 。 $jobv1 = 'V'$ または $jobvr = 'V'$ の場合、 $lwork \geq \max(1, 4n)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。
$rwork$	REAL (cggev の場合) DOUBLE PRECISION (zggev の場合) ワークスペース配列、次元は $\max(1, 2n)$ 以上。 複素数型でのみ使用される。

出力パラメーター

a	終了時に、中間結果で上書きされる。
wr, wi	REAL (sggev の場合) DOUBLE PRECISION (dgev の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。 計算された固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。
w	COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。 配列、次元は $\max(1, n)$ 以上。 計算された固有値が格納される。
$v1, vr$	REAL (sggev の場合) DOUBLE PRECISION (dgev の場合) COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。

配列：

$v1(ldv1, *)$ 。 $v1$ の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$jobv1 = 'V'$ の場合、左固有ベクトル $u(j)$ は、それらの固有値と同じ順序で、 $v1$ の列に次々と格納される。 $jobv1 = 'N'$ の場合、 $v1$ は参照されない。

実数型の場合：

j 番目の固有値が実数の場合、 $u(j) = v1(:, j)$ ($v1$ の j 番目の列) である。 j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、 $u(j) = v1(:, j) + i * v1(:, j+1)$ 、 $u(j+1) = v1(:, j) - i * v1(:, j+1)$ である。ここで、 $i = \sqrt{-1}$ である。

複素数型の場合：

$u(j) = v1(:, j)$ ($v1$ の j 番目の列) である。

$vr(ldvr, *)$ 。 vr の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$jobv1 = 'V'$ の場合、右固有ベクトル $v(j)$ は、それらの固有値と同じ順序で、 vr の列に次々と格納される。 $jobv1 = 'N'$ の場合、 vr は参照されない。

実数型の場合：

j 番目の固有値が実数の場合、 $v(j) = vr(:, j)$ (vr の j 番目の列) である。 j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、 $v(j) = vr(:, j) + i * vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i * vr(:, j+1)$ である。ここで、 $i = \sqrt{-1}$ である。

複素数型の場合：

$v(j) = vr(:, j)$ (vr の j 番目の列) である。

$work(1)$ 終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。

$info$

INTEGER。

$info = 0$ の場合、正常に終了したことを示す。

$info = -i$ の場合、 i 番目のパラメーターの値が不正である。

$info = i$ の場合、QR アルゴリズムが固有値をすべて計算できず、固有ベクトルが計算されなかったことを示す。収束した固有値は、 wr と wi (実数型の場合) または w (複素数型の場合) の $i+1:n$ の成分に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geev` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
wr	長さ (n) のベクトルを格納する。実数型でのみ使用される。
wi	長さ (n) のベクトルを格納する。実数型でのみ使用される。
w	長さ (n) のベクトルを格納する。複素数型でのみ使用される。

<code>v1</code>	サイズ (n,n) の行列 VL を格納する。
<code>vr</code>	サイズ (n,n) の行列 VR を格納する。
<code>jobv1</code>	引数 <code>v1</code> の存在に基づいて以下のように復元される。 <code>jobv1 = 'V'</code> (<code>v1</code> が存在する場合)、 <code>jobv1 = 'N'</code> (<code>v1</code> が省略された場合)。
<code>jobvr</code>	引数 <code>vr</code> の存在に基づいて以下のように復元される。 <code>jobvr = 'V'</code> (<code>vr</code> が存在する場合)、 <code>jobvr = 'N'</code> (<code>vr</code> が省略された場合)。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

?geevx

あらかじめ行列を平衡化して、一般行列の固有値と左/右の固有ベクトル、固有値と右固有ベクトルに対する条件数の逆数を計算する。

構文

Fortran 77:

```
call sgeevx(balanc, jobv1, jobvr, sense, n, a, lda, wr, wi, vl, ldvl, vr,
           ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, iwork,
           info)
call dgeevx(balanc, jobv1, jobvr, sense, n, a, lda, wr, wi, vl, ldvl, vr,
           ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, iwork,
           info)
call cgeevx(balanc, jobv1, jobvr, sense, n, a, lda, w, vl, ldvl, vr,
           ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, rwork,
           info)
call zgeevx(balanc, jobv1, jobvr, sense, n, a, lda, w, vl, ldvl, vr,
           ldvr, ilo, ihi, scale, abnrm, rconde, rcondv, work, lwork, rwork,
           info)
```

Fortran 95:

```
call geevx(a, wr, wi [,vl] [,vr] [,balanc] [,ilo] [,ihi] [,scale]
           [,abnrm] [,rconde] [,rcondv] [,info])
call geevx(a, w [,vl] [,vr] [,balanc] [,ilo] [,ihi] [,scale] [,abnrm]
           [,rconde] [,rcondv] [,info])
```

説明

このルーチンは、 $n \times n$ の実/複素非対称行列 A について、固有値と (オプションで) 左/右の固有ベクトルを計算する。

また、このルーチンは、平衡化変換を計算して、固有値と固有ベクトル (*ilo*、*ihi*、*scale*、*abnrm*)、固有値に対する条件数の逆数 (*rconde*)、右固有ベクトルに対する条件数の逆数 (*rcondv*) の条件付けも改善できる。

A の右固有ベクトル $v(j)$ は、以下を満たす。

$$A * v(j) = \lambda(j) * v(j)$$

$\lambda(j)$ は、その固有値である。

A の左固有ベクトル $u(j)$ は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H$$

$u(j)^H$ は、 $u(j)$ の共役転置を示す。

計算された固有ベクトルは、1 に等しいユークリッド・ノルムと最大コンポーネント実数を持つように正規化される。

行列の平衡化は、行と列を置換して行列を上三角により近づけ、対角相似変換 $D A D^{-1}$ (D は対角行列) を適用して行と列をノルム内でより近づけ、その固有値と固有ベクトルの条件数をより小さくすることを意味する。計算された条件数の逆数は、平衡化された行列に一致する。

行と列を置換しても条件数は変更されないが (正確な演算において)、対角をスケールリングすると条件数は変更される。平衡化の詳細は、4.10 の [\[LUG\]](#) を参照。

入力パラメーター

balanc CHARACTER*1。'N'、'P'、'S'、または 'B' でなければならない。入力行列を対角的にスケールリングする方法、または入力行列を置換してその固有値の条件付けを改善する方法を指定する。

balanc = 'N' の場合、対角的なスケールリングまたは置換は実行されない。

balanc = 'P' の場合、置換を実行して行列を上三角により近づける。対角的なスケールリングは実行されない。

balanc = 'S' の場合、行列に対して対角的なスケールリングが実行される。つまり、 A が $D A D^{-1}$ で置き換えられる (D は、 A の行と列をノルム内でより等しくするために選択された対角行列)。置換は実行されない。

balanc = 'B' の場合、対角的なスケールリングが実行されるとともに、 A が置換される。

計算された条件数の逆数は、平衡化または置換を実行後の行列に使用される。置換を実行しても条件数は変更されないが (正確な演算では)、平衡化を実行すると条件数が変更される。

jobvl CHARACTER*1。'N' または 'V' でなければならない。
jobvl = 'N' の場合、 A の左固有ベクトルは計算されない。
jobvl = 'V' の場合、 A の左固有ベクトルは計算される。
sense = 'E' または 'B' の場合、*jobvl* は 'V' でなければならない。

<i>jobvr</i>	<p>CHARACTER*1。'N' または 'V' でなければならない。 <i>jobvr</i>='N' の場合、A の右固有ベクトルは計算されない。 <i>jobvr</i>='V' の場合、A の右固有ベクトルは計算される。 <i>sense</i>='E' または 'B' の場合、<i>jobvr</i> は 'V' でなければならない。</p>
<i>sense</i>	<p>CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。</p> <p><i>sense</i>='N' の場合、計算は実行されない。 <i>sense</i>='E' の場合、固有値についてのみ計算が実行される。 <i>sense</i>='V' の場合、右固有ベクトルについてのみ計算が実行される。 <i>sense</i>='B' の場合、固有値と右固有ベクトルについて計算が実行される。</p> <p><i>sense</i> が 'E' または 'B' の場合、左固有ベクトルと右固有ベクトルの両方についても計算を実行する必要がある (<i>jobvl</i>='V' と <i>jobvr</i>='V')。</p>
<i>n</i>	INTEGER。行列 A の次数 ($n \geq 0$)。
<i>a, work</i>	<p>REAL (sgeevx の場合) DOUBLE PRECISION (dgeevx の場合) COMPLEX (cggeevx の場合) DOUBLE COMPLEX (zgeevx の場合) 配列: <i>a</i>(<i>lda</i>,*) は、$n \times n$ の行列 A を格納する配列である。 <i>a</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。</p>
<i>ldvl, ldvr</i>	<p>INTEGER。それぞれ、出力配列 <i>vl</i> と <i>vr</i> のリーディング・ディメンジョン。次の制約がある。 $ldvl \geq 1, ldvr \geq 1$。 <i>jobvl</i>='V' の場合、$ldvl \geq \max(1, n)$。 <i>jobvr</i>='V' の場合、$ldvr \geq \max(1, n)$。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。 実数型の場合: <i>sense</i>='N' または 'E' の場合、$lwork \geq \max(1, 2n)$、 <i>jobvl</i>='V' または <i>jobvr</i>='V' の場合、$lwork \geq 3n$、 <i>sense</i>='V' または 'B' の場合、$lwork \geq n(n+6)$。 パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。</p> <p>複素数型の場合: <i>sense</i>='N' または 'E' の場合、$lwork \geq \max(1, 2n)$、 <i>sense</i>='V' または 'B' の場合、$lwork \geq n^2+2n$。 パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。</p>

$lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。

$rwork$ REAL (cggeevx の場合)
DOUBLE PRECISION (zggeevx の場合)
ワークスペース配列、次元は $\max(1, 2n)$ 以上。複素数型でのみ使用される。

$iwork$ INTEGER。
ワークスペース配列、次元は $\max(1, 2n-2)$ 以上。実数型でのみ使用される。 $sense = 'N'$ または $'E'$ の場合は参照されない。

出力パラメーター

a 終了時に上書きされる。 $jobvl = 'V'$ または $jobvr = 'V'$ の場合、平衡化した入力行列 A の実数 Schur/Schur 形式が格納される。

wr, wi REAL (sggeevx の場合)
DOUBLE PRECISION (dsggeevx の場合)
配列、次元はそれぞれ $\max(1, n)$ 以上。
計算された固有値の実数部と虚数部がそれぞれ格納される。固有値の複素共役ペアは、虚数部が正である固有値に続いて格納される。

w COMPLEX (cggeevx の場合)
DOUBLE COMPLEX (zggeevx の場合)。
配列、次元は $\max(1, n)$ 以上。
計算された固有値が格納される。

vl, vr REAL (sggeevx の場合)
DOUBLE PRECISION (dsggeevx の場合)
COMPLEX (cggeevx の場合)
DOUBLE COMPLEX (zggeevx の場合)
配列：
 $vl(ldvl, *)$ 。 vl の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $jobvl = 'V'$ の場合、左固有ベクトル $u(j)$ は、それらの固有値と同じ順序で、 vl の列に次々と格納される。
 $jobvl = 'N'$ の場合、 vl は参照されない。
実数型の場合：
 j 番目の固有値が実数の場合、 $u(j) = vl(:, j)$ (vl の j 番目の列) である。 j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、 $u(j) = vl(:, j) + i*vl(:, j+1)$ 、 $u(j+1) = vl(:, j) - i*vl(:, j+1)$ である。ここで、 $i = \sqrt{-1}$ である。
複素数型の場合：
 $u(j) = vl(:, j)$ (vl の j 番目の列) である。
 $vr(ldvr, *)$ 。 vr の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$jobv = 'V'$ の場合、右固有ベクトル $v(j)$ は、それらの固有値と同じ順序で、 vr の列に次々と格納される。

$jobv = 'N'$ の場合、 vr は参照されない。

実数型の場合:

j 番目の固有値が実数の場合、 $v(j) = vr(:, j)$ (vr の j 番目の列) である。 j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、 $v(j) = vr(:, j) + i*vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i*vr(:, j+1)$ である。ここで、 $i = \sqrt{-1}$ である。

複素数型の場合:

$v(j) = vr(:, j)$ (vr の j 番目の列) である。

ilo, ihi

INTEGER。

ilo と ihi は、 A を平衡化したときに決定される整数値である。 $i > j$ と $j = 1, \dots, ilo-1$ または $i = ihi+1, \dots, n$ の場合、平衡化した $A(i, j) = 0$ である。

$balanc = 'N'$ または $'S'$ の場合、 $ilo = 1$ と $ihi = n$ である。

$scale$

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元は $\max(1, n)$ 以上。

A の平衡化時に適用された置換とスケーリング係数の各成分が格納される。 $P(j)$ が、行と列 j で交換された行と列のインデックスである場合、 $D(j)$ は、行と列 j に適用されたスケーリング係数であり、

$scale(j) = P(j)$ 、($j = 1, \dots, ilo-1$)

$= D(j)$ 、($j = ilo, \dots, ihi$)

$= P(j)$ 、($j = ihi+1, \dots, n$) である。

交換を実行する順序は、 $n \sim ihi+1$ 、次に $1 \sim ilo-1$ である。

$abnrm$

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

平衡化した行列の 1- ノルム (任意の列の成分に対する絶対値の合計の最大値)。

$rconde, rcondv$

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元はそれぞれ $\max(1, n)$ 以上。

$rconde(j)$ は、 j 番目の固有値の条件数の逆数である。

$rcondv(j)$ は、 j 番目の右固有ベクトルの条件数の逆数である。

$work(1)$

終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。

$info$

INTEGER。

$info = 0$ の場合、正常に終了したことを示す。

$info = -i$ の場合、 i 番目のパラメーターの値が不正である。

$info = i$ の場合、 QR アルゴリズムが固有値をすべて計算できず、固有ベクトルまたは条件数が計算されなかったことを示す。

収束した固有値は、 wr と wi (実数型の場合) または w (複素数型の場合) の $1:ilo-1$ と $i+1:n$ の成分に格納される。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `geevx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>wr</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>wi</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>w</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>vl</code>	サイズ (n, n) の行列 VL を格納する。
<code>vr</code>	サイズ (n, n) の行列 VR を格納する。
<code>scale</code>	長さ (n) のベクトルを格納する。
<code>rconde</code>	長さ (n) のベクトルを格納する。
<code>rcondv</code>	長さ (n) のベクトルを格納する。
<code>balanc</code>	'N'、'B'、'P'、または 'S' のいずれかでなければならない。 デフォルト値は 'N'。
<code>jobvl</code>	引数 <code>vl</code> の存在に基づいて以下のように復元される。 <code>jobvl = 'V'</code> (<code>vl</code> が存在する場合)、 <code>jobvl = 'N'</code> (<code>vl</code> が省略された場合)。
<code>jobvr</code>	引数 <code>vr</code> の存在に基づいて以下のように復元される。 <code>jobvr = 'V'</code> (<code>vr</code> が存在する場合)、 <code>jobvr = 'N'</code> (<code>vr</code> が省略された場合)。
<code>sense</code>	引数 <code>rconde</code> と <code>rcondv</code> の存在に基づいて以下のように復元される。 <code>sense = 'B'</code> (<code>rconde</code> と <code>rcondv</code> の両方が存在する場合)、 <code>sense = 'E'</code> (<code>rconde</code> が存在し、 <code>rcondv</code> が省略された場合)、 <code>sense = 'V'</code> (<code>rconde</code> が省略され、 <code>rcondv</code> が存在する場合)、 <code>sense = 'N'</code> (<code>rconde</code> と <code>rcondv</code> の両方が省略された場合)。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

特異値分解

このセクションでは、特異値を解くために使用する LAPACK ドライバールーチンについて説明する。これらの問題を解くために呼び出すことが可能な [計算ルーチン](#) も参照のこと。

表 4-12 に Fortran-77 インターフェイスのすべてのドライバールーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-12 特異値分解 (SVD) 用のドライバールーチン

ルーチン名	機能
?gesvd	一般矩形行列の特異値分解を計算する。
?gesdd	分割統治法を使用して、一般矩形行列の特異値分解を計算する。
?ggsvd	一般矩形行列ペアの汎用特異値分解を計算する。

?gesvd

一般矩形行列の特異値分解を計算する。

構文

Fortran 77:

```
call sgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call dgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, info)
call cgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info)
call zgesvd(jobu, jobvt, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork, info)
```

Fortran 95:

```
call gesvd(a, s [,u] [,vt] [,ww] [,job] [,info])
```

説明

このルーチンは、実 / 複素行列 $A (m \times n)$ の特異値分解 (SVD) と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。SVD は次のように記述される。

$$A = U \Sigma V^H$$

ここで、 Σ はその $\min(m, n)$ 対角成分を除いてゼロの $m \times n$ の行列、 U は $m \times m$ の直交 / ユニタリー行列、 V は $n \times n$ の直交 / ユニタリー行列である。 Σ の対角成分は、 A の特異値である。これらは、実数かつ非負であり、降順で返される。 U と V の先頭から $\min(m, n)$ の列は、 A の左特異ベクトルと右特異ベクトルである。

ルーチンは、 V ではなく、 V^H を戻すことに注意が必要である。

入力パラメーター

<i>jobu</i>	<p>CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。行列 U のすべて、または一部を計算するオプションを指定する。</p> <p><i>jobu</i>='A' の場合、U の m 個の列がすべて配列 u に返される。 <i>jobu</i>='S' の場合、U の先頭から $\min(m, n)$ の列 (左特異ベクトル) が配列 u に返される。 <i>jobu</i>='O' の場合、U の先頭から $\min(m, n)$ の列 (左特異ベクトル) が配列 a 上で上書きされる。 <i>jobu</i>='N' の場合、U の列 (左特異ベクトル) は計算されない。</p>
<i>jobvt</i>	<p>CHARACTER*1。'A'、'S'、'O'、または 'N' でなければならない。行列 V^H のすべてまたは一部を計算するオプションを指定する。</p> <p><i>jobvt</i>='A' の場合、V^H の n 個の行がすべて配列 vt に返される。 <i>jobvt</i>='S' の場合、V^H の先頭から $\min(m, n)$ の行 (右特異ベクトル) が配列 vt に返される。 <i>jobvt</i>='O' の場合、V^H の先頭から $\min(m, n)$ の行 (右特異ベクトル) が配列 a 上で上書きされる。 <i>jobvt</i>='N' の場合、V^H の行 (右特異ベクトル) は計算されない。</p> <p><i>jobvt</i> と <i>jobu</i> は、どちらも 'O' にできない。</p>
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 A の列数 ($n \geq 0$)。
<i>a, work</i>	<p>REAL (sgesvd の場合) DOUBLE PRECISION (dgesvd の場合) COMPLEX (cgesvd の場合) DOUBLE COMPLEX (zgesvd の場合)。 配列: <i>a</i>(<i>lda</i>,*) は、$m \times n$ の行列 A を格納する配列である。 <i>a</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	<p>INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上でなければならない。</p>
<i>ldu, ldvt</i>	<p>INTEGER。それぞれ、出力配列 u と vt のリーディング・ディメンジョン。次の制約がある。 $ldu \geq 1$; $ldvt \geq 1$。 <i>jobu</i>='S' または 'A' の場合、$ldu \geq m$。 <i>jobvt</i>='A' の場合、$ldvt \geq n$。 <i>jobvt</i>='S' の場合、$ldvt \geq \min(m, n)$。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。$lwork \geq 1$。次の制約がある。 $lwork \geq \max(3 * \min(m, n) + \max(m, n), 5 * \min(m, n))$ (実数型の場合)。 $lwork \geq 2 * \min(m, n) + \max(m, n)$ (複素数型の場合)。 パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、<i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。</p>

rwork REAL (cgesvd の場合)
 DOUBLE PRECISION (zgesvd の場合)
 ワークスペース配列、次元は $\max(1, 5 * \min(m, n))$ 以上。
 複素数型でのみ使用される。

出力パラメーター

a 終了時に、
 $jobu = 'O'$ の場合、*a* は、*U* の先頭から $\min(m, n)$ の列 (列方向に格納される左特異ベクトル) で上書きされる。
 $jobvt = 'O'$ の場合、*a* は、 V^H の先頭から $\min(m, n)$ の行 (行方向に格納される右特異ベクトル) で上書きされる。
 $jobu \neq 'O'$ かつ $jobvt \neq 'O'$ の場合、*a* の内容は破棄される。

s REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)。
 配列、次元は $\max(1, \min(m, n))$ 以上。
 $s(i) \geq s(i+1)$ となるようにソートされて *A* の特異値が格納される。

u, vt REAL (sgesvd の場合)
 DOUBLE PRECISION (dgesvd の場合)
 COMPLEX (cgesvd の場合)
 DOUBLE COMPLEX (zgesvd の場合)。
 配列：
 $u(ldu, *)$ 。*u* の第 2 次元は、 $\max(1, m)$ 以上 ($jobu = 'A'$ の場合) または $\max(1, \min(m, n))$ ($jobu = 'S'$ の場合) 以上でなければならない。
 $jobu = 'A'$ の場合、*u* には、 $m \times m$ の直交 / ユニタリー行列 *U* が格納される。
 $jobu = 'S'$ の場合、*u* には、*U* の先頭から $\min(m, n)$ の列 (列方向に格納される左特異ベクトル) が格納される。
 $jobu = 'N'$ または $'O'$ の場合、*u* は参照されない。
 $vt(ldvt, *)$ 。*vt* の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $jobvt = 'A'$ の場合、*vt* には、 $n \times n$ の直交 / ユニタリー行列 V^H が格納される。
 $jobvt = 'S'$ の場合、*vt* には、 V^H の先頭から $\min(m, n)$ の行 (行方向に格納される右特異ベクトル) が格納される。
 $jobvt = 'N'$ または $'O'$ の場合、*vt* は参照されない。

work 終了時に、 $info = 0$ の場合、*work(1)* は、*lwork* の必要最小サイズを返す。
 実数型の場合：
 $info > 0$ の場合、*work(2:min(m, n))* には、対角が (必ずしもソートされていない) *s* 内にある上二重対角行列 *B* の非収束優対角成分が格納される。*B* は、 $A = u * B * vt$ を満たすため、*A* と同じ特異値を持ち、*u* と *vt* で関連付けられた特異ベクトルを持つ。

<i>rwork</i>	終了時 (複素数型の場合)、 $info > 0$ であれば、 $rwork(1:\min(m, n)-1)$ には、対角が (必ずしもソートされていない) s 内にある上二重対角行列 B の非収束優対角成分が格納される。 B は、 $A = u * B * vt$ を満たすため、 A と同じ特異値を持ち、 u と vt で関連付けられた特異ベクトルを持つ。
<i>info</i>	INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ で、 <code>?bdsqr</code> が収束していなかった場合、 i は、ゼロに収束しなかった中間二重対角形式 B の優対角の数である。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gesvd` のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (m, n) の行列 A を格納する。
<i>s</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>u</i>	サイズ $(m, \min(m, n))$ の行列 U を格納する。
<i>vt</i>	サイズ $(\min(m, n), n)$ の行列 VT を格納する。
<i>ww</i>	長さ $(\min(m, n)-1)$ のベクトルを格納する。
<i>ww</i>	長さ $\min(m, n)-1$ のベクトルを格納する。 ww には、対角が (必ずしもソートされていない) s 内にある上二重対角行列 B の非収束優対角成分が格納される。 B は、 $A = U * B * VT$ を満たすため、 A と同じ特異値を持ち、 U と VT で関連付けられた特異ベクトルを持つ。
<i>job</i>	'N'、'U'、または 'V' でなければならない。デフォルト値は 'N'。 $job = 'U'$ で、 u が存在しない場合、 u は配列 a に返される。 $job = 'V'$ で、 vt が存在しない場合、 vt は配列 a に返される。
<i>jobu</i>	引数 u の存在、 job の値、配列 u と a のサイズに基づいて以下のように復元される。 $jobu = 'A'$ (u が存在し、 u の列数が a の行数と等しい場合)、 $jobu = 'S'$ (u が存在し、 u の列数が a の行数と等しくない場合)、 $jobu = 'O'$ (u が省略され、 job が 'U' と等しい場合)、 $jobu = 'N'$ (u が省略され、 job が 'U' と等しくない場合)。
<i>jobvt</i>	引数 vt の存在、 job の値、配列 vt と a のサイズに基づいて以下のように復元される。 $jobvt = 'A'$ (vt が存在し、 vt の列数が a の行数と等しい場合)、 $jobvt = 'S'$ (vt が存在し、 vt の列数が a の行数と等しくない場合)、 $jobvt = 'O'$ (vt が省略され、 job が 'V' と等しい場合)、 $jobvt = 'N'$ (vt が省略され、 job が 'V' と等しくない場合)。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

?gesdd

分割統治法を使用して、一般矩形行列の特異値分解を計算する。

構文

Fortran 77:

```
call sgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork,
            info)
call dgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, iwork,
            info)
call cgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork,
            iwork, info)
call zgesdd(jobz, m, n, a, lda, s, u, ldu, vt, ldvt, work, lwork, rwork,
            iwork, info)
```

Fortran 95:

```
call gesdd(a, s [,u] [,vt] [,jobz] [,info])
```

説明

このルーチンは、実 / 複素行列 A ($m \times n$) の特異値分解 (SVD) と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。特異ベクトルを計算する場合、分割統治アルゴリズムを使用する。

SVD は次のように記述される。

$$A = U \Sigma V^H$$

ここで、 Σ はその $\min(m, n)$ 対角成分を除いてゼロの $m \times n$ の行列、 U は $m \times m$ の直交 / ユニタリー行列、 V は $n \times n$ の直交 / ユニタリー行列である。 Σ の対角成分は、 A の特異値である。これらは、実数かつ非負であり、降順で返される。 U と V の先頭から $\min(m, n)$ の列は、 A の左特異ベクトルと右特異ベクトルである。ルーチンは、 V ではなく、 V^H を戻すことに注意が必要である。

入力パラメーター

jobz CHARACTER*1. 'A'、'S'、'O'、または 'N' でなければならない。行列 U のすべてまたは一部を計算するオプションを指定する。
jobz = 'A' の場合、 U の m 個の列、 V^T の n 個の行は、すべて配列 *u* と *vt* に返される。
jobz = 'S' の場合、 U の先頭から $\min(m, n)$ の列、 V^T の先頭から $\min(m, n)$ の行は、配列 *u* と *vt* に返される。
jobz = 'O' の場合、

$m \geq n$ であれば、 U の先頭から n 個の列が配列 a 上で上書きされ、 V^T の行がすべて配列 vt に返される。

$m < n$ であれば、 U の列がすべて配列 u に返され、 V^T の先頭から m 個の行が配列 vt 内で上書きされる。

$jobz = 'N'$ の場合、 U の列、または V^T の行は計算されない。

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。行列 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgesdd の場合) DOUBLE PRECISION (dgesdd の場合) COMPLEX (cgesdd の場合) DOUBLE COMPLEX (zgesdd の場合)。 配列: $a(lda, *)$ は、 $m \times n$ の行列 A を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
lda	INTEGER。配列 a の第 1 次元。 $\max(1, m)$ 以上でなければならない。
$ldu, ldvt$	INTEGER。それぞれ、出力配列 u と vt のリーディング・ディメンジョン。次の制約がある。 $ldu \geq 1$; $ldvt \geq 1$ 。 $jobz = 'S'$ または $'A'$ の場合、または $jobz = 'O'$ の場合、 $m < n$ であれば、 $ldu \geq m$ 。 $jobz = 'A'$ または $jobz = 'O'$ の場合、 $m \geq n$ であれば、 $ldvt \geq n$ 。 $jobz = 'S'$ の場合、 $ldvt \geq \min(m, n)$ である。
$lwork$	INTEGER。配列 $work$ の次元。 $lwork \geq 1$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。 $lwork$ の推奨値は、「アプリケーション・ノート」を参照。
$rwork$	REAL (cgesdd の場合) DOUBLE PRECISION (zgesdd の場合) $jobz = 'N'$ の場合、ワークスペース配列、次元は $\max(1, 5 * \min(m, n))$ 以上。それ以外の場合、 $rwork$ の次元は、 $5 * (\min(m, n))^2 + 7 * \min(m, n)$ 以上でなければならない。 この配列は、複素数型でのみ使用される。
$iwork$	INTEGER。ワークスペース配列、次元は $\max(1, 8 * \min(m, n))$ 以上。

出力パラメーター

a	終了時に、 $jobz = 'O'$ の場合、 $m \geq n$ であれば、 a は、 U の先頭から n 個の列 (列方向に格納される左特異ベクトル) で上書きされる。 $m < n$ であれば、 a は、 V^T の先頭から m 個の行 (行方向に格納される右特異ベクトル) で上書きされる。 $jobz \neq 'O'$ の場合、 a の内容は破棄される。
-----	--

<i>s</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元は $\max(1, \min(m, n))$ 以上。 $s(i) \geq s(i+1)$ となるようにソートされて <i>A</i> の特異値が格納される。
<i>u, vt</i>	REAL (<i>sgesdd</i> の場合) DOUBLE PRECISION (<i>dgesdd</i> の場合) COMPLEX (<i>cgesdd</i> の場合) DOUBLE COMPLEX (<i>zgesdd</i> の場合)。 配列: <i>u(ldu, *)</i> 。 <i>u</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない (<i>jobz</i> = 'A' または <i>jobz</i> = 'O' であり、 $m < n$ の場合)。 <i>jobz</i> = 'S' の場合、 <i>u</i> の第 2 次元は、 $\max(1, \min(m, n))$ 以上でなければならない。 <i>jobz</i> = 'A' または <i>jobz</i> = 'O' の場合、 $m < n$ であれば、 <i>u</i> には、 $m \times m$ の直交 / ユニタリー行列 <i>U</i> が格納される。 <i>jobz</i> = 'S' の場合、 <i>u</i> には、 <i>U</i> の先頭から $\min(m, n)$ の列 (列方向に格納される左特異ベクトル) が格納される。 <i>jobz</i> = 'O' かつ $m \geq n$ 、または <i>jobz</i> = 'N' の場合、 <i>u</i> は参照されない。 <i>vt(ldvt, *)</i> 。 <i>vt</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'A' または <i>jobz</i> = 'O' の場合、 $m \geq n$ であれば、 <i>vt</i> には、 $n \times n$ の直交 / ユニタリー行列 <i>V^T</i> が格納される。 <i>jobz</i> = 'S' の場合、 <i>vt</i> には、 <i>V^T</i> の先頭から $\min(m, n)$ の行 (行方向に格納される右特異ベクトル) が格納される。 <i>jobz</i> = 'O' かつ $m < n$ 、または <i>jobz</i> = 'N' の場合、 <i>vt</i> は参照されない。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> で、?bdsdc が収束していなかった場合、更新プロセスが失敗したことを示す。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *gesdd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m, n</i>) の行列 <i>A</i> を格納する。
<i>s</i>	長さ $\min(m, n)$ のベクトルを格納する。
<i>u</i>	サイズ (<i>m, min(m, n)</i>) の行列 <i>U</i> を格納する。
<i>vt</i>	サイズ ($\min(m, n), n$) の行列 <i>VT</i> を格納する。

job 'N'、'A'、'S'、または 'O' でなければならない。
デフォルト値は 'N'。

アプリケーション・ノート

実数型の場合：

jobz = 'N' の場合、 $lwork \geq 3 * \min(m,n) + \max(\max(m,n), 6 * \min(m,n))$ 。

jobz = 'O' の場合、 $lwork \geq 3 * (\min(m,n))^2 +$

$\max(\max(m,n), 5 * (\min(m,n))^2 + 4 * \min(m,n))$ 。

jobz = 'S' または 'A' の場合、 $lwork \geq 3 * (\min(m,n))^2 +$

$\max(\max(m,n), 4 * (\min(m,n))^2 + 4 * \min(m,n))$ 。

複素数型の場合：

jobz = 'N' の場合、 $lwork \geq 2 * \min(m,n) + \max(m,n)$ 。

jobz = 'O' の場合、 $lwork \geq 2 * (\min(m,n))^2 + \max(m,n) + 2 * \min(m,n)$ 。

jobz = 'S' または 'A' の場合、 $lwork \geq (\min(m,n))^2 + \max(m,n) + 2 * \min(m,n)$ 。

パフォーマンスを改善するには、一般に *lwork* に上記以上の値が必要である。
配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?ggsvd

一般矩形行列ペアの汎用特異値分解を計算する。

構文

Fortran 77:

```
call sggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
           beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
```

```
call dggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
           beta, u, ldu, v, ldv, q, ldq, work, iwork, info)
```

```
call cggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
           beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
```

```
call zggsvd(jobu, jobv, jobq, m, n, p, k, l, a, lda, b, ldb, alpha,
           beta, u, ldu, v, ldv, q, ldq, work, rwork, iwork, info)
```

Fortran 95:

```
call ggsvd(a, b, alpha, beta [,k] [,l] [,u] [,v] [,q] [,iwork] [,info])
```

説明

このルーチンは、 $m \times n$ の実 / 複素行列 A と $p \times n$ の実 / 複素行列 B の汎用特異値分解 (GSVD) を計算する。

$$U^H A Q = D_1 * (0 \ R), \quad V^H B Q = D_2 * (0 \ R)$$

ここで、 U 、 V 、および Q は直交 / ユニタリー行列である。

$k+1=$ が行列 $(A^H, B^H)^H$ の有効な数値ランクの場合、 R は、 $(k+1) \times (k+1)$ の非特異上三角行列となり、 D_1 と D_2 は、 $m \times (k+1)$ の "対角" 行列と $p \times (k+1)$ の "対角" 行列となる。 D_1 と D_2 はそれぞれ、次の構造で示される。

$m-k-1 \geq 0$ の場合、

$$D_1 = \begin{matrix} & k & 1 \\ & I & 0 \\ m-k-1 & 0 & C \\ & 0 & 0 \end{matrix}$$

$$D_2 = \begin{matrix} & k & 1 \\ 1 & 0 & S \\ p-1 & 0 & 0 \end{matrix}$$

$$(0 \ R) = \begin{matrix} & n-k-1 & k & 1 \\ k & 0 & R_{11} & R_{12} \\ l & 0 & 0 & R_{22} \end{matrix},$$

ここで、

$$\begin{aligned} C &= \text{diag}(\alpha(k+1), \dots, \alpha(k+1)) \\ S &= \text{diag}(\beta(k+1), \dots, \beta(k+1)) \\ C^2 + S^2 &= I \end{aligned}$$

R は、終了時に、 $a(1:k+1, n-k-1+1:n)$ に格納される。

$m-k-1 < 0$ の場合、

$$D_1 = \begin{matrix} & k & m-k & k+1-m \\ & I & 0 & 0 \\ m-k & 0 & C & 0 \end{matrix}$$

$$D_2 = \begin{matrix} & k & m-k & k+1-m \\ m-k & 0 & S & 0 \\ k+1-m & 0 & 0 & I \\ p-1 & 0 & 0 & 0 \end{matrix}$$

$$(0 \ R) = \begin{matrix} & n-k-l & k & m-k & k+l-m \\ & k & \begin{pmatrix} 0 & R_{11} & R_{12} & R_{13} \\ 0 & 0 & R_{22} & R_{23} \\ 0 & 0 & 0 & R_{33} \end{pmatrix} \\ m-k & & & & \\ k+l-m & & & & \end{matrix}$$

ここで、

$$\begin{aligned} C &= \text{diag}(\alpha(k+1), \dots, \alpha(m)), \\ S &= \text{diag}(\beta(k+1), \dots, \beta(m)), \\ C^2 + S^2 &= I \end{aligned}$$

終了時に、 $\begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \end{pmatrix}$ は $a(1:m, n-k-l+1:n)$ に格納され、 R_{33} は

$b(m-k+1:l, n+m-k-l+1:n)$ に格納される。

このルーチンは、 C 、 S 、 R と (オプションで) 直交 / ユニタリー変換行列 U 、 V 、 Q を計算する。

特に、 B が $n \times n$ の非特異行列の場合、 A と B の GSVD は、暗黙的に AB^{-1} の SVD を提供する。

$$AB^{-1} = U(D_1 D_2^{-1}) V^H$$

$(A^H, B^H)^H$ が直交列を持っている場合、 A と B の GSVD も A と B の CS 分解に等しくなる。更に、GSVD を使用すると、固有値問題の解を導き出すことができる。

$$A^H A x = \lambda B^H B x$$

入力パラメーター

<i>jobu</i>	CHARACTER*1。'U' または 'N' でなければならない。 <i>jobu</i> ='U' の場合、直交 / ユニタリー行列 U が計算される。 <i>jobu</i> ='N' の場合、 U は計算されない。
<i>jobv</i>	CHARACTER*1。'V' または 'N' でなければならない。 <i>jobv</i> ='V' の場合、直交 / ユニタリー行列 V が計算される。 <i>jobv</i> ='N' の場合、 V は計算されない。
<i>jobq</i>	CHARACTER*1。'Q' または 'N' でなければならない。 <i>jobq</i> ='Q' の場合、直交 / ユニタリー行列 Q が計算される。 <i>jobq</i> ='N' の場合、 Q は計算されない。
<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 A と B の列数 ($n \geq 0$)。
<i>p</i>	INTEGER。行列 B の行数 ($p \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sggsvd の場合) DOUBLE PRECISION (dggsvd の場合) COMPLEX (cggsvd の場合) DOUBLE COMPLEX (zggsvd の場合)。

配列:

$a(lda,*)$ には、 $m \times n$ の行列 A を格納する。
 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(ldb,*)$ には、 $p \times n$ の行列 B を格納する。
 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(*)$ は、ワークスペース配列である。 $work$ の次元は、 $\max(3n, m, p)+n$ 以上でなければならない。

lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
ldb	INTEGER。 b の第 1 次元。 $\max(1, p)$ 以上。
ldu	INTEGER。 配列 u の第 1 次元。 $jobu='U'$ の場合、 $ldu \geq \max(1, m)$ 、それ以外の場合、 $ldu \geq 1$ 。
ldv	INTEGER。 配列 v の第 1 次元。 $jobv='V'$ の場合、 $ldv \geq \max(1, p)$ 、それ以外の場合、 $ldv \geq 1$ 。
ldq	INTEGER。 配列 q の第 1 次元。 $jobq='Q'$ の場合、 $ldq \geq \max(1, n)$ 、それ以外の場合、 $ldq \geq 1$ 。
$iwork$	INTEGER。 ワークスペース配列、次元は $\max(1, n)$ 以上。
$rwork$	REAL (cggsvd の場合) DOUBLE PRECISION (zggsvd の場合)。 ワークスペース配列、次元は $\max(1, 2n)$ 以上。 複素数型でのみ使用される。

出力パラメーター

k, l	INTEGER。 終了時に、 k と l は、サブブロックの次元を指定する。 $k+1$ の合計は、 $(A^H, B^H)^H$ の有効数値ランクに等しい。
a	終了時に、 a には、三角行列 R または R の一部が格納される。
b	終了時に、 $m-k-l < 0$ の場合、 b には、三角行列 R の一部が格納される。
$alpha, beta$	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。 A と B の汎用特異値ペアが、次のように格納される。 $alpha(1:k) = 1$ $beta(1:k) = 0$ $m-k-l \geq 0$ の場合、 $alpha(k+1:k+1) = C$ 、 $beta(k+1:k+1) = S$ 、 $m-k-l < 0$ の場合、 $alpha(k+1:m) = C$ 、 $alpha(m+1:k+1) = 0$ $beta(k+1:m) = S$ 、 $beta(m+1:k+1) = 1$ および、 $alpha(k+l+1:n) = 0$ $beta(k+l+1:n) = 0$

<i>u, v, q</i>	REAL (sggsvd の場合) DOUBLE PRECISION (dggsvd の場合) COMPLEX (cggsvd の場合) DOUBLE COMPLEX (zggsvd の場合)。 配列： <i>u(ldu,*)</i> 。 <i>u</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobu='U'</i> の場合、 <i>u</i> には、 $m \times m$ の直交 / ユニタリー行列 <i>U</i> が格納される。 <i>jobu='N'</i> の場合、 <i>u</i> は参照されない。 <i>v(ldv,*)</i> 。 <i>v</i> の第 2 次元は、 $\max(1, p)$ 以上でなければならない。 <i>jobv='V'</i> の場合、 <i>v</i> には、 $p \times p$ の直交 / ユニタリー行列 <i>V</i> が格納される。 <i>jobv='N'</i> の場合、 <i>v</i> は参照されない。 <i>q(ldq,*)</i> 。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobq='Q'</i> の場合、 <i>q</i> には、 $n \times n$ の直交 / ユニタリー行列 <i>Q</i> が格納される。 <i>jobq='N'</i> の場合、 <i>q</i> は参照されない。
<i>iwork</i>	終了時に、 <i>iwork</i> には、ソート情報が格納される。
<i>info</i>	INTEGER。 <i>info=0</i> の場合、実行は正常に終了したことを示す。 <i>info=-i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info=1</i> の場合、Jacobi 型プロシージャラーが収束できなかったことを示す。詳細は、サブルーチン ?tgsja を参照。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ggsvd* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>m, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>p, n</i>) の行列 <i>B</i> を格納する。
<i>alpha</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>beta</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>u</i>	サイズ (<i>m, m</i>) の行列 <i>U</i> を格納する。
<i>v</i>	サイズ (<i>p, p</i>) の行列 <i>V</i> を格納する。
<i>q</i>	サイズ (<i>n, n</i>) の行列 <i>Q</i> を格納する。
<i>iwork</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>jobu</i>	引数 <i>u</i> の存在に基づいて以下のように復元される。 <i>jobu='U'</i> (<i>u</i> が存在する場合)、 <i>jobu='N'</i> (<i>u</i> が省略された場合)。
<i>jobv</i>	引数 <i>v</i> の存在に基づいて以下のように復元される。 <i>jobv='V'</i> (<i>v</i> が存在する場合)、 <i>jobv='N'</i> (<i>v</i> が省略された場合)。

jobq

引数 *q* の存在に基づいて以下のように復元される。

jobq = 'Q' (*q* が存在する場合)、

jobq = 'N' (*q* が省略された場合)。

汎用対称固有値問題

このセクションでは、汎用対称固有値問題を解くために使用する LAPACK ドライバールーチンについて説明する。これらの問題を解くために呼び出すことが可能な[計算ルーチン](#)も参照のこと。

表 4-13 に Fortran-77 インターフェイスのすべてのドライバールーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-13 汎用対称固有値問題を解くためのドライバールーチン

ルーチン名	機能
?ssygv / ?hesgv	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。
?ssygvd / ?hesgvd	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?ssygvx / ?hesgvx	実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。
?spgv / ?hpgv	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。
?spgvd / ?hpgvd	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?spgvx / ?hpgvx	圧縮格納形式の行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。
?sbgv / ?hbgv	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。
?sbgvd / ?hbgvd	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。
?sbgvx / ?hbgvx	帯行列に関する実数 / 複素数の汎用対称 / エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

[?ssygv](#)

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssygv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, info)
call dsygv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, info)
```

Fortran 95:

```
call sygv(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B は対称であると仮定する。また、 B は正定値である。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 a と b は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 a と b は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (ssygv の場合) DOUBLE PRECISION (dsygv の場合)。 配列: $a(lda,*)$ には、 <i>uplo</i> の値に従って、対称行列 A の上三角または下三角を格納する。 a の第2次元は、 $\max(1, n)$ 以上でなければならない。 $b(l,db,*)$ には、 <i>uplo</i> の値に従って、対称正定値行列 B の上三角または下三角を格納する。 b の第2次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 b の第1次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 3n-1)$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。

出力パラメーター

<i>a</i>	終了時に、 <i>jobz</i> ='V' の場合、 <i>info</i> =0 であれば、 <i>a</i> には、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、次のように正規化される。 <i>itype</i> = 1 または 2 の場合、 $Z^T B Z = I$ 。 <i>itype</i> = 3 の場合、 $Z^T B^{-1} Z = I$ 。 <i>jobz</i> ='N' の場合、終了時に、 <i>A</i> の上三角 (<i>uplo</i> ='U' の場合)、または下三角 (<i>uplo</i> ='L' の場合) が対角を含めて破棄される。
<i>b</i>	終了時に、 <i>info</i> ≤ <i>n</i> の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>w</i>	REAL (<i>ssygv</i> の場合) DOUBLE PRECISION (<i>dsygv</i> の場合)。 配列、次元は max(1, <i>n</i>) 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 の場合、 <i>spotrf/dpotrf</i> と <i>ssyev/dsyev</i> によって、エラーコードが返される。 <i>info</i> = <i>i</i> ≤ <i>n</i> の場合、 <i>ssyev/dsyev</i> が収束せず、中間三重対角の <i>i</i> 個の非対角成分がゼロに収束しなかったことを示す。 <i>info</i> = <i>n</i> + <i>i</i> (1 ≤ <i>i</i> ≤ <i>n</i>) の場合、 <i>B</i> について先頭から次数 <i>i</i> の小行列が正定値でないことを示す。また、 <i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sygv* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+2)*n$$

ここで、 nb は、`ilaenv` が `ssytrd/dsytrd` に対して返したブロックサイズである。

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

?hegv

複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chegv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork,
           info)
call zhegv(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork,
           info)
```

Fortran 95:

```
call hegv(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B はエルミートであると仮定する。また、 B は正定値である。

入力パラメーター

<code>itype</code>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <code>itype = 1</code> の場合、問題のタイプは $Ax = \lambda Bx$ である。 <code>itype = 2</code> の場合、問題のタイプは $ABx = \lambda x$ である。 <code>itype = 3</code> の場合、問題のタイプは $BAx = \lambda x$ である。
<code>jobz</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>uplo</code>	CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、配列 a と b は、 A と B の上三角を格納する。 <code>uplo = 'L'</code> の場合、配列 a と b は、 A と B の下三角を格納する。
<code>n</code>	INTEGER。行列 A と B の次数 ($n \geq 0$)。

<i>a, b, work</i>	COMPLEX (chegv の場合) DOUBLE COMPLEX (zhegv の場合)。 配列： <i>a(lda,*)</i> には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b(ldb,*)</i> には、 <i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(lwork)</i> は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。 配列 <i>work</i> の次元。 $lwork \geq \max(1, 2n-1)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (chegv の場合) DOUBLE PRECISION (zhegv の場合)。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

出力パラメーター

<i>a</i>	終了時に、 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>a</i> には、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、次のように正規化される。 <i>itype</i> = 1 または 2 の場合、 $Z^H B Z = I$ 。 <i>itype</i> = 3 の場合、 $Z^H B^{-1} Z = I$ 。 <i>jobz</i> = 'N' の場合、終了時に、 <i>A</i> の上三角 (<i>uplo</i> = 'U' の場合)、または下三角 (<i>uplo</i> = 'L' の場合) が対角を含めて破棄される。
<i>b</i>	終了時に、 <i>info</i> ≤ <i>n</i> の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>w</i>	REAL (chegv の場合) DOUBLE PRECISION (zhegv の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 の場合、 <i>cpotrf/zpotrf</i> と <i>cheev/zheev</i> によって、エラーコードが返される。

$info = i \leq n$ の場合、cheev/zheev が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i (1 \leq i \leq n)$ の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hegv のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (n, n) の行列 A を格納する。
<i>b</i>	サイズ (n, n) の行列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

アプリケーション・ノート

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、 nb は、ilaenv が chetrd/zhetrd に対して返したブロックサイズである。配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?sygvd

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call ssygvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, iwork,
           liwork, info)
call dsygvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, iwork,
           liwork, info)
```

Fortran 95:

```
call sygvd(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B は対称であると仮定する。また、 B は正定値である。

固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 a と b は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 a と b は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (ssygvd の場合) DOUBLE PRECISION (dsygvd の場合)。 配列: $a(lda, *)$ には、 <i>uplo</i> の値に従って、対称行列 A の上三角または下三角を格納する。 a の第2次元は、 $\max(1, n)$ 以上でなければならない。 $b(l db, *)$ には、 <i>uplo</i> の値に従って、対称正定値行列 B の上三角または下三角を格納する。 b の第2次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<i>lda</i>	INTEGER。 a の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 b の第1次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> = 'N' かつ $n > 1$ の場合、 $lwork \geq 2n+1$ 。 <i>jobz</i> = 'V' で $n > 1$ の場合、 $lwork \geq 2n^2+6n+1$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。

liwork INTEGER。配列 *iwork* の次元。次の制約がある。
 $n \leq 1$ の場合、 $liwork \geq 1$ 。
 $jobz = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。
 $jobz = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n+3$ 。
 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエンタリーとして返す。xerbla は *liwork* に関するエラーメッセージを生成しない。

出力パラメーター

a 終了時に、 $jobz = 'V'$ の場合、 $info = 0$ であれば、*a* には、固有ベクトルの行列 *Z* が格納される。固有ベクトルは、次のように正規化される。
 $itype = 1$ または 2 の場合、 $Z^T B Z = I$ 。
 $itype = 3$ の場合、 $Z^T B^{-1} Z = I$ 。
 $jobz = 'N'$ の場合、終了時に、*A* の上三角 ($uplo = 'U'$ の場合)、または下三角 ($uplo = 'L'$ の場合) が対角を含めて破棄される。

b 終了時に、 $info \leq n$ の場合、行列を格納している *b* の一部が、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 *U* または *L* で上書きされる。

w REAL (ssygvd の場合)
DOUBLE PRECISION (dsygvd の場合)。
配列、次元は $\max(1, n)$ 以上。
 $info = 0$ の場合、固有値が昇順で格納される。

work(1) 終了時に、 $info = 0$ の場合、*work(1)* は *lwork* の必要最小サイズを返す。

iwork(1) 終了時に、 $info = 0$ の場合、*iwork(1)* は *liwork* の必要最小サイズを返す。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、*i* 番目の引数が不正であったことを示す。
 $info > 0$ の場合、spotrf/dpotrf と ssyev/dsyev によって、エラーコードが返される。
 $info = i \leq n$ の場合、ssyev/dsyev が収束せず、中間三重対角の *i* 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i$ ($1 \leq i \leq n$) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sygvd` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n,n) の行列 A を格納する。
<code>b</code>	サイズ (n,n) の行列 B を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>itype</code>	1、2、または3でなければならない。デフォルト値は1。
<code>jobz</code>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。

?hegvd

複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call chegvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork,
            lrwork, iwork, liwork, info)
call zhegvd(itype, jobz, uplo, n, a, lda, b, ldb, w, work, lwork, rwork,
            lrwork, iwork, liwork, info)
```

Fortran 95:

```
call hegvd(a, b, w [,itype] [,jobz] [,uplo] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と B はエルミートであると仮定する。また、 B は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<code>itype</code>	INTEGER。1、2、または3のいずれかでなければならない。解決する問題のタイプを指定する。 <code>itype = 1</code> の場合、問題のタイプは $Ax = \lambda Bx$ である。 <code>itype = 2</code> の場合、問題のタイプは $ABx = \lambda x$ である。 <code>itype = 3</code> の場合、問題のタイプは $BAx = \lambda x$ である。
<code>jobz</code>	CHARACTER*1。'N' または 'V' でなければならない。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	COMPLEX (chegvd の場合) DOUBLE COMPLEX (zhegvd の場合)。 配列: <i>a</i> (<i>lda</i> ,*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $lwork \geq n+1$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $lwork \geq n^2+2n$ 。 <i>lwork</i> =-1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (chegvd の場合) DOUBLE PRECISION (zhegvd の場合)。 ワークスペース配列、次元は (<i>lrwork</i>)。
<i>lrwork</i>	INTEGER。配列 <i>rwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $lrwork \geq n$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $lrwork \geq 2n^2+5n+1$ 。 <i>lrwork</i> =-1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>lrwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n+3$ 。

$liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $iwork$ 配列の最適サイズだけを計算し、その値を $iwork$ 配列の最初のエンタリーとして返す。xerbla は $liwork$ に関するエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、 $jobz = 'V'$ の場合、 $info = 0$ であれば、 a には、固有ベクトルの行列 Z が格納される。固有ベクトルは、次のように正規化される。
 $itype = 1$ または 2 の場合、 $Z^H B Z = I$ 。
 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。
- $jobz = 'N'$ の場合、終了時に、 A の上三角 ($uplo = 'U'$ の場合)、または下三角 ($uplo = 'L'$ の場合) が対角を含めて破棄される。
- b** 終了時に、 $info \leq n$ の場合、行列を格納している b の一部が、コレスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 U または L で上書きされる。
- w** REAL (chegvd の場合)
DOUBLE PRECISION (zhegvd の場合)。
配列、次元は $\max(1, n)$ 以上。
 $info = 0$ の場合、固有値が昇順で格納される。
- work(1)** 終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。
- rwork(1)** 終了時に、 $info = 0$ の場合、 $rwork(1)$ は、 $lrwork$ の必要最小サイズを返す。
- iwork(1)** 終了時に、 $info = 0$ の場合、 $iwork(1)$ は $liwork$ の必要最小サイズを返す。
- info** INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目の引数が不正であったことを示す。
 $info > 0$ の場合、cpotrf/zpotrf と cheev/zheev によって、エラーコードが返される。
 $info = i \leq n$ の場合、cheev/zheev が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i$ ($1 \leq i \leq n$) の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hegvd のインターフェイスの詳細を以下に示す。

- a** サイズ (n, n) の行列 A を格納する。

<i>b</i>	サイズ (n,n) の行列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>jobz</i>	'N' または 'V' でなければならない。デフォルト値は 'N'。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。

?sygvx

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call ssygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu,
            abstol, m, w, z, ldz, work, lwork, iwork, ifail, info)
call dsygvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu,
            abstol, m, w, z, ldz, work, lwork, iwork, ifail, info)
```

Fortran 95:

```
call sygvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m]
           [,ifail] [,abstol] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B は対称であると仮定する。また、 B は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または 3 のいずれかでなければならない。解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。

	<p><i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $v_l < \lambda_i \leq v_u$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。</p>
<i>uplo</i>	<p>CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は、<i>A</i> と <i>B</i> の下三角を格納する。</p>
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	<p>REAL (ssygvd の場合) DOUBLE PRECISION (dsygvd の場合)。 配列: <i>a</i>(<i>lda</i>,*) には、<i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>b</i>(<i>ldb</i>,*) には、<i>uplo</i> の値に従って、対称正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>work</i>(<i>lwork</i>) は、ワークスペース配列である。</p>
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>v_l</i> , <i>vu</i>	<p>REAL (ssygvx の場合) DOUBLE PRECISION (dsygvx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v_l < v_u$。 <i>range</i> = 'A' または 'I' の場合、<i>v_l</i> と <i>vu</i> は参照されない。</p>
<i>il</i> , <i>iu</i>	<p>INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il = 1$ かつ $iu = 0$ ($n = 0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、<i>il</i> と <i>iu</i> は参照されない。</p>
<i>abstol</i>	<p>REAL (ssygvx の場合) DOUBLE PRECISION (dsygvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。</p>
<i>ldz</i>	<p>INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$。 <i>jobz</i> = 'V' の場合、$ldz \geq \max(1, n)$。</p>
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 8n)$。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルー</p>

チンは *work* 配列の最適サイズだけを計算し、その値を *work* 配列の最初のエントリーとして返し、*xerbla* は *lwork* に関するエラーメッセージを生成しない。

lwork の推奨値は、「アプリケーション・ノート」を参照。

iwork

INTEGER。

ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

a

終了時に、*A* の上三角 (*uplo* = 'U' の場合)、または下三角 (*uplo* = 'L' の場合) が対角を含めて上書きされる。

b

終了時に、*info* ≤ *n* の場合、行列を格納している *b* の一部が、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 *U* または *L* で上書きされる。

m

INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。

range = 'A' の場合、*m* = *n*、*range* = 'I' の場合、*m* = *iu* - *il* + 1。

w, *z*

REAL (*ssygvx* の場合)

DOUBLE PRECISION (*dsygvx* の場合)。

配列:

w(*)、次元は $\max(1, n)$ 以上。

w の先頭から *m* 個の成分には、指定された固有値が昇順で格納される。

z(*ldz*,*)。 *z* の第 2 次元は、 $\max(1, m)$ 以上でなければならない。*jobz* = 'V' の場合、*info* = 0 であれば、*z* の先頭の *m* 列に、行列 *A* の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、*w*(*i*) に対応する固有ベクトルが、*z* の *i* 番目の列に格納される。固有ベクトルは、次のように正規化される。

itype = 1 または 2 の場合、 $Z^T B Z = I$ 。

itype = 3 の場合、 $Z^T B^{-1} Z = I$ 。

jobz = 'N' の場合、*z* は参照されない。

固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に返される。

注: 配列 *z* には、 $\max(1, m)$ 以上の列が提供されなければならない。*range* = 'V' の場合、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

work(1)

終了時に、*info* = 0 の場合、*work*(1) は *lwork* の必要最小サイズを返す。

ifail

INTEGER。

配列、次元は $\max(1, n)$ 以上。

jobz = 'V' の場合、*info* = 0 であれば、*ifail* の先頭から *m* 個の成分はゼロになる。*info* > 0 であれば、収束しなかった固有ベクトルのインデックスが *ifail* に格納される。

jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目の引数が不正であったことを示す。
info > 0 の場合、spotrf/dpotrf と ssyevx/dsyevx によって、エラーコードが返される。
info = $i \leq n$ の場合、ssyevx/dsyevx が収束せず、*i* 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 *ifail* に格納される。
info = $n + i$ ($1 \leq i \leq n$) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sygvx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は $vu = HUGE(v1)$ である。
<i>i1</i>	この引数のデフォルト値は、 $i1 = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+3)*n$$

ここで、 nb は、 $ilaenv$ が $ssytrd/dsytrd$ に対して返したブロックサイズである。

配列 $work$ に必要なワークスペースの大きさがわからない場合は、最初の実行で $lwork$ を大きめの値に設定する。終了時に、 $work(1)$ の値を確認し、これ以降の実行にはその値を使用する。

?hegvx

複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call chegvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu,
           abstol, m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
call zhegvx(itype, jobz, range, uplo, n, a, lda, b, ldb, vl, vu, il, iu,
           abstol, m, w, z, ldz, work, lwork, rwork, iwork, ifail, info)
```

Fortran 95:

```
call hegvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m]
           [,ifail] [,abstol] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B はエルミートであると仮定する。また、 B は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半开区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>i1</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	COMPLEX (chegvx の場合) DOUBLE COMPLEX (zhegvx の場合)。 配列： <i>a</i> (<i>lda</i> ,*) には、 <i>uplo</i> の値に従って、エルミート行列 <i>A</i> の上三角または下三角を格納する。 <i>a</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には、 <i>uplo</i> の値に従って、エルミート正定値行列 <i>B</i> の上三角または下三角を格納する。 <i>b</i> の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。 <i>a</i> の第1次元。 $\max(1, n)$ 以上。
<i>ldb</i>	INTEGER。 <i>b</i> の第1次元。 $\max(1, n)$ 以上。
<i>v1</i> , <i>vu</i>	REAL (chegvx の場合) DOUBLE PRECISION (zhegvx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>i1</i> , <i>iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq i1 \leq iu \leq n$ ($n > 0$ の場合)。 <i>i1</i> = 1 かつ <i>iu</i> = 0 ($n = 0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>i1</i> と <i>iu</i> は参照されない。

<i>abstol</i>	REAL (<i>chegvx</i> の場合) DOUBLE PRECISION (<i>zhegvx</i> の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 2n-1)$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。 <i>lwork</i> の推奨値は、「アプリケーション・ノート」を参照。
<i>rwork</i>	REAL (<i>chegvx</i> の場合) DOUBLE PRECISION (<i>zhegvx</i> の場合)。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>a</i>	終了時に、 <i>A</i> の上三角 (<i>uplo</i> = 'U' の場合)、または下三角 (<i>uplo</i> = 'L' の場合) が対角を含めて上書きされる。
<i>b</i>	終了時に、 $info \leq n$ の場合、行列を格納している <i>b</i> の一部が、コレスキー因子分解 $B = U^H U$ または $B = L L^H$ からの三角係数 <i>U</i> または <i>L</i> で上書きされる。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (<i>chegvx</i> の場合) DOUBLE PRECISION (<i>zhegvx</i> の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>w</i> の先頭から <i>m</i> 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	COMPLEX (<i>chegvx</i> の場合) DOUBLE COMPLEX (<i>zhegvx</i> の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, m)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 $info = 0$ であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルは、次のように正規化される。 <i>itype</i> = 1 または 2 の場合、 $Z^H B Z = I$ 。 <i>itype</i> = 3 の場合、 $Z^H B^{-1} Z = I$ 。

jobz = 'N' の場合、*z* は参照されない。
固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に返される。
注：配列 *z* には、 $\max(1, m)$ 以上の列が提供されなければならない。*range* = 'V' の場合、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

work(1) 終了時に、*info* = 0 の場合、*work(1)* は *lwork* の必要最小サイズを返す。

ifail INTEGER。
配列、次元は $\max(1, n)$ 以上。
jobz = 'V' の場合、*info* = 0 であれば、*ifail* の先頭から *m* 個の成分はゼロになる。*info* > 0 であれば、収束しなかった固有ベクトルのインデックスが *ifail* に格納される。
jobz = 'N' の場合、*ifail* は参照されない。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目の引数が不正であったことを示す。
info > 0 の場合、*cpotrf/zpotrf* と *cheevx/zheevx* によって、エラーコードが返される。

info = *i* ≤ *n* の場合、*cheevx/zheevx* が収束せず、*i* 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 *ifail* に格納される。
info = *n* + *i* (1 ≤ *i* ≤ *n*) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hegvx* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n, n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n, n</i>) の行列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n, m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>itype</i>	1、2、または3でなければならない。デフォルト値は1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE(<i>v1</i>) である。
<i>vu</i>	この成分のデフォルト値は <i>vu</i> = HUGE(<i>v1</i>) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。

<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。 <i>ifail</i> が存在し、 z が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 $v1$ 、 vu 、 $i1$ 、および iu の存在に基づいて以下のように復元される。 $range = 'V'$ ($v1$ と vu のいずれかまたは両方が存在する場合)、 $range = 'I'$ ($i1$ と iu のいずれかまたは両方が存在する場合)、 $range = 'A'$ ($v1$ 、 vu 、 $i1$ 、および iu がすべて存在しない場合)。 $v1$ と vu のいずれかまたは両方が存在し、同時に $i1$ と iu のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\epsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

最適なパフォーマンスを得るには、次の値を使用する。

$$lwork \geq (nb+1)*n$$

ここで、 nb は、[ilaenv](#) が `chetrd/zhetrd` に対して返したブロックサイズである。

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work*(1) の値を確認し、これ以降の実行にはその値を使用する。

?spgv

圧縮格納形式の行列に関する実数の汎用対称
固有値問題について、固有値と (オプションで)
固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call sspgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)
call dspgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, info)
```

Fortran 95:

```
call spgv(a, b, w [,itype] [,uplo] [,z] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と B は対称であり、圧縮形式で格納されると仮定する。また、 B は正定値である。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ap</i> , <i>bp</i> , <i>work</i>	REAL (sspgv の場合) DOUBLE PRECISION (dspgv の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 A の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>uplo</i> の値に従って、対称行列 B の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 3n)$ 以上。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、 B と同じ格納形式で格納される。
<i>w</i> , <i>z</i>	REAL (sspgv の場合) DOUBLE PRECISION (dspgv の場合)。 配列: <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。 <i>z</i> (<i>ldz</i> , *)。 z の第2次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 z には、固有ベクトルの行列 Z が格納される。固有ベクトルは、次のように正

規化される。

$itype = 1$ または 2 の場合、 $Z^T B Z = I$ 。

$itype = 3$ の場合、 $Z^T B^{-1} Z = I$ 。

$jobz = 'N'$ の場合、 z は参照されない。

info

INTEGER。

$info = 0$ の場合、正常に終了したことを示す。

$info = -i$ の場合、 i 番目の引数が不正であったことを示す。

$info > 0$ の場合、`sppstrf/dppstrf` と `sspev/dspev` によって、エラーコードが返される。

$info = i \leq n$ の場合、`sspev/dspev` が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。

$info = n + i$ ($1 \leq i \leq n$) の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgv` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 a_p を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 b_p を意味する。 サイズ $(n*(n+1)/2)$ の配列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 Z を格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。

?hpgv

圧縮格納形式の行列に関する複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chpgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork, info)
call zhpgv(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, rwork, info)
```

Fortran 95:

```
call hpgv(a, b, w [,itype] [,uplo] [,z] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と B はエルミートであり、圧縮形式で格納されると仮定する。また、 B は正定値である。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ap</i> , <i>bp</i> , <i>work</i>	COMPLEX (chpgv の場合) DOUBLE COMPLEX (zhpgv の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 B の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

	$work(*)$ は、ワークスペース配列、次元は $\max(1, 2n-1)$ 以上。
ldz	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq 1$ 。 $jobz = 'V'$ の場合、 $ldz \geq \max(1, n)$ 。
$rwork$	REAL (chpgv の場合) DOUBLE PRECISION (zhpgv の場合)。 ワークスペース配列、次元は $\max(1, 3n-2)$ 以上。

出力パラメーター

ap	終了時に、 ap の内容は上書きされる。
bp	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、 B と同じ格納形式で格納される。
w	REAL (chpgv の場合) DOUBLE PRECISION (zhpgv の場合)。 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。
z	COMPLEX (chpgv の場合) DOUBLE COMPLEX (zhpgv の場合)。 配列 $z(ldz, *)$ 。 z の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ を指定した場合に $info = 0$ であると、 z には、固有ベクトルの行列 Z が格納される。固有ベクトルは、次のように正規化される。 $itype = 1$ または 2 の場合、 $Z^H B Z = I$ 。 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。 $jobz = 'N'$ の場合、 z は参照されない。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目の引数が不正であったことを示す。 $info > 0$ の場合、cpptrf/zpptrf と chpev/zhpev によって、エラーコードが返される。 $info = i \leq n$ の場合、chpev/zhpev が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。 $info = n + i (1 \leq i \leq n)$ の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hpgv のインターフェイスの詳細を以下に示す。

a	Fortran 77 インターフェイスでの引数 ap を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
-----	---

<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n, n</i>) の行列 <i>Z</i> を格納する。
<i>itype</i>	1、2、または3でなければならない。デフォルト値は1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?spgvd

圧縮格納形式の行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call sspgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, iwork,
           liwork, info)
call dspgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, iwork,
           liwork, info)
```

Fortran 95:

```
call spgvd(a, b, w [,itype] [,uplo] [,z] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と *B* は対称であり、圧縮形式で格納されると仮定する。また、*B* は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。

<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>ap</i> と <i>bp</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>ap, bp, work</i>	REAL (sspgvd の場合) DOUBLE PRECISION (dspgvd の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>B</i> の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> ='V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $lwork \geq 2n$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $lwork \geq 2n^2+6n+1$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='N' で $n > 1$ の場合、 $liwork \geq 1$ 。 <i>jobz</i> ='V' で $n > 1$ の場合、 $liwork \geq 5n+3$ 。 <i>liwork</i> = -1 の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエントリーとして返す。 <i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。

出力パラメーター

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> が、 <i>B</i> と同じ格納形式で格納される。
<i>w, z</i>	REAL (sspgv の場合) DOUBLE PRECISION (dspgv の場合)。 配列: <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。

$z(ldz,*)$ 。 z の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $jobz='V'$ を指定した場合に $info=0$ であると、 z には、固有ベクトルの行列 Z が格納される。固有ベクトルは、次のように正規化される。
 $itype=1$ または 2 の場合、 $Z^T B Z = I$ 。
 $itype=3$ の場合、 $Z^T B^{-1} Z = I$ 。
 $jobz='N'$ の場合、 z は参照されない。

$work(1)$ 終了時に、 $info=0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。

$iwork(1)$ 終了時に、 $info=0$ の場合、 $iwork(1)$ は $liwork$ の必要最小サイズを返す。

$info$ INTEGER。
 $info=0$ の場合、正常に終了したことを示す。
 $info=-i$ の場合、 i 番目の引数が不正であったことを示す。
 $info>0$ の場合、 $sppturf/dppturf$ と $sspevd/dspevd$ によって、エラーコードが返される。
 $info=i \leq n$ の場合、 $sspevd/dspevd$ が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。
 $info=n+i$ ($1 \leq i \leq n$) の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgvd` のインターフェイスの詳細を以下に示す。

a	Fortran 77 インターフェイスでの引数 ap を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
b	Fortran 77 インターフェイスでの引数 bp を意味する。 サイズ $(n*(n+1)/2)$ の配列 B を格納する。
w	長さ (n) のベクトルを格納する。
z	サイズ (n, n) の行列 Z を格納する。
$itype$	1、2、または 3 でなければならない。デフォルト値は 1。
$uplo$	'U' または 'L' でなければならない。デフォルト値は 'U'。
$jobz$	引数 z の存在に基づいて以下のように復元される。 $jobz='V'$ (z が存在する場合)、 $jobz='N'$ (z が省略された場合)。

?hpgvd

圧縮格納形式の行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call chpgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, rwork,
           lrwork, iwork, liwork, info)
call zhpgvd(itype, jobz, uplo, n, ap, bp, w, z, ldz, work, lwork, rwork,
           lrwork, iwork, liwork, info)
```

Fortran 95:

```
call hpgvd(a, b, w [,itype] [,uplo] [,z] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

$$Ax = \lambda Bx, ABx = \lambda x, \text{ または } BAx = \lambda x$$

A と B はエルミートであり、圧縮形式で格納されると仮定する。また、 B は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ap, bp, work</i>	COMPLEX (chpgvd の場合) DOUBLE COMPLEX (zhpgvd の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

$bp(*)$ には、 $uplo$ の値に従って、エルミート行列 B の上三角または下三角を圧縮形式で格納する。

bp の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。

$work(lwork)$ は、ワークスペース配列である。

ldz	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq 1$ 。 $jobz = 'V'$ の場合、 $ldz \geq \max(1, n)$ 。
$lwork$	INTEGER。配列 $work$ の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $lwork \geq n$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $lwork \geq 2n$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエンタリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。
$rwork$	REAL (chpgvd の場合) DOUBLE PRECISION (zhpgvd の場合)。 ワークスペース配列、次元は ($lrwork$)。
$lrwork$	INTEGER。配列 $rwork$ の次元。次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $lrwork \geq n$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $lrwork \geq 2n^2 + 5n + 1$ 。 $lrwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $rwork$ 配列の最適サイズだけを計算し、その値を $rwork$ 配列の最初のエンタリーとして返す。 $xerbla$ は $lrwork$ に関するエラーメッセージを生成しない。
$iwork$	INTEGER。 ワークスペース配列、次元は ($liwork$)。
$liwork$	INTEGER。配列 $iwork$ の次元。次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$ 。 $jobz = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。 $jobz = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n + 3$ 。 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは $iwork$ 配列の最適サイズだけを計算し、その値を $iwork$ 配列の最初のエンタリーとして返す。 $xerbla$ は $liwork$ に関するエラーメッセージを生成しない。

出力パラメーター

ap	終了時に、 ap の内容は上書きされる。
bp	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 U または L が、 B と同じ格納形式で格納される。
w	REAL (chpgvd の場合) DOUBLE PRECISION (zhpgvd の場合)。 配列、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。

<i>z</i>	COMPLEX (chpgvd の場合) DOUBLE COMPLEX (zhpgvd の場合)。 配列 $z(ldz,*)$ 。 z の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ を指定した場合に $info = 0$ であると、 z には、固有ベクトルの行列 Z が格納される。 固有ベクトルは、次のように正規化される。 $itype = 1$ または 2 の場合、 $Z^H B Z = I$ 。 $itype = 3$ の場合、 $Z^H B^{-1} Z = I$ 。 $jobz = 'N'$ の場合、 z は参照されない。
<i>work(1)</i>	終了時に、 $info = 0$ の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>rwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>rwork(1)</i> は、 <i>lrwork</i> の必要最小サイズを返す。
<i>iwork(1)</i>	終了時に、 $info = 0$ の場合、 <i>iwork(1)</i> は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目の引数が不正であったことを示す。 $info > 0$ の場合、 <i>cpptrf/zpptrf</i> と <i>chpevd/zhpevd</i> によって、エラーコードが返される。 $info = i \leq n$ の場合、 <i>chpevd/zhpevd</i> が収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。 $info = n + i$ ($1 \leq i \leq n$) の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpgvd* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 a_p を意味する。 サイズ $(n*(n+1)/2)$ の配列 A を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 b_p を意味する。 サイズ $(n*(n+1)/2)$ の配列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 Z を格納する。
<i>itype</i>	1、2、または 3 でなければならない。デフォルト値は 1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。

?spgvx

圧縮格納形式の行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call sspgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol,
            m, w, z, ldz, work, iwork, ifail, info)
call dspgvx(itype, jobz, range, uplo, n, ap, bp, vl, vu, il, iu, abstol,
            m, w, z, ldz, work, iwork, ifail, info)
```

Fortran 95:

```
call spgvx(a, b, w [,itype] [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m]
           [,ifail] [,abstol] [,info])
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と B は対称であり、圧縮形式で格納されると仮定する。また、 B は正定値である。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>i1</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。

<i>ap, bp, work</i>	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>A</i> の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>uplo</i> の値に従って、対称行列 <i>B</i> の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 8n)$ 以上。
<i>v1, vu</i>	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $v1 < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 次の制約がある。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> が、 <i>B</i> と同じ格納形式で格納される。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w, z</i>	REAL (sspgvx の場合) DOUBLE PRECISION (dspgvx の場合)。 配列: <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。

$z(ldz,*)$ 。 z の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $jobz='V'$ の場合、 $info=0$ であれば、 z の先頭の m 列に、行列 A の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 $w(i)$ に対応する固有ベクトルが、 z の i 番目の列に格納される。固有ベクトルは、次のように正規化される。

$itype=1$ または 2 の場合、 $Z^T B Z = I$ 。
 $itype=3$ の場合、 $Z^T B^{-1} Z = I$ 。

$jobz='N'$ の場合、 z は参照されない。

固有ベクトルが収束できない場合、 z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは $ifail$ に返される。

注：配列 z には、 $\max(1, m)$ 以上の列が提供されなければならない。 $range='V'$ の場合、 m の正確な値が事前にわからないため、上限値を使用する必要がある。

ifail

INTEGER。

配列、次元は $\max(1, n)$ 以上。

$jobz='V'$ の場合、 $info=0$ であれば、 $ifail$ の先頭から m 個の成分はゼロになる。 $info>0$ であれば、収束しなかった固有ベクトルのインデックスが $ifail$ に格納される。

$jobz='N'$ の場合、 $ifail$ は参照されない。

info

INTEGER。

$info=0$ の場合、正常に終了したことを示す。

$info=-i$ の場合、 i 番目の引数が不正であったことを示す。

$info>0$ の場合、 $sppturf/dppturf$ と $sspevx/dspevx$ によって、エラーコードが返される。

$info=i \leq n$ の場合、 $sspevx/dspevx$ が収束せず、 i 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 $ifail$ に格納される。

$info=n+i$ ($1 \leq i \leq n$) の場合、 B について先頭から次数 i の小行列が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `spgvx` のインターフェイスの詳細を以下に示す。

- | | |
|--------------|--|
| <i>a</i> | Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。
サイズ $(n*(n+1)/2)$ の配列 A を格納する。 |
| <i>b</i> | Fortran 77 インターフェイスでの引数 <i>bp</i> を意味する。
サイズ $(n*(n+1)/2)$ の配列 B を格納する。 |
| <i>w</i> | 長さ (n) のベクトルを格納する。 |
| <i>z</i> | サイズ (n, m) の行列 Z を格納する。ここで、値 n と m は有意である。 |
| <i>ifail</i> | 長さ (n) のベクトルを格納する。 |
| <i>itype</i> | 1、2、または 3 でなければならない。デフォルト値は 1。 |

<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
<code>vu</code>	この成分のデフォルト値は $vu = HUGE(v1)$ である。
<code>il</code>	この引数のデフォルト値は、 $il = 1$ である。
<code>iu</code>	この引数のデフォルト値は、 $iu = n$ である。
<code>abstol</code>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。 <code>ifail</code> が存在し、 z が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 vu 、 il 、および iu の存在に基づいて以下のように復元される。 <code>range = 'V'</code> ($v1$ と vu のいずれかまたは両方が存在する場合)、 <code>range = 'I'</code> (il と iu のいずれかまたは両方が存在する場合)、 <code>range = 'A'</code> ($v1$ 、 vu 、 il 、および iu がすべて存在しない場合)。 $v1$ と vu のいずれかまたは両方が存在し、同時に il と iu のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\epsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の2倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?hpgvx

圧縮格納形式の行列に関する汎用エルミート
固有値問題について、固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call chpgvx(itype, jobz, range, uplo, n, ap, bp, v1, vu, il, iu, abstol,
           m, w, z, ldz, work, rwork, iwork, ifail, info)
call zhpgvx(itype, jobz, range, uplo, n, ap, bp, v1, vu, il, iu, abstol,
           m, w, z, ldz, work, rwork, iwork, ifail, info)
```

Fortran 95:

```
call hpgvx(a, b, w [,itype] [,uplo] [,z] [,v1] [,vu] [,i1] [,iu] [,m]
          [,ifail] [,abstol] [,info])
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

$$Ax = \lambda Bx, \quad ABx = \lambda x, \quad \text{または} \quad BAx = \lambda x$$

A と B はエルミートであり、圧縮形式で格納されると仮定する。また、 B は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

<i>itype</i>	INTEGER。1、2、または3のいずれかでなければならない。 解決する問題のタイプを指定する。 <i>itype</i> = 1 の場合、問題のタイプは $Ax = \lambda Bx$ である。 <i>itype</i> = 2 の場合、問題のタイプは $ABx = \lambda x$ である。 <i>itype</i> = 3 の場合、問題のタイプは $BAx = \lambda x$ である。
<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半开区間 $v1 < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>i1</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ap</i> と <i>bp</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ap</i> , <i>bp</i> , <i>work</i>	COMPLEX (chpgvx の場合) DOUBLE COMPLEX (zhpgvx の場合)。 配列: <i>ap</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 A の上三角または下三角を圧縮形式で格納する。 <i>ap</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>bp</i> (*) には、 <i>uplo</i> の値に従って、エルミート行列 B の上三角または下三角を圧縮形式で格納する。 <i>bp</i> の次元は、 $\max(1, n*(n+1)/2)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, 2n)$ 以上。

<i>vl, vu</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $vl < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>ap</i>	終了時に、 <i>ap</i> の内容は上書きされる。
<i>bp</i>	終了時に、コレスキー因子分解 $B = U^T U$ または $B = L L^T$ からの三角係数 <i>U</i> または <i>L</i> が、 <i>B</i> と同じ格納形式で格納される。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (chpgvx の場合) DOUBLE PRECISION (zhpgvx の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chpgvx の場合) DOUBLE COMPLEX (zhpgvx の場合)。 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> の先頭の <i>m</i> 列に、行列 <i>A</i> の選択された固有値に対応する正規直交固有ベクトルが格納される。すなわち、 <i>w</i> (<i>i</i>) に対応する固有ベクトルが、 <i>z</i> の <i>i</i> 番目の列に格納される。固有ベクトルは、次のように正規化される。 <i>itype</i> = 1 または 2 の場合、 $Z^H B Z = I$ 。 <i>itype</i> = 3 の場合、 $Z^H B^{-1} Z = I$ 。

jobz = 'N' の場合、*z* は参照されない。

固有ベクトルが収束できない場合、*z* のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは *ifail* に返される。

注：配列 *z* には、 $\max(1, m)$ 以上の列が提供されなければならない。*range* = 'V' の場合、*m* の正確な値が事前にわからないため、上限値を使用する必要がある。

ifail

INTEGER。

配列、次元は $\max(1, n)$ 以上。

jobz = 'V' の場合、*info* = 0 であれば、*ifail* の先頭から *m* 個の成分はゼロになる。*info* > 0 であれば、収束しなかった固有ベクトルのインデックスが *ifail* に格納される。

jobz = 'N' の場合、*ifail* は参照されない。

info

INTEGER。

info = 0 の場合、正常に終了したことを示す。

info = -*i* の場合、*i* 番目の引数が不正であったことを示す。

info > 0 の場合、*cpptrf/zpptrf* と *chpevx/zhpevx* によって、エラーコードが返される。

info = $i \leq n$ の場合、*chpevx/zhpevx* が収束せず、*i* 個の固有ベクトルが収束しなかったことを示す。これらのインデックスは、配列 *ifail* に格納される。

info = $n + i$ ($1 \leq i \leq n$) の場合、*B* について先頭から次数 *i* の小行列が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *hpgvx* のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ap</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bp</i> を意味する。 サイズ $(n*(n+1)/2)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n,m</i>) の行列 <i>Z</i> を格納する。ここで、値 <i>n</i> と <i>m</i> は有意である。
<i>ifail</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>itype</i>	1、2、または3でなければならない。デフォルト値は1。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は <i>v1</i> = -HUGE(<i>v1</i>) である。
<i>vu</i>	この成分のデフォルト値は <i>vu</i> = HUGE(<i>v1</i>) である。
<i>i1</i>	この引数のデフォルト値は、 <i>i1</i> = 1 である。

<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<i>jobz</i>	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。 <i>ifail</i> が存在し、 z が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 $v1$ 、 vu 、 $i1$ 、および iu の存在に基づいて以下のように復元される。 $range = 'V'$ ($v1$ と vu のいずれかまたは両方が存在する場合)、 $range = 'I'$ ($i1$ と iu のいずれかまたは両方が存在する場合)、 $range = 'A'$ ($v1$ 、 vu 、 $i1$ 、および iu がすべて存在しない場合)。 $v1$ と vu のいずれかまたは両方が存在し、同時に $i1$ と iu のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \varepsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ε はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\varepsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?sbgv

帯行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call ssbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           info)
call dsbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           info)
```

Fortran 95:

```
call sbgv(a, b, w [,uplo] [,z] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。 A と B は対称で帯であると仮定する。また、 B は正定値である。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ($kb \geq 0$)。
<i>ab, bb, work</i>	REAL (ssbgv の場合) DOUBLE PRECISION (dsbgv の場合) 配列： <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> (<i>ldb</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) はワークスペース配列、次元は $\max(1, 3n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。

出力パラメーター

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 spbstf/dpbstf によって返された分割コレスキー因子分解 $B = S^T S$ からの係数 <i>S</i> が格納される。
<i>w, z</i>	REAL (ssbgv の場合) DOUBLE PRECISION (dsbgv の場合) 配列： <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。 <i>z</i> (<i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 <i>z</i> には、 <i>w</i> (<i>i</i>) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、 $Z^T B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重

対角の i 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i$ ($1 \leq i \leq n$) の場合、[spbstf/dpbstf](#) によって
 $info = i$ が返され、 B が正定値でないことを示す。また、 B の因
 子分解が完了できず、固有値または固有ベクトルが計算されな
 かった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbgv` のインターフェイスの詳細を以下に示す。

<code>a</code>	Fortran 77 インターフェイスでの引数 <code>ab</code> を意味する。 サイズ $(ka+1, n)$ の配列 A を格納する。
<code>b</code>	Fortran 77 インターフェイスでの引数 <code>bb</code> を意味する。 サイズ $(kd+1, n)$ の配列 B を格納する。
<code>w</code>	長さ (n) のベクトルを格納する。
<code>z</code>	サイズ (n, n) の行列 Z を格納する。
<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。

?hbgv

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

構文

Fortran 77:

```
call chbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           rwork, info)
call zhbgv(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           rwork, info)
```

Fortran 95:

```
call hbgv(a, b, w [,uplo] [,z] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。 A と B はエルミートで帯であると仮定する。また、 B は正定値である。

入力パラメーター

<i>jobz</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ($kb \geq 0$)。
<i>ab, bb, work</i>	COMPLEX (<i>chbgv</i> の場合) DOUBLE COMPLEX (<i>zhbgv</i> の場合) 配列： <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> (<i>ldb</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>rwork</i>	REAL (<i>chbgv</i> の場合) DOUBLE PRECISION (<i>zhbgv</i> の場合)。 ワークスペース配列、次元は $\max(1, 3n)$ 以上。

出力パラメーター

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 cpbstf/zpbstf によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 <i>S</i> が格納される。
<i>w</i>	REAL (<i>chbgv</i> の場合) DOUBLE PRECISION (<i>zhbgv</i> の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (<i>chbgv</i> の場合) DOUBLE COMPLEX (<i>zhbgv</i> の場合) 配列 <i>z</i> (<i>ldz</i> ,*)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 <i>z</i> には、 <i>w</i> (<i>i</i>) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、

固有ベクトルの行列 Z が格納される。
 固有ベクトルは、 $Z^H B Z = I$ のように正規化される。
 $jobz = 'N'$ の場合、 z は参照されない。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、 i 番目の引数が不正であったことを示す。
 $info > 0$ で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i$ ($1 \leq i \leq n$) の場合、[cpbstf/zpbstf](#) によって $info = i$ が返され、 B が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbgv` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(ka+1, n)$ の配列 A を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ $(kd+1, n)$ の配列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 Z を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 z の存在に基づいて以下のように復元される。 $jobz = 'V'$ (z が存在する場合)、 $jobz = 'N'$ (z が省略された場合)。

?sbgvd

帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call ssbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           lwork, iwork, liwork, info)
call dsbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
           lwork, iwork, liwork, info)
```

Fortran 95:

```
call sbgvd(a, b, w [,uplo] [,z] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。 A と B は対称で帯であると仮定する。また、 B は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> と <i>bb</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> と <i>bb</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 B の優対角成分または劣対角成分の数 ($kb \geq 0$)。
<i>ab, bb, work</i>	REAL (ssbgvd の場合) DOUBLE PRECISION (dsbgvd の場合) 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 A の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> (<i>ldbb</i> ,*) は、(<i>uplo</i> の値に従って) 対称行列 B の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$ 。 <i>jobz</i> = 'N' で $n > 1$ の場合、 $lwork \geq 3n$ 。 <i>jobz</i> = 'V' で $n > 1$ の場合、 $lwork \geq 2n^2 + 5n + 1$ 。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエントリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。

liwork INTEGER。配列 *iwork* の次元。次の制約がある。
 $n \leq 1$ の場合、 $liwork \geq 1$ 。
 $jobz = 'N'$ で $n > 1$ の場合、 $liwork \geq 1$ 。
 $jobz = 'V'$ で $n > 1$ の場合、 $liwork \geq 5n+3$ 。
 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは *iwork* 配列の最適サイズだけを計算し、その値を *iwork* 配列の最初のエントリーとして返す。xerbla は *liwork* に関するエラーメッセージを生成しない。

出力パラメーター

ab 終了時に、*ab* の内容は上書きされる。

bb 終了時に、[spbstf/dpbstf](#) によって返された分割コレスキー因子分解 $B = S^T S$ からの係数 *S* が格納される。

w, z REAL (ssbgvd の場合)
DOUBLE PRECISION (dsbgvd の場合)
配列：
w(*)、次元は $\max(1, n)$ 以上。
 $info = 0$ の場合、固有値が昇順で格納される。
z(*ldz*,*)。 *z* の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $jobz = 'V'$ を指定した場合に $info = 0$ であると、*z* には、*w*(*i*) に関連する固有ベクトルを格納している *z* の *i* 番目の列とともに、固有ベクトルの行列 *Z* が格納される。
固有ベクトルは、 $Z^T B Z = I$ のように正規化される。
 $jobz = 'N'$ の場合、*z* は参照されない。

work(1) 終了時に、 $info = 0$ の場合、*work*(1) は *lwork* の必要最小サイズを返す。

iwork(1) 終了時に、 $info = 0$ の場合、*iwork*(1) は *liwork* の必要最小サイズを返す。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = -i$ の場合、*i* 番目の引数が不正であったことを示す。
 $info > 0$ で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の *i* 個の非対角成分がゼロに収束しなかったことを示す。
 $info = n + i$ ($1 \leq i \leq n$) の場合、[spbstf/dpbstf](#) によって $info = i$ が返され、*B* が正定値でないことを示す。また、*B* の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *sbgvd* のインターフェイスの詳細を以下に示す。

a Fortran 77 インターフェイスでの引数 *ab* を意味する。
サイズ ($ka+1, n$) の配列 *A* を格納する。

<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ $(kd+1, n)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 <i>Z</i> を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。

?hbgvd

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。固有ベクトルを計算する場合は、分割統治法を使用する。

構文

Fortran 77:

```
call chbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
            lwork, rwork, lrwork, iwork, liwork, info)
call zhbgvd(jobz, uplo, n, ka, kb, ab, ldab, bb, ldbb, w, z, ldz, work,
            lwork, rwork, lrwork, iwork, liwork, info)
```

Fortran 95:

```
call hbgvd(a, b, w [,uplo] [,z] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。*A* と *B* はエルミートで帯であると仮定する。また、*B* は正定値である。固有ベクトルを計算する場合は、分割統治法を使用する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> と <i>bb</i> は、 <i>A</i> と <i>B</i> の下三角を格納する。
<i>n</i>	INTEGER。行列 <i>A</i> と <i>B</i> の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ($kb \geq 0$)。

<i>ab, bb, work</i>	<p>COMPLEX (chbgvd の場合) DOUBLE COMPLEX (zhbgvd の場合) 配列: <i>ab(ldab,*)</i> は、(<i>uplo</i> の値に従って)エルミート行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p><i>bb(lddb,*)</i> は、(<i>uplo</i> の値に従って)エルミート行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p> <p><i>work(lwork)</i> は、ワークスペース配列である。</p>
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>lwork</i>	<p>INTEGER。配列 <i>work</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lwork \geq 1$。 $jobz$ = 'N' で $n > 1$ の場合、 $lwork \geq n$。 $jobz$ = 'V' で $n > 1$ の場合、 $lwork \geq 2n^2$。</p> <p>$lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、<i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。</p>
<i>rwork</i>	<p>REAL (chbgvd の場合) DOUBLE PRECISION (zhbgvd の場合)。 ワークスペース配列、次元は (<i>lrwork</i>)。</p>
<i>lrwork</i>	<p>INTEGER。配列 <i>rwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $lrwork \geq 1$。 $jobz$ = 'N' で $n > 1$ の場合、 $lrwork \geq n$。 $jobz$ = 'V' で $n > 1$ の場合、 $lrwork \geq 2n^2+5n+1$。</p> <p>$lrwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>rwork</i> 配列の最適サイズだけを計算し、その値を <i>rwork</i> 配列の最初のエンタリーとして返す。<i>xerbla</i> は <i>lrwork</i> に関するエラーメッセージを生成しない。</p>
<i>iwork</i>	<p>INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。</p>
<i>liwork</i>	<p>INTEGER。配列 <i>iwork</i> の次元。次の制約がある。 $n \leq 1$ の場合、 $liwork \geq 1$。 $jobz$ = 'N' で $n > 1$ の場合、 $liwork \geq 1$。 $jobz$ = 'V' で $n > 1$ の場合、 $liwork \geq 5n+3$。 $liwork = -1$ の場合、ワークスペースのクエリーとみなされ、ルーチンは <i>iwork</i> 配列の最適サイズだけを計算し、その値を <i>iwork</i> 配列の最初のエンタリーとして返す。<i>xerbla</i> は <i>liwork</i> に関するエラーメッセージを生成しない。</p>

出力パラメーター

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 cpbstf/zpbstf によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 <i>S</i> が格納される。
<i>w</i>	REAL (chbgvd の場合) DOUBLE PRECISION (zhbgvd の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z</i>	COMPLEX (chbgvd の場合) DOUBLE COMPLEX (zhbgvd の場合) 配列 <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' の場合、 <i>info</i> = 0 であれば、 <i>z</i> には、 <i>w</i> (<i>i</i>) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 <i>Z</i> が格納される。 固有ベクトルは、 $Z^H B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>work</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>work</i> (1) は <i>lwork</i> の必要最小サイズを返す。
<i>rwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>rwork</i> (1) は、 <i>lrwork</i> の必要最小サイズを返す。
<i>iwork</i> (1)	終了時に、 <i>info</i> = 0 の場合、 <i>iwork</i> (1) は <i>liwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目の引数が不正であったことを示す。 <i>info</i> > 0 で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の <i>i</i> 個の非対角成分がゼロに収束しなかったことを示す。 <i>info</i> = $n + i$ ($1 \leq i \leq n$) の場合、 cpbstf/zpbstf によって <i>info</i> = <i>i</i> が返され、 <i>B</i> が正定値でないことを示す。また、 <i>B</i> の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン hbgvd のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ (<i>ka</i> +1, <i>n</i>) の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ (<i>kd</i> +1, <i>n</i>) の配列 <i>B</i> を格納する。
<i>w</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>z</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>Z</i> を格納する。

<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。

?sbgvx

帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call ssbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl,
            vu, il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
call dsbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, vl,
            vu, il, iu, abstol, m, w, z, ldz, work, iwork, ifail, info)
```

Fortran 95:

```
call sbgvx(a, b, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
           [,q] [,abstol] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する実数の汎用対称固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。 A と B は対称で帯であると仮定する。また、 B は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

入力パラメーター

<code>jobz</code>	CHARACTER*1. 'N' または 'V' でなければならない。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>range</code>	CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <code>range = 'A'</code> の場合、固有値をすべて計算する。 <code>range = 'V'</code> の場合、半開区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <code>range = 'I'</code> の場合、インデックス il から iu の固有値を計算する。
<code>uplo</code>	CHARACTER*1. 'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、配列 <code>ab</code> と <code>bb</code> は、 A と B の上三角を格納する。 <code>uplo = 'L'</code> の場合、配列 <code>ab</code> と <code>bb</code> は、 A と B の下三角を格納する。
<code>n</code>	INTEGER. 行列 A と B の次数 ($n \geq 0$)。

<i>ka</i>	INTEGER。 <i>A</i> の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 <i>B</i> の優対角成分または劣対角成分の数 ($kb \geq 0$)。
<i>ab, bb, work</i>	REAL (ssbgvd の場合) DOUBLE PRECISION (dsbgvd の場合) 配列： <i>ab(ldab,*)</i> は、(<i>uplo</i> の値に従って) 対称行列 <i>A</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb(lddb,*)</i> は、(<i>uplo</i> の値に従って) 対称行列 <i>B</i> の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work(*)</i> は、ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldbb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。
<i>vl, vu</i>	REAL (ssbgvx の場合) DOUBLE PRECISION (dsbgvx の場合)。 <i>range = 'V'</i> の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $vl < vu$ 。 <i>range = 'A'</i> または <i>'I'</i> の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range = 'I'</i> の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 $il=1$ かつ $iu=0$ ($n=0$ の場合)。 <i>range = 'A'</i> または <i>'V'</i> の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (ssbgvx の場合) DOUBLE PRECISION (dsbgvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz = 'V'</i> の場合、 $ldz \geq \max(1, n)$ 。
<i>ldq</i>	INTEGER。出力配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ 。 <i>jobz = 'V'</i> の場合、 $ldq \geq \max(1, n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。

出力パラメーター

<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 spbstf/dpbstf によって返された分割コレスキー因子分解 $B = S^T S$ からの係数 <i>S</i> が格納される。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range = 'A'</i> の場合、 $m = n$ 、 <i>range = 'I'</i> の場合、 $m = iu - il + 1$ 。

w, z, q	<p>REAL (ssbgvx の場合) DOUBLE PRECISION (dsbgvx の場合) 配列: $w(*)$、次元は $\max(1, n)$ 以上。 $info = 0$ の場合、固有値が昇順で格納される。</p> <p>$z(ldz, *)$。 z の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ の場合、$info = 0$ であれば、z には、$w(i)$ に関連する固有ベクトルを格納している z の i 番目の列とともに、固有ベクトルの行列 Z が格納される。固有ベクトルは、$Z^T B Z = I$ のように正規化される。 $jobz = 'N'$ の場合、z は参照されない。</p> <p>$q(ldq, *)$。 q の第 2 次元は、$\max(1, n)$ 以上でなければならない。 $jobz = 'V'$ の場合、q には、$Ax = \lambda Bx$ を標準形式、すなわち $Cx = \lambda x$ に縮退させて、C を三重対角形式に縮退させるために使用する $n \times n$ の行列が格納される。 $jobz = 'N'$ の場合、q は参照されない。</p>
<i>ifail</i>	<p>INTEGER。 配列、次元は $\max(1, n)$ 以上。 $jobz = 'V'$ の場合、$info = 0$ であれば、<i>ifail</i> の先頭から m 個の成分はゼロになる。$info > 0$ であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 $jobz = 'N'$ の場合、<i>ifail</i> は参照されない。</p>
<i>info</i>	<p>INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、i 番目の引数が不正であったことを示す。 $info > 0$ で、$i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。 $info = n + i$ ($1 \leq i \leq n$) の場合、spbstf/dpbstf によって $info = i$ が返され、B が正定値でないことを示す。また、B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。</p>

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `sbgvx` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(ka+1, n)$ の配列 <i>A</i> を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ $(kd+1, n)$ の配列 <i>B</i> を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 <i>Z</i> を格納する。
<i>ifail</i>	長さ (n) のベクトルを格納する。
<i>q</i>	サイズ (n, n) の行列 <i>Q</i> を格納する。

<code>uplo</code>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<code>v1</code>	この成分のデフォルト値は $v1 = -\text{HUGE}(v1)$ である。
<code>vu</code>	この成分のデフォルト値は $vu = \text{HUGE}(v1)$ である。
<code>il</code>	この引数のデフォルト値は、 $il = 1$ である。
<code>iu</code>	この引数のデフォルト値は、 $iu = n$ である。
<code>abstol</code>	この成分のデフォルト値は $abstol = 0.0_WP$ である。
<code>jobz</code>	引数 z の存在に基づいて以下のように復元される。 <code>jobz = 'V'</code> (z が存在する場合)、 <code>jobz = 'N'</code> (z が省略された場合)。 <code>ifail</code> または q が存在し、 z が省略された場合、エラー条件がセットされる。
<code>range</code>	引数 $v1$ 、 vu 、 il 、および iu の存在に基づいて以下のように復元される。 <code>range = 'V'</code> ($v1$ と vu のいずれかまたは両方が存在する場合)、 <code>range = 'I'</code> (il と iu のいずれかまたは両方が存在する場合)、 <code>range = 'A'</code> ($v1$ 、 vu 、 il 、および iu がすべて存在しない場合)。 $v1$ と vu のいずれかまたは両方が存在し、同時に il と iu のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\epsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の2倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつか収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

?hbgvx

帯行列に関する複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

構文

Fortran 77:

```
call chbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, v1,
           vu, il, iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)
call zhbgvx(jobz, range, uplo, n, ka, kb, ab, ldab, bb, ldbb, q, ldq, v1,
           vu, il, iu, abstol, m, w, z, ldz, work, rwork, iwork, ifail, info)
```

Fortran 95:

```
call hbgvx(a, b, w [,uplo] [,z] [,vl] [,vu] [,il] [,iu] [,m] [,ifail]
          [,q] [,abstol] [,info])
```

説明

このルーチンは、 $Ax = \lambda Bx$ の形式で示される帯行列に関する複素数の汎用エルミート固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。 A と B はエルミートで帯であると仮定する。また、 B は正定値である。

固有値と固有ベクトルを選択するには、希望する固有値について、すべての固有値、値の範囲、またはインデックスの範囲を指定する。

入力パラメーター

<i>jobz</i>	CHARACTER*1。'N' または 'V' でなければならない。 <i>jobz</i> = 'N' の場合、固有値のみを計算する。 <i>jobz</i> = 'V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、半开区間 $vl < \lambda_i \leq vu$ の固有値 λ_i を計算する。 <i>range</i> = 'I' の場合、インデックス <i>il</i> から <i>iu</i> の固有値を計算する。
<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、配列 <i>ab</i> と <i>bb</i> は、 A と B の上三角を格納する。 <i>uplo</i> = 'L' の場合、配列 <i>ab</i> と <i>bb</i> は、 A と B の下三角を格納する。
<i>n</i>	INTEGER。行列 A と B の次数 ($n \geq 0$)。
<i>ka</i>	INTEGER。 A の優対角成分または劣対角成分の数 ($ka \geq 0$)。
<i>kb</i>	INTEGER。 B の優対角成分または劣対角成分の数 ($kb \geq 0$)。
<i>ab, bb, work</i>	COMPLEX (chbgvx の場合) DOUBLE COMPLEX (zhbgvx の場合) 配列: <i>ab</i> (<i>ldab</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 A の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>bb</i> (<i>ldb</i> ,*) は、(<i>uplo</i> の値に従って) エルミート行列 B の上三角部分または下三角部分を帯格納形式で格納する配列である。 配列 <i>bb</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $\max(1, n)$ 以上。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元。 $ka+1$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 <i>bb</i> の第 1 次元。 $kb+1$ 以上でなければならない。

<i>vl, vu</i>	REAL (chbgvx の場合) DOUBLE PRECISION (zhbgvx の場合)。 <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 次の制約がある。 $vl < vu$ 。 <i>range</i> = 'A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。次の制約がある。 $1 \leq il \leq iu \leq n$ ($n > 0$ の場合)。 <i>il</i> =1 かつ <i>iu</i> =0 ($n=0$ の場合)。 <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	REAL (chbgvx の場合) DOUBLE PRECISION (zhbgvx の場合)。 固有値に対する絶対エラー許容値。 詳細は、「アプリケーション・ノート」を参照。
<i>ldz</i>	INTEGER。出力配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldz \geq \max(1, n)$ 。
<i>ldq</i>	INTEGER。出力配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ 。 <i>jobz</i> = 'V' の場合、 $ldq \geq \max(1, n)$ 。
<i>rwork</i>	REAL (chbgvx の場合) DOUBLE PRECISION (zhbgvx の場合)。 ワークスペース配列、次元は $\max(1, 7n)$ 以上。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $\max(1, 5n)$ 以上。
出力パラメーター	
<i>ab</i>	終了時に、 <i>ab</i> の内容は上書きされる。
<i>bb</i>	終了時に、 cpbstf / zpbstf によって返された分割コレスキー因子分解 $B = S^H S$ からの係数 <i>S</i> が格納される。
<i>m</i>	INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。 <i>range</i> = 'A' の場合、 $m = n$ 、 <i>range</i> = 'I' の場合、 $m = iu - il + 1$ 。
<i>w</i>	REAL (chbgvx の場合) DOUBLE PRECISION (zhbgvx の場合)。 配列 <i>w</i> (*)、次元は $\max(1, n)$ 以上。 <i>info</i> = 0 の場合、固有値が昇順で格納される。
<i>z, q</i>	COMPLEX (chbgvx の場合) DOUBLE COMPLEX (zhbgvx の場合) 配列： <i>z</i> (<i>ldz</i> , *)。 <i>z</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobz</i> = 'V' を指定した場合に <i>info</i> = 0 であると、 <i>z</i> には、 <i>w</i> (<i>i</i>) に関連する固有ベクトルを格納している <i>z</i> の <i>i</i> 番目の列とともに、固有ベクトルの行列 <i>Z</i> が格納される。固有ベクトルは、 $Z^H B Z = I$ のように正規化される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。

$q(ldq,*)$ 。 q の第2次元は、 $\max(1, n)$ 以上でなければならない。
 $jobz='V'$ の場合、 q には、 $Ax=\lambda Bx$ を標準形式、すなわち
 $Cx=\lambda x$ に縮退させて、 C を三重対角形式に縮退させるために使用
する $n \times n$ の行列が格納される。
 $jobz='N'$ の場合、 q は参照されない。

<i>ifail</i>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 $jobz='V'$ の場合、 $info=0$ であれば、 <i>ifail</i> の先頭から m 個の成分はゼロになる。 $info>0$ であれば、収束しなかった固有ベクトルのインデックスが <i>ifail</i> に格納される。 $jobz='N'$ の場合、 <i>ifail</i> は参照されない。
<i>info</i>	INTEGER。 $info=0$ の場合、正常に終了したことを示す。 $info=-i$ の場合、 i 番目の引数が不正であったことを示す。 $info>0$ で、 $i \leq n$ の場合、アルゴリズムが収束せず、中間三重対角の i 個の非対角成分がゼロに収束しなかったことを示す。 $info=n+i$ ($1 \leq i \leq n$) の場合、 cpbstf/zpbstf によって $info=i$ が返され、 B が正定値でないことを示す。また、 B の因子分解が完了できず、固有値または固有ベクトルが計算されなかった。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `hbgvx` のインターフェイスの詳細を以下に示す。

<i>a</i>	Fortran 77 インターフェイスでの引数 <i>ab</i> を意味する。 サイズ $(ka+1, n)$ の配列 A を格納する。
<i>b</i>	Fortran 77 インターフェイスでの引数 <i>bb</i> を意味する。 サイズ $(kd+1, n)$ の配列 B を格納する。
<i>w</i>	長さ (n) のベクトルを格納する。
<i>z</i>	サイズ (n, n) の行列 Z を格納する。
<i>ifail</i>	長さ (n) のベクトルを格納する。
<i>q</i>	サイズ (n, n) の行列 Q を格納する。
<i>uplo</i>	'U' または 'L' でなければならない。デフォルト値は 'U'。
<i>v1</i>	この成分のデフォルト値は $v1 = -HUGE(v1)$ である。
<i>vu</i>	この成分のデフォルト値は $vu = HUGE(v1)$ である。
<i>il</i>	この引数のデフォルト値は、 $il = 1$ である。
<i>iu</i>	この引数のデフォルト値は、 $iu = n$ である。
<i>abstol</i>	この成分のデフォルト値は $abstol = 0.0_WP$ である。

<i>jobz</i>	引数 <i>z</i> の存在に基づいて以下のように復元される。 <i>jobz</i> = 'V' (<i>z</i> が存在する場合)、 <i>jobz</i> = 'N' (<i>z</i> が省略された場合)。 <i>ifail</i> または <i>q</i> が存在し、 <i>z</i> が省略された場合、エラー条件がセットされる。
<i>range</i>	引数 <i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> の存在に基づいて以下のように復元される。 <i>range</i> = 'V' (<i>v1</i> と <i>vu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'I' (<i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合)、 <i>range</i> = 'A' (<i>v1</i> 、 <i>vu</i> 、 <i>i1</i> 、および <i>iu</i> がすべて存在しない場合)。 <i>v1</i> と <i>vu</i> のいずれかまたは両方が存在し、同時に <i>i1</i> と <i>iu</i> のいずれかまたは両方が存在する場合、エラー条件がセットされる。

アプリケーション・ノート

$abstol + \epsilon * \max(|a|, |b|)$ 以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (ϵ はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol$ がゼロ以下の場合、 $\epsilon * \|T\|_1$ は所定の場所で使用される (T は、 A を三重対角形式に縮退して獲得した三重対角行列)。

固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * ?lamch('S')$ に設定されたときである。このルーチンで $info > 0$ が返された場合、固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * ?lamch('S')$ に設定して、やり直してみる。

汎用非対称固有値問題

このセクションでは、汎用非対称固有値問題を解くために使用する LAPACK ドライバールーチンについて説明する。これらの問題を解くために呼び出すことが可能な [計算ルーチン](#) も参照のこと。

表 4-14 に Fortran-77 インターフェイスのすべてのドライバールーチンを示す。Fortran-95 インターフェイスのそれぞれのルーチン名には、最初の記号をつけない（「[ルーチン命名規則](#)」を参照）。

表 4-14 汎用非対称固有値問題を解くためのドライバールーチン

ルーチン名	機能
?qges	非対称行列のペアについて、汎用固有値、Schur 形式、左 / 右の Schur ベクトルを計算する。
?qgesx	汎用固有値、Schur 形式と (オプションで) Schur ベクトルの左 / 右の行列を計算する。
?qgev	非対称行列のペアについて、汎用固有値、左 / 右の汎用固有ベクトルを計算する。
?qgevx	汎用固有値と (オプションで) 左 / 右の汎用固有ベクトルを計算する。

?qges

非対称行列のペアについて、汎用固有値、Schur 形式、左 / 右の Schur ベクトルを計算する。

構文

Fortran 77:

```
call sgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alphas,
           alphai, beta, vs1, ldvsl, vsr, ldvsr, work, lwork, bwork, info)
call dgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alphas,
           alphai, beta, vs1, ldvsl, vsr, ldvsr, work, lwork, bwork, info)
call cgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alpha,
           beta, vs1, ldvsl, vsr, ldvsr, work, lwork, rwork, bwork, info)
call zgges(jobvsl, jobvsr, sort, selctg, n, a, lda, b, ldb, sdim, alpha,
           beta, vs1, ldvsl, vsr, ldvsr, work, lwork, rwork, bwork, info)
```

Fortran 95:

```
call gges(a, b, alphas, alphai, beta [,vs1] [,vsr] [,select] [,sdim]
          [,info])
call gges(a, b, alpha, beta [,vs1] [,vsr] [,select] [,sdim] [,info])
```

説明

このルーチンは、 $n \times n$ の実 / 複素非対称行列 (A, B) のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式 (S, T) と (オプションで) Schur ベクトルの左 / 右の行列 ($vs1$ と vsr) を計算する。これにより、汎用 Schur 因子分解 $(A, B) = (vs1 * S * vsr^H, vs1 * T * vsr^H)$ が提供される。

また、このルーチンは、固有値の選択した束が上準三角行列 S と上三角行列 T の主対角ブロックにくるように固有値を順序付けられる。 $vs1$ と vsr の先頭の列は、対応する左と右の固有空間 (収縮部分空間) に対する直交 / ユニタリー基を形成する。(汎用固有値だけが必要な場合、より高速なドライバー [zggev](#) を代わりに使用できる)。行列 (A, B) のペアの汎用固有値は、 $A - w*B$ が特異であるようなスカラー w または比 $\alpha / \beta = w$ である。 $\beta=0$ または両方がゼロである妥当な解釈が存在するため、通常は、ペア (α, β) として表現される。 T が非負の対角を持つ上三角で S が 1×1 と 2×2 のブロックを持つブロック上三角である場合、行列 (S, T) のペアは、汎用実数 Schur 形式である。 1×1 ブロックは実数の汎用固有値に対応するが、 S の 2×2 ブロックは、 T の対応する成分を次の形式にすることで「標準化」される。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、 S と T の対応する 2×2 ブロックのペアは、汎用固有値の複素共役ペアを持つ。 S と T が上三角であり、 T の対角が非負の実数の場合、行列 (S, T) のペアは、汎用複素数 Schur 形式になる。

入力パラメーター

jobvs1 CHARACTER*1. 'N' または 'V' でなければならない。
jobvs1='N' の場合、左 Schur ベクトルは計算されない。
jobvs1='V' の場合、左 Schur ベクトルは計算される。

jobvsr CHARACTER*1. 'N' または 'V' でなければならない。
jobvsr='N' の場合、右 Schur ベクトルは計算されない。
jobvsr='V' の場合、右 Schur ベクトルは計算される。

sort CHARACTER*1. 'N' または 'S' でなければならない。
 汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。
sort='N' の場合、固有値は順序付けされない。
sort='S' の場合、固有値は順序付けされる (*select* を参照)。

selctg 3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)。
 2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)。
selctg は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。
sort='S' の場合、**selctg** は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。
sort='N' の場合、**selctg** は参照されない。

実数型の場合:
selctg(*alphan*(j), *alpha*(j), *beta*(j)) が真の場合、固有値 (*alphan*(j) + *alpha*(j))/*beta*(j) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。
 悪条件の場合、指定された複素数の固有値は、順序付けの実行後、**selctg**(*alphan*(j), *alpha*(j), *beta*(j)) = .TRUE. を満たさないことがある。その場合、*info* に $n+2$ を設定する。

	<p>複素数型の場合: <code>selectg(alpha(j), beta(j))</code> が真の場合、固有値 $\alpha(j) / \beta(j)$ が選択される。 順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、<code>selectg(alpha(j), beta(j)) = .TRUE.</code> を満たさないことがある。その場合、<code>info</code> に $n+2$ が設定される (以下の <code>info</code> を参照)。</p>
<code>n</code>	INTEGER。行列 A 、 B 、 $vs1$ 、および vsr の次数 ($n \geq 0$)。
<code>a, b, work</code>	REAL (sgges の場合) DOUBLE PRECISION (dggges の場合) COMPLEX (cgges の場合) DOUBLE COMPLEX (zgges の場合)。 配列: $a(lda, *)$ は、 $n \times n$ の行列 A (行列のペアの 1 番目) を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $b(ldb, *)$ は、 $n \times n$ の行列 B (行列のペアの 2 番目) を格納する配列である。 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(lwork)$ は、ワークスペース配列である。
<code>lda</code>	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<code>ldb</code>	INTEGER。配列 b の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<code>ldvs1, ldvsr</code>	INTEGER。それぞれ、出力行列 $vs1$ と vsr の第 1 次元。 次の制約がある。 $ldvs1 \geq 1$ 。 $jobvs1 = 'V'$ の場合、 $ldvs1 \geq \max(1, n)$ 。 $ldvsr \geq 1$ 。 $jobvsr = 'V'$ の場合、 $ldvsr \geq \max(1, n)$ 。
<code>lwork</code>	INTEGER。配列 $work$ の次元。 $lwork \geq \max(1, 8n+16)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 <code>xerbla</code> は $lwork$ に関するエラーメッセージを生成しない。
<code>rwork</code>	REAL (cgges の場合) DOUBLE PRECISION (zgges の場合) ワークスペース配列、次元は $\max(1, 8n)$ 以上。 この配列は、複素数型でのみ使用される。
<code>bwork</code>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 $sort = 'N'$ の場合は参照されない。

出力パラメーター

<i>a</i>	終了時に、その汎用 Schur 形式 <i>S</i> で上書きされる。
<i>b</i>	終了時に、その汎用 Schur 形式 <i>T</i> で上書きされる。
<i>sdim</i>	INTEGER。 <i>sort</i> = 'N' の場合、 <i>sdim</i> = 0 である。 <i>sort</i> = 'S' の場合、 <i>sdim</i> は、 <i>selctg</i> が真の (ソート後の) 固有値の数に等しくなる。 実数型の場合、どちらかの固有値について <i>selctg</i> が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。
<i>alphan, alphai</i>	REAL (sgges の場合) DOUBLE PRECISION (dggges の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>alpha</i>	COMPLEX (cgges の場合) DOUBLE COMPLEX (zgges の場合)。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>beta</i>	REAL (sgges の場合) DOUBLE PRECISION (dggges の場合) COMPLEX (cgges の場合) DOUBLE COMPLEX (zgges の場合)。 配列、次元は $\max(1, n)$ 以上。 実数型の場合： 終了時に、 $(\text{alphan}(j) + \text{alphai}(j)*i)/\text{beta}(j)$ 、 $j=1, \dots, n$ は、汎用固有値になる。 $\text{alphan}(j) + \text{alphai}(j)*i$ と $\text{beta}(j)$ 、 $j=1, \dots, n$ は、 (A, B) の実数汎用 Schur 形式の 2×2 の対角ブロックを複素数のユニタリー変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 (S, T) の対角である。 $\text{alphai}(j)$ がゼロの場合、 j 番目の固有値が実数であり、正の場合、 j 番目と $(j+1)$ 番目の固有値が複素共役ペアである ($\text{alphai}(j+1)$ は負)。 複素数型の場合： 終了時に、 $\text{alpha}(j)/\text{beta}(j)$ 、 $j=1, \dots, n$ は、汎用固有値になる。 $\text{alpha}(j)$ 、 $j=1, \dots, n$ と $\text{beta}(j)$ 、 $j=1, \dots, n$ は、cgges/zgges による複素数の Schur 形式 (S, T) 出力の対角である。 $\text{beta}(j)$ は、非負の実数になる。 次の「アプリケーション・ノート」も参照のこと。
<i>vs1, vsr</i>	REAL (sgges の場合) DOUBLE PRECISION (dggges の場合) COMPLEX (cgges の場合) DOUBLE COMPLEX (zgges の場合)。 配列： $\text{vs1}(\text{ldvs1}, *)$ 、 vs1 の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>jobvs1</i> = 'V' の場合、左 Schur ベクトルが格納される。 <i>jobvs1</i> = 'N' の場合、 <i>vs1</i> は参照されない。

	$vsl(ldvsl, *)$ 、 vsl の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
	$jobvsl = 'V'$ の場合、右 Schur ベクトルが格納される。
	$jobvsl = 'N'$ の場合、 vsl は参照されない。
$work(1)$	終了時に、 $info = 0$ の場合、 $work(1)$ は $lwork$ の必要最小サイズを返す。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i$ 、かつ $i \leq n$: QZ の繰り返し失敗した。 (A, B) は Schur 形式でないが、 $\alpha_{phar}(j)$ 、 $\alpha_{hai}(j)$ (実数型の場合)、または $\alpha(j)$ (複素数型の場合)、および $\beta(j)$ 、 $j = info + 1, \dots, n$ は正しい。 $i > n$ の場合：一般に LAPACK の問題を示すエラー。 $i = n + 1$ の場合： ?hgeqz で QZ の繰り返し以外の操作に失敗した。 $i = n + 2$ の場合：順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が $selctg = .TRUE.$ を満たさなくなった。これは、スケールリングによっても引き起こされる。 $i = n + 3$ の場合： ?tqsen で順序変更失敗した。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `gges` のインターフェイスの詳細を以下に示す。

a	サイズ (n, n) の行列 A を格納する。
b	サイズ (n, n) の行列 B を格納する。
α_{phar}	長さ (n) のベクトルを格納する。実数型でのみ使用される。
α_{hai}	長さ (n) のベクトルを格納する。実数型でのみ使用される。
α	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
β	長さ (n) のベクトルを格納する。
vsl	サイズ (n, n) の行列 VSL を格納する。
vsl	サイズ (n, n) の行列 VSR を格納する。
$jobvsl$	引数 vsl の存在に基づいて以下のように復元される。 $jobvsl = 'V'$ (vsl が存在する場合)、 $jobvsl = 'N'$ (vsl が省略された場合)。
$jobvsl$	引数 vsl の存在に基づいて以下のように復元される。 $jobvsl = 'V'$ (vsl が存在する場合)、 $jobvsl = 'N'$ (vsl が省略された場合)。

sort 引数 *select* の存在に基づいて以下のように復元される。
sort = 'S' (*select* が存在する場合)、
sort = 'N' (*select* が省略された場合)。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

一般に、商 $\alpha(j)/\beta(j)$ と $\alpha_{\text{real}}(j)/\beta(j)$ は、簡単にオーバーフローまたはアンダーフローし、 $\beta(j)$ がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、 α と α_{real} は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、 β は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

?ggesx

汎用固有値、Schur 形式と (オプションで) Schur ベクトルの左/右の行列を計算する。

構文

Fortran 77:

```
call sggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
            sdim, alphas, alpha, beta, vs1, ldvsl, vsr, ldvsr, rconde, rcondv,
            work, lwork, iwork, liwork, bwork, info)

call dggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
            sdim, alphas, alpha, beta, vs1, ldvsl, vsr, ldvsr, rconde, rcondv,
            work, lwork, iwork, liwork, bwork, info)

call cggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
            sdim, alpha, beta, vs1, ldvsl, vsr, ldvsr, rconde, rcondv, work,
            lwork, rwork, iwork, liwork, bwork, info)

call zggesx (jobvsl, jobvsr, sort, selctg, sense, n, a, lda, b, ldb,
            sdim, alpha, beta, vs1, ldvsl, vsr, ldvsr, rconde, rcondv, work,
            lwork, rwork, iwork, liwork, bwork, info)
```

Fortran 95:

```
call ggesx(a, b, alphas, alpha, beta [,vs1] [,vsr] [,select] [,sdim]
            [,rconde] [,rcondv] [,info])

call ggesx(a, b, alpha, beta [,vs1] [,vsr] [,select] [,sdim] [,rconde]
            [,rcondv] [,info])
```

説明

このルーチンは、 $n \times n$ の実 / 複素非対称行列 (A, B) のペアについて、汎用固有値、汎用実数 / 複素数 Schur 形式 (S, T) と (オプションで) Schur ベクトルの左 / 右の行列 (vs1 と vsr) を計算する。これにより、汎用 Schur 因子分解 $(A, B) = (vs1 * S * vsr^H, vs1 * T * vsr^H)$ が提供される。

このルーチンは、固有値の選択した束が上準三角行列 S と上三角行列 T の主対角ブロックにくるように固有値を順序付けられる。また、指定された固有値の平均に対する条件数の逆数 ($rconde$) を計算し、指定された固有値に対応する右と左の収縮部分空間に対する条件数の逆数 ($rcondv$) を計算することもできる。 $vs1$ と vsr の先頭の列は、対応する左と右の固有空間 (収縮部分空間) に対する直交 / ユニタリー基を形成する。

行列 (A, B) のペアの汎用固有値は、 $A - w*B$ が特異であるようなスカラー w または比 $alpha / beta = w$ である。 $beta=0$ または両方がゼロである妥当な解釈が存在するため、通常は、ペア ($alpha, beta$) として表現される。

T が非負の対角を持つ上三角で S が 1×1 と 2×2 のブロックを持つブロック上三角である場合、行列 (S, T) のペアは、汎用実数 Schur 形式である。 1×1 ブロックは実数の汎用固有値に対応するが、 S の 2×2 ブロックは、 T の対応する成分を次の形式にすることで「標準化」される。

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

また、 S と T の対応する 2×2 ブロックのペアは、汎用固有値の複素共役ペアを持つ。 S と T が上三角であり、 T の対角が非負の実数の場合、行列 (S, T) のペアは、汎用複素数 Schur 形式になる。

入力パラメーター

<code>jobvs1</code>	CHARACTER*1. 'N' または 'V' でなければならない。 <code>jobvs1='N'</code> の場合、左 Schur ベクトルは計算されない。 <code>jobvs1='V'</code> の場合、左 Schur ベクトルは計算される。
<code>jobvsr</code>	CHARACTER*1. 'N' または 'V' でなければならない。 <code>jobvsr='N'</code> の場合、右 Schur ベクトルは計算されない。 <code>jobvsr='V'</code> の場合、右 Schur ベクトルは計算される。
<code>sort</code>	CHARACTER*1. 'N' または 'S' でなければならない。 汎用 Schur 形式の対角上の固有値を順序付けるかどうかを指定する。 <code>sort='N'</code> の場合、固有値は順序付けされない。 <code>sort='S'</code> の場合、固有値は順序付けされる (<code>select</code> を参照)。
<code>selctg</code>	3 つの REAL 引数の LOGICAL FUNCTION (実数型の場合)。 2 つの COMPLEX 引数の LOGICAL FUNCTION (複素数型の場合)。 <code>selctg</code> は、呼び出しサブルーチン内で EXTERNAL として宣言する必要がある。 <code>sort='S'</code> の場合、 <code>selctg</code> は、Schur 形式の左上に対してソートを実行する固有値を選択するために使用される。 <code>sort='N'</code> の場合、 <code>selctg</code> は参照されない。 実数型の場合: <code>selctg(alphar(j), alphai(j), beta(j))</code> が真の場合、固有値 ($alphar(j) + alphai(j)/beta(j)$) が選択される。つまり、固有値の複素共役ペアの一方が選択されると、複素数の固有値が両方選択される。 悪条件の場合、指定された複素数の固有値は、順序付けの実行後、 <code>selctg(alphar(j), alphai(j), beta(j)) = .TRUE.</code> を満たさないことがある。その場合、 <code>info</code> に $n+2$ を設定する。

	<p>複素数型の場合: <code>selctg(alpha(j), beta(j))</code> が真の場合、固有値 $\alpha(j) / \beta(j)$ が選択される。 順序付けを行うと、複素数の固有値の値が変更される可能性がある (特に、固有値が悪い条件の場合)。そのため、順序付けを実行した後、指定された複素数の固有値は、<code>selctg(alpha(j), beta(j)) = .TRUE.</code> を満たさないことがある。その場合、<code>info</code> に $n+2$ が設定される (以下の <code>info</code> を参照)。</p>
<code>sense</code>	<p>CHARACTER*1。'N'、'E'、'V'、または 'B' でなければならない。どの条件数の逆数を計算するのかを決定する。</p> <p><code>sense = 'N'</code> の場合、計算は実行されない。 <code>sense = 'E'</code> の場合、指定された固有値の平均についてのみ計算が実行される。 <code>sense = 'V'</code> の場合、指定された収縮部分空間についてのみ計算が実行される。 <code>sense = 'B'</code> の場合、両方について計算が実行される。 <code>sense</code> が 'E'、'V'、または 'B' の場合、<code>sort</code> は 'S' に等しくなければならない。</p>
<code>n</code>	INTEGER。行列 <i>A</i> 、 <i>B</i> 、 <i>vs1</i> 、および <i>vsr</i> の次数 ($n \geq 0$)。
<code>a, b, work</code>	<p>REAL (sggesx の場合) DOUBLE PRECISION (dggex の場合) COMPLEX (cggesx の場合) DOUBLE COMPLEX (zggesx の場合)。 配列: <code>a(lda,*)</code> は、$n \times n$ の行列 <i>A</i> (行列のペアの 1 番目) を格納する配列である。 <code>a</code> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <code>b(l db,*)</code> は、$n \times n$ の行列 <i>B</i> (行列のペアの 2 番目) を格納する配列である。 <code>b</code> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <code>work(lwork)</code> は、ワークスペース配列である。</p>
<code>lda</code>	<p>INTEGER。配列 <code>a</code> の第 1 次元。 $\max(1, n)$ 以上でなければならない。</p>
<code>ldb</code>	<p>INTEGER。配列 <code>b</code> の第 1 次元。 $\max(1, n)$ 以上でなければならない。</p>
<code>ldvs1, ldvsr</code>	<p>INTEGER。それぞれ、出力行列 <i>vs1</i> と <i>vsr</i> の第 1 次元。 次の制約がある。 <code>ldvs1</code> ≥ 1。 <code>jobvs1 = 'V'</code> の場合、<code>ldvs1</code> $\geq \max(1, n)$。 <code>ldvsr</code> ≥ 1。 <code>jobvsr = 'V'</code> の場合、<code>ldvsr</code> $\geq \max(1, n)$。</p>
<code>lwork</code>	<p>INTEGER。配列 <code>work</code> の次元。 実数型の場合: <code>lwork</code> $\geq \max(1, 8(n+1)+16)$。 <code>sense = 'E'、'V'、または 'B'</code> の場合、 <code>lwork</code> $\geq \max(8(n+1)+16, 2*sdim*(n-sdim))$。</p>

複素数型の場合:

$lwork \geq \max(1, 2n)$ 。

$sense = 'E'$ 、 $'V'$ 、または $'B'$ の場合、

$lwork \geq \max(2n, 2 * sdim * (n - sdim))$ 。

パフォーマンスを改善するには、一般に $lwork$ に上記以上の値が必要である。

<i>rwork</i>	REAL (cggesx の場合) DOUBLE PRECISION (zggesx の場合) ワークスペース配列、次元は $\max(1, 8n)$ 以上。 この配列は、複素数型でのみ使用される。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。 $sense = 'N'$ の場合は参照されない。
<i>liwork</i>	INTEGER。配列 <i>iwork</i> の次元。 $liwork \geq n + 6$ (実数型の場合)。 $liwork \geq n + 2$ (複素数型の場合)。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 $sort = 'N'$ の場合は参照されない。

出力パラメーター

<i>a</i>	終了時に、その汎用 Schur 形式 <i>S</i> で上書きされる。
<i>b</i>	終了時に、その汎用 Schur 形式 <i>T</i> で上書きされる。
<i>sdim</i>	INTEGER。 $sort = 'N'$ の場合、 $sdim = 0$ である。 $sort = 'S'$ の場合、 <i>sdim</i> は、 <i>selctg</i> が真の (ソート後の) 固有値の数に等しくなる。 実数型の場合、どちらかの固有値について <i>selctg</i> が真の複素共役ペアは、2 としてカウントされるのに注意が必要である。
<i>alphar, alphai</i>	REAL (sggesx の場合) DOUBLE PRECISION (dggesx の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>alpha</i>	COMPLEX (cggesx の場合) DOUBLE COMPLEX (zggesx の場合)。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。
<i>beta</i>	REAL (sggesx の場合) DOUBLE PRECISION (dggesx の場合) COMPLEX (cggesx の場合) DOUBLE COMPLEX (zggesx の場合)。 配列、次元は $\max(1, n)$ 以上。 実数型の場合: 終了時に、 $(alphar(j) + alphai(j)*i)/beta(j)$ 、 $j = 1, \dots, n$ は、汎用

固有値になる。

$\alpha(j) + \alpha_i(j) \cdot i$ と $\beta(j)$ 、 $j=1, \dots, n$ は、 (A, B) の実数汎用 Schur 形式の 2×2 の対角ブロックを複素数のユニタリー変換を使用して三角形式に更に縮退する場合に生じる複素数 Schur 形式 (S, T) の対角である。 $\alpha_i(j)$ がゼロの場合、 j 番目の固有値が実数であり、正の場合、 j 番目と $(j+1)$ 番目の固有値が複素共役ペアである ($\alpha_i(j+1)$ は負)。

複素数型の場合：

終了時に、 $\alpha(j)/\beta(j)$ 、 $j=1, \dots, n$ は、汎用固有値になる。 $\alpha(j)$ 、 $j=1, \dots, n$ と $\beta(j)$ 、 $j=1, \dots, n$ は、`cggesx/zggesx` による複素数の Schur 形式 (S, T) 出力の対角である。 $\beta(j)$ は、非負の実数になる。

次の「アプリケーション・ノート」も参照のこと。

`vs1, vsr`

REAL (`sggesx` の場合)

DOUBLE PRECISION (`dggex` の場合)

COMPLEX (`cggesx` の場合)

DOUBLE COMPLEX (`zggesx` の場合)。

配列：

`vs1(ldvs1,*)`、`vs1` の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

`jobvs1='V'` の場合、左 Schur ベクトルが格納される。

`jobvs1='N'` の場合、`vs1` は参照されない。

`vsr(ldvsr,*)`、`vsr` の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

`jobvsr='V'` の場合、右 Schur ベクトルが格納される。

`jobvsr='N'` の場合、`vsr` は参照されない。

`rconde, rcondv`

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列は、次元はそれぞれ (2)

`sense='E'` または `'B'` の場合、`rconde(1)` と `rconde(2)` には、指定された固有値の平均に対する条件数の逆数が格納される。

`sense='N'` または `'V'` の場合は参照されない。

`sense='V'` または `'B'` の場合、`rcondv(1)` と `rcondv(2)` には、指定された収縮部分空間に対する条件数の逆数が格納される。

`sense='N'` または `'E'` の場合は参照されない。

`work(1)`

終了時に、`info=0` の場合、`work(1)` は `lwork` の必要最小サイズを返す。

`info`

INTEGER。

`info=0` の場合、正常に終了したことを示す。

`info=-i` の場合、 i 番目のパラメーターの値が不正である。

`info=i`、かつ $i \leq n$ ：

`QZ` の繰り返し失敗した。 (A, B) は Schur 形式でないが、 $\alpha(j)$ 、 $\alpha_i(j)$ (実数型の場合)、または $\alpha(j)$ (複素数型の場合)、および $\beta(j)$ 、 $j=info+1, \dots, n$ は正しい。

$i > n$ の場合：一般に LAPACK の問題を示すエラー。

$i = n+1$ の場合: [?hgeqz](#) で QZ の繰り返し以外の操作に失敗した。

$i = n+2$ の場合: 順序変更後、丸め操作によりいくつかの複素数の固有値の値が変更され、その結果、汎用 Schur 形式の先頭の固有値が `selctg = .TRUE.` を満たさなくなった。これは、スケーリングによっても引き起こされる。

$i = n+3$ の場合: [?tgsen](#) で順序変更失敗した。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン `ggesx` のインターフェイスの詳細を以下に示す。

<code>a</code>	サイズ (n, n) の行列 A を格納する。
<code>b</code>	サイズ (n, n) の行列 B を格納する。
<code>alphar</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>alphai</code>	長さ (n) のベクトルを格納する。実数型でのみ使用される。
<code>alpha</code>	長さ (n) のベクトルを格納する。複素数型でのみ使用される。
<code>beta</code>	長さ (n) のベクトルを格納する。
<code>vs1</code>	サイズ (n, n) の行列 VSL を格納する。
<code>vsr</code>	サイズ (n, n) の行列 VSR を格納する。
<code>rconde</code>	長さ (2) のベクトルを格納する。
<code>rcondv</code>	長さ (2) のベクトルを格納する。
<code>jobvs1</code>	引数 <code>vs1</code> の存在に基づいて以下のように復元される。 <code>jobvs1 = 'V'</code> (<code>vs1</code> が存在する場合)、 <code>jobvs1 = 'N'</code> (<code>vs1</code> が省略された場合)。
<code>jobvsr</code>	引数 <code>vsr</code> の存在に基づいて以下のように復元される。 <code>jobvsr = 'V'</code> (<code>vsr</code> が存在する場合)、 <code>jobvsr = 'N'</code> (<code>vsr</code> が省略された場合)。
<code>sort</code>	引数 <code>select</code> の存在に基づいて以下のように復元される。 <code>sort = 'S'</code> (<code>select</code> が存在する場合)、 <code>sort = 'N'</code> (<code>select</code> が省略された場合)。
<code>sense</code>	引数 <code>rconde</code> と <code>rcondv</code> の存在に基づいて以下のように復元される。 <code>sense = 'B'</code> (<code>rconde</code> と <code>rcondv</code> の両方が存在する場合)、 <code>sense = 'E'</code> (<code>rconde</code> が存在し、 <code>rcondv</code> が省略された場合)、 <code>sense = 'V'</code> (<code>rconde</code> が省略され、 <code>rcondv</code> が存在する場合)、 <code>sense = 'N'</code> (<code>rconde</code> と <code>rcondv</code> の両方が省略された場合)。

`rconde` または `rcondv` が存在し、`select` が省略された場合、エラー条件がセットされる。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

一般に、商 $\text{alphan}(j)/\text{betan}(j)$ と $\text{alphai}(j)/\text{betai}(j)$ は、簡単にオーバーフローまたはアンダーフローし、 $\text{betan}(j)$ がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、 alphan と alphai は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、 betan は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

?ggev

非対称行列のペアについて、汎用固有値、左/右の汎用固有ベクトルを計算する。

構文

Fortran 77:

```
call sggev(jobvl, jobvr, n, a, lda, b, ldb, alphan, alphai, betan, vl,
           ldvl, vr, ldvr, work, lwork, info)
call dggev(jobvl, jobvr, n, a, lda, b, ldb, alphan, alphai, betan, vl,
           ldvl, vr, ldvr, work, lwork, info)
call cggev(jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl, vr,
           ldvr, work, lwork, rwork, info)
call zggev(jobvl, jobvr, n, a, lda, b, ldb, alpha, beta, vl, ldvl, vr,
           ldvr, work, lwork, rwork, info)
```

Fortran 95:

```
call ggev(a, b, alphan, alphai, betan [,vl] [,vr] [,info])
call ggev(a, b, alpha, beta [,vl] [,vr] [,info])
```

説明

このルーチンは、 $n \times n$ の実/複素非対称行列 (A, B) のペアについて、汎用固有値と (オプションで) 左/右の汎用固有ベクトルを計算する。

行列 (A, B) のペアの汎用固有値は、 $A - \lambda * B$ が特異であるようなスカラー λ または比 $\text{alpha} / \text{beta} = \lambda$ である。 $\text{beta}=0$ および両方がゼロである妥当な解釈が存在するため、通常は、ペア (alpha, beta) として表現される。

(A, B) の汎用固有値 $\lambda(j)$ に対応する右汎用固有ベクトル $v(j)$ は、以下を満たす。

$$A * v(j) = \lambda(j) * B * v(j)$$

(A, B) の汎用固有値 $\lambda(j)$ に対応する左汎用固有ベクトル $u(j)$ は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H * B$$

$u(j)^H$ は、 $u(j)$ の共役転置を示す。

入力パラメーター

<i>jobvl</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobvl</i> ='N' の場合、左汎用固有ベクトルは計算されない。 <i>jobvl</i> ='V' の場合、左汎用固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobvr</i> ='N' の場合、右汎用固有ベクトルは計算されない。 <i>jobvr</i> ='V' の場合、右汎用固有ベクトルは計算される。
<i>n</i>	INTEGER。行列 <i>A</i> 、 <i>B</i> 、 <i>v1</i> 、 <i>vr</i> の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (sggev の場合) DOUBLE PRECISION (dggev の場合) COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。 配列: <i>a</i> (<i>lda</i> ,*) は、 $n \times n$ の行列 <i>A</i> (行列のペアの 1 番目) を格納する配列である。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) は、 $n \times n$ の行列 <i>B</i> (行列のペアの 2 番目) を格納する配列である。 <i>b</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>lwork</i>) は、ワークスペース配列である。
<i>lda</i>	INTEGER。配列 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 <i>b</i> の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldv1</i> , <i>ldvr</i>	INTEGER。それぞれ、出力行列 <i>v1</i> と <i>vr</i> の第 1 次元。 次の制約がある。 $ldv1 \geq 1$ 。 <i>jobvl</i> ='V' の場合、 $ldv1 \geq \max(1, n)$ 。 $ldvr \geq 1$ 。 <i>jobvr</i> ='V' の場合、 $ldvr \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 <i>work</i> の次元。 $lwork \geq \max(1, 8n+16)$ (実数型の場合)。 $lwork \geq \max(1, 2n)$ (複素数型の場合)。 パフォーマンスを改善するには、一般に <i>lwork</i> に上記以上の値が必要である。 <i>lwork</i> = -1 の場合はワークスペースのクエリーとみなされ、ルーチンは <i>work</i> 配列の最適サイズだけを計算し、その値を <i>work</i> 配列の最初のエンタリーとして返し、 <i>xerbla</i> は <i>lwork</i> に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (cggev の場合) DOUBLE PRECISION (zggev の場合) ワークスペース配列、次元は $\max(1, 8n)$ 以上。 この配列は、複素数型でのみ使用される。

出力パラメーター

a, *b* 終了時に上書きされる。

<i>alphar, alphai</i>	<p>REAL (sggev の場合) DOUBLE PRECISION (dggev の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。beta を参照。</p>
<i>alpha</i>	<p>COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。beta を参照。</p>
<i>beta</i>	<p>REAL (sggev の場合) DOUBLE PRECISION (dggev の場合) COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>実数型の場合:</i> 終了時に、$(\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$、$j=1, \dots, n$ は、汎用固有値になる。 $\text{alphai}(j)$ がゼロの場合、j 番目の固有値が実数であり、正の場合、j 番目と $(j+1)$ 番目の固有値が複素共役ペアである ($\text{alphai}(j+1)$ は負)。 <i>複素数型の場合:</i> 終了時に、$\text{alpha}(j)/\text{beta}(j)$、$j=1, \dots, n$ は、汎用固有値になる。 次の「アプリケーション・ノート」も参照のこと。</p>
<i>v1, vr</i>	<p>REAL (sggev の場合) DOUBLE PRECISION (dggev の場合) COMPLEX (cggev の場合) DOUBLE COMPLEX (zggev の場合)。 配列： $v1(ldv1, *)$。v1 の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>jobv1='V' の場合</i>、左汎用固有ベクトル $u(j)$ は、それらの固有値と同じ順序で、v1 の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$ となるようにスケールされる。 <i>jobv1='N' の場合</i>、v1 は参照されない。 <i>実数型の場合:</i> j 番目の固有値が実数の場合、$u(j) = v1(:, j)$ (v1 の j 番目の列) である。j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、$u(j) = v1(:, j) + i*v1(:, j+1)$、$u(j+1) = v1(:, j) - i*v1(:, j+1)$ である。ここで、$i = \sqrt{-1}$ である。 <i>複素数型の場合:</i> $u(j) = v1(:, j)$ (v1 の j 番目の列) である。 $vr(ldvr, *)$。vr の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>jobvr='V' の場合</i>、右汎用固有ベクトル $v(j)$ は、それらの固有値と同じ順序で、vr の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$ となるようにスケールされる。</p>

	<i>jobvr</i> = 'N' の場合、 <i>vr</i> は参照されない。 実数型の場合： <i>j</i> 番目の固有値が実数の場合、 $v(j) = vr(:, j)$ (<i>vr</i> の <i>j</i> 番目の列) である。 <i>j</i> 番目と (<i>j</i> +1) 番目の固有値が複素共役ペアとなる場合は、 $v(j) = vr(:, j) + i*vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i*vr(:, j+1)$ である。 複素数型の場合： $v(j) = vr(:, j)$ (<i>vr</i> の <i>j</i> 番目の列) である。
<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> 、かつ $i \leq n$ ： <i>QZ</i> の繰り返しが失敗した。固有ベクトルは計算されなかったが、 <i>alphar(j)</i> 、 <i>alphai(j)</i> (実数型の場合)、または <i>alpha(j)</i> (複素数型の場合)、および <i>beta(j)</i> 、 $j = info+1, \dots, n$ は正しい。 $i > n$ の場合：一般に LAPACK の問題を示すエラー。 $i = n+1$ の場合： ?hgeqz で <i>QZ</i> の繰り返し以外の操作に失敗した。 $i = n+2$ の場合： ?tqevc からエラーが返された。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ggev* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>B</i> を格納する。
<i>alphar</i>	長さ (<i>n</i>) のベクトルを格納する。実数型でのみ使用される。
<i>alphai</i>	長さ (<i>n</i>) のベクトルを格納する。実数型でのみ使用される。
<i>alpha</i>	長さ (<i>n</i>) のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>v1</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>VR</i> を格納する。
<i>jobv1</i>	引数 <i>v1</i> の存在に基づいて以下のように復元される。 <i>jobv1</i> = 'V' (<i>v1</i> が存在する場合)、 <i>jobv1</i> = 'N' (<i>v1</i> が省略された場合)。
<i>jobvr</i>	引数 <i>vr</i> の存在に基づいて以下のように復元される。 <i>jobvr</i> = 'V' (<i>vr</i> が存在する場合)、 <i>jobvr</i> = 'N' (<i>vr</i> が省略された場合)。

アプリケーション・ノート

配列 `work` に必要なワークスペースの大きさがわからない場合は、最初の実行で `lwork` を大きめの値に設定する。終了時に、`work(1)` の値を確認し、これ以降の実行にはその値を使用する。

一般に、商 `alphan(j)/beta(j)` と `alphai(j)/beta(j)` は、簡単にオーバーフローまたはアンダーフローし、`beta(j)` がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、`alphan` と `alphai` (実数型の場合) または `alpha` (複素数型の場合) は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、`beta` は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

?ggev

汎用固有値と (オプションで) 左/右の汎用固有ベクトルを計算する。

構文

Fortran 77:

```
call sggev(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphan,
           alphai, beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm,
           bbnrm, rconde, rcondv, work, lwork, iwork, bwork, info)
call dggev(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alphan,
           alphai, beta, vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm,
           bbnrm, rconde, rcondv, work, lwork, iwork, bwork, info)
call cggev(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha, beta,
           vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde,
           rcondv, work, lwork, rwork, iwork, bwork, info)
call zggev(balanc, jobvl, jobvr, sense, n, a, lda, b, ldb, alpha, beta,
           vl, ldvl, vr, ldvr, ilo, ihi, lscale, rscale, abnrm, bbnrm, rconde,
           rcondv, work, lwork, rwork, iwork, bwork, info)
```

Fortran 95:

```
call ggev(a, b, alphan, alphai, beta [,vl] [,vr] [,balanc] [,ilo] [,ihi]
         [,lscale] [,rscale] [,abnrm] [,bbnrm] [,rconde] [,rcondv] [,info])
call ggev(a, b, alpha, beta [,vl] [,vr] [,balanc] [,ilo] [,ihi]
         [,lscale] [,rscale] [,abnrm] [,bbnrm] [,rconde] [,rcondv] [,info])
```

説明

このルーチンは、 $n \times n$ の実/複素非対称行列 (A, B) のペアについて、汎用固有値と (オプションで) 左/右の汎用固有ベクトルを計算する。

また、このルーチンは、平衡化のための変換を計算し、固有値と固有ベクトル (`ilo`、`ihi`、`lscale`、`rscale`、`abnrm`、`bbnrm`)、固有値に対する条件数の逆数 (`rconde`)、右固有ベクトルに対する条件数の逆数 (`rcondv`) の条件付けを改善できる。

行列 (A, B) のペアの汎用固有値は、 $A - \lambda * B$ が特異であるようなスカラー λ または比 $\alpha / \beta = \lambda$ である。 $\beta=0$ および両方がゼロである妥当な解釈が存在するため、通常は、ペア (α, β) として表現される。

(A, B) の汎用固有値 $\lambda(j)$ に対応する右汎用固有ベクトル $v(j)$ は、以下を満たす。

$$A * v(j) = \lambda(j) * B * v(j)$$

(A, B) の汎用固有値 $\lambda(j)$ に対応する左汎用固有ベクトル $u(j)$ は、以下を満たす。

$$u(j)^H * A = \lambda(j) * u(j)^H * B$$

$u(j)^H$ は、 $u(j)$ の共役転置を示す。

入力パラメーター

<i>balanc</i>	CHARACTER*1。 'N'、 'P'、 'S'、 または 'B' でなければならない。 実行する平衡化オプションを指定する。 <i>balanc</i> = 'N' の場合、対角的なスケーリングまたは置換は実行されない。 <i>balanc</i> = 'P' の場合、置換のみが実行される。 <i>balanc</i> = 'S' の場合、スケーリングのみが実行される。 <i>balanc</i> = 'B' の場合、置換とスケーリングがともに実行される。 計算された条件数の逆数は、平衡化または置換を実行後の行列に使用される。置換を実行しても条件数は変更されないが (正確な演算では)、平衡化を実行すると条件数が変更される。
<i>jobvl</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvl</i> = 'N' の場合、左汎用固有ベクトルは計算されない。 <i>jobvl</i> = 'V' の場合、左汎用固有ベクトルは計算される。
<i>jobvr</i>	CHARACTER*1。 'N' または 'V' でなければならない。 <i>jobvr</i> = 'N' の場合、右汎用固有ベクトルは計算されない。 <i>jobvr</i> = 'V' の場合、右汎用固有ベクトルは計算される。
<i>sense</i>	CHARACTER*1。 'N'、 'E'、 'V'、 または 'B' でなければならない。 どの条件数の逆数を計算するのかを決定する。 <i>sense</i> = 'N' の場合、計算は実行されない。 <i>sense</i> = 'E' の場合、固有値についてのみ計算が実行される。 <i>sense</i> = 'V' の場合、固有ベクトルについてのみ計算が実行される。 <i>sense</i> = 'B' の場合、固有値と固有ベクトルについて計算が実行される。
<i>n</i>	INTEGER。 行列 A 、 B 、 $v1$ 、 vr の次数 ($n \geq 0$)。
<i>a</i> , <i>b</i> , <i>work</i>	REAL (<i>sggev</i> の場合) DOUBLE PRECISION (<i>dggev</i> の場合) COMPLEX (<i>cggev</i> の場合) DOUBLE COMPLEX (<i>zggev</i> の場合)。 配列: $a(lda, *)$ は、 $n \times n$ の行列 A (行列のペアの 1 番目) を格納する配列である。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$b(ldb,*)$ は、 $n \times n$ の行列 B (行列のペアの 2 番目) を格納する配列である。
 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $work(lwork)$ は、ワークスペース配列である。

<i>lda</i>	INTEGER。配列 a の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 b の第 1 次元。 $\max(1, n)$ 以上でなければならない。
<i>ldv1, ldvr</i>	INTEGER。それぞれ、出力行列 $v1$ と vr の第 1 次元。 次の制約がある。 $ldv1 \geq 1$ 。 $jobv1 = 'V'$ の場合、 $ldv1 \geq \max(1, n)$ 。 $ldvr \geq 1$ 。 $jobvr = 'V'$ の場合、 $ldvr \geq \max(1, n)$ 。
<i>lwork</i>	INTEGER。配列 $work$ の次元。 実数型の場合： $lwork \geq \max(1, 6n)$ 。 $sense = 'E'$ の場合、 $lwork \geq 12n$ 。 $sense = 'V'$ または $'B'$ の場合、 $lwork \geq 2n^2 + 12n + 16$ 。 複素数型の場合： $lwork \geq \max(1, 2n)$ 。 $sense = 'N'$ または $'E'$ の場合、 $lwork \geq 2n$ 。 $sense = 'V'$ または $'B'$ の場合、 $lwork \geq 2n^2 + 2n$ 。 $lwork = -1$ の場合はワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最適サイズだけを計算し、その値を $work$ 配列の最初のエントリーとして返し、 $xerbla$ は $lwork$ に関するエラーメッセージを生成しない。
<i>rwork</i>	REAL (cggevx の場合) DOUBLE PRECISION (zggevx の場合) ワークスペース配列、次元は $\max(1, 6n)$ 以上。 この配列は、複素数型でのみ使用される。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(n+6)$ 以上 (実数型の場合) または $(n+2)$ 以上 (複素数型の場合)。 $sense = 'E'$ の場合は参照されない。
<i>bwork</i>	LOGICAL。 ワークスペース配列、次元は $\max(1, n)$ 以上。 $sense = 'N'$ の場合は参照されない。

出力パラメーター

<i>a, b</i>	終了時に上書きされる。 $jobv1 = 'V'$ または $jobvr = 'V'$ (またはその両方) の場合、 a には、平衡化した入力 A と B の実数 Schur 形式の 1 番目の部分が格納され、 b には、その 2 番目の部分が格納される。
-------------	---

<i>alphar, alphai</i>	<p>REAL (sggevx の場合) DOUBLE PRECISION (dggevx の場合)。 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型の汎用固有値を形成する値が格納される。 <i>beta</i> を参照。</p>
<i>alpha</i>	<p>COMPLEX (cggevx の場合) DOUBLE COMPLEX (zggevx の場合)。 配列、次元は $\max(1, n)$ 以上。複素数型の汎用固有値を形成する値が格納される。<i>beta</i> を参照。</p>
<i>beta</i>	<p>REAL (sggevx の場合) DOUBLE PRECISION (dggevx の場合) COMPLEX (cggevx の場合) DOUBLE COMPLEX (zggevx の場合)。 配列、次元は $\max(1, n)$ 以上。 <i>実数型の場合:</i> 終了時に、$(\text{alphar}(j) + \text{alphai}(j)*i)/\text{beta}(j)$、$j=1, \dots, n$ は、汎用固有値になる。 <i>alphai(j)</i> がゼロの場合、j 番目の固有値が実数であり、正の場合、j 番目と $(j+1)$ 番目の固有値が複素共役ペアである (<i>alphai(j+1)</i> は負)。 <i>複素数型の場合:</i> 終了時に、$\text{alpha}(j)/\text{beta}(j)$、$j=1, \dots, n$ は、汎用固有値になる。 次の「アプリケーション・ノート」も参照のこと。</p>
<i>v1, vr</i>	<p>REAL (sggevx の場合) DOUBLE PRECISION (dggevx の場合) COMPLEX (cggevx の場合) DOUBLE COMPLEX (zggevx の場合)。 配列: $v1(ldv1, *)$。<i>v1</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>jobv1='V'</i> の場合、左汎用固有ベクトル $u(j)$ は、それらの固有値と同じ順序で、<i>v1</i> の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$ となるようにスケールされる。<i>jobv1='N'</i> の場合、<i>v1</i> は参照されない。 <i>実数型の場合:</i> j 番目の固有値が実数の場合、$u(j) = v1(:, j)$ (<i>v1</i> の j 番目の列) である。j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合、$u(j) = v1(:, j) + i*v1(:, j+1)$、$u(j+1) = v1(:, j) - i*v1(:, j+1)$ である。ここで、$i = \sqrt{-1}$ である。 <i>複素数型の場合:</i> $u(j) = v1(:, j)$ (<i>v1</i> の j 番目の列) である。 $vr(ldvr, *)$。<i>vr</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>jobvr='V'</i> の場合、右汎用固有ベクトル $v(j)$ は、それらの固有値と同じ順序で、<i>vr</i> の列に次々と格納される。各固有ベクトルは、最大のコンポーネントが $\text{abs}(\text{Re}) + \text{abs}(\text{Im}) = 1$ となるようにスケールされる。<i>jobvr='N'</i> の場合、<i>vr</i> は参照されない。</p>

実数型の場合:

j 番目の固有値が実数の場合、 $v(j) = vr(:, j)$ (vr の j 番目の列) である。 j 番目と $(j+1)$ 番目の固有値が複素共役ペアとなる場合は、 $v(j) = vr(:, j) + i*vr(:, j+1)$ 、 $v(j+1) = vr(:, j) - i*vr(:, j+1)$ である。

複素数型の場合:

$v(j) = vr(:, j)$ (vr の j 番目の列) である。

ilo, ihi

INTEGER。

ilo と *ihi* は、終了時に、 $A(i, j) = 0$ および $B(i, j) = 0$ ($i > j$ かつ $j = 1, \dots, ilo-1$ または $i = ihi+1, \dots, n$ の場合) であるような整数値である。

balanc = 'N' または 'S' の場合、*ilo* = 1 と *ihi* = n である。

lscale, rscale

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元はそれぞれ $\max(1, n)$ 以上。

lscale には、 A と B の左側に適用される置換およびスケーリング係数の各成分が格納される。

$PL(j)$ は行 j と交換される行のインデックスであり、 $DL(j)$ は行 j に適用されるスケーリング係数である。したがって、

$$\begin{aligned} lscale(j) &= PL(j), \quad j = 1, \dots, ilo-1 \\ &= DL(j), \quad j = ilo, \dots, ihi \\ &= PL(j), \quad j = ihi+1, \dots, n \end{aligned}$$

交換を実行する順序は、 $n \sim ihi+1$ 、次に $1 \sim ilo-1$ である。

rscale には、 A と B の右側に適用される置換およびスケーリング係数の各成分が格納される。

$PR(j)$ は列 j と交換される列のインデックスであり、 $DR(j)$ は列 j に適用されるスケーリング係数である。したがって、

$$\begin{aligned} rscale(j) &= PR(j), \quad j = 1, \dots, ilo-1 \\ &= DR(j), \quad j = ilo, \dots, ihi \\ &= PR(j), \quad j = ihi+1, \dots, n \end{aligned}$$

交換を実行する順序は、 $n \sim ihi+1$ 、次に $1 \sim ilo-1$ である。

abnrm, bbnrm

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

それぞれ、平衡化した行列 A と B の 1- ノルムである。

rconde, rcondv

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)。

配列、次元はそれぞれ $\max(1, n)$ 以上。

sense = 'E' または 'B' の場合、*rconde* には、指定された固有値の条件数の逆数を格納する。これらは、配列の連続する成分に格納される。固有値の複素共役ペアの場合、*rconde* の 2 つの連続する成分には同じ値が設定される。したがって、*rconde(j)*、*rcondv(j)*、 $v1$ と vr の j 番目の列はすべて、同じ固有ペアに対応している (ただし、選択しなかった固有ペアが存在する場合は、一般に、 j 番目の固有ペアにはならない)。

sense = 'V' の場合、*rconde* は参照されない。

sense = 'V' または 'B' の場合、*rcondv* には、指定された固有ベクトルについての見積りの条件数の逆数が格納される。これらは、配列の連続する成分に格納される。複素数の固有ベクトルの場合、*rcondv* の 2 つの連続する成分には同じ値が設定される。*rcondv(j)* を計算するために固有値の順序を変更できない場合は、*rcondv(j)* にゼロが設定される。これは、真の値が非常に小さい場合にだけ発生する可能性がある。

sense = 'E' の場合、*rcondv* は参照されない。

<i>work(1)</i>	終了時に、 <i>info</i> = 0 の場合、 <i>work(1)</i> は <i>lwork</i> の必要最小サイズを返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> 、かつ $i \leq n$: <i>QZ</i> の繰り返し失敗した。固有ベクトルは計算されなかったが、 <i>alphar(j)</i> 、 <i>alphai(j)</i> (実数型の場合)、または <i>alpha(j)</i> (複素数型の場合)、および <i>beta(j)</i> 、 $j = info + 1, \dots, n$ は正しい。 <i>i</i> > <i>n</i> の場合：一般に LAPACK の問題を示すエラー。 <i>i</i> = <i>n</i> + 1 の場合： ?hgeqz で <i>QZ</i> の繰り返し以外の操作に失敗した。 <i>i</i> = <i>n</i> + 2 の場合： ?tgevc からエラーが返された。

Fortran 95 インターフェイス・ノート

Fortran 95 インターフェイスのルーチンは、Fortran 77 に比べて、呼び出しシーケンスの引数が少ない。冗長な引数または復元可能な引数をスキップする一般的な規則に関する詳細は、「[Fortran-95 インターフェイス規則](#)」を参照のこと。

ルーチン *ggev* のインターフェイスの詳細を以下に示す。

<i>a</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>A</i> を格納する。
<i>b</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>B</i> を格納する。
<i>alphar</i>	長さ (<i>n</i>) のベクトルを格納する。実数型でのみ使用される。
<i>alphai</i>	長さ (<i>n</i>) のベクトルを格納する。実数型でのみ使用される。
<i>alpha</i>	長さ (<i>n</i>) のベクトルを格納する。複素数型でのみ使用される。
<i>beta</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>vl</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>VL</i> を格納する。
<i>vr</i>	サイズ (<i>n</i> , <i>n</i>) の行列 <i>VR</i> を格納する。
<i>lscale</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>rscale</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>rconde</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>rcondv</i>	長さ (<i>n</i>) のベクトルを格納する。
<i>balanc</i>	'N'、'B'、または 'P' でなければならない。デフォルト値は 'N'。

<i>jobvl</i>	引数 <i>v1</i> の存在に基づいて以下のように復元される。 <i>jobvl</i> = 'V' (<i>v1</i> が存在する場合)、 <i>jobvl</i> = 'N' (<i>v1</i> が省略された場合)。
<i>jobvr</i>	引数 <i>vr</i> の存在に基づいて以下のように復元される。 <i>jobvr</i> = 'V' (<i>vr</i> が存在する場合)、 <i>jobvr</i> = 'N' (<i>vr</i> が省略された場合)。
<i>sense</i>	引数 <i>rconde</i> と <i>rcondv</i> の存在に基づいて以下のように復元される。 <i>sense</i> = 'B' (<i>rconde</i> と <i>rcondv</i> の両方が存在する場合)、 <i>sense</i> = 'E' (<i>rconde</i> が存在し、 <i>rcondv</i> が省略された場合)、 <i>sense</i> = 'V' (<i>rconde</i> が省略され、 <i>rcondv</i> が存在する場合)、 <i>sense</i> = 'N' (<i>rconde</i> と <i>rcondv</i> の両方が省略された場合)。

アプリケーション・ノート

配列 *work* に必要なワークスペースの大きさがわからない場合は、最初の実行で *lwork* を大きめの値に設定する。終了時に、*work(1)* の値を確認し、これ以降の実行にはその値を使用する。

一般に、商 $\text{alphan}(j)/\text{betan}(j)$ と $\text{alphai}(j)/\text{betai}(j)$ は、簡単にオーバーフローまたはアンダーフローし、 $\text{betan}(j)$ がゼロになる場合もある。したがって、単純に比を計算するのは避けなければならない。ただし、 alphan と alphai (実数型の場合) または alpha (複素数型の場合) は、大きさがノルム (A) より常に小さくなり、通常はノルム (A) と比較できる。また、 beta は、ノルム (B) より常に小さくなり、通常はノルム (B) と比較できる。

LAPACK 補助ルーチンと ユーティリティー・ルーチン

5

この章では LAPACK [補助ルーチン](#)と[ユーティリティー・ルーチン](#)に関するインテル® マス・カーネル・ライブラリーの実装について説明する。ライブラリーは実数と複素の両方のデータに対応した補助ルーチンで構成される。

補助ルーチン

LAPACK 補助ルーチンで使用されているルーチン名の規則、数学表記、行列の格納形式は、これまでの章に記載されているドライバーおよび計算ルーチンのものと同一である。

次の表は、利用可能な LAPACK 補助ルーチンについての情報をまとめたものである。

表 5-1 LAPACK 補助ルーチン

ルーチン名	データ型	説明
?lacgv	c, z	複素ベクトルを共役する。
?lacrm	c, z	複素行列に正方実数行列を乗算する。
?lacrt	c, z	複素ベクトル対の線形変換を実行する。
?laesy	c, z	2 × 2 の複素数対称行列の固有値と固有ベクトルを計算する。
?rot	c, z	複素ベクトル対に実余弦と複素正弦を用いて面回転を適用する。
?spsmv	c, z	複素対称圧縮行列を使用して複素ベクトルに対する行列 - ベクトル積を計算する。
?spr	c, z	対称圧縮行列の複素数対称階数 1 の更新を実行する。
?symv	c, z	複素対称行列に対して行列 - ベクトル積を計算する。
?syr	c, z	複素対称行列の対称階数 1 の更新を実行する。
i?max1	c, z	実数部分が最大絶対値を持つベクトルの成分のインデックスを検出する。
?sum1	sc, dz	真の絶対値を用いて複素ベクトルの 1- ノルムを実行する。
?gbtf2	s, d, c, z	非ブロック化バージョンのアルゴリズムを使って一般帯行列の LU 因子分解を計算する。
?gebd2	s, d, c, z	非ブロック化アルゴリズムを使って一般行列を二重対角形式に縮退させる。
?gehd2	s, d, c, z	非ブロック化アルゴリズムを使用して一般正方行列を上 Hessenberg 形式に縮退させる。
?gelq2	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の LQ 因子分解を計算する。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?geql2	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の QL 因子分解を計算する。
?geqr2	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の QR 因子分解を計算する。
?gerq2	s, d, c, z	非ブロック化アルゴリズムを使用して一般矩形行列の RQ 因子分解を計算する。
?gesc2	s, d, c, z	?getc2 で計算された完全ピボット演算による LU 因子分解を使用して連立線形方程式を解く。
?getc2	s, d, c, z	一般 $n \times n$ 行列の完全ピボット演算による LU 因子分解を計算する。
?getf2	s, d, c, z	一般 $m \times n$ 行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する (非ブロック化アルゴリズム)。
?gtts2	s, d, c, z	?gttrf によって行われた LU 因子分解を使用して、三重対角行列を係数行列とする連立線形方程式を解く。
?labrd	s, d, c, z	一般行列の最初の nb 行と列を二重対角形式に縮退させる。
?lacon	s, d, c, z	行列 - ベクトル積の評価にリバース・コミュニケーションを使用して正方行列の 1- ノルムを推定する。
?lacpy	s, d, c, z	1 つの 2 次元配列の一部または全部を他にコピーする。
?ladiv	s, d, c, z	不必要なオーバーフローを回避して、実数演算で複素除算を実行する。
?lae2	s, d	2×2 対称行列の固有値を計算する。
?laebz	s, d	実数対称三重対角行列の指定された値以下の固有値の個数を計算し、また ?stebz ルーチンが必要とする他の処理を実行する。
?laed0	s, d, c, z	?stedc で使用される。縮退されていない対称三重対角行列の固有値と対応する固有ベクトルを分割統治法を使用してすべて求める。
?laed1	s, d	sstedc/dstedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が三重対角の場合に使用される。
?laed2	s, d	sstedc/dstedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が三重対角の場合に使用される。
?laed3	s, d	sstedc/dstedc で使用される。永年方程式の根を探し固有ベクトルを更新する。元の行列が三重対角の場合に使用される。
?laed4	s, d	sstedc/dstedc で使用される。永年方程式の単一根を見つける。
?laed5	s, d	sstedc/dstedc で使用される。 2×2 の永年方程式を解く。
?laed6	s, d	sstedc/dstedc で使用される。永久方程式の解の中から Newton ステップの 1 つを計算する。
?laed7	s, d, c, z	?stedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が密の場合に使用される。
?laed8	s, d, c, z	?stedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が密の場合に使用される。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?laed9	s, d	sstedc/dstedc で使用される。 永年方程式の根を探し固有ベクトルを更新する。元の行列が密の場合に使用される。
?laeda	s, d	?stedc で使用される。対角行列の階数 1 更新を決定する Z ベクトルを計算する。元の行列が密の場合に使用される。
?laein	s, d, c, z	上 Hessenberg 行列の指定された右または左固有ベクトルを逆反復法を用いて計算する。
?laev2	s, d, c, z	2 × 2 の対称 / エルミート行列の固有値と固有ベクトルを計算する。
?laexc	s, d	Schur 標準形となっている実数の上準三角行列の隣接対角ブロックを、直交相似変換によって交換する。
?lag2	s, d	2 × 2 一般固有値問題の固有値を、オーバーフロー / アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。
?lags2	s, d	2 × 2 直交行列 U 、 V 、 Q を計算し、変換された A と B の行が平行であるような A と B にそれらを適用する。
?lagtf	s, d	行交換を伴う部分ピボット演算を用いて行列 $T-\lambda I$ の LU 因子分解を計算する。 T は一般三重対角行列、 λ はスカラー。
?lagtm	s, d, c, z	$C = \alpha AB + \beta C$ の形式で示される行列 = 行列積を実行し、ここで A は三重対角行列、 B と C は矩形行列、 α と β はスカラーで 0、1、または -1 である。
?lagts	s, d	?lagtf で計算された ΛY 因子分解を用いて連立方程式 $(T-\lambda I)x = y$ または $(T-\lambda I)^T x = y$ を解く。ここで T は一般三重対角行列、 λ はスカラー。
?lagv2	s, d	実数の 2 × 2 行列束 (A, B) の汎用 Schur 因子分解を計算する。ここで B は上三角である。
?lahqr	s, d, c, z	ダブルシフト / シングルシフト QR アルゴリズムを用いて、上 Hessenberg 行列の固有値と Schur 因子分解を計算する。
?lahrd	s, d, c, z	k 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の nb 列を縮退させ、 A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。
?laicl	s, d, c, z	増加条件推定を 1 ステップ適用する。
?laln2	s, d	指定された形式の 1 × 1 または 2 × 2 の連立線形方程式を解く。
?lals0	s, d, c, z	最小二乗問題を分割統治 SVD 法を使用して解く過程で乗率を逆に適用する。?gelsd で使用される。
?lalsa	s, d, c, z	コンパクト形式で係数行列の SVD を計算する。 ?gelsd で使用される。
?lalsd	s, d, c, z	最小二乗問題を解くために A の特異値分解を使用する。
?lamrc	s, d	2 つの独立するソートされたセットの成分を単一のソートされたセットに昇順で併合する置換リストを生成する。
?langb	s, d, c, z	一般帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lange	s, d, c, z	一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?langt	s, d, c, z	一般三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?lanhs	s, d, c, z	上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lansb	s, d, c, z	対称帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lanhb	c, z	エルミート帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lansp	s, d, c, z	圧縮形式で与えられる対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lanhp	c, z	圧縮形式で与えられる複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lanst/?lanht	s, d/c, z	実数対称または複素エルミート三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lansy	s, d, c, z	実数 / 複素対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lanhe	c, z	複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lantb	s, d, c, z	三角帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lantp	s, d, c, z	圧縮形式で与えられる三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lantr	s, d, c, z	台形または三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
?lanv2	s, d	2 × 2 実数非対称行列の Schur 因子分解を標準形式で計算する。
?lapll	s, d, c, z	2 つのベクトルの線形従属性を測定する。
?lapmt	s, d, c, z	行列の列に対して順方向または逆方向置換を実行する。
?lapy2	s, d	$\sqrt{x^2+y^2}$ を返す。
?lapy3	s, d	$\sqrt{x^2+y^2+z^2}$ を返す。
?laqgb	s, d, c, z	?gbequ で計算された行と列のスケール係数を使って一般帯行列をスケールリングする。
?laqge	s, d, c, z	?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。
?laqp2	s, d, c, z	行列ブロックに対して列ピボット演算を用いた QR 因子分解を計算する。
?laqps	s, d, c, z	BLAS レベル 3 を使って実数 $m \times n$ 行列 A に対して列ピボット演算を用いた QR 因子分解のステップを計算する。
?laqsb	s, d, c, z	?pbequ で計算されたスケール係数を用いて対称/エルミート帯行列をスケールリングする。
?laqsp	s, d, c, z	?ppequ で計算されたスケール係数を用いて圧縮格納形式にある対称/エルミート行列をスケールリングする。
?laqsy	s, d, c, z	?poequ で計算されたスケール係数を用いて対称/エルミート行列をスケールリングする。
?laqtr	s, d	実数準三角連立方程式または特殊な形式の複素準三角連立方程式を実数演算で解く。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?larlv	s, d, c, z	三重対角行列 $LDL^T - \sigma I$ の行 b_1 から b_n にある部分行列に対して逆行列の (スケールされた) r 番目の列を計算する。
?lar2v	s, d, c, z	一連の 2×2 対称 / エルミート行列に、実数余弦と実数 / 複素正弦を持つ面回転のベクトルを両側から適用する。
?larf	s, d, c, z	一般矩形行列に基本リフレクターを適用する。
?larfb	s, d, c, z	ブロック・リフレクターまたはその転置 / 共役転置を一般矩形行列に適用する。
?larfg	s, d, c, z	基本リフレクター (Householder 行列) を生成する
?larft	s, d, c, z	ブロック・リフレクター $H = I - VTV^H$ の三角係数 T を生成する。
?larfx	s, d, c, z	リフレクターが次数 ≤ 10 を持つ場合、ループの繰り返しを避けながら、基本リフレクターを一般矩形行列に適用する。
?larqv	s, d, c, z	実数余弦および実数 / 複素正弦を持つ面回転のベクトルを生成する。
?larnv	s, d, c, z	乱数ベクトルを一様分布または正規分布で返す。
?larrb	s, d	より高い精度で固有値を見つけるための限定二分法を与える。
?larre	s, d	三重対角行列 T が与えられたとき、小さい非対角成分をゼロに設定し、縮退されていないブロック T_i に対して本表現と固有値を探す。
?larrf	s, d	少なくとも 1 つの固有値が相対的に分離されているような新しい比較的安定な表現を探す。
?larrv	s, d, c, z	L 、 D 、 LDL^T の固有値が与えられたとき、三重対角行列 $T = LDL^T$ の固有ベクトルを計算する。
?lartq	s, d, c, z	実数余弦と実数 / 複素正弦を持つ面回転を生成する。
?lartv	s, d, c, z	実数余弦と実数 / 複素制限を持つ面回転のベクトルをベクトル対の成分に適用する。
?laruv	s, d	n 個の実数乱数のベクトルを一様分布で返す。
?larz	s, d, c, z	(?tzzrf が返したとおりの) 基本リフレクターを一般行列に適用する。
?larzb	s, d, c, z	ブロック・リフレクターまたはその転置 / 共役転置を一般矩形行列に適用する。
?larzt	s, d, c, z	ブロック・リフレクター $H = I - VTV^H$ の三角係数 T を生成する。
?las2	s, d	2×2 三角行列の特異値を計算する。
?lascl	s, d, c, z	一般矩形行列に C_{to}/C_{from} として定義される実数スカラーを乗算する。
?lasd0	s, d	対角 d と非対角 e を持つ実数上二重対角 $n \times m$ 行列 B の特異値を計算する。?bdsdc で使用される。
?lasd1	s, d	指定サイズの上二重対角行列 B の SVD を計算する。?bdsdc で使用される。
?lasd2	s, d	2 つの特異値のセットをソートされた単一のセットに併合する。?bdsdc で使用される。
?lasd3	s, d	D と Z 内の値によって定義されている永年方程式に対して根のすべての平方根を求め、次に行列乗算によって特異ベクトルを更新する。?bdsdc で使用される。
?lasd4	s, d	正値対角行列に対する正値対称階数 1 更新の、 i 番目の更新された固有値の平方根を計算する。?bdsdc で使用される。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?lasd5	s, d	2 × 2 対角行列の正値対称階数 1 更新から、i 番目の固有値の平方根を計算する。?bdsdc で使用される。
?lasd6	s, d	2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列の SVD を計算する。?bdsdc で使用される。
?lasd7	s, d	2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。?bdsdc で使用される。
?lasd8	s, d	永年方程式の根の平方根を求め、D の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。
?lasd9	s, d	永年方程式の根の平方根を求め、D の各成分に対して最も近い 2 つの極までの距離を格納する。?bdsdc で使用される。
?lasda	s, d	対角 d と非対角 e を持つ実数上二重対角行列の特異値分解 (SVD) を計算する。?bdsdc で使用される。
?lasdq	s, d	対角 d と非対角 e を持つ実二重対角行列を計算する。?bdsdc で使用される。
?lasdt	s, d	二重対角分割統治に対する部分問題のツリーを生成する。?bdsdc で使用される。
?laset	s, d, c, z	行列の非対角成分と対角成分を 指定された値 に初期化する。
?lasq1	s, d	実数平方二重対角行列の特異値を計算する。?bdsqr で使用される。
?lasq2	s, d	相対的に高い精度で、 qd 配列 z に関する対称正定値三重対角行列のすべての固有値を計算する。?bdsqr と ?stegr で使用される。
?lasq3	s, d	収縮をチェックし、シフトを計算し、 $dqds$ を呼び出す。?bdsqr で使用される。
?lasq4	s, d	以前の変換で得られた d の値を用いて最小固有値に対する近似を計算する。?bdsqr で使用される。
?lasq5	s, d	ping-pong 形式で $dqds$ 変換を 1 つ計算する。?bdsqr と ?stegr で使用される。
?lasq6	s, d	ping-pong 形式で dqd 変換を 1 つ計算する。?bdsqr と ?stegr で使用される。
?lasr	s, d, c, z	一般矩形行列に一連の面回転を適用する。
?lasrt	s, d	数を昇順または降順でソートする。
?lassq	s, d, c, z	スケーリング形式で表現された二乗和を更新する。
?lasv2	s, d	2 × 2 三角行列の特異値分解を計算する。
?laswp	s, d, c, z	一般矩形行列に対して一連の行交換を実行する。
?lasy2	s, d	行列の次数が 1 または 2 のシルベスター行列式を解く。
?lasyf	s, d, c, z	対角ピボット演算法を使って実数 / 複素対称行列の部分因子分解を計算する。
?lahef	c, z	対角ピボット演算法を使って複素エルミート無限行列の部分因子分解を計算する。
?latbs	s, d, c, z	三角帯連立方程式を解く。
?latdf	s, d, c, z	?getc2 による $n \times n$ 行列の LU 因子分解を使い、Dif 推定値の逆数に対する影響を計算する。
?latps	s, d, c, z	圧縮形式で格納されている行列を持った三角連立方程式を解く。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?lattrd	s, d, c, z	直交 / ユニタリー相似変換を用いて、対称 / エルミート行列 A の最初の nb 列と nb 行を実数三重対角形式に縮退する。
?latrs	s, d, c, z	オーバーフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。
?latrz	s, d, c, z	上台形行列を直交 / ユニタリー変換によって因子分解する。
?lauu2	s, d, c, z	積 UU^H または L^HL を計算する。ここで U と L は上三角または下三角行列 (非ブロック化アルゴリズム)。
?lauum	s, d, c, z	積 UU^H または L^HL を計算する。ここで U と L は上三角または下三角行列 (ブロック化アルゴリズム)。
?orgq2l/?ungq2l	s, d/c, z	?geqlf で求めた QL 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
?orgq2r/?ungq2r	s, d/c, z	?geqrf で求めた QR 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
?orgl2/?ungl2	s, d/c, z	?gelqf で求めた LQ 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
?orgqr2/?unqr2	s, d/c, z	?gerqf で求めた RQ 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
?orm2l/?unm2l	s, d/c, z	一般行列と ?geqlf で求めた QL 因子分解の直交 / ユニタリー行列を乗算する (非ブロック化アルゴリズム)。
?orm2r/?unm2r	s, d/c, z	一般行列と ?geqrf で求めた QR 因子分解の直交 / ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。
?orml2/?unml2	s, d/c, z	一般行列と ?gelqf で求めた LQ 因子分解の直交 / ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。
?ormr2/?unmr2	s, d/c, z	一般行列と ?gerqf で求めた RQ 因子分解の直交 / ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。
?ormr3/?unmr3	s, d/c, z	一般行列と ?tzzrf で求めた RZ 因子分解の直交 / ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。
?pbt2	s, d, c, z	対称 / エルミート正定値帯行列のコレスキー因子分解を計算する (非ブロック化アルゴリズム)。
?potf2	s, d, c, z	対称 / エルミート正定値行列のコレスキー因子分解を行う (非ブロック化アルゴリズム)。
?ptts2	s, d, c, z	?pttrf で計算した L 、 D 、 L^H 因子分解を用いて、形式 $AX = B$ の三重対角連立方程式を解く。
?rscl	s, d, cs, zd	ベクトルに実スカラーの逆数を掛ける。
?svgs2/?hegs2	s, d/c, z	?potrf で得られた因子分解の結果を用いて、対称 / エルミート汎用固有値問題を標準形式に縮退させる (非ブロック化アルゴリズム)。
?sytd2/?hetd2	s, d/c, z	直交 / ユニタリー相似変換を用いて、対称 / エルミート行列を実数対称三重対角形式に縮退させる (非ブロック化アルゴリズム)。
?sytf2	s, d, c, z	対角ピボット演算法を使用して、実数 / 複素不定値対称行列の因子分解を計算する (非ブロック化アルゴリズム)。
?hetf2	c, z	対角ピボット演算法を使用して、複素エルミート行列の因子分解を計算する (非ブロック化アルゴリズム)。
?tgex2	s, d, c, z	直交 / ユニタリー等価変換を使用して、(準)上三角行列のペア中の隣接対角ブロックを交換する。

表 5-1 LAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?tqsy2	s, d, c, z	汎用シルベスター式を解く (非ブロック化アルゴリズム)。
?trti2	s, d, c, z	三角行列の逆行列を計算する (非ブロック化アルゴリズム)。

?lacgv

複素ベクトルを共役する。

構文

```
call clacgv( n, x, incx )
```

```
call zlacgv( n, x, incx )
```

説明

このルーチンは、長さ n 、増分 $incx$ の複素ベクトル x を共役する (付録 B の「[BLAS のベクトル引数](#)」を参照)。

入力パラメーター

n INTEGER。ベクトル x の長さ ($n \geq 0$)。

x COMPLEX (clacgv の場合)
 COMPLEX*16 (zlacgv の場合)
 配列、次元は $(1+(n-1)*|incx|)$ 。
 共役する長さ n のベクトルを格納する。

$incx$ INTEGER。 x の連続する成分の間隔。

出力パラメーター

x conjg(x) で上書きされる。

?lacrm

複素行列に正方実数行列を乗算する。

構文

```
call clacrm( m, n, a, lda, b, ldb, c, ldc, rwork )
```

```
call zlacrm( m, n, a, lda, b, ldb, c, ldc, rwork )
```

説明

このルーチンは、次の形式の単純な行列 - 行列乗算を実行する。

$$C = A * B$$

ここで、 A は $m \times n$ の複素行列、 B は $n \times n$ の実行列、 C は $m \times n$ の複素行列である。

入力パラメーター

<i>m</i>	INTEGER。行列 <i>A</i> の行数と行列 <i>C</i> の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。行列 <i>B</i> の列数と行数および行列 <i>C</i> の列数 ($n \geq 0$)。
<i>a</i>	COMPLEX (clacrm の場合) COMPLEX*16 (zlacrm の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 $m \times n$ 行列 <i>A</i> が格納される。
<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。
<i>b</i>	REAL (clacrm の場合) DOUBLE PRECISION (zlacrm の場合) 配列、次元は (<i>ldb</i> , <i>n</i>)。 $n \times n$ 行列 <i>B</i> が格納される。
<i>ldb</i>	INTEGER。配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。
<i>ldc</i>	INTEGER。出力配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, n)$ 。
<i>rwork</i>	REAL (clacrm の場合) DOUBLE PRECISION (zlacrm の場合) ワークスペース配列、次元は ($2 * m * n$)。

出力パラメーター

<i>c</i>	COMPLEX (clacrm の場合) COMPLEX*16 (zlacrm の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 $m \times n$ 行列 <i>C</i> が格納される。
----------	---

?lacrt

複素ベクトル対の線形変換を実行する。

構文

```
call clacrt( n, cx, incx, cy, incy, c, s )
call zlacrt( n, cx, incx, cy, incy, c, s )
```

説明

このルーチンは次の変換を実行する。

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix},$$

C と *S* は複素スカラー、*x* と *y* は複素ベクトルである。

入力パラメーター

<i>n</i>	INTEGER。ベクトル <i>cx</i> とベクトル <i>cy</i> の成分の数 ($n \geq 0$)。
<i>cx</i> , <i>cy</i>	COMPLEX (<i>clacrt</i> の場合) COMPLEX*16 (<i>zlacrt</i> の場合) 配列、次元は (<i>n</i>)。 それぞれ入力ベクトル <i>x</i> と <i>y</i> を格納する。
<i>incx</i>	INTEGER。 <i>cx</i> の連続する成分間の増分。
<i>incy</i>	INTEGER。 <i>cy</i> の連続する成分間の増分。
<i>c</i> , <i>s</i>	COMPLEX (<i>clacrt</i> の場合) COMPLEX*16 (<i>zlacrt</i> の場合) 変換行列を定義する複素スカラー

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

出力パラメーター

<i>cx</i>	$c*x + s*y$ で上書きされる。
<i>cy</i>	$-s*x + c*y$ で上書きされる。

?laesy

2×2 の複素対称行列の固有値と固有ベクトルを計算し、固有ベクトルの行列のノルムがしきい値より大きいことを確認する。

構文

```
call claesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
call zlaesy (a, b, c, rt1, rt2, evscal, cs1, sn1)
```

説明

このルーチンは、固有ベクトルの行列のノルムがしきい値より大きければ、次の 2×2 対称行列の固有分解を実行する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

rt1 は絶対値が大きい方の固有値、*rt2* は絶対値が小さい方の固有値である。固有ベクトルが計算されると、(*cs1*, *sn1*) は *rt1* の単位固有ベクトルで上書きされ、ゆえに、

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

入力パラメーター

a, b, c COMPLEX (claesy の場合)
 COMPLEX*16 (zlaesy の場合)

入力行列の成分。

出力パラメーター

rt1, rt2 COMPLEX (claesy の場合)
 COMPLEX*16 (zlaesy の場合)

それぞれ、大きい方(モジュラス)の固有値、小さい方(モジュラス)の固有値。

evscal COMPLEX (claesy の場合)
 COMPLEX*16 (zlaesy の場合)

固有ベクトル行列を正規直交化するためにスケーリングした複素値。*evscal* がゼロの場合、固有ベクトルは計算されていない。これには2つの理由が考えられる。2×2行列を対角化できなかったか、スケーリング前の固有ベクトルの行列のノルムがしきい値 *thresh* (0.1E0 に設定) よりも大きかったからである。

cs1, sn1 COMPLEX (claesy の場合)
 COMPLEX*16 (zlaesy の場合)

evscal がゼロでない場合、(*cs1, sn1*) は *rt1* の単位右固有ベクトルである。

?rot

複素ベクトル対に実余弦と複素正弦を用いて面回転を適用する。

構文

```
call crot( n, cx, incx, cy, incy, c, s )
call zrot( n, cx, incx, cy, incy, c, s )
```

説明

このルーチンは面回転を適用する。ここで余弦(*c*)は実数、正弦(*s*)は複素数、ベクトル *cx* と *cy* は複素数である。このルーチンと実数において等価な関数が BLAS にある(第2章の「[?rot](#)」を参照)。

入力パラメーター

n INTEGER。ベクトル *cx* とベクトル *cy* の成分の数。

cx, cy COMPLEX (crot の場合)
 COMPLEX*16 (zrot の場合)
 次元 (*n*) の配列で、それぞれ入力ベクトル *x* と *y* を格納する。

incx INTEGER。 *cx* の連続する成分間の増分。

<i>incy</i>	INTEGER。 <i>cy</i> の連続する成分間の増分。
<i>c</i>	REAL (<i>crot</i> の場合) DOUBLE PRECISION (<i>zrot</i> の場合)
<i>s</i>	COMPLEX (<i>crot</i> の場合) COMPLEX*16 (<i>zrot</i> の場合)
	回転を定義する値で、
	$\begin{bmatrix} c & s \\ -\text{conjg}(s) & c \end{bmatrix}$
	$c*c + s*\text{conjg}(s) = 1.0$ 。

出力パラメーター

<i>cx</i>	$c*x + s*y$ で上書きされる。
<i>cy</i>	$-\text{conjg}(s)*x + c*y$ で上書きされる。

?spmv

複素対称圧縮行列を使用して複素ベクトルに対する行列-ベクトル積を計算する。

構文

```
call cspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
call zspmv( uplo, n, alpha, ap, x, incx, beta, y, incy )
```

説明

これらのルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \text{alpha} * a * x + \text{beta} * y,$$

ここで、

alpha と *beta* は、複素スカラーである。

x と *y* は *n* 個の成分を持つ複素ベクトルである。

a は、圧縮形式で与える $n \times n$ の対称行列である。

このルーチンと実数において等価な関数が BLAS にある (第 2 章の「[?spmv](#)」を参照)。

入力パラメーター

uplo CHARACTER*1。次に示すように、行列 *a* の上三角部分と下三角部分のどちらを圧縮形式の配列 *ap* において与えるかを指定する。

	<i>uplo</i> = 'U' または 'u' の場合、行列 <i>a</i> の上三角部分を配列 <i>ap</i> に格納する。 <i>uplo</i> = 'L' または 'l' の場合、行列 <i>a</i> の下三角部分を配列 <i>ap</i> に格納する。
<i>n</i>	INTEGER。行列 <i>a</i> の次数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha, beta</i>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合) 複素スカラー <i>alpha</i> と <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>ap</i>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合) 配列、次元は $((n*(n+1))/2)$ 以上。 <i>uplo</i> = 'U' または 'u' と指定した場合は、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (1, 2) が、 <i>ap</i> (3) に <i>a</i> (2, 2) がそれぞれ格納されるように、配列 <i>ap</i> には順番にパックされた対称行列の上三角部分を列ごとに格納しなければならない。また、 <i>uplo</i> = 'L' または 'l' と指定した場合は、 <i>ap</i> (1) に <i>a</i> (1, 1) が、 <i>ap</i> (2) に <i>a</i> (2, 1) が、 <i>ap</i> (3) に <i>a</i> (3, 1) がそれぞれ格納されるように、配列 <i>ap</i> には順番にパックされた対称行列の下三角部分を列ごとに格納しなければならない。
<i>x</i>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合) 配列、次元は $(1 + (n - 1)*abs(incx))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>y</i>	COMPLEX (cspmv の場合) COMPLEX*16 (zspmv の場合) 配列、次元は $(1 + (n - 1)*abs(incy))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。
出力パラメーター	
<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。

?spr

対称圧縮行列の複素数対称階数1の更新を実行する。

構文

```
call cspr( uplo, n, alpha, x, incx, ap )
call zspr( uplo, n, alpha, x, incx, ap )
```

説明

?spr ルーチンは、次のように定義される行列 - ベクトル演算を実行する。

$$a := \alpha * x * \text{conjg}(x') + a$$

ここで、

α は複素数のスカラーである。

x は、 n 個の成分を持つ複素数ベクトルである。

a は、圧縮形式で与える $n \times n$ の対称行列である。

このルーチンと実数において等価な関数が BLAS にある (第 2 章の「[?spr](#)」を参照) 。

入力パラメーター

<code>uplo</code>	CHARACTER*1。次に示すように、行列 a の上三角部分と下三角部分のどちらを圧縮形式の配列 ap において与えるかを指定する。 <code>uplo = 'U'</code> または <code>'u'</code> の場合、行列 a の上三角部分を配列 ap に格納する。 <code>uplo = 'L'</code> または <code>'l'</code> の場合、行列 a の下三角部分を配列 ap に格納する。
<code>n</code>	INTEGER。行列 a の次数を指定する。 n の値は、ゼロ以上でなければならない。
<code>alpha</code>	COMPLEX (<code>cspr</code> の場合) COMPLEX*16 (<code>zspr</code> の場合) スカラー α を指定する。
<code>x</code>	COMPLEX (<code>cspr</code> の場合) COMPLEX*16 (<code>zspr</code> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(incx))$ 以上。このルーチンに入る前に、増分された配列 x に n 個の成分を持つベクトル x を格納しなければならない。
<code>incx</code>	INTEGER。 x の成分に対する増分を指定する。 <code>incx</code> の値は、ゼロであってはならない。
<code>ap</code>	COMPLEX (<code>cspr</code> の場合) COMPLEX*16 (<code>zspr</code> の場合)

配列、次元は $((n*(n+1))/2)$ 以上。 `uplo = 'U'` または `'u'` と指定した場合は、このルーチンに入る前に、`ap(1)` に $a(1, 1)$ が、`ap(2)` に $a(1, 2)$ が、`ap(3)` に $a(2, 2)$ がそれぞれ格納されるように、配列 `ap` には、順番に圧縮された対称行列の上三角部分を列ごとに格納しなければならない。

`uplo = 'L'` または `'l'` と指定した場合は、このルーチンに入る前に、`ap(1)` に $a(1,1)$ が、`ap(2)` に $a(2,1)$ が、`ap(3)` に $a(3,1)$ がそれぞれ格納されるように、配列 `ap` には、順番にパックされた対称行列の下三角部分を列ごとに格納しなければならない。

直交成分の虚数部分はゼロとみなすため設定する必要はなく、また、出力ではゼロに設定される。

出力パラメーター

`ap` `uplo = 'U'` または `'u'` と指定した場合は、更新された行列の上三角部分によって上書きされる。
`uplo = 'L'` または `'l'` と指定した場合は、更新された行列の下三角部分によって上書きされる。

?symv

複素対称行列に対して行列-ベクトル積を計算する。

構文

```
call csymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
call zsymv( uplo, n, alpha, a, lda, x, incx, beta, y, incy )
```

説明

これらのルーチンは、次のように定義される行列-ベクトル演算を実行する。

$$y := \alpha * a * x + \beta * y,$$

ここで、

`alpha` と `beta` は、複素スカラーである。

`x` と `y` は n 個の成分を持つ複素ベクトルである。

`a` は、 $n \times n$ の対称複素行列である。

このルーチンと実数において等価な関数が BLAS にある (第2章の「[?symv](#)」を参照)。

入力パラメーター

`uplo` CHARACTER*1。次に示すように、配列 `a` の上三角部分と下三角部分のどちらが参照されるかを指定する。

	<i>uplo</i> = 'U' または 'u' の場合、配列 <i>a</i> の上三角部分が参照される。
	<i>uplo</i> = 'L' または 'l' の場合、配列 <i>a</i> の下三角部分が参照される。
<i>n</i>	INTEGER。行列 <i>a</i> の次数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha, beta</i>	COMPLEX (<i>csymv</i> の場合) COMPLEX*16 (<i>zsymv</i> の場合) スカラー <i>alpha</i> と <i>beta</i> を指定する。 <i>beta</i> をゼロに設定した場合は、 <i>y</i> を設定する必要はない。
<i>a</i>	COMPLEX (<i>csymv</i> の場合) COMPLEX*16 (<i>zsymv</i> の場合) 配列、次元は (<i>lda, n</i>)。 <i>uplo</i> = 'U' または 'u' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の上三角部分には対称行列の上三角部分を格納しなければならず、 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' または 'l' と指定した場合は、このルーチンに入る前に、配列 <i>a</i> の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならず、 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおり、 <i>a</i> の第 1 次元を指定する。 <i>lda</i> の値は、 $\max(1, n)$ 以上でなければならない。
<i>x</i>	COMPLEX (<i>csymv</i> の場合) COMPLEX*16 (<i>zsymv</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に <i>n</i> 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>y</i>	COMPLEX (<i>csymv</i> の場合) COMPLEX*16 (<i>zsymv</i> の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incy}))$ 以上。このルーチンに入る前に、増分された配列 <i>y</i> に <i>n</i> 個の成分を持つベクトル <i>y</i> を格納しなければならない。
<i>incy</i>	INTEGER。 <i>y</i> の成分に対する増分を指定する。 <i>incy</i> の値は、ゼロであってはならない。

出力パラメーター

<i>y</i>	更新されたベクトル <i>y</i> によって上書きされる。
----------	--------------------------------

?syr

複素対称行列の対称階数 1 の更新を実行する。

構文

```
call csyr( uplo, n, alpha, x, incx, a, lda )
call zsyr( uplo, n, alpha, x, incx, a, lda )
```

説明

このルーチンは、次のように定義される対称階数 1 の演算を実行する。

$$a := \alpha * x * x' + a$$

ここで、

α は複素数のスカラーである。

x は、 n 個の成分を持つ複素数ベクトルである。

a は、 $n \times n$ の複素対称行列である。

このルーチンと実数において等価な関数が BLAS にある (第 2 章の「[?syr](#)」を参照)。

入力パラメーター

<i>uplo</i>	CHARACTER*1。次に示すように、配列 <i>a</i> の上三角部分と下三角部分のどちらが参照されるかを指定する。 <i>uplo</i> = 'U' または 'u' の場合、配列 <i>a</i> の上三角部分が参照される。 <i>uplo</i> = 'L' または 'l' の場合、配列 <i>a</i> の下三角部分が参照される。
<i>n</i>	INTEGER。行列 <i>a</i> の次数を指定する。 <i>n</i> の値は、ゼロ以上でなければならない。
<i>alpha</i>	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合) スカラー <i>alpha</i> を指定する。
<i>x</i>	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合) 配列、次元は $(1 + (n - 1) * \text{abs}(\text{incx}))$ 以上。このルーチンに入る前に、増分された配列 <i>x</i> に n 個の成分を持つベクトル <i>x</i> を格納しなければならない。
<i>incx</i>	INTEGER。 <i>x</i> の成分に対する増分を指定する。 <i>incx</i> の値は、ゼロであってはならない。
<i>a</i>	COMPLEX (csyr の場合) COMPLEX*16 (zsyr の場合)

配列、次元は (lda, n) 。 $uplo = 'U'$ または $'u'$ と指定した場合は、このルーチンに入る前に、配列 A の先頭の $n \times n$ の上三角部分には対称行列の上三角部分を格納しなければならず、 A の厳密な下三角部分は参照されない。

$uplo = 'L'$ または $'l'$ と指定した場合は、このルーチンに入る前に、配列 a の先頭の $n \times n$ の下三角部分には対称行列の下三角部分を格納しなければならず、 A の厳密な上三角部分は参照されない。

lda INTEGER。呼び出し元の (サブ) プログラムで宣言されているとおり、 a の第 1 次元を指定する。 lda の値は、 $\max(1, n)$ 以上でなければならない。

出力パラメーター

a $uplo = 'U'$ または $'u'$ と指定した場合は、配列 a の上三角部分が、更新された行列の上三角部分によって上書きされる。

$uplo = 'L'$ または $'l'$ と指定した場合は、配列 a の下三角部分が、更新された行列の下三角部分によって上書きされる。

i?max1

実数部分が最大絶対値を持つベクトルの成分のインデックスを検出する。

構文

```
index = icmax1( n, cx, incx )
```

```
index = izmax1( n, cx, incx )
```

説明

複素ベクトル cx が与えられたとき、 $i?max1$ 関数は実数部分が最大絶対値を持つベクトル成分のインデックスを返す。これら関数は BLAS 関数 $icamax/izamax$ を元にしていて、実数部分の絶対値を使用している。 $clacon/zlacon$ との併用を意図している。

入力パラメーター

n INTEGER。ベクトル cx の成分の数を指定する。

cx COMPLEX ($icmax1$ の場合)
COMPLEX*16 ($izmax1$ の場合)

配列、次元は $(1+(n-1)*abs(incx))$ 以上。

入力ベクトルが格納される。

$incx$ INTEGER。 cx の連続する成分の間隔を指定する。

出力パラメーター

index INTEGER。実数部分が最大絶対値を持つベクトル成分のインデックスが格納される。

?sum1

真の絶対値を用いて複素ベクトルの1- ノルムを実行する。

構文

```
res = ssum1( n, cx, incx )
res = dsum1( n, cx, incx )
```

説明

複素ベクトル *cx* が与えられたとき、*ssum1*/*dsum1* 関数は、ベクトル成分の絶対値の合計を取り、それぞれ単精度/倍精度の結果を返す。これら関数はレベル 1 BLAS の [scasum/dzasum](#) を元になっているが、真の絶対値を使用し、[clacon/zlacon](#) と併せて使用するように設計されている。

入力パラメーター

n INTEGER。ベクトル *cx* の成分の数を指定する。

cx COMPLEX (*ssum1* の場合)
COMPLEX*16 (*dsum1* の場合)
配列、次元は $(1+(n-1)*abs(incx))$ 以上。
成分を合計する入力ベクトルを格納する。

incx INTEGER。*cx* の連続する成分間の間隔を指定する (*incx* > 0)。

出力パラメーター

res REAL (*ssum1* の場合)
DOUBLE PRECISION (*dsum1* の場合)
絶対値の合計が格納される。

?gbtf2

非ブロック化バージョンのアルゴリズムを使って一般帯行列の LU 因子分解を計算する。

構文

```
call sgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )
call dgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )
```

```
call cgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )
call zgbtf2( m, n, kl, ku, ab, ldab, ipiv, info )
```

説明

このルーチンは、 kl 個の劣対角と ku 個の優対角を含む $m \times n$ の一般実数 / 複素帯行列 A の LU 因子分解を実行する。このルーチンは行交換を伴う部分ピボット演算を使用しており、また、非ブロック化ルーチンのアルゴリズムを実装している。レベル 2 BLAS を呼び出す。

参照

[?gbtrf](#)。

入力パラメーター

<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。 A の列数 ($n \geq 0$)。
<i>kl</i>	INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。
<i>ku</i>	INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。
<i>ab</i>	REAL (sgbtf2 の場合) DOUBLE PRECISION (dgbtf2 の場合) COMPLEX (cgbtf2 の場合) COMPLEX*16 (zgbtf2 の場合) 配列、次元は (<i>ldab</i> , *)。 配列 <i>ab</i> には、行列 A が帯形式で格納される (「 行列指数 」を参照)。 <i>ab</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldab</i>	INTEGER。配列 <i>ab</i> の第 1 次元 ($ldab \geq 2kl + ku + 1$)

出力パラメーター

<i>ab</i>	因子分解の詳細で上書きされる。 U の対角成分と $kl + ku$ 個の優対角成分は、 <i>ab</i> の最初の $1 + kl + ku$ 行に格納される。因子分解に使用した乗数は、次の kl 行に格納される。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス: 行 i は行 $ipiv(i)$ と交換される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - i の場合、 i 番目のパラメーターの値が不正である。 <i>info</i> = i の場合、 u_{ii} は 0 である。因子分解は完了したが U は完全に特異である。連立線形方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

?geb2

非ブロック化アルゴリズムを使って一般行列を二重対角形式に縮退させる。

構文

```
call sgeb2( m, n, a, lda, d, e, tauq, taup, work, info )
call dgeb2( m, n, a, lda, d, e, tauq, taup, work, info )
call cgeb2( m, n, a, lda, d, e, tauq, taup, work, info )
call zgeb2( m, n, a, lda, d, e, tauq, taup, work, info )
```

説明

このルーチンは直交/ユニタリー変換を用いて、 $m \times n$ の一般行列 A を上または下二重対角形式 B に縮退させる。 $Q'AP = B$

$m \geq n$ の場合、 B は上二重対角である。 $m < n$ の場合、 B は下二重対角である。

このルーチンでは、行列 Q と P を明示的な形式では表現せず、基本リフレクターの積として表現する。 $m \geq n$ の場合、

$$Q = H(1)H(2)\dots H(n) \quad \text{および} \quad P = G(1)G(2)\dots G(n-1)$$

$m < n$ の場合、

$$Q = H(1)H(2)\dots H(m-1) \quad \text{および} \quad P = G(1)G(2)\dots G(m)$$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$$H(i) = I - \tau u v' \quad \text{および} \quad G(i) = I - \tau u' u$$

τu と $\tau u'$ はスカラー (sgeb2/dgeb2 では実数、cgeb2/zgeb2 では複素)、 v と u はベクトルである (sgeb2/dgeb2 では実数、cgeb2/zgeb2 では複素)。

入力パラメーター

m INTEGER。行列 A の行数 ($m \geq 0$)。

n INTEGER。 A の列数 ($n \geq 0$)。

$a, work$ REAL (sgeb2 の場合)
DOUBLE PRECISION (dgeb2 の場合)
COMPLEX (cgeb2 の場合)
COMPLEX*16 (zgeb2 の場合)

配列:

$a(lda, *)$ には縮退する $m \times n$ の一般行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

$work(*)$ はワークスペース配列、 $work$ の値は $\max(1, m, n)$ 以上でなければならない。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

<i>a</i>	<p>$m \geq n$ の場合、<i>a</i> の対角および第 1 の優対角成分は上二重対角行列 <i>B</i> によって上書きされる。対角よりも下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。また、第 1 の優対角よりも上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>P</i> を表現する。</p> <p>$m < n$ の場合、<i>a</i> の対角線と第 1 の劣対角成分は、下二重対角行列 <i>B</i> によって上書きされる。第 1 の劣対角よりも下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。また、対角よりも上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>P</i> を表現する。</p>
<i>d</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, \min(m, n))$ 以上。 二重対角行列 <i>B</i> の対角成分 ($d(i) = a(i, i)$) が格納される。</p>
<i>e</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(1, \min(m, n) - 1)$ 以上。 二重対角行列 <i>B</i> の非対角成分が次のように格納される。</p> <p>$m \geq n$ のとき、$i = 1, 2, \dots, n-1$ に対して $e(i) = a(i, i+1)$、 $m < n$ のとき、$i = 1, 2, \dots, m-1$ に対して $e(i) = a(i+1, i)$</p>
<i>tauq, taup</i>	<p>REAL (sgebd2 の場合) DOUBLE PRECISION (dgebd2 の場合) COMPLEX (cgebd2 の場合) COMPLEX*16 (zgebd2 の場合) 配列、次元は $(1, \min(m, n))$ 以上。 直交/ユニタリー行列 <i>Q</i> と <i>P</i> をそれぞれ表す基本リフレクターのスカラー係数が格納される。</p>
<i>info</i>	<p>INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = -<i>i</i> の場合、<i>i</i> 番目のパラメーターの値が不正である。</p>

?gehd2

非ブロック化アルゴリズムを使用して一般正方行列を上 Hessenberg 形式に縮退させる。

構文

```
call sgehd2( n, ilo, ihi, a, lda, tau, work, info )
call dgehd2( n, ilo, ihi, a, lda, tau, work, info )
call cgehd2( n, ilo, ihi, a, lda, tau, work, info )
call zgehd2( n, ilo, ihi, a, lda, tau, work, info )
```

説明

このルーチンは、直交またはユニタリーの相似変換 $Q^T A Q = H$ によって、実数 / 複素一般行列 A を上 Hessenberg 形式 H に縮退させる。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は基本リフレクターの積として表現される。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
ilo, ihi	INTEGER。 A はすでに行 $1:ilo-1$ と列 $ihi+1:n$ で上三角になっていると仮定する。 A が ?gebal から出力された場合、 ilo と ihi に、そのルーチンから返された値を設定しなければならない。それ以外の場合は $ilo=1$ かつ $ihi=n$ を設定する。制約: $1 \leq ilo \leq ihi \leq \max(1, n)$
$a, work$	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列: $a(lda, *)$ には縮退させる $n \times n$ の行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(n)$ は、ワークスペース配列。
lda	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。

出力パラメーター

a	A の上三角と第 1 の劣対角は、上 Hessenberg 行列 H および第 1 の劣対角よりも下の成分で上書きされ、配列 tau とともに、基本リフレクターの積として直交 / ユニタリーの行列 Q を表現する。次の「アプリケーション・ノート」を参照。
tau	REAL (sgehd2 の場合) DOUBLE PRECISION (dgehd2 の場合) COMPLEX (cgehd2 の場合) COMPLEX*16 (zgehd2 の場合) 配列、次元は $\max(1, n-1)$ 以上。 基本リフレクターのスカラー係数が格納される。次の「アプリケーション・ノート」を参照。
$info$	INTEGER。 $info=0$ の場合、実行は正常に終了したことを示す。 $info=-i$ の場合、 i 番目のパラメーターの値が不正である。

アプリケーション・ノート

?gexrd は、行列 Q を $(ihi-ilo)$ 個の基本リフレクターの積として表現する。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

τ は実数 / 複素スカラー、 v は実数 / 複素ベクトルで、 $v(1:i) = 0$ 、 $v(i+1) = 1$ および $v(ihi+1:n) = 0$ である。

ルーチン終了時には、 $v(i+2:ihi)$ は $a(i+2:ihi, i)$ に格納され、 τ は $\tau(i)$ に格納される。

$n = 7$ 、 $ilo = 2$ 、 $ihi = 6$ の場合、 a の内容を次の例に示す。

入力

出力

$$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$$

$$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & & h & h & h & h & h \\ v_2 & h & h & h & h & h & h \\ v_2 & v_3 & h & h & h & h & h \\ v_2 & v_3 & v_4 & h & h & h & h \\ & & & & & & a \end{bmatrix}$$

ここで a は元の行列 A の成分を表し、 h は上 Hessenberg 行列 H の更新された成分を表し、 v_i は $H(i)$ を定義するベクトルの成分を表している。

?gelq2

非ブロック化アルゴリズムを使用して一般矩形行列の LQ 因子分解を計算する。

構文

```
call sgelq2( m, n, a, lda, tau, work, info )
call dgelq2( m, n, a, lda, tau, work, info )
call cgelq2( m, n, a, lda, tau, work, info )
call zgelq2( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ 行列 A の LQ 因子分解を $A = LQ$ として計算する。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は $\min(m, n)$ の基本リフレクターの積として表現される。

$Q = H(k) \dots H(2) H(1)$ (または複素の場合、 $Q = H(k)' \dots H(2)' H(1)'$)、ここで $k = \min(m, n)$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

τ は $\tau(i)$ に格納される実数 / 複素スカラー、 v は $v(1:i-1) = 0$ および $v(i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時に $v(i+1:n)$ は $a(i, i+1:n)$ に格納される。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
$a, work$	REAL (sgelq2 の場合) DOUBLE PRECISION (dgelq2 の場合) COMPLEX (cgelq2 の場合) COMPLEX*16 (zgelq2 の場合) 配列: $a(lda, *)$ には、 $m \times n$ の行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $work(m)$ は、ワークスペース配列。
lda	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

a	因子分解のデータによって、次のように上書きされる。 配列 a の対角成分とその下の成分は $m \times \min(n, m)$ の下台形行列 L ($n \geq m$ の場合、 L は下三角行列) で上書きされる。対角よりも上の成分は、配列 τ とともに、 $\min(n, m)$ の基本リフレクター積として直交 / ユニタリーの行列 Q を表現する。
τ	REAL (sgelq2 の場合) DOUBLE PRECISION (dgelq2 の場合) COMPLEX (cgelq2 の場合) COMPLEX*16 (zgelq2 の場合) 配列、次元は $\max(1, \min(m, n))$ 以上。 基本リフレクターのスカラー係数が入る。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

?geql2

非ブロック化アルゴリズムを使用して一般矩形行列の QL 因子分解を計算する。

構文

```
call sgeql2( m, n, a, lda, tau, work, info )
call dgeql2( m, n, a, lda, tau, work, info )
```

```
call cgeql2( m, n, a, lda, tau, work, info )
call zgeql2( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ 行列 A の QL 因子分解を $A = QL$ として計算する。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。

$Q = H(k) \dots H(2) H(1)$ 、ここで $k = \min(m, n)$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

τ は $\tau(i)$ に格納される実数 / 複素スカラー、 v は $v(m-k+i+1:m) = 0$ および $v(m-k+i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時には $v(1:m-k+i-1)$ は $a(1:m-k+i-1, n-k+i)$ に格納される。

入力パラメーター

m INTEGER。行列 A の行数 ($m \geq 0$)。

n INTEGER。 A の列数 ($n \geq 0$)。

$a, work$ REAL (sgeql2 の場合)
DOUBLE PRECISION (dgeql2 の場合)
COMPLEX (cgeql2 の場合)
COMPLEX*16 (zgeql2 の場合)
配列：
 $a(lda, *)$ には、 $m \times n$ の行列 A を格納する。
 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
 $work(m)$ は、ワークスペース配列。

lda INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

a 因子分解のデータによって、次のように上書きされる。
 $m \geq n$ の場合は、部分配列 $a(m-n+1:m, 1:n)$ の下三角部分に $n \times n$ の下三角行列 L が格納される。
 $m < n$ の場合は、 $(n-m)$ 番目の優対角成分とその下の各成分に $m \times n$ の下台形行列 L が格納される。残りの成分は、配列 τ とともに、基本リフレクターの積として直交 / ユニタリー行列 Q を表現する。

τ REAL (sgeql2 の場合)
DOUBLE PRECISION (dgeql2 の場合)
COMPLEX (cgeql2 の場合)
COMPLEX*16 (zgeql2 の場合)
配列、次元は $\max(1, \min(m, n))$ 以上。
基本リフレクターのスカラー係数が入る。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正である。

?geqr2

非ブロック化アルゴリズムを使用して一般矩形行列の QR 因子分解を計算する。

構文

```
call sgeqr2( m, n, a, lda, tau, work, info )
call dgeqr2( m, n, a, lda, tau, work, info )
call cgeqr2( m, n, a, lda, tau, work, info )
call zgeqr2( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数 / 複素の $m \times n$ 行列 A の QR 因子分解を $A = QR$ として計算する。

このルーチンでは、行列 Q を明示的な形式では表現しない。代わりに、 Q は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。

$Q = H(1)H(2) \dots H(k)$ 、ここで $k = \min(m, n)$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

τ は $\tau(i)$ に格納される実数 / 複素スカラー、 v は $v(1:i-1) = 0$ および $v(i) = 1$ を持つ実数 / 複素ベクトルである。

ルーチン終了時に $v(i+1:m)$ は $a(i+1:m, i)$ に格納される。

入力パラメーター

m INTEGER。行列 A の行数 ($m \geq 0$)。

n INTEGER。 A の列数 ($n \geq 0$)。

a, *work* REAL (sgeqr2 の場合)
DOUBLE PRECISION (dgeqr2 の場合)
COMPLEX (cgeqr2 の場合)
COMPLEX*16 (zgeqr2 の場合)
配列：
a(*lda*,*) には、 $m \times n$ の行列 A を格納する。
a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
work(*n*) は、ワークスペース配列。

lda INTEGER。 *a* の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

<i>a</i>	因子分解のデータによって、次のように上書きされる。 行列の対角成分とその上の成分は、 $\min(n, m) \times n$ の上台形行列 <i>R</i> によって上書きされる ($m \geq n$ のとき <i>R</i> は上三角行列)。 対角よりも下の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交/ユニタリー行列 <i>Q</i> を表現する。
<i>tau</i>	REAL (sgeqr2 の場合) DOUBLE PRECISION (dgeqr2 の場合) COMPLEX (cgeqr2 の場合) COMPLEX*16 (zgeqr2 の場合) 配列、次元は $\max(1, \min(m, n))$ 以上。 基本リフレクターのスカラー係数が入る。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

?gerq2

非ブロック化アルゴリズムを使用して一般矩形行列の *RQ* 因子分解を計算する。

構文

```
call sgerq2( m, n, a, lda, tau, work, info )
call dgerq2( m, n, a, lda, tau, work, info )
call cgerq2( m, n, a, lda, tau, work, info )
call zgerq2( m, n, a, lda, tau, work, info )
```

説明

このルーチンは実数/複素の $m \times n$ 行列 *A* の *RQ* 因子分解を $A = QR$ として計算する。

このルーチンでは、行列 *Q* を明示的な形式では表現しない。代わりに、*Q* は、 $\min(m, n)$ 個の基本リフレクターの積として表現される。

$Q = H(1)H(2) \dots H(k)$ 、ここで $k = \min(m, n)$

それぞれの *H*(*i*) は次のような形式を持つ。

$H(i) = I - \tau v v'$

tau は *tau*(*i*) に格納される実数/複素スカラー、*v* は $v(n-k+i+1:n) = 0$ および $v(n-k+i) = 1$ を持つ実数/複素ベクトルである。

ルーチン終了時に $v(1:n-k+i-1)$ は $a(m-k+i, 1:n-k+i-1)$ に格納される。

入力パラメーター

m INTEGER。行列 *A* の行数 ($m \geq 0$)。

<i>n</i>	INTEGER。 <i>A</i> の列数 ($n \geq 0$)。
<i>a</i> , <i>work</i>	REAL (sgerq2 の場合) DOUBLE PRECISION (dgerq2 の場合) COMPLEX (cgerq2 の場合) COMPLEX*16 (zgerq2 の場合) 配列 : <i>a</i> (<i>lda</i> ,*) には、 $m \times n$ の行列 <i>A</i> を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (<i>m</i>) は、ワークスペース配列。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

<i>a</i>	因子分解のデータによって、次のように上書きされる。 $m \leq n$ の場合は、部分配列 <i>a</i> (1: <i>m</i> , <i>n</i> - <i>m</i> +1: <i>n</i>) の上三角部分に、 $m \times m$ の上三角行列 <i>R</i> が格納される。 $m > n$ の場合は、(<i>m</i> - <i>n</i>) 番目の劣対角成分とその上の各成分に、 $m \times n$ の上台形行列 <i>R</i> が格納される。残りの成分は、配列 <i>tau</i> とともに、 $\min(m, n)$ 個の基本リフレクターの積として直交/ユニタリ行列 <i>Q</i> を表現する。
<i>tau</i>	REAL (sgerq2 の場合) DOUBLE PRECISION (dgerq2 の場合) COMPLEX (cgerq2 の場合) COMPLEX*16 (zgerq2 の場合) 配列、次元は $\max(1, \min(m, n))$ 以上。 基本リフレクターのスカラー係数が入る。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

?gesc2

?getc2 で計算された完全ピボット演算による LU 因子分解を使用して連立線形方程式を解く。

構文

```
call sgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call dgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call cgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
call zgetc2( n, a, lda, rhs, ipiv, jpiv, scale )
```

説明

このルーチンは、次の連立線形方程式を解く。

$$AX = scale * RHS$$

`?getc2` で計算された完全ピボット演算による LU 因子分解を使用して、一般 $n \times n$ 行列 A について解く。

入力パラメーター

<code>n</code>	INTEGER。行列 A の次数。
<code>a, rhs</code>	REAL (<code>sgetc2</code> の場合) DOUBLE PRECISION (<code>dgetc2</code> の場合) COMPLEX (<code>cgetc2</code> の場合) COMPLEX*16 (<code>zgetc2</code> の場合) 配列: $a(lda, *)$ には <code>?getc2</code> で計算された $n \times n$ 行列 A の因子分解の LU 部分を格納する。 $A = PLUQ$ a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $rhs(n)$ には連立 1 次方程式の右辺ベクトルを格納する。
<code>lda</code>	INTEGER。 a の第 1 次元。 $\max(1, n)$ 以上。
<code>ipiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 i は行 $ipiv(i)$ と交換されている。
<code>jpiv</code>	INTEGER。 配列、次元は $\max(1, n)$ 以上。 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 j は列 $jpiv(j)$ と交換されている。

出力パラメーター

<code>rhs</code>	解のベクトル X で上書きされる。
<code>scale</code>	REAL (<code>sgetc2/cgetc2</code> の場合) DOUBLE PRECISION (<code>dgetc2/zgetc2</code> の場合) スケール係数が格納される。 <code>scale</code> は解におけるオーバーフローを防ぐため $0 \leq scale \leq 1$ の範囲で選択される。

?getc2

一般 $n \times n$ 行列の完全ピボット演算による LU 因子分解を計算する。

構文

```
call sgetc2( n, a, lda, ipiv, jpiv, info )
call dgetc2( n, a, lda, ipiv, jpiv, info )
call cgetc2( n, a, lda, ipiv, jpiv, info )
call zgetc2( n, a, lda, ipiv, jpiv, info )
```

説明

このルーチンは一般 $n \times n$ 行列 A の完全ピボット演算による LU 因子分解を計算する。因子分解は形式 $A = P * L * U * Q$ を持ち、 P と Q は置換行列、 L は単位対角成分を持つ下三角、 U は上三角である。

このルーチンで計算される LU 因子分解は、[?latdf](#) より Dif 推定値の逆数に対する影響を計算するのに用いられる。

入力パラメーター

n INTEGER。行列 A の次数 ($n \geq 0$)。

a REAL (sgetc2 の場合)
 DOUBLE PRECISION (dgetc2 の場合)
 COMPLEX (cgetc2 の場合)
 COMPLEX*16 (zgetc2 の場合)
 配列 $a(lda, *)$ には因子分解する $n \times n$ 行列 A を格納する。
 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

lda INTEGER。 *a* の第 1 次元。 $\max(1, n)$ 以上。

出力パラメーター

a 因子分解 $A = P * L * U * Q$ で得られた係数 L と U で上書きされる。
 L の単位対角成分は格納されない。 $U(k, k)$ が $smin$ 未満の場合、
 $U(k, k)$ が $smin$ の値として与えられる。すなわち、非特異の摂動
 連立方程式を与える。

ipiv INTEGER。
 配列、次元は $\max(1, n)$ 以上。
 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 i は行
 $ipiv(i)$ と交換されている。

jpiv INTEGER
 配列、次元は $\max(1, n)$ 以上。
 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 j は列
 $jpiv(j)$ と交換されている。

info INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info = k > 0$ の場合、 $Ax = b$ で x を解こうとすると $U(k, k)$ はおそ
 らくオーバーフローを生む。そこで、オーバーフローを防止す
 るために U は摂動となる。

?getf2

一般 $m \times n$ 行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する (非ブロック化アルゴリズム)。

構文

```
call sgetf2( m, n, a, lda, ipiv, info )
call dgetf2( m, n, a, lda, ipiv, info )
call cgetf2( m, n, a, lda, ipiv, info )
call zgetf2( m, n, a, lda, ipiv, info )
```

説明

このルーチンは一般 $m \times n$ 行列 A の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する。因子分解の形式は次のとおりである。

$$A = PLU,$$

ここで、 P は置換行列、 L は単位対角成分を含む下三角 ($m > n$ の場合は下台形)、 U は上三角 ($m < n$ の場合上台形) である。

入力パラメーター

<i>m</i>	INTEGER。行列 A の行数 ($m \geq 0$)。
<i>n</i>	INTEGER。 A の列数 ($n \geq 0$)。
<i>a</i>	REAL (sgetf2 の場合) DOUBLE PRECISION (dgetf2 の場合) COMPLEX (cgetf2 の場合) COMPLEX*16 (zgetf2 の場合) 配列、次元は (<i>lda</i> , *)。因子分解の対象となる行列 A が格納される。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, m)$ 以上。

出力パラメーター

<i>a</i>	L と U によって上書きされる。 L の単位対角成分は格納されない。
<i>ipiv</i>	INTEGER。 配列、次元は $\max(1, \min(m, n))$ 以上。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 i は行 $ipiv(i)$ と交換されている。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> > 0 の場合、 u_{ii} はゼロである。因子分解は完了したが U は完全に特異である。連立線形方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

?gtts2

?gttrf によって行われた LU 因子分解を使用して、三重対角行列を係数行列とする連立線形方程式を解く。

構文

```
call sgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call dgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call cgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
call zgtts2( itrans, n, nrhs, dl, d, du, du2, ipiv, b, ldb )
```

説明

このルーチンは、複数の右辺を持つ以下の連立線形方程式のうち 1 つを X について解く。

$AX=B$ $A^T X=B$ または $A^H X=B$ (複素行列のみ)、
 A は三重対角行列で [?gttrf](#) で計算した LU 因子分解を使用する。

入力パラメーター

itrans INTEGER。ゼロ、1、または 2 のいずれかでなければならない。
 方程式の形式を指定する。
itrans = 0 の場合、 $AX=B$ (転置なし)
itrans = 1 の場合、 $A^T X=B$ (転置)
itrans = 2 の場合、 $A^H X=B$ (共役転置)

n INTEGER。行列 A の次数 ($n \geq 0$)。

nrhs INTEGER。右辺の数、すなわち、 B の列数 ($nrhs \geq 0$)。

dl, d, du, du2, b REAL (sgtts2 の場合)
 DOUBLE PRECISION (dgtts2 の場合)
 COMPLEX (cgtts2 の場合)
 COMPLEX*16 (zgtts2 の場合)
 配列: *dl*($n-1$), *d*(n), *du*($n-1$), *du2*($n-2$), *b*(*ldb*, *nrhs*)
 配列 *dl* には、 A の LU 因子分解で得られた行列 L を定義する
 ($n-1$) 個の乗数が格納される。
 配列 *d* には、 A の LU 因子分解で得られた上三角行列 U の n 個の
 対角成分が格納される。
 配列 *du* には、 U の最初の優対角成分の ($n-1$) 個の成分が格納さ
 れる。
 配列 *du2* には、 U の 2 番目の優対角成分の ($n-2$) 個の成分が格納
 される。
 配列 *b* には、行列 B が格納される。この行列の列は、連立方程
 式の右辺である。

<code>ldb</code>	INTEGER。 b のリーディング・ディメンション $ldb \geq \max(1, n)$ 。
<code>ipiv</code>	INTEGER。 配列、次元は (n) 。 ?qtrf によって返されるピボットのインデックス配列。

出力パラメーター

<code>b</code>	解の行列 X によって上書きされる。
----------------	----------------------

?labrd

一般行列の最初の nb 行と列を二重対角形式に縮退させる。

構文

```
call slabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call dlabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call clabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
call zlabrd( m, n, nb, a, lda, d, e, tauq, taup, x, ldx, y, ldy )
```

説明

このルーチンは、直交/ユニタリー変換 $Q'AP$ によって一般 $m \times n$ 行列 A の最初の nb 行と列を上または下二重対角形式に縮退させ、 A の縮退されていない部分に変換を適用するために必要な行列 X と Y を返す。

$m \geq n$ の場合、 A は上二重対角形式に縮退され、 $m < n$ の場合、下二重対角形式に縮退される。

行列 Q と P は基本リフレクターの積として表現される。

$Q = H(1)H(2) \dots H(nb)$ および $P = G(1)G(2) \dots G(nb)$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$H(i) = I - \tau_{i,q} v v'$ および $G(i) = I - \tau_{i,p} u u'$

$\tau_{i,q}$ と $\tau_{i,p}$ はスカラー、 v と u はベクトルである。

ベクトル v と u の成分は $m \times nb$ の行列 V と $nb \times n$ の行列 U' を形成する。行列 A の縮退されていない部分に次の形式のブロック更新を使って変換を適用するために、これらの行列と行列 X および Y が必要となる。 $A := A - V * U' - X * U'$

このルーチンは、[?gebrd](#) から呼び出される補助ルーチンである。

入力パラメーター

<code>m</code>	INTEGER。 行列 A の行数 ($m \geq 0$)。
<code>n</code>	INTEGER。 A の列数 ($n \geq 0$)。
<code>nb</code>	INTEGER。 縮退の対象となる A の先頭の行と列の数。

<i>a</i>	REAL (slabrd の場合) DOUBLE PRECISION (dlabrd の場合) COMPLEX (clabrd の場合) COMPLEX*16 (zlabrd の場合) 配列 $a(lda, *)$ には縮退される行列 A を格納する。 a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 a の第 1 次元。 $\max(1, m)$ 以上。
<i>ldx</i>	INTEGER。 出力配列 x の第 1 次元のサイズ。 $\max(1, m)$ 以上でなければならない。
<i>ldy</i>	INTEGER。 出力配列 y の第 1 次元のサイズ。 $\max(1, n)$ 以上でなければならない。

出力パラメーター

<i>a</i>	行列の最初の nb 行と列は上書きされる。配列の残りの部分は変更されない。 $m \geq n$ の場合、最初の nb 列の対角線上とその下の成分は、配列 $tauq$ とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表現する。また、最初の nb 行の対角よりも上の成分は、配列 $taup$ とともに、基本リフレクターの積として直交/ユニタリー行列 P を表現する。 $m < n$ の場合、最初の nb 列の対角よりも下の成分は、配列 $tauq$ とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表現する。また、最初の nb 行の対角線上とその上の成分は、配列 $taup$ とともに、基本リフレクターの積として直交/ユニタリー行列 P を表現する。
<i>d, e</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元はそれぞれ (nb)。 配列 d には縮退された行列の最初の nb 行と列の対角成分が格納される。 $d(i) = a(i, i)$ 。 配列 e には縮退された行列の最初の nb 行と列の非対角成分が格納される。
<i>tauq, taup</i>	REAL (slabrd の場合) DOUBLE PRECISION (dlabrd の場合) COMPLEX (clabrd の場合) COMPLEX*16 (zlabrd の場合) 配列、次元はそれぞれ (nb)。 直交/ユニタリー行列 Q と P をそれぞれ表す基本リフレクターのスカラ係数が格納される。
<i>x, y</i>	REAL (slabrd の場合) DOUBLE PRECISION (dlabrd の場合) COMPLEX (clabrd の場合) COMPLEX*16 (zlabrd の場合)

配列、次元は $x(ldx, nb)$, $y(ldy, nb)$ 。

配列 x には、 A の縮退されていない部分を更新するために必要となる $m \times nb$ の行列 X が格納される。

配列 y には、 A の縮退されていない部分を更新するために必要となる $n \times nb$ の行列 Y が格納される。

アプリケーション・ノート

$m \geq n$ の場合、基本リフレクター $H(i)$ と $G(i)$ に対して、

$v(1:i-1) = 0$, $v(i) = 1$, $v(i:m)$ は $a(i:m, i)$ に格納される。

$u(1:i) = 0$, $u(i+1) = 1$, $u(i+1:n)$ は $a(i, i+1:n)$ に格納される。

$tauq$ は $tauq(i)$ に、 $taup$ は $taup(i)$ に格納される。

$m < n$ の場合、

$v(1:i) = 0$, $v(i+1) = 1$, $v(i+1:m)$ は $a(i+2:m, i)$ に格納される。

$u(1:i-1) = 0$, $u(i) = 1$, $u(i:n)$ は $a(i, i+1:n)$ に格納される。

$tauq$ は $tauq(i)$ に、 $taup$ は $taup(i)$ に格納される。

ルーチン終了後の a の内容は、 $nb = 2$ において次の例に示される。

$m = 6, n = 5 (m > n)$

$m = 5, n = 6 (m < n)$

$$\begin{bmatrix} 1 & 1 & u_1 & u_1 & u_1 \\ v_1 & 1 & 1 & u_2 & u_2 \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

$$\begin{bmatrix} 1 & u_1 & u_1 & u_1 & u_1 & u_1 \\ 1 & 1 & u_2 & u_2 & u_2 & u_2 \\ v_1 & 1 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \\ v_1 & v_2 & a & a & a & a \end{bmatrix}$$

a は元の行列の変更されていない成分を意味し、 v_i は $H(i)$ を定義するベクトルの成分を意味し、 u_i は $G(i)$ を定義するベクトルの成分を意味する。

?lacon

行列-ベクトル積の評価にリバーズ・コミュニケーションを使用して正方行列の1-ノルムを推定する。

構文

```
call slacon( n, v, x, isgn, est, kase )
call dlacon( n, v, x, isgn, est, kase )
call clacon( n, v, x, est, kase )
call zlacon( n, v, x, est, kase )
```

説明

このルーチンは実数/複素の正方行列 A の 1- ノルムを推定する。行列 - ベクトル積の評価にはリバース・コミュニケーションが使用される。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 1$)。
v, x	REAL (slacon の場合) DOUBLE PRECISION (dlacon の場合) COMPLEX (clacon の場合) COMPLEX*16 (zlacon の場合) 配列、次元はそれぞれ、(n)。 v は、ワークスペース配列。 x は中間戻り後は入力として使用される。
$isgn$	INTEGER。ワークスペース配列、次元は (n)、実数の場合のみ使用される。
$kase$	INTEGER。?lacon を初めて呼び出すときは $kase$ はゼロでなければならない。

出力パラメーター

est	REAL (slacon/clacon の場合) DOUBLE PRECISION (dlacon/zlacon の場合) ノルム (A) の推定 (下限)。
$kase$	中間戻りでは、 $kase$ は 1 か 2 となり、 x が $A * x$ または $A' * x$ によって上書きされるかどうかを示す。?lacon からの最終戻りでは、 $kase$ は再びゼロになる。
v	最終戻りでは $v = A * w$ 、ここで $est = \text{norm}(v) / \text{norm}(w)$ (w は返されない)。
x	中間戻りでは、 x は次の値で上書きされる。 $A * x$ 、 $kase = 1$ の場合、 $A' * x$ 、 $kase = 2$ の場合、 (複素では A' は A の共役転置)、また他のパラメーターは変更しない状態で ?lacon を再度呼び出さなければならない。

?lacpy

1 つの 2 次元配列の一部または全部を他にコピーする。

構文

```
call slacpy( uplo, m, n, a, lda, b, ldb )
call dlacpy( uplo, m, n, a, lda, b, ldb )
call clacpy( uplo, m, n, a, lda, b, ldb )
```

```
call zlacpy( uplo, m, n, a, lda, b, ldb )
```

説明

このルーチンは 2 次元行列 A の一部または全部を他の行列 B にコピーする。

入力パラメーター

<code>uplo</code>	CHARACTER*1。 行列 A のうち、 B にコピーする部分を指定する。 <code>uplo = 'U'</code> の場合、 A の上三角部分をコピーする。 <code>uplo = 'L'</code> の場合、 A の下三角部分をコピーする。 それ以外の場合は、行列 A の全部がコピーする。
<code>m</code>	INTEGER。行列 A の行数 ($m \geq 0$)。
<code>n</code>	INTEGER。 A の列数 ($n \geq 0$)。
<code>a</code>	REAL (<code>slacpy</code> の場合) DOUBLE PRECISION (<code>dlacpy</code> の場合) COMPLEX (<code>clacpy</code> の場合) COMPLEX*16 (<code>zlacpy</code> の場合) 配列 <code>a(lda, *)</code> には $m \times n$ の行列 A を格納する。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。 <code>uplo = 'U'</code> の場合、三角または台形の上側だけがアクセスされる。 <code>uplo = 'L'</code> の場合、三角または台形の下側だけがアクセスされる。
<code>lda</code>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, m)$
<code>ldb</code>	INTEGER。出力配列 b の第 1 次元のサイズ。 $ldb \geq \max(1, m)$

出力パラメーター

<code>b</code>	REAL (<code>slacpy</code> の場合) DOUBLE PRECISION (<code>dlacpy</code> の場合) COMPLEX (<code>clacpy</code> の場合) COMPLEX*16 (<code>zlacpy</code> の場合) 配列 <code>b(ldb, *)</code> には $m \times n$ の行列 B が格納される。 b の第 2 次元は $\max(1, n)$ 以上でなければならない。 <code>uplo</code> で指定された部分において $B = A$ で上書きされる。
----------------	---

?ladiv

不必要なオーバーフローを回避して、実数演算で複素除算を実行する。

構文

```
call sladiv( a, b, c, d, p, q )
call dladiv( a, b, c, d, p, q )
```

```
res = cladiv( x, y )
res = zladiv( x, y )
```

説明

ルーチン `sladiv/dladiv` は、次式として複素除算を実数演算で実行する。

$$p + iq = \frac{a + ib}{c + id}$$

複素関数 `cladiv/zladiv` は結果を次式として計算する。

$$res = x/y ,$$

x と y は複素である。 x/y の計算は、結果がオーバーフローしない限り中間過程でオーバーフローは生じない。

入力パラメーター

a, b, c, d	REAL (<code>sladiv</code> の場合) DOUBLE PRECISION (<code>dladiv</code> の場合) 上の式にあるスカラー a, b, c, d (実数型のみ)
x, y	COMPLEX (<code>cladiv</code> の場合) COMPLEX*16 (<code>zladiv</code> の場合) 複素スカラー x と y (複素型のみ)

出力パラメーター

p, q	REAL (<code>sladiv</code> の場合) DOUBLE PRECISION (<code>dladiv</code> の場合) 上の式にあるスカラー p と q (実数型のみ)
res	COMPLEX (<code>cladiv</code> の場合) DOUBLE COMPLEX (<code>zladiv</code> の場合) x/y の除算の結果が格納される。

?lae2

2×2 対称行列の固有値を計算する。

構文

```
call slae2( a, b, c, rt1, rt2 )
call dlae2( a, b, c, rt1, rt2 )
```

説明

ルーチン `slae2/dlae2` は 2×2 対称行列の固有値を計算する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix}$$

出力において、 $rt1$ は大きい方の絶対値の固有値、 $rt2$ は小さい方の絶対値の固有値で上書きされる。

入力パラメーター

a, b, c REAL (slae2 の場合)
DOUBLE PRECISION (dlae2 の場合)
上記 2×2 行列の成分 a, b, c

出力パラメーター

$rt1, rt2$ REAL (slae2 の場合)
DOUBLE PRECISION (dlae2 の場合)
大きい方の絶対値、小さい方の絶対値に対してそれぞれ計算された固有値。

アプリケーション・ノート

$rt1$ はオーバーフロー/アンダーフローが起こらなければ、わずかな ulp (最小位置単位) に対して正確である。 $rt2$ は行列式 $a*c-b*b$ で、大幅な桁落ちがある場合に不正確となる可能性がある。 $rt2$ を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。

オーバーフローは、 $rt1$ がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こり得る。アンダーフローは入力データがゼロか $underflow_threshold / macheps$ を超えた場合には影響とはならない。

?laebz

実数対称三重対角行列の指定された値以下の固有値の個数を計算し、また ?stebz ルーチンが必要とする他の処理を実行する。

構文

```
call slaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol,
            reltol, pivmin, d, e, e2, nval, ab, c, mout, nab,
            work, iwork, info )
```

```
call dlaebz( ijob, nitmax, n, mmax, minp, nbmin, abstol,
            reltol, pivmin, d, e, e2, nval, ab, c, mout, nab,
            work, iwork, info )
```

説明

ルーチン ?laebz は関数 $N(w)$ を使った反復ループで構成される。関数 $N(w)$ は、対称三重対角行列 T から、引数 w 以下の固有値を計数する。ループには 2 つのタイプがある。

$ijob=1$ 、このあと続けて、

ijob = 2: 区間リストを入力として受け取り、元の区間の和集合と同一の固有値の和集合を持つ十分小さい区間のリストを返す。入力区間は $(ab(j,1), ab(j,2)]$, $j=1, \dots, minp$ である。出力区間 $(ab(j,1), ab(j,2)]$ は固有値 $nab(j,1)+1, \dots, nab(j,2)$ を含み、ここで $1 \leq j \leq mout$ である。

ijob = 3: 各入力区間 $(ab(j,1), ab(j,2)]$ 内で $N(w(j))=nval(j)$ となる $w(j)$ 点に対してバイナリサーチを実行し、サーチでは $c(j)$ を開始点として使用する。そのような $w(j)$ が見つかった場合、出力は $ab(j,1)=ab(j,2)=w$ となる。 $w(j)$ が見つからなかった場合、出力 $(ab(j,1), ab(j,2)]$ は、初期区間外に点がない限り $N(w)$ が $nval(j)$ を飛び越える点を含んだ小区間となる。前記区間はどのような場合でも半开区間である。

すなわち、形式 $(a,b]$ において b は含まれるが a は含まれないことに注意する。

アンダーフローを防ぐために、行列を、その最大の成分が $overflow^{**}(1/2) * underflow^{**}(1/4)$ よりも絶対値において超えないようにスケールしなければならない。小さい固有値の計算を最も精度高く行うには、行列を上記条件よりもはるかに小さくスケールしてはならない。

注: 通常、引数の妥当性は確認されない。

入力パラメーター

<i>ijob</i>	INTEGER。処理方法を指定する。 = 1: 初期区間に対して <i>nab</i> を計算する。 = 2: <i>T</i> の固有値を探すために二分法の反復を実行する。 = 3: $N(w)$ を反転するため、すなわち、指定した個数の <i>T</i> の固有値を左辺に持つ点を探すため、二分法の反復を実行する。 そのほかの値の場合、 <i>?laebz</i> は <i>info</i> = -1 を返す。
<i>nitmax</i>	INTEGER 実行する二分法の「レベル」の最大数で、すなわち、幅 <i>W</i> の区間を $2^{(-nitmax)} * W$ よりも小さくしない。 <i>nitmax</i> 反復後にすべての区間が収束しなかった場合、 <i>info</i> には非収束区間数が設定される。
<i>n</i>	INTEGER。 三重対角行列 <i>T</i> の次元 n 。この値は 1 以上でなければならない。
<i>mmax</i>	INTEGER。 区間の最大数。 <i>mmax</i> より大きい区間が生成された場合、 <i>?laebz</i> は <i>info</i> = <i>mmax</i> +1 を返して終了する。
<i>minp</i>	INTEGER。 区間の初期数。 <i>mmax</i> よりも大きくてはならない。
<i>nbmin</i>	INTEGER。 ベクトルループを使って処理されるべき、区間の最小値。ゼロの場合、スカラーループのみが使用される。
<i>abstol</i>	REAL (<i>s</i> laebz の場合) DOUBLE PRECISION (<i>d</i> laebz の場合) 区間の最小(絶対)幅。区間が <i>abstol</i> よりも狭い場合、または(大きさで)大きい方の終点の <i>reltol</i> 倍より狭い場合、区間は十分に小さい、すなわち収束としてみなされる。この値は 0 以上でなければならない。

<i>reltol</i>	<p>REAL (slaebz の場合) DOUBLE PRECISION (dlaebz の場合) 区間の最小相対幅。区間が <i>abstol</i> よりも狭い場合、または(大きさで)大きい方の終点の <i>reltol</i> 倍より狭い場合、区間は十分に小さい、すなわち収束としてみなされる。<i>radix*machine epsilon</i> 以上でなければならない。</p>
<i>pivmin</i>	<p>REAL (slaebz の場合) DOUBLE PRECISION (dlaebz の場合) Sturm シーケンスループにおける「ピボット」の最小絶対値。 この値は $e(j)**2 * safe_min$ 以上でかつ <i>safe_min</i> 以上でなければならず、ここで <i>safe_min</i> オーバーフローなしで除算できる最小値である。</p>
<i>d, e, e2</i>	<p>REAL (slaebz の場合) DOUBLE PRECISION (dlaebz の場合) 配列、次元はそれぞれ、(n) 配列 <i>d</i> には三重対角行列 <i>T</i> の対角成分を格納する。 配列 <i>e</i> には 1 から <i>n-1</i> の位置にある三重対角行列 <i>T</i> の非対角成分を格納する。<i>e(n)</i> は任意である。 配列 <i>e2</i> には三重対角行列 <i>T</i> の非対角成分の正方を格納する。<i>e2(n)</i> は無視される。</p>
<i>nval</i>	<p>INTEGER。 配列、次元は (<i>minp</i>) <i>ijob</i> = 1 または 2 の場合は参照されない。 <i>ijob</i> = 3 の場合、<i>N(w)</i> の求める値。</p>
<i>ab</i>	<p>REAL (slaebz の場合) DOUBLE PRECISION (dlaebz の場合) 配列、次元は (<i>mmax,2</i>) 区間の終点。<i>ab(j, 1)</i> は <i>j</i> 番目の区間の左側終点 <i>a(j)</i>、<i>ab(j, 2)</i> は <i>j</i> 番目の区間の右側終点 <i>b(j)</i></p>
<i>c</i>	<p>REAL (slaebz の場合) DOUBLE PRECISION (dlaebz の場合) 配列、次元は (<i>mmax</i>) <i>ijob</i> = 1 の場合は無視される。 <i>ijob</i> = 2 の場合、ワークスペース。 <i>ijob</i> = 3 の場合、<i>c(j)</i> はバイナリーサーチにおける最初の検索点に初期化されなければならない。</p>
<i>nab</i>	<p>INTEGER。 配列、次元は (<i>mmax,2</i>) <i>ijob</i>=2 の場合は <i>nab(i,j)</i> を設定しなければならない。次の条件を満たす必要がある。 $N(ab(i,1)) \leq nab(i,1) \leq nab(i,2) \leq N(ab(i,2))$、これは区間 <i>i</i> で、固有値 <i>nab(i,1)+1,...,nab(i,2)</i> のみが考慮されることを意味する。 通常、先に <i>ijob</i> = 1 で ?laebz を呼び出しておけば、<i>nab(i, j) = N(ab(i, j))</i> となる。 <i>ijob</i> = 3 の場合、?laebz を呼び出す前に、<i>nab</i> に別の値を通常は設定しなければならない。</p>

<i>work</i>	REAL (<i>s</i> laebz の場合) DOUBLE PRECISION (<i>d</i> laebz の場合) ワークスペース配列、次元は (<i>mmax</i>)。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (<i>mmax</i>)。

出力パラメーター

<i>nval</i>	<i>nval</i> の成分は <i>ab</i> 内の区間に対応して順序が変更される。そのため、出力の <i>nval</i> (<i>j</i>) は一般に入力の <i>nval</i> (<i>j</i>) とは同一ではないが、出力の区間 (<i>ab</i> (<i>j</i> , 1), <i>ab</i> (<i>j</i> , 2)] に対応する。
<i>ab</i>	入力区間は一般に、計算によって、変更、分割、または順序の変更が生じる。
<i>mout</i>	INTEGER。 <i>ijob</i> =1 の場合、区間内の固有値の個数。 <i>ijob</i> =2 または 3 の場合、区間出力の個数 <i>ijob</i> =3 の場合、 <i>mout</i> は <i>minp</i> と等しい。
<i>nab</i>	<i>ijob</i> =1 の場合、 <i>nab</i> (<i>i</i> , <i>j</i>) は <i>N</i> (<i>ab</i> (<i>i</i> , <i>j</i>)) に設定される。 <i>ijob</i> =2 の場合、 <i>nab</i> (<i>i</i> , <i>j</i>) には $\max(\mathit{na}(k), \mathit{nb}(k), N(\mathit{ab}(i,j)))$ が格納され、 <i>k</i> は出力区間 (<i>ab</i> (<i>j</i> ,1), <i>ab</i> (<i>j</i> ,2)] をもたらした入力区間のインデックス、 <i>na</i> (<i>k</i>) と <i>nb</i> (<i>k</i>) は <i>nab</i> (<i>k</i> ,1) と <i>nab</i> (<i>k</i> ,2) の入力値である。 <i>ijob</i> =3 の場合、 <i>nab</i> (<i>i</i> , 1) が変更されない (すなわち出力値は入力値と同じ (モジュロの順序変更、 <i>nval</i> と <i>ab</i> 参照)) すべてのサーチ点 <i>w</i> に対して <i>N</i> (<i>w</i>) > <i>nval</i> (<i>i</i>) でない限り、または、 <i>nab</i> (<i>i</i> , 2) が変更されないすべてのサーチ点 <i>w</i> に対して <i>N</i> (<i>w</i>) < <i>nval</i> (<i>i</i>) でない限り、 <i>nab</i> (<i>i</i> , <i>j</i>) には <i>N</i> (<i>ab</i> (<i>i</i> , <i>j</i>)) が格納される。
<i>info</i>	INTEGER。 0: すべての区間は収束した。 1 ~ <i>mmax</i> : 最後の <i>info</i> 区間は収束しなかった。 <i>mmax</i> +1: <i>mmax</i> を超える区間が生成された。

アプリケーション・ノート

このルーチンは他の LAPACK ルーチンからの呼び出しのみを想定しているため、インターフェイスが使いにくくなっている。目的は次の 2 つである。

(a) 固有値を探す。?laebz は 1 つ以上の初期区間を *ab* に持っていなければならない、また ?laebz を *ijob*=1 で呼び出さなければならない。これは *nab* を設定し、また固有値をカウントする。固有値を持たない区間は通常この時点で捨てられる。また、区間 *i* のすべての固有値が必要ではない場合、*nab*(*i*, 1) を増やすか *nab*(*i*, 2) を減らせる。例えば、最大固有値を得るために *nab*(*i*, 1)=*nab*(*i*, 2)-1 と設定する。次に、*ijob*=2 とし、*mmax* を *ijob*=1 の呼び出しで得た *mout* 値以上にして ?laebz を呼び出す。*ijob*=2 の呼び出し後、固有値 *nab*(*i*, 1)+1 から *nab*(*i*, 2) は、*abstol* と *reltol* で指定された許容値で、およそ *ab*(*i*, 1) (または *ab*(*i*, 2)) となる。

(b) 固有値 *w*(*f*), ..., *w*(*l*) を含む区間 [*a*', *b*'] を探す。ここでは Gershgorin interval (*a*,*b*) 区間から開始する。*ab* に 2 個のサーチ区間を設定し、どちらも最初は (*a*,*b*) とする。*nval* の 1 つの成分は *f*-1 を、その他の成分には 1 を格納しなければならない一方で、*c* には *a* と *b*

がそれぞれ格納されなければならない。求めた区間が (a, b) にない場合をエラーとするため、 $nab(i, 1)$ は -1 でなければならない、 $nab(i, 2)$ は $n+1$ でなければならない。そして、 $ijob=3$ で `?laebz` を呼び出す。出力で、 $w(f-1) < w(f)$ ならば、区間の 1 つ j は $ab(j, 1)=ab(j, 2)$ と $nab(j, 1)=nab(j, 2)=f-1$ を持ち、一方、指定された許容値に対して、 $w(f-k)=\dots=w(f+r)$ 、 $k > 0$ および $r \geq 0$ ならば、区間は $N(ab(j, 1))=nab(j, 1)=f-k$ と $N(ab(j, 2))=nab(j, 2)=f+r$ を持つ。 $w(l) < w(l+1)$ と $w(l-r)=\dots=w(l+k)$ は同様に扱われる。

?laed0

`?stedc` で使用される。縮退されていない対称三重対角行列の固有値と対応する固有ベクトルを分割統治法を使用してすべて求める。

構文

```
call slaed0(  icompq, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork, info )
call dlaed0(  icompq, qsiz, n, d, e, q, ldq, qstore, ldqs, work, iwork, info )
call claed0(  qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork, info )
call zlaed0(  qsiz, n, d, e, q, ldq, qstore, ldqs, rwork, iwork, info )
```

説明

実数型の本ルーチンは、対称三重対角行列の固有値と (オプションとして) 対応する固有ベクトルを分割統治法を使用してすべて求める。

複素型の `claed0/zlaed0` は、密または帯エルミート行列を縮退して得られる 1 個の対角ブロックである対称三重対角行列のすべての固有値と、密または帯行列の対応する固有ベクトルを求める

入力パラメーター

<code>icompq</code>	INTEGER。実数型でのみ使用される。 <code>icompq = 0</code> の場合、固有値のみ計算する。 <code>icompq = 1</code> の場合、元の密対称行列の固有ベクトルも計算する。配列 <code>q</code> には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。 <code>icomp = 2</code> の場合、三重対角行列の固有値と固有ベクトルを計算する。
<code>qsiz</code>	INTEGER。 フル行列から三重対角行列への縮退に使用した直交 / ユニタリー行列の次元で、 $qsiz \geq n$ (実数型の場合。 <code>icompq = 1</code> の場合は $qsiz \geq n$)
<code>n</code>	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。

<i>d, e, rwork</i>	<p>REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: <i>d</i>(*) には三重対角行列の主対角を格納する。<i>d</i> の次元は、$\max(1, n)$ 以上でなければならない。 <i>e</i>(*) には三重対角行列の非対角成分を格納する。<i>e</i> の次元は、$\max(1, n-1)$ 以上でなければならない。 <i>rwork</i>(*) は複素型でのみ使用されるワークスペース配列である。<i>rwork</i> の次元は $(1 + 3n + 2n \lg(n) + 3n^2)$ 以上でなければならない、ここで $\lg(n) = 2^k \geq n$ となる最小の整数 <i>k</i>。</p>
<i>q, qstore</i>	<p>REAL (slaed0 の場合) DOUBLE PRECISION (dlaed0 の場合) COMPLEX (claed0 の場合) COMPLEX*16 (zlaed0 の場合) 配列: <i>q</i>(<i>ldq</i>, *), <i>qstore</i>(<i>ldqs</i>, *)。これらの配列の第 2 次元は $\max(1, n)$ 以上でなければならない。 実数型の場合: <i>icompq</i> = 0 ならば <i>q</i> は参照されない。 <i>icompq</i> = 1 の場合、<i>q</i> は、現時点で分解の対象となるフル行列の部分集合に対応する、フル行列から三重対角形式への縮退に使用する直交行列の列の部分集合を格納する。 <i>icompq</i> = 2 の場合、<i>q</i> は単位行列を格納する。 配列 <i>qstore</i> はワークスペース配列で、<i>icomp</i> = 1 の場合のみ参照される。更新行列乗算の実行時に固有ベクトル行列の部分を格納するために使用される。 複素数型の場合: <i>q</i> には列がユニタリーに直交している <i>qsiz</i> × <i>n</i> 行列を格納しなければならない。これは、フル密エルミート行列を (縮退可能な) 対称三重対角行列に縮退させるユニタリー行列の一部である。 配列 <i>qstore</i> は、更新行列乗算の実行時に固有ベクトル行列の部分を格納するためにワークスペースとして使用される。</p>
<i>ldq</i>	INTEGER。配列 <i>q</i> の第一次元。 $ldq \geq \max(1, n)$ 。
<i>ldqs</i>	INTEGER。配列 <i>qstore</i> の第 1 次元。 $ldqs \geq \max(1, n)$
<i>work</i>	<p>REAL (slaed0 の場合) DOUBLE PRECISION (dlaed0 の場合) ワークスペース配列で実数型でのみ使用される。 <i>icompq</i> = 0 または 1 の場合、<i>work</i> の次元は $(1 + 3n + 2n \lg(n) + 2n^2)$ 以上でなければならない、ここで $\lg(n) = 2^k \geq n$ となる最小の整数 <i>k</i> である。 <i>icompq</i> = 2 の場合、<i>work</i> の次元は $(4n + n^2)$ 以上でなければならない。</p>
<i>iwork</i>	<p>INTEGER。 ワークスペース配列。 実数型で <i>icomp</i> = 0 または 1 の場合、および複素型の場合、</p>

iwork の次元は $(6 + 6n + 5n \lg(n))$ 以上でなければならない。
実数型の場合は、*icompg* = 2 ならば *iwork* の次元は $(3 + 5n)$ 以上でなければならない。

出力パラメーター

<i>d</i>	昇順に並べられた固有値によって上書きされる。
<i>e</i>	配列を削除する。
<i>q</i>	<i>icompg</i> = 2 の場合、 <i>q</i> は三重対角行列の固有ベクトルで上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = <i>i</i> > 0 の場合、 <i>i</i> /(<i>n</i> +1) から $\text{mod}(i, n+1)$ の行と列からなる部分行列の操作中に、アルゴリズムが固有値の計算に失敗したことを示す。

?laed1

sstedc/dstedc で使用される。階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が三重対角の場合に使用される。

構文

```
call slaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info )
call dlaed1( n, d, q, ldq, indxq, rho, cutpnt, work, iwork, info )
```

説明

ルーチン ?laed1 は、階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。このルーチンは、三重対角行列のすべての固有値と固有ベクトルを必要とする固有問題に対してのみ使用される。[?laed7](#) は、フル対称行列の固有値のみを必要とする場合、または (三重対角形式に縮退された) 固有値と固有ベクトルを必要とする場合を取り扱う。

$$T = Q(\text{in}) (D(\text{in}) + \text{rho} * Z * Z') Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$z = Q'u$ で、*u* は *cutpnt* 番目と (*cutpnt* + 1) 番目の成分が 1 で残りがゼロの長さ *n* のベクトルである。元の行列の固有ベクトルは *Q* に格納され、固有値は *D* に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または *z* 内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン [?laed2](#) によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン [?laed4](#) ([?laed3](#) として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

入力パラメーター

<i>n</i>	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。
<i>d</i> , <i>q</i> , <i>work</i>	REAL (slaed1 の場合) DOUBLE PRECISION (dlaed1 の場合) 配列: <i>d</i> (*) には階数 1 摂動行列の固有値を格納する。 <i>d</i> の次元は、 $\max(1, n)$ 以上でなければならない。 <i>q</i> (<i>ldq</i> , *) には階数 1 摂動行列の固有ベクトルを格納する。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) は、ワークスペース配列、次元は $(4n+n^2)$ 以上。
<i>ldq</i>	INTEGER。配列 <i>q</i> の第一次元。 $ldq \geq \max(1, n)$ 。
<i>indxq</i>	INTEGER。配列、次元は (<i>n</i>)。 <i>d</i> に入っている 2 つの部分問題を昇順で別々にソートする置換。
<i>rho</i>	REAL (slaed1 の場合) DOUBLE PRECISION (dlaed1 の場合) 階数 1 更新を生成するために使用する劣対角エンタリー。
<i>cutpnt</i>	INTEGER。 先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq cutpnt \leq n/2$
<i>iwork</i>	INTEGER。ワークスペース配列、次元は ($4n$)。

出力パラメーター

<i>d</i>	修復された行列の固有値で上書きされる。
<i>q</i>	<i>q</i> は修復された三重対角行列の固有ベクトルで上書きされる。
<i>indxq</i>	ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、 <i>d</i> (<i>indxq</i> (<i>i</i> = 1, <i>n</i>)) は昇順となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、固有値は収束しなかったことを示す。

?laed2

sstedc/dstedc で使用される。固有値を併合し永年方程式を収縮させる。元の行列が三重対角の場合に使用される。

構文

```
call slaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx, indxc,
            indxp, coltyp, info )
call dlaed2( k, n, n1, d, q, ldq, indxq, rho, z, dlamda, w, q2, indx, indxc,
            indxp, coltyp, info )
```

説明

このルーチン ?laed2 は 2 つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の固有値が互いに近い場合、または、 z ベクトルに微小エントリーがある場合である。そのような場合ごとに、関連する永年方程式問題の次元は 1 だけ縮小される。

入力パラメーター

k	INTEGER。収縮されていない固有値の個数、および関連する永年方程式の次数 ($0 \leq k \leq n$)。
n	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。
$n1$	INTEGER。先頭の部分行列にある最後の固有値の位置で、 $\min(1, n) \leq n1 \leq n/2$
d, q, z	REAL (slaed2 の場合) DOUBLE PRECISION (dlaed2 の場合) 配列: $d(*)$ には結合対象となる 2 つの部分行列の固有値を格納する。 d の次元は、 $\max(1, n)$ 以上でなければならない。 $q(ldq, *)$ には、(1,1), (n1,n1) と (n1+1,n1+1), (n,n) の隅にある 2 つの平方ブロック内の 2 個の部分行列の固有ベクトルを格納する。 q の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 $z(*)$ には更新ベクトルを格納する (最初の部分固有ベクトル行列の最後の行と、2 番目の部分固有ベクトル行列の最初の行)。
ldq	INTEGER。配列 q の第一次元。 $ldq \geq \max(1, n)$ 。
$indxq$	INTEGER。配列、次元は (n)。 d に入っている 2 つの部分問題を昇順で別々にソートする置換。 この置換の後半にある成分は、それら値に加算された $n1$ を最初に持っていなければならない。
rho	REAL (slaed2 の場合) DOUBLE PRECISION (dlaed2 の場合) 今回再結合の対象となっている 2 つの部分行列を、そもそも分割する階数 1 切断に関連した非対角成分。

<i>indx, indxp</i>	INTEGER。 ワークスペース配列、次元はそれぞれ、(n)。 配列 <i>indx</i> には <i>dlambda</i> の内容を昇順にソートするために使用した置換を格納する。 配列 <i>indxp</i> には <i>d</i> の収縮された値を配列の終わりに配置するために使用した置換を格納する。 <i>indxp(1:k)</i> は収縮されていない <i>d</i> 値を指し、 <i>indxp(k+1:n)</i> は収縮された固有値を指す。
<i>coltyp</i>	INTEGER。ワークスペース配列、次元は (n)。 実行中に、 <i>q2</i> 行列中の次のタイプの列を示すラベルである。 1: 上半分のみゼロ 2: 密 3: 下半分のみ非ゼロ 4: 収縮
出力パラメーター	
<i>d</i>	<i>d</i> には後続の (n-k) 個の更新された固有値 (収縮された) が昇順で上書きされる。
<i>q</i>	<i>q</i> の最後の n-k 列には、後続の (n-k) 個の更新された固有値 (収縮された) が上書きされる。
<i>indxq</i>	壊される。
<i>rho</i>	?1aed3 が必要とした値に <i>rho</i> は変更されている。
<i>dlambda, w, q2</i>	REAL (s1aed2 の場合) DOUBLE PRECISION (d1aed2 の場合) 配列: <i>dlambda</i> (n), <i>w</i> (n), <i>q2</i> (n1 ² +(n-n1) ²) 配列 <i>dlambda</i> には最初の k 個の固有値のコピーが格納され、永年方程式を形成するために ?1aed3 で使用される。 配列 <i>w</i> には収縮更新された最終的な z ベクトルの最初の k 個の値が格納され、?1aed3 に渡される。 配列 <i>q2</i> には最初の k 個の固有ベクトルのコピーが格納され、新しい固有ベクトルを解くために ?1aed3 の行列乗算 (sgemm/dgemm) で使用される。
<i>indx</i>	INTEGER。配列、次元は (n)。 収縮された <i>q</i> 行列の列を 3 つのグループに整理するために使用された置換。第 1 のグループには n1 とその上のみ非ゼロの成分が格納され、第 2 のグループには n1 より下のみ非ゼロの成分が格納され、第 3 のグループは密である。
<i>coltyp</i>	<i>coltyp(i)</i> は、i = 1 から 4 のみに対するタイプ i の列数で上書きされる (タイプの定義については入力パラメーターの <i>coltyp</i> の説明を参照)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = -i の場合、i 番目のパラメーターの値が不正である。

?laed3

sstedc/dstedc で使用される。
 永年方程式の根を探し固有ベクトルを更新する。
 元の行列が三重対角の場合に使用される。

構文

```
call slaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s, info )
call dlaed3( k, n, n1, d, q, ldq, rho, dlamda, q2, indx, ctot, w, s, info )
```

説明

ルーチン ?laed3 は、1 と k の範囲に対して、 d 、 w 、 ρ の値によって定義されているとおり、永年方程式の根を探す。?laed4 を適切に呼び出し、ここで解かれる $k \times k$ 連立方程式の固有ベクトルの行列によって結合される固有方程式ペアから固有ベクトルの行列を乗算して、その固有ベクトルを更新する。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリーマシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了することが考えられるが、そのような前例はない。

入力パラメーター

k	INTEGER。?laed4 で解かれる有利関数の項目数 ($k \geq 0$)。
n	INTEGER。 q 行列の行と列の数。 $n \geq k$ (収縮で $v > k$ になる場合がある)。
$n1$	INTEGER。先頭の部分行列にある最後の固有値の位置で、 $\min(1, n) \leq n1 \leq n/2$ 。
q	REAL (slaed3 の場合) DOUBLE PRECISION (dlaed3 の場合) 配列 $q(ldq, *)$ 。 q の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 初期時には、この配列の最初の k 列はワークスペースとして使用される。
ldq	INTEGER。配列 q の第一次元。 $ldq \geq \max(1, n)$ 。
ρ	REAL (slaed3 の場合) DOUBLE PRECISION (dlaed3 の場合) 階数 1 の更新方程式にあるパラメーターの値。 $\rho \geq 0$ が必要。
$d\lambda, q2, w$	REAL (slaed3 の場合) DOUBLE PRECISION (dlaed3 の場合) 配列: $d\lambda(k), q2(ldq2, *), w(k)$ 配列 $d\lambda$ の最初の k 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。

配列 $q2$ の最初の k 列には、分割問題に対する収縮されていない固有ベクトルを格納する。 $q2$ の第 2 次元は $\max(1, n)$ 以上でなければならない。

配列 w の最初の k 個の成分には、収縮調整更新ベクトルの成分を格納する。

<i>indx</i>	INTEGER。配列、次元は (n) 。 収縮された q 行列の列を 3 つのグループに整理するために使用された置換 (<i>?laed2</i> 参照)。 <i>?laed4</i> によって求められた固有ベクトルの行は、行列乗算が行われる前に同様に置換されなければならない。
<i>ctot</i>	INTEGER。配列 A 次元は (4) 。 <i>indx</i> に記述されている、 q に含まれる列のタイプのそれぞれの合計。4 番目の列タイプは収縮された任意の列である。
<i>s</i>	REAL (<i>slaed3</i> の場合) DOUBLE PRECISION (<i>dlaed3</i> の場合) ワークスペース配列、次元は $(n1+1)*k$ 。 連立方程式を更新するために過去に蓄積された固有ベクトルで乗算される、修復された行列の固有ベクトルを格納する。

出力パラメーター

<i>d</i>	REAL (<i>slaed3</i> の場合) DOUBLE PRECISION (<i>dlaed3</i> の場合) 配列、次元は $\max(1, n)$ 以上。 $d(i)$ には $1 \leq i \leq k$ に対する更新された固有値が格納される。
<i>q</i>	q の列 1 から k は更新された固有ベクトルで上書きされる。
<i>dlamda</i>	前述のとおり、Cray X-MP、Cray Y-MP、Cray-2、または Cray C-90 では、ゼロに設定された最下位ビットを持っていると変更される場合がある。
<i>w</i>	削除される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - i の場合、 i 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、固有値は収束しなかったことを示す。

?laed4

sstedc/dstedc で使用される。
永年方程式の単一根を見つける。

構文

```
call slaed4(n, i, d, z, delta, rho, dlam, info )
call dlaed4(n, i, d, z, delta, rho, dlam, info )
```

説明

このサブルーチンは、成分が配列 d で与えられている対角行列に対する、対称階数 1 の更新の i 番目の更新された固有値を計算し、

ここで、 $i < j$ では $D(i) < D(j)$ 、

および $\rho > 0$ である。これは呼び出しルーチンによって配置され、一般性が失われることはない。階数 1 の更新された連立方程式はゆえに、

$$\text{diag}(D) + \rho * Z * \text{transpose}(Z).$$

Z のユークリッド・ノルムを 1 とする。

この方法は、単純補間の有利関数による永久方程式内の有利関数の近似を含む。

入力パラメーター

n	INTEGER。全配列の長さ。
i	INTEGER。計算される固有値のインデックスで、 $1 \leq i \leq n$
d, z	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 配列、次元はそれぞれ、 (n) 。 配列 d には元の固有値を格納する。それらは、 $i < j$ では $d(i) < d(j)$ のように順序どおりに並べられていると仮定される。 配列 z には更新ベクトル Z の成分を格納する。
ρ	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 対称更新式におけるスカラー。

出力パラメーター

δ	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 配列、次元は (n) 。 $n \neq 1$ の場合、 δ の j 番目の成分に $(d(j) - \lambda_i)$ が格納される。 $n = 1$ の場合、 $\delta(1) = 1$ 。ベクトル δ には固有ベクトルを構成するために必要な情報が格納される。
d_{lam}	REAL (slaed4 の場合) DOUBLE PRECISION (dlaed4 の場合) 計算された λ_i 、 i 番目の更新された固有値。
$info$	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = 1$ の場合、更新処理に失敗したことを示す。

?laed5

sstedc/dstedc で使用される。
 2×2 の永年方程式を解く。

構文

```
call slaed5(i, d, z, delta, rho, dlam )
call dlaed5(i, d, z, delta, rho, dlam )
```

説明

このサブルーチンは、 2×2 対角行列の対称階数 1 の更新の i 番目の固有値を計算する。

$\text{diag}(D) + \rho * Z * \text{transpose}(Z)$ 。

配列 D における対角成分は満たされるとみなされる。

ここで、 $i < j$ では $D(i) < D(j)$ 、

さらに、 $\rho > 0$ 、ベクトル Z のユークリッド・ノルムを 1 と仮定する。

入力パラメーター

i INTEGER。計算される固有値のインデックスで、
 $1 \leq i \leq 2$

d, z REAL (slaed5 の場合)
 DOUBLE PRECISION (dlaed5 の場合)
 配列、次元はそれぞれ、(2)。
 配列 d には元の固有値を格納する。 $d(1) < d(2)$ と仮定する。
 配列 z には更新ベクトルの成分を格納する。

ρ REAL (slaed5 の場合)
 DOUBLE PRECISION (dlaed5 の場合)
 対称更新式におけるスカラー。

出力パラメーター

δ REAL (slaed5 の場合)
 DOUBLE PRECISION (dlaed5 の場合)
 配列、次元は (2)。
 ベクトル δ には固有ベクトルを構成するために必要な情報が格納される。

$d\lambda$ REAL (slaed5 の場合)
 DOUBLE PRECISION (dlaed5 の場合)
 計算された λ_i 、 i 番目の更新された固有値。

?laed6

sstedc/dstedc で使用される。
永久方程式の解の中から Newton ステップの 1 つ
を計算する。

構文

```
call slaed6( kniter, orgati, rho, d, z, finit, tau, info )
call dlaed6( kniter, orgati, rho, d, z, finit, tau, info )
```

説明

このルーチンは次の式の正または負の根 (元にもっとも近い) を計算する。

$$f(x) = \text{rho} + \frac{z(1)}{d(1) - x} + \frac{z(2)}{d(2) - x} + \frac{z(3)}{d(3) - x}$$

orgati = .TRUE の場合、根は *d*(2) と *d*(3) の間にあると仮定される。それ以外では *d*(1) と *d*(2) の間にあると仮定される。
このルーチンは [?laed4](#) から必要に応じて呼び出される。多くの場合、根の探索は大きさにおいて最も小さいが、きわめて稀な状況ではないと考えられる。

入力パラメーター

<i>kniter</i>	INTEGER。 大きさに関して ?laed4 を参照。
<i>orgati</i>	LOGICAL。 <i>orgati</i> = .TRUE. の場合、必要となる根は <i>d</i> (2) と <i>d</i> (3) の間にあり、それ以外では <i>d</i> (1) と <i>d</i> (2) の間にある。詳細は ?laed4 を参照。
<i>rho</i>	REAL (<i>slaed6</i> の場合) DOUBLE PRECISION (<i>dlaed6</i> の場合) 上記の <i>f</i> (<i>x</i>) の式を参照。
<i>d</i> , <i>z</i>	REAL (<i>slaed6</i> の場合) DOUBLE PRECISION (<i>dlaed6</i> の場合) 配列、次元はそれぞれ、(3)。 <i>d</i> (1) < <i>d</i> (2) < <i>d</i> (3) を満たす配列 <i>d</i> 配列 <i>z</i> の各成分は正でなければならない。
<i>finit</i>	REAL (<i>slaed6</i> の場合) DOUBLE PRECISION (<i>dlaed6</i> の場合) ゼロにおける <i>f</i> (<i>x</i>) の値。このルーチン内部を評価した値よりも正確であること (そのようなことを望む場合)。

出力パラメーター

<i>tau</i>	REAL (s _{laed6} の場合) DOUBLE PRECISION (d _{laed6} の場合) <i>f(x)</i> に対する式の根。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、収束に失敗したことを示す。

?_{laed7}

?_{stedc} で使用される。

階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。元の行列が密の場合に使用される。

構文

```
call slaed7( icompg, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq,
            indxq, rho, cutpnt, qstore, qptr, prmptr, perm, givptr, givcol,
            givnum, work, iwork, info )
call dlaed7( icompg, n, qsiz, tlvls, curlvl, curpbm, d, q, ldq,
            indxq, rho, cutpnt, qstore, qptr, prmptr, perm, givptr, givcol,
            givnum, work, iwork, info )
call claed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
            indxq, qstore, qptr, prmptr, perm, givptr, givcol, givnum,
            work, rwork, iwork, info )
call zlaed7( n, cutpnt, qsiz, tlvls, curlvl, curpbm, d, q, ldq, rho,
            indxq, qstore, qptr, prmptr, perm, givptr, givcol, givnum,
            work, rwork, iwork, info )
```

説明

ルーチン ?_{laed7} は、階数 1 対称行列による更新後に対角行列の更新された固有連立方程式を計算する。このルーチンは、三重対角形式に縮退された密対称/エルミート行列のすべての固有値と、オプションで固有ベクトルを必要とする固有問題に対してのみ使用される。実数型では、s_{laed1}/d_{laed1} は、対称三重対角行列のすべての固有値と固有ベクトルが必要な場合を取り扱う。

$$T = Q(\text{in}) (D(\text{in}) + \rho * Z * Z') Q'(\text{in}) = Q(\text{out}) * D(\text{out}) * Q'(\text{out})$$

$Z = Q'u$ で、 u は *cutpnt* 番目と (*cutpnt* + 1) 番目の成分が 1 で残りがゼロの長さ n のベクトルである。元の行列の固有ベクトルは Q に格納され、固有値は D に格納される。アルゴリズムは次の 3 過程で構成される。

最初の過程では、固有値が複数ある場合、または Z 内にゼロがある場合、問題の大きさの収縮が行われる。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン s_{laed8}/d_{laed8} (実数型)、またはルーチン s_{laed2}/d_{laed2} (複素型) によって実行される。

次の過程では更新された固有値を計算する。これは、ルーチン ?1aed4 (?1aed3 または ?1aed9 として呼び出される) を介して永年方程式の根を見つけることによって行われる。このルーチンは現在の問題の固有ベクトルも計算する。

最後の過程では、更新された固有値を直接使用して更新された固有ベクトルを計算する。現在の問題に対する固有ベクトルは、全体問題から得られる固有ベクトルで乗算される。

入カパラメーター

<i>icompg</i>	INTEGER。実数型でのみ使用される。 <i>icompg</i> = 0 の場合、固有値のみ計算する。 <i>icompg</i> = 1 の場合、元の密対称行列の固有ベクトルも計算する。配列 <i>q</i> には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。
<i>n</i>	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。
<i>cutpnt</i>	INTEGER。先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq \text{cutpnt} \leq n$
<i>qsiz</i>	INTEGER。フル行列から三重対角行列への縮退に使用した直交/ユニタリ行列の次元で、 $qsiz \geq n$ (実数型の場合。 <i>icompg</i> = 1 の場合は $qsiz \geq n$)
<i>tlvls</i>	INTEGER。分割統治ツリー全体の併合レベルの合計数。
<i>curlvl</i>	INTEGER。併合ルーチン全体の現在のレベルで、 $0 \leq \text{curlvl} \leq \text{tlvls}$
<i>curpbm</i>	INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。
<i>d</i>	REAL (s1aed7/c1aed7 の場合) DOUBLE PRECISION (d1aed7/z1aed7 の場合) 配列、次元は $\max(1, n)$ 以上。 配列 <i>d</i> (*) には階数 1 置換行列の固有値を格納する。
<i>q, work</i>	REAL (s1aed7 の場合) DOUBLE PRECISION (d1aed7 の場合) COMPLEX (c1aed7 の場合) COMPLEX*16 (z1aed7 の場合) 配列 : <i>q</i> (1 <i>dq</i> , *) には階数 1 置換行列の固有ベクトルを格納する。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>work</i> (*) はワークスペース配列、次元は、実数の場合は $(3n+qsiz*n)$ 以上、複素数の場合は $(qsiz*n)$ 以上。
<i>ldq</i>	INTEGER。配列 <i>q</i> の第一次元。 $ldq \geq \max(1, n)$ 。
<i>rho</i>	REAL (s1aed7/c1aed7 の場合) DOUBLE PRECISION (d1aed7/z1aed7 の場合) 階数 1 更新を生成するために使用された部分対角成分。

<i>qstore</i>	REAL (slaed7/claed7 の場合) DOUBLE PRECISION (dlaed7/zlaed7 の場合) 配列、次元は (n^2+1) 。出力パラメーターとしても働く。 分割統治中に遭遇した部分行列の固有ベクトルを圧縮して格納する。 <i>qptr</i> は部分行列の始まりを指す。
<i>qptr</i>	INTEGER。配列、次元は $(n+2)$ 。出力パラメーターとしても働く。 <i>qstore</i> に格納されている部分行列の始まりを指すインデックス・リスト。部分行列は、分割統治ツリーの左下を先頭にして、左から右へ、下から上へ番号が振られる。
<i>prmptr, perm, givptr</i>	INTEGER。配列、次元はそれぞれ、 $(n \lg n)$ 。 配列 <i>prmptr</i> (*) には、あるレベルの置換が <i>perm</i> 内のどこに格納されているかを示すポインターリストを格納する。 <i>prmptr</i> (i+1) - <i>prmptr</i> (i) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。 配列 <i>perm</i> (*) には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。 配列 <i>givptr</i> (*) には、あるレベルの Givens 回転が <i>givcol</i> 内のどこに格納されているかを示すポインターリストを格納する。 <i>givptr</i> (i+1) - <i>givptr</i> (i) は Givens 回転の回数を示す。
<i>givcol</i>	INTEGER。配列、次元は $(2, n \lg n)$ 。 それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL (slaed7/claed7 の場合) DOUBLE PRECISION (dlaed7/zlaed7 の場合) 配列、次元は $(2, n \lg n)$ 。 それぞれの数は対応する Givens 回転で使用される <i>S</i> 値を示す。
<i>iwork</i>	INTEGER。ワークスペース配列、次元は $(4n)$ 。
<i>rwork</i>	REAL (claed7 の場合) DOUBLE PRECISION (zlaed7 の場合) ワークスペース配列、次元は $(3n+2qsiz*n)$ 。複素数型でのみ使用される。

出力パラメーター

<i>d</i>	修復された行列の固有値で上書きされる。
<i>q</i>	<i>q</i> は修復された三重対角行列の固有ベクトルで上書きされる。
<i>indxq</i>	INTEGER。配列、次元は (n) 。 ソートされた順に部分問題を再統合する置換で上書きされ、すなわち、 $d(\text{indxq}(i=1, n))$ は昇順となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、固有値は収束しなかったことを示す。

?laed8

?stedc で使用される。

固有値を併合し永年方程式を収縮させる。元の行列が密の場合に使用される。

構文

```
call slaed8(  icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,
             dlamda, q2, ldq2, w, perm, givptr, givcol, givnum, indx,
             info )
call dlaed8(  icompq, k, n, qsiz, d, q, ldq, indxq, rho, cutpnt, z,
             dlamda, q2, ldq2, w, perm, givptr, givcol, givnum, indx,
             info )
call claed8(  k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2,
             ldq2, w, indxp, indx, indxq, perm, givptr, givcol, givnum,
             info )
call zlaed8(  k, n, qsiz, q, ldq, d, rho, cutpnt, z, dlamda, q2,
             ldq2, w, indxp, indx, indxq, perm, givptr, givcol, givnum,
             info )
```

説明

このルーチンは2つの固有値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには2つの場合がある。2個以上の固有値が互いに近い場合、または、**z** ベクトルに微小エントリーがある場合である。そのような場合ごとに、関連する永年方程式問題の次元は1だけ縮小される。

入力パラメーター

<i>icompq</i>	INTEGER。実数型でのみ使用される。 <i>icompq</i> = 0 の場合、固有値のみ計算する。 <i>icompq</i> = 1 の場合、元の密対称行列の固有ベクトルも計算する。配列 <i>q</i> には、元の行列から三重対角形式への縮退に使用した直交行列を格納しなければならない。
<i>n</i>	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。
<i>cutpnt</i>	INTEGER。先頭の部分行列にある最後の固有値の位置。 $\min(1, n) \leq \text{cutpnt} \leq n$
<i>qsiz</i>	INTEGER。フル行列から三重対角行列への縮退に使用した直交 / ユニタリー行列の次元で、 $qsiz \geq n$ (実数型の場合。 <i>icompq</i> = 1 の場合は $qsiz \geq n$)
<i>d, z</i>	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 配列、次元はそれぞれ、 $\max(1, n)$ 以上。 配列 <i>d</i> (*) には結合する2つの部分行列の固有値を格納する。 <i>z</i> (*) には更新ベクトルを格納する (最初の部分固有ベクトル行列の最後の行と、2番目の部分固有ベクトル行列の最初の行)。 <i>z</i> の内容は更新過程で壊される。

<i>q</i>	REAL (slaed8 の場合) DOUBLE PRECISION (dlaed8 の場合) COMPLEX (claed8 の場合) COMPLEX*16 (zlaed8 の場合) 配列 $q(ldq, *)$ 。 q の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 q には、他の部分的に解かれた固有連立方程式との行列乗算によって、すでに更新された部分的に解かれた連立方程式の固有ベクトルを格納する。 実数型の場合は、 $icompq = 0$ ならば q は参照されない。
<i>ldq</i>	INTEGER。配列 q の第 1 次元。 $ldq \geq \max(1, n)$ 。
<i>ldq2</i>	INTEGER。出力配列 $q2$ の第 1 次元のサイズ。 $ldq2 \geq \max(1, n)$
<i>indxq</i>	INTEGER。配列、次元は (n) 。 d に入っている 2 つの部分問題を昇順で別々にソートする置換。 この置換の後半にある成分は、精度を得るために、それら値に加算された <i>cutpnt</i> を最初に持っていなければならない。
<i>rho</i>	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 今回再結合の対象となっている 2 つの部分行列を、そもそも分割する階数 1 切断に関連した非対角成分。
出力パラメーター	
<i>k</i>	INTEGER。非収縮の固有値の数、関連永年方程式の次数。
<i>d</i>	後続の $(n-k)$ 個の更新された固有値 (収縮された) が昇順で上書きされる。
<i>q</i>	q の最後の $n-k$ 列には、後続の $(n-k)$ 個の更新された固有値 (収縮された) が上書きされる。
<i>rho</i>	?1aed3 が必要とした値に ρ は変更されている。
<i>dlambda, w</i>	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 配列、次元はそれぞれ、 (n) 配列 $dlambda(*)$ には最初の k 個の固有値のコピーが格納され、永年方程式を形成するために ?1aed3 で使用される。 配列 $w(*)$ は収縮更新された最終的な z ベクトルの最初の k 個の値を保持し、?1aed3 に渡される。
<i>q2</i>	REAL (slaed8 の場合) DOUBLE PRECISION (dlaed8 の場合) COMPLEX (claed8 の場合) COMPLEX*16 (zlaed8 の場合) 配列 $q2(ldq2, *)$ 。 $q2$ の第 2 次元は $\max(1, n)$ 以上でなければならない。 新しい固有値の更新のために、slaed7/dlaed7 の行列乗算

	(sgemm/dgemm) で使用される最初の k 個の固有ベクトルのコピーが格納される。 実数型の場合は、 $icompg = 0$ ならば $q2$ は参照されない。
<i>indxp, indx</i>	INTEGER。ワークスペース配列、次元はそれぞれ、 (n) 。 配列 <i>indxp</i> (*) には、 d の収縮した値を配列の終わりに配置するために使用した置換を格納する。 <i>indxp</i> (1: k) は収縮されていない d 値を指し、 <i>indxp</i> ($k+1:n$) は収縮された固有値を指す。 配列 <i>indx</i> (*) には d の内容を昇順にソートするために使用した置換を格納する。
<i>perm</i>	INTEGER。配列、次元は (n) 。 各固有ブロックに適用される (収縮とソートによる) 置換が格納される。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。
<i>givcol</i>	INTEGER。配列、次元は $(2, n)$ 。 それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL (slaed8/claed8 の場合) DOUBLE PRECISION (dlaed8/zlaed8 の場合) 配列、次元は $(2, n)$ 。 それぞれの数は対応する Givens 回転で使用される S 値を示す。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。

?laed9

sstedc/dstedc で使用される。
永年方程式の根を探し固有ベクトルを更新する。
元の行列が密の場合に使用される。

構文

```
call slaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds, info )
call dlaed9( k, kstart, kstop, n, d, q, ldq, rho, dlamda, w, s, lds, info )
```

説明

ルーチン ?laed3 は、 $kstart$ と $kstop$ の範囲で、 d 、 w 、 rho の値によって定義されているとおり、永年方程式の根を探す。slaed4/dlaed4 への適切な呼び出しを行い、続いて、 z ベクトルの次のレベルの計算で用いる固有ベクトルの新しい行列を格納する。

入力パラメーター

k INTEGER。slaed4/dlaed4 で解かれる有利関数の項目数 ($k \geq 0$)。

<i>kstart, kstop</i>	INTEGER。更新された固有値 $\lambda(i)$ 、 $kstart \leq i \leq kstop$ で計算される。 $1 \leq kstart \leq kstop \leq k$
<i>n</i>	INTEGER。Q 行列の行と列の数。 $n \geq k$ (収縮で $n > k$ になる場合がある)。
<i>q</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) ワークスペース配列、次元は (<i>ldq</i> , *)。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldq</i>	INTEGER。配列 <i>q</i> の第 1 次元。 $ldq \geq \max(1, n)$ 。
<i>rho</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 階数 1 の更新方程式にあるパラメーターの値。 $\rho \geq 0$ が必要。
<i>dlambda, w</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元はそれぞれ (<i>k</i>)。 配列 <i>dlambda</i> (*) の最初の <i>k</i> 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。 配列 <i>w</i> (*) の最初の <i>k</i> 個の成分には、収縮調整更新ベクトルの成分を格納する。
<i>lds</i>	INTEGER。出力配列 <i>s</i> の第 1 次元のサイズ。 $lds \geq \max(1, k)$

出力パラメーター

<i>d</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元は (<i>n</i>)。 <i>d</i> (<i>i</i>) には $kstart \leq i \leq kstop$ に対する更新された固有値が格納される。
<i>s</i>	REAL (slaed9 の場合) DOUBLE PRECISION (dlaed9 の場合) 配列、次元は (<i>lds</i> , *)。 <i>s</i> の第 2 次元は $\max(1, k)$ 以上でなければならない。 続く <i>z</i> ベクトル計算のために格納され、また、連立方程式の更新のために過去に蓄積された固有ベクトルで乗算される、修復された行列の固有ベクトルが格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。 <i>info</i> = 1 の場合、固有値は収束しなかったことを示す。

?laeda

?stedc で使用される。対角行列の階数1更新を決定するZベクトルを計算する。元の行列が密の場合に使用される。

構文

```
call slaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
            givnum, q, qptr, z, ztemp, info )
call dlaeda( n, tlvls, curlvl, curpbm, prmptr, perm, givptr, givcol,
            givnum, q, qptr, z, ztemp, info )
```

説明

?laeda ルーチンは、*curpbm* 番目問題に対する *tlvls* ステップを用いて、併合過程の *curlvl* 番目のステップ中の併合ステップに対応した *z* ベクトルを計算する。

入カパラメーター

<i>n</i>	INTEGER。対称三重対角行列の次元 ($n \geq 0$)。
<i>tlvls</i>	INTEGER。分割統治ツリー全体の併合レベルの合計数。
<i>curlvl</i>	INTEGER。併合ルーチン全体の現在のレベルで、 $0 \leq \text{curlvl} \leq \text{tlvls}$
<i>curpbm</i>	INTEGER。併合ルーチン全体の現在のレベルにおける現在の問題 (左上から右下に向かって計数)。
<i>prmptr, perm, givptr</i>	INTEGER。配列、次元はそれぞれ、($n \lg n$)。 配列 <i>prmptr</i> (*) には、あるレベルの置換が <i>perm</i> 内のどこに格納されているかを示すポインターリストを格納する。 <i>prmptr</i> (<i>i</i> +1) - <i>prmptr</i> (<i>i</i>) は置換のサイズを表し、また収縮されていない全問題の大きさも表す。 配列 <i>perm</i> (*) には各固有ブロックに適用される (収縮とソートによる) 置換を格納する。
<i>givcol</i>	INTEGER。配列、次元は ($2, n \lg n$)。 それぞれの数のペアは Givens 回転で対象となる列ペアを示す。
<i>givnum</i>	REAL (<i>slaeda</i> の場合) DOUBLE PRECISION (<i>dlaeda</i> の場合) 配列、次元は ($2, n \lg n$)。 それぞれの数は対応する Givens 回転で使用される <i>S</i> 値を示す。

<i>q</i>	REAL (slaeda の場合) DOUBLE PRECISION (dlaeda の場合) 配列、次元は (n^2) 。 以前のレベルから得られる平方固有ブロックを格納する。ブ ロックの開始位置は <i>qptr</i> で与えられる。
<i>qptr</i>	INTEGER。配列、次元は $(n+2)$ 。固有ブロックが <i>q</i> のどこに格納 されているかを示すポインタのリストを格納する。 $\text{sqrt}(qptr(i+1) - qptr(i))$ はブロックの大きさを示す。
<i>ztemp</i>	REAL (slaeda の場合) DOUBLE PRECISION (dlaeda の場合) ワークスペース配列、次元は (n) 。

出力パラメーター

<i>z</i>	REAL (slaeda の場合) DOUBLE PRECISION (dlaeda の場合) 配列、次元は (n) 。更新ベクトルが格納される (最初の部分固有 ベクトル行列の最後の行と、2 番目の部分固有ベクトル行列の最 初の行)。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正である。

?laein

上 Hessenberg 行列の指定された右または左固有ベ
クトルを逆反復法を用いて計算する。

構文

```
call slaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work, eps3,
  smlnum, bignum, info )
call dlaein( rightv, noinit, n, h, ldh, wr, wi, vr, vi, b, ldb, work, eps3,
  smlnum, bignum, info )
call claein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3, smlnum,
  info )
call zlaein( rightv, noinit, n, h, ldh, w, v, b, ldb, rwork, eps3, smlnum,
  info )
```

説明

?laein ルーチンは、実数の上 Hessenberg 行列 H (実数型 slaein/dlaein) の固有値 (*wr*,
wi)、または複素の上 Hessenberg 行列 H (複素型 claein/zlaein) の固有値 *w* に対応した
右または左固有ベクトルを逆反復法を使用して探す。

入カパラメーター

<i>rightv</i>	LOGICAL。 <i>rightv</i> = .TRUE. の場合、右固有ベクトルを計算する。 <i>rightv</i> = .FALSE. の場合、左固有ベクトルを計算する。
<i>noinit</i>	LOGICAL。 <i>noinit</i> = .TRUE. の場合、(<i>vr</i> , <i>vi</i>) または <i>v</i> (複素型) において初期ベクトルは与えられない。 <i>noinit</i> = .FALSE. の場合、(<i>vr</i> , <i>vi</i>) または <i>v</i> (複素型) において初期ベクトルが与えられる。
<i>n</i>	INTEGER。行列 <i>H</i> の次数 ($n \geq 0$)。
<i>h</i>	REAL (<i>slaein</i> の場合) DOUBLE PRECISION (<i>dlaein</i> の場合) COMPLEX (<i>claein</i> の場合) COMPLEX*16 (<i>zlaein</i> の場合) 配列 <i>h</i> (<i>ldh</i> , *)。 <i>h</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。Hessenberg 上行列 <i>H</i> が格納される。
<i>ldh</i>	INTEGER。配列 <i>h</i> の第 1 次元。 $ldh \geq \max(1, n)$
<i>wr, wi</i>	REAL (<i>slaein</i> の場合) DOUBLE PRECISION (<i>dlaein</i> の場合) その対応する右または左の固有ベクトルが、計算対象となっている <i>H</i> の固有値の実数部分および虚数部分 (実数型ルーチン)。
<i>w</i>	COMPLEX (<i>claein</i> の場合) COMPLEX*16 (<i>zlaein</i> の場合) その対応する右または左の固有ベクトルが計算対象となっている <i>H</i> の固有値 (複素型ルーチン)。
<i>vr, vi</i>	REAL (<i>slaein</i> の場合) DOUBLE PRECISION (<i>dlaein</i> の場合) 配列、次元はそれぞれ、(<i>n</i>)。実数型でのみ使用される。 <i>noinit</i> = .FALSE. かつ <i>wi</i> = 0.0 の場合、実数固有値 <i>wr</i> を使った逆反復のために、 <i>vr</i> には実数開始ベクトルを格納しなければならない。 <i>noinit</i> = .FALSE. かつ <i>wi</i> \neq 0.0 の場合、複素固有値 (<i>wr</i> , <i>wi</i>) を使った逆反復のために、 <i>vr</i> と <i>vi</i> には複素開始ベクトルの実数と虚数部分を格納しなければならない。それ以外の場合は <i>vr</i> と <i>vi</i> を設定する必要はない。
<i>v</i>	COMPLEX (<i>claein</i> の場合) COMPLEX*16 (<i>zlaein</i> の場合) 配列、次元は (<i>n</i>)。複素型でのみ使用される。 <i>noinit</i> = .FALSE. 場合、 <i>v</i> には逆反復のために開始ベクトルを格納しなければならない。それ以外の場合は <i>v</i> を設定する必要はない。
<i>b</i>	REAL (<i>slaein</i> の場合) DOUBLE PRECISION (<i>dlaein</i> の場合) COMPLEX (<i>claein</i> の場合)

	COMPLEX*16 (zlaein の場合) ワークスペース配列 $b(l\text{db}, *)$ 。 b の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldb</i>	INTEGER。配列 b の第 1 次元。 実数型は $ldb \geq n+1$ 複素型は $ldb \geq \max(1, n)$
<i>work</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) ワークスペース配列、次元は (n) 。実数型でのみ使用される。
<i>rwork</i>	REAL (claein の場合) DOUBLE PRECISION (zlaein の場合) ワークスペース配列、次元は (n) 。複素型でのみ使用される。
<i>eps3, smlnum</i>	REAL (slaein/claein の場合) DOUBLE PRECISION (dlaein/zlaein の場合) <i>eps3</i> は小さなマシン依存値で、値が近い固有値を摂動するため、およびゼロピットを交換するために使われる。 <i>smlnum</i> はマシン依存値でアンダーフローしきい値に近い。
<i>bignum</i>	REAL (slaein の場合) DOUBLE PRECISION (dlaein の場合) <i>bignum</i> はマシン依存値でオーバーフローしきい値に近い。実数型でのみ使用される。

出力パラメーター

<i>vr, vi</i>	$w_i = 0.0$ (実数固有値) の場合、 vr は計算された実数固有ベクトルで上書きされ、 $w_i \neq 0.0$ (複素固有値) の場合、 vr と vi は計算された複素固有ベクトルの実数部と虚数部で上書きされる。固有ベクトルは最大の大きさの成分が大きさ 1 を持つように正規化される。ここで、複素値 (x, y) の大きさは $ x + y $ として求められる。 $w_i = 0.0$ の場合、 vi は参照されない。
<i>v</i>	v は、最大の大きさの成分が大きさ 1 を持つように正規化された、計算された固有ベクトルで上書きされる。ここで複素値 (x, y) の大きさは $ x + y $ として求められる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、逆反復は収束しなかったことを示す。実数型の場合、 vr は最後の反復が設定され、 vi も $w_i \neq 0.0$ の場合と同じである。複素型の場合、 v には最後の反復が設定される。

?laev2

2×2 の対称/エルミート行列の固有値と固有ベクトルを計算する。

構文

```
call slaev2( a, b, c, rt1, rt2, cs1, sn1 )
call dlaev2( a, b, c, rt1, rt2, cs1, sn1 )
call claev2( a, b, c, rt1, rt2, cs1, sn1 )
call zlaev2( a, b, c, rt1, rt2, cs1, sn1 )
```

説明

このルーチンは、固有ベクトルの行列のノルムがしきい値より大きければ、次の 2×2 対称行列の固有分解を実行する。

$$\begin{bmatrix} a & b \\ b & c \end{bmatrix} \text{ (slaev2/dlaev2 の場合)、またはエルミート行列 } \begin{bmatrix} a & b \\ \text{conjg}(b) & c \end{bmatrix}$$

(claev2/zlaev2 の場合)

rt1 は大きい方の絶対値の固有値で上書きされ、rt2 は小さい方の絶対値の固有値で上書きされ、(cs1, sn1) は rt1 に対する単位右固有ベクトルで次の分解を与える。

$$\begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ b & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -sn1 \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(slaev2/dlaev2 の場合)、

または

$$\begin{bmatrix} cs1 & \text{conjg}(sn1) \\ -sn1 & cs1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ \text{conjg}(b) & c \end{bmatrix} \cdot \begin{bmatrix} cs1 & -\text{conjg}(sn1) \\ sn1 & cs1 \end{bmatrix} = \begin{bmatrix} rt1 & 0 \\ 0 & rt2 \end{bmatrix}$$

(claev2/zlaev2 の場合)

入力パラメーター

a, b, c	REAL (slaev2 の場合)
	DOUBLE PRECISION (dlaev2 の場合)
	COMPLEX (claev2 の場合)
	COMPLEX*16 (zlaev2 の場合)
	入力行列の成分。

出力パラメーター

<i>rt1, rt2</i>	REAL (slaev2/claev2 の場合) DOUBLE PRECISION (dlaev2/zlaev2 の場合) それぞれ、大きい方および小さい方の値の固有値。
<i>cs1</i>	REAL (slaev2/claev2 の場合) DOUBLE PRECISION (dlaev2/zlaev2 の場合)
<i>sn1</i>	REAL (slaev2 の場合) DOUBLE PRECISION (dlaev2 の場合) COMPLEX (claev2 の場合) COMPLEX*16 (zlaev2 の場合) ベクトル (<i>cs1, sn1</i>) は <i>rt1</i> に対する単位右固有ベクトル。

アプリケーション・ノート

rt1 はオーバーフロー/アンダーフローが起こらなければ、わずかな ulp (最小位置単位) に対して正確である。*rt2* は行列式 $a*c-b*b$ で、大幅な桁落ちがある場合に不正確となる可能性がある。*rt2* を精度高く計算するには、あらゆる場合で高精度または適切な丸めまたは適切な切捨て計算が必要になる。*cs1* と *sn1* はオーバーフロー/アンダーフローが起こらなければわずかな ulp に対して正確である。オーバーフローは、*rt1* がオーバーフローに係数 5 を掛けた値の範囲内にあるときに起こり得る。アンダーフローは入力データがゼロか *underflow_threshold* / *macheps* を超えた場合には影響とはならない。

?laexc

Schur 標準形となっている実数の上準三角行列の隣接対角ブロックを、直交相似変換によって交換する。

構文

```
call slaexc( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
call dlaexc( wantq, n, t, ldt, q, ldq, j1, n1, n2, work, info )
```

説明

このルーチンは、上準三角行列 T にある次数 1 または 2 の隣接対角ブロック T_{11} と T_{22} を、直交相似変換によって交換する。

T は Schur 標準形でなければならず、すなわち、 1×1 と 2×2 の対角ブロックを持つブロック上三角である。それぞれの 2×2 対角ブロックは、対角成分が等しく、かつ非対角成分が逆の符号を持つ。

入力パラメーター

<i>wantq</i>	LOGICAL。 <i>wantq</i> = .TRUE. の場合、行列 Q 内の変換を蓄積する。 <i>wantq</i> = .FALSE. の場合、変換を蓄積しない。
<i>n</i>	INTEGER。行列 T の次数 ($n \geq 0$)。

<i>t</i> , <i>q</i>	REAL (slaexc の場合) DOUBLE PRECISION (dlaexc の場合) 配列: <i>t</i> (<i>ldt</i> ,*) には、上準三角行列 <i>T</i> を Shur 標準形で格納する。 <i>t</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。 <i>wantq</i> = .TRUE. の場合、 <i>q</i> (<i>ldq</i> , *) には直交行列 <i>Q</i> を格納する。 <i>wantq</i> = .FALSE. の場合、 <i>q</i> は参照されない。 <i>q</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>ldt</i>	INTEGER。 <i>t</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldq</i>	INTEGER。 <i>q</i> の第 1 次元。 <i>wantq</i> = .FALSE. の場合、 <i>ldq</i> ≥ 1 <i>wantq</i> = .TRUE. の場合、 <i>ldq</i> $\geq \max(1, n)$ 。
<i>j1</i>	INTEGER。 第 1 のブロック <i>T</i> ₁₁ の第 1 行のインデックス。
<i>n1</i>	INTEGER。 第 1 のブロック <i>T</i> ₁₁ の次数 (<i>n1</i> = 0、1、または 2)。
<i>n2</i>	INTEGER。 第 2 のブロック <i>T</i> ₂₂ の次数 (<i>n2</i> = 0、1、または 2)。
<i>work</i>	REAL (slaexc の場合) DOUBLE PRECISION (dlaexc の場合) ワークスペース配列、次元は (<i>n</i>)。

出力パラメーター

<i>t</i>	schur 標準形を持つ更新された行列 <i>T</i> で上書きされる。
<i>q</i>	<i>wantq</i> = .TRUE. の場合、更新された行列 <i>Q</i> で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、変換された行列 <i>T</i> は Schur 形式に遠すぎたことを示す。ブロックは交換されず、また <i>T</i> と <i>Q</i> は変更されない。

?lag2

2 × 2 一般固有値問題の固有値を、オーバーフロー/アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。

構文

```
call slag2( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
call dlag2( a, lda, b, ldb, safmin, scale1, scale2, wr1, wr2, wi )
```

説明

このルーチンは、2 × 2 一般固有値問題 $A - wB$ の固有値を、オーバーフロー/アンダーフローが発生しないように必要に応じてスケーリングを行い計算する。スケール係数 *s* によって固有値の式は次のように変更される。

$$sA - wB$$

s は、 w 、 wB 、 sA がオーバーフローを起こさないように、かつ可能ならばアンダーフローも起こさないように選ばれた非負のスケール係数である。

入力パラメーター

a, b REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
配列:
 $a(lda, 2)$ には 2×2 の行列 A を格納する。この 1- ノルムは $1/safmin$ よりも小さいとする。 $\sqrt{safmin} \cdot \text{norm}(A)$ より小さい成分はゼロとして取り扱われる。
 $b(l db, 2)$ には 2×2 上三角行列 B を格納する。この 1- ノルムは $1/safmin$ よりも小さいとする。対角は B の (絶対値で) 最大成分の \sqrt{safmin} 倍以上でなければならない。もし対角がこれよりも小さい場合は、対角が変わって $\pm \sqrt{safmin}$ が使用される。

lda INTEGER。 a の第 1 次元。 $lda \geq 2$

ldb INTEGER。 b の第 1 次元。 $ldb \geq 2$

$safmin$ REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
 $1/safmin$ をオーバーフローさせない最も小さな正の値。(これは常に $?lamch('S')$ でなければならない。 $?lamch$ を頻繁に呼び出すのを避ける引数である。)

出力パラメーター

$scale1$ REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
第 1 の固有値を定義する固有値の式で、オーバーフロー/アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、固有値は $(wr1 \pm wi i) / scale1$ (マシンの指数範囲を超えるときがある) で、 $scale1 = scale2$ となり、 $scale1$ は常に正となる。
固有値が実数の場合、第 1 (実数) の固有値は $wr1 / scale1$ となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が十分に大きい場合、 $scale1$ はゼロかアンダーフローしきい値未満になることがある。

$scale2$ REAL (slag2 の場合)
DOUBLE PRECISION (dlag2 の場合)
第 2 の固有値を定義する固有値の式で、オーバーフロー/アンダーフローを防ぐために使用されたスケール係数。固有値が複素の場合、 $scale1 = scale2$ となる。固有値が実数の場合、第 2 (実数) の固有値は $wr2 / scale2$ となるが、これはオーバーフローかアンダーフローを起こすときがあり、実際、正確な固有値が十分に大きいと、 $scale2$ はゼロかアンダーフローしきい値未満になることがある。

<i>wr1</i>	REAL (slag2 の場合) DOUBLE PRECISION (dlag2 の場合) 固有値が実数の場合、 <i>wr1</i> は AB^{-1} の (2, 2) 成分に最も近い固有値の <i>scale1</i> 倍となる。固有値が複素の場合、 <i>wr1</i> は固有値の実数部の <i>scale1</i> 倍となり $wr1 = wr2$ である。
<i>wr2</i>	REAL (slag2 の場合) DOUBLE PRECISION (dlag2 の場合) 固有値が実数の場合、 <i>wr2</i> はその他の固有値の <i>scale2</i> 倍になる。固有値が複素の場合、 <i>wr1</i> は固有値の実数部の <i>scale1</i> 倍となり $wr1 = wr2$ である。
<i>wi</i>	REAL (slag2 の場合) DOUBLE PRECISION (dlag2 の場合) 固有値が実数の場合、 <i>wi</i> はゼロである。固有値が複素の場合、 <i>wi</i> は固有値の虚数部の <i>scale1</i> 倍となる。 <i>wi</i> は常に非負である。

?lags2

2×2 直交行列 *U*、*V*、*Q* を計算し、変換された *A* と *B* の行が平行であるような *A* と *B* にそれらを適用する。

構文

```
call slags2( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq, snq )
call dlags2( upper, a1, a2, a3, b1, b2, b3, csu, snu, csv, snv, csq, snq )
```

説明

このルーチンは 2×2 直交行列 *U*、*V*、*Q* を計算し、*upper* = .TRUE. であれば、

$$U' * A * Q = U' * \begin{bmatrix} A_1 & A_2 \\ 0 & A_3 \end{bmatrix} * Q = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix}$$

および

$$V' * B * Q = V' * \begin{bmatrix} B_1 & B_2 \\ 0 & B_3 \end{bmatrix} * Q = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix}$$

または *upper* = .FALSE. であれば、

$$U' * A * Q = U' * \begin{bmatrix} A_1 & 0 \\ A_2 & A_3 \end{bmatrix} * Q = \begin{bmatrix} x & x \\ 0 & x \end{bmatrix}$$

および

$$V' * B * Q = V' * \begin{bmatrix} B_1 & 0 \\ B_2 & B_3 \end{bmatrix} * Q = \begin{bmatrix} x & x \\ 0 & x \end{bmatrix}$$

変換された A と B の行は平行で、ここで

$$U = \begin{bmatrix} csu & snu \\ -snu & csu \end{bmatrix}, V = \begin{bmatrix} csv & snv \\ -snv & csv \end{bmatrix}, Q = \begin{bmatrix} csq & snq \\ -snq & csq \end{bmatrix}$$

Z' は Z の転置を表わす。

入力パラメーター

<i>upper</i>	LOGICAL。 <i>upper</i> = .TRUE. の場合、入力行列 A と B は上三角である。 <i>upper</i> = .FALSE. の場合、入力行列 A と B は下三角である。
<i>a1, a2, a3</i>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) <i>a1, a2, a3</i> には入力 2×2 上(下)三角行列 A の成分を格納する。
<i>b1, b2, b3</i>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) <i>b1, b2, b3</i> には入力 2×2 上(下)三角行列 B の成分を格納する。

出力パラメーター

<i>csu, snu</i>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) 目的の直交行列 U 。
<i>csv, snv</i>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) 目的の直交行列 V 。
<i>csq, snq</i>	REAL (slags2 の場合) DOUBLE PRECISION (dlags2 の場合) 目的の直交行列 Q 。

?lagtf

行交換を伴なう部分ピボット演算を用いて行列 $T-\lambda I$ の LU 因子分解を計算する。 T は一般三重対角行列、 λ はスカラー。

構文

```
call slagtf( n, a, lambda, b, c, tol, d, in, info )
call dlagtf( n, a, lambda, b, c, tol, d, in, info )
```

説明

このルーチンは次のように行列 ($T - \lambda I$) を因子分解する。 T は $n \times n$ 三重対角行列、 λ はスカラーである。

$$T - \lambda I = PLU$$

P は置換行列、 L は非ゼロの劣対角成分を列あたり多くても 1 つしか持たない単位下三重対角行列、 U は非ゼロの優対角成分を列あたり多くても 1 つしか持たない上三重対角行列である。因子分解は部分ピボット演算と黙示的な行スケールを用いたガウス消去によって実行される。 T の固有ベクトルを逆反復によって取得するために、 ?lagtf と ?lagts が使用できるようにパラメーター λ はルーチンに格納されている。

入力パラメーター

n INTEGER。行列 T の次数 ($n \geq 0$)。

a, b, c REAL (slagtf の場合)
DOUBLE PRECISION (dlagtf の場合)
配列、次元は $a(n), b(n-1), c(n-1)$:
 $a(*)$ には行列 T の対角成分を格納しなければならない。
 $b(*)$ には行列 T の $(n-1)$ 個の優対角成分を格納しなければならない。
 $c(*)$ には行列 T の $(n-1)$ 個の劣対角成分を格納しなければならない。

tol REAL (slagtf の場合)
DOUBLE PRECISION (dlagtf の場合)
行列 ($T - \lambda I$) がほとんど特異値かどうかを示すために使われる相対許容値。
 tol は通常、 T の成分におけるおおよその最大相対誤差として選択される。例えば、 T の成分がおおよそ 4 桁の有効数字で正しい場合、 tol はおおよそ 5×10^{-4} に設定しなければならない。 tol が相対マシン精度 eps よりも小さく与えられた場合は tol の代わりに eps の値が使われる。

出力パラメーター

a T を因子分解した上三角行列 U の n 個の対角成分で上書きされる。

b T を因子分解した行列 U の $n-1$ 個の優対角成分で上書きされる。

<i>c</i>	<i>T</i> を因子分解した行列 <i>L</i> の <i>n</i> -1 個の劣対角成分で上書きされる。
<i>d</i>	REAL (slagtf の場合) DOUBLE PRECISION (dlagtf の場合) 配列、次元は (<i>n</i> -2)。 <i>T</i> を因子分解した行列 <i>U</i> の <i>n</i> -2 個の第 2 の優対角成分で上書きされる。
<i>in</i>	INTEGER。 配列、次元は (<i>n</i>)。 <i>in</i> には置換行列 <i>P</i> の詳細が格納される。消去の <i>k</i> 番目の過程で交換が起こった場合は <i>in</i> (<i>k</i>) = 1 となり、それ以外では <i>in</i> (<i>k</i>) = 0 となる。 成分 <i>in</i> (<i>n</i>) は以下を満たすような最小の正の整数 <i>j</i> を返す。 $\text{abs}(u(j,j)) \leq \text{norm}(T - \text{lambda} * I(j)) * \text{tol},$ <i>norm</i> (<i>A</i> (<i>j</i>)) は行列 <i>A</i> の <i>j</i> 行目の絶対値の総和を示す。そのような <i>j</i> が存在しない場合は <i>in</i> (<i>n</i>) はゼロとして返される。 <i>in</i> (<i>n</i>) が正で返された場合、 <i>U</i> の対角成分は小さく、(<i>T</i> - <i>lambda</i> * <i>I</i>) が特異値またはほとんど特異値であることを示す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>k</i> の場合、 <i>k</i> 番目のパラメーターの値が不正だったことを示す。

?lagtm

$X = \alpha AB + \beta C$ の形式で示される行列 = 行列積を実行し、ここで *A* は三重対角行列、*B* と *C* は矩形行列、*a* と *b* はスカラーで 0、1、または -1 である。

構文

```
call slagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call dlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call clagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
call zlagtm( trans, n, nrhs, alpha, dl, d, du, x, ldx, beta, b, ldb )
```

説明

このルーチンは次の形式で示される行列 - ベクトル積を実行する。

$$B := \text{alpha} * A * X + \text{beta} * B$$

ここで *A* は次数 *n* の三重対角行列、*B* と *X* は *n* × *nrhs* 行列、*alpha* と *beta* は実数のスカラーで、それぞれ 0、1、または -1 である。

入力パラメーター

<i>trans</i>	CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、 $B := \alpha * A * X + \beta * B$ (転置なし) <i>trans</i> = 'T' の場合、 $B := \alpha * A^T * X + \beta * B$ (転置) <i>trans</i> = 'C' の場合、 $B := \alpha * A^H * X + \beta * B$ (共役転置)
<i>n</i>	INTEGER。行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の個数。すなわち、 <i>X</i> と <i>B</i> にある列の数 ($nrhs \geq 0$)。
<i>alpha, beta</i>	REAL (slagtm/clagtm の場合) DOUBLE PRECISION (dlagtm/zlagtm の場合) スカラー α と β 。alpha は 0、1、または -1 のいずれかでなければならない、それ以外の場合は 0 とみなされる。beta は 0、1、または -1 のいずれかでなければならない、それ以外の場合は 1 とみなされる。
<i>d1, d, du</i>	REAL (slagtm の場合) DOUBLE PRECISION (dlagtm の場合) COMPLEX (clagtm の場合) COMPLEX*16 (zlagtm の場合) 配列: <i>d1</i> (<i>n</i> -1), <i>d</i> (<i>n</i>), <i>du</i> (<i>n</i> -1)。 配列 <i>d1</i> には <i>T</i> の (<i>n</i> -1) 個の劣対角成分を格納する。 配列 <i>d</i> には <i>T</i> の <i>n</i> 個の対角成分を格納する。 配列 <i>du</i> には <i>T</i> の (<i>n</i> -1) 個の優対角成分を格納する。
<i>x, b</i>	REAL (slagtm の場合) DOUBLE PRECISION (dlagtm の場合) COMPLEX (clagtm の場合) COMPLEX*16 (zlagtm の場合) 配列: <i>x</i> (<i>ldx</i> ,*) には $n \times nrhs$ の行列 <i>X</i> を格納する。 <i>x</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。 <i>b</i> (<i>ldb</i> ,*) には $n \times nrhs$ の行列 <i>B</i> を格納する。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>ldx</i>	INTEGER。配列 <i>x</i> のリーディング・ディメンジョン。 $ldx \geq \max(1, n)$
<i>ldb</i>	INTEGER。配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$

出力パラメーター

<i>b</i>	次の行列式で上書きされる。 $B := \alpha * A * X + \beta * B$
----------	--

?lagts

?lagtf で計算された LU 因子分解を用いて連立方程式 $(T-\lambda)x = y$ または $(T-\lambda)^T x = y$ を解く。ここで T は一般三重対角行列、 λ はスカラー。

構文

```
call slagts( job, n, a, b, c, d, in, y, tol, info )
call dlagts( job, n, a, b, c, d, in, y, tol, info )
```

説明

このルーチンは次のどちらかの連立方程式を x について解くために使用される。

$(T - \lambda I)x = y$ または $(T - \lambda I)^T x = y$ 、
ここで T は $n \times n$ の三重対角行列で、 $(T - \lambda I)$ の因子分解から続く。

$$T - \lambda I = PLU$$

?lagtf で、として計算された

解く式は引数 *job* によって選択し、どちらの場合もゼロへの摂動か U のきわめて小さい対角成分のオプションがあり、このオプションは逆反復のようなアプリケーションでの使用を意図したものである。

入力パラメーター

job INTEGER。?lagts で実行される内容を次の中から指定する。
 = 1 の場合、式 $(T - \lambda I)x = y$ を解くが U の対角成分を摂動させない。
 = -1 の場合、式 $(T - \lambda I)x = y$ を解き、かつ、オーバーフローが起きるなら U の対角成分を摂動させる。下記の引数 *tol* を参照のこと。
 = 2 の場合、式 $(T - \lambda I)^T x = y$ を解くが U の対角成分を摂動させない。
 = -2 の場合、 $(T - \lambda I)^T x = y$ を解き、かつ、オーバーフローが起きるなら U の対角成分を摂動させる。下記の引数 *tol* を参照のこと。

n INTEGER。行列 T の次数 ($n \geq 0$)。

a, b, c, d REAL (slagts の場合)
 DOUBLE PRECISION (dlagts の場合)
 配列、次元は $a(n)$ 、 $b(n-1)$ 、 $c(n-1)$ 、 $d(n-2)$ ：
a(*) には、?lagtf から返されたとおりの、 U の対角成分を格納しなければならない。
b(*) には、?lagtf から返されたとおりの、 U の第 1 優対角成分を格納しなければならない。
c(*) には、?lagtf から返されたとおりの、 L の劣対角成分を格納

しなければならない。

$d(*)$ には、?lagtf から返されたとおり、 U の第 2 優対角成分を格納しなければならない。

<i>in</i>	INTEGER。 配列、次元は (n)。 $in(*)$ には、?lagtf から返されたとおり、行列 P の詳細を格納しなければならない。
<i>y</i>	REAL (slagts の場合) DOUBLE PRECISION (dlagts の場合) 配列、次元は (n)。ベクトル y の右辺である。
<i>tol</i>	REAL (slagtf の場合) DOUBLE PRECISION (dlagtf の場合) $job < 0$ の場合、 tol は、 U のきわめて小さな対角成分に対して作られる最小摂動でなければならない。 tol は通常およそ $eps * \text{norm}(U)$ として選ばれ、ここで eps は相対マシン精度であるが、 tol が 0 未満として与えられた場合は $eps * \max(\text{abs}(u(i, j)))$ に設定しなおされる。 $job > 0$ の場合、 tol は参照されない。

出力パラメーター

<i>y</i>	y は解ベクトル x によって上書きされる。
<i>tol</i>	上記入力パラメーターのセクションで説明したとおり、 tol が入力において 0 未満の場合のみ tol は変更される。それ以外では tol は変更されない。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正である。 $info = i > 0$ の場合、解ベクトル x の i 番目の成分の計算でオーバーフローが起こったことを示す。これは job が正として与えられ、 U の対角成分がきわめて小さいかベクトル y の右辺の成分がきわめて大きい場合にのみ起こる

?lagv2

実数の 2×2 行列束 (A, B) の汎用 Schur 因子分解を計算する。ここで B は上三角である。

構文

```
call slagv2( a, lda, b, ldb, alphas, alphai, beta, csl, snl, csr, snr )
call dlagv2( a, lda, b, ldb, alphas, alphai, beta, csl, snl, csr, snr )
```

説明

このルーチンは 2×2 行列束 (A, B) の汎用 Schur 因子分解を計算する。ここで B は上三角である。ルーチンは、次を満たすような csl, snl と csr, snr で与えられる直交 (回転) 行列を計算する。

1) 束 (A, B) が 2 個の実数固有値 (0/0 または 1/0 タイプを含む) を持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

2) 束 (A, B) が複素共役固有値のペアを持つ場合、

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & 0 \\ 0 & b_{22} \end{bmatrix} = \begin{bmatrix} cs1 & sn1 \\ -sn1 & cs1 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ 0 & b_{22} \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix}$$

ここで $b_{11} \geq b_{22} > 0$ 。

入力パラメーター

a, b	REAL (slagv2 の場合) DOUBLE PRECISION (dlagv2 の場合) 配列: $a(lda, 2)$ には 2×2 行列 A を格納する。 $b(l db, 2)$ には上三角 2×2 行列 B を格納する。
lda	INTEGER。配列 a のリーディング・ディメンジョン。 $lda \geq 2$
ldb	INTEGER。配列 b のリーディング・ディメンジョン。 $ldb \geq 2$

出力パラメーター

a	a は汎用 Schur 形式の "A-部分" で上書きされる。
b	b は汎用 Schur 形式の "B-部分" で上書きされる。
$alpha, alpha_i, beta$	REAL (slagv2 の場合) DOUBLE PRECISION (dlagv2 の場合) 配列、次元はそれぞれ、(2)。 $(alpha(k) + i * alpha_i(k)) / beta(k)$ は束 (A, B) の固有値で、 $k=1, 2$ および $i = \sqrt{-1}$ である。 $beta(k)$ はゼロとなる点がある点に 注意する。

<i>cs1, sn1</i>	REAL (slagv2 の場合) DOUBLE PRECISION (dlagv2 の場合) それぞれ、左回転行列の余弦と正弦である。
<i>csr, snr</i>	REAL (slagv2 の場合) DOUBLE PRECISION (dlagv2 の場合) それぞれ、右回転行列の余弦と正弦である。

?lahqr

ダブルシフト/シングルシフト QR アルゴリズム
を用いて、上 Hessenberg 行列の固有値と Schur 因子分解を計算する。

構文

```
call slahqr( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz, z, ldz,
            info )
call dlahqr( wantt, wantz, n, ilo, ihi, h, ldh, wr, wi, iloz, ihiz, z, ldz,
            info )
call clahqr( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z, ldz, info )
call zlahqr( wantt, wantz, n, ilo, ihi, h, ldh, w, iloz, ihiz, z, ldz, info )
```

説明

このルーチンは [?hseqr](#) から呼び出される補助ルーチンで、*ilo* から *ihi* の行と列にある Hessenberg 部分行列を対象として、[?hseqr](#) によって過去に計算された固有値と Schur 分解を更新する。

入力パラメーター

<i>wantt</i>	LOGICAL。 <i>wantt</i> = .TRUE. の場合、すべての Schur 形式 <i>T</i> が必要である。 <i>wantt</i> = .FALSE. の場合、固有値のみが必要である。
<i>wantz</i>	LOGICAL。 <i>wantz</i> = .TRUE. の場合、Schur ベクトル <i>z</i> が必要である。 <i>wantz</i> = .FALSE. の場合、Schur ベクトルは必要ない。
<i>n</i>	INTEGER。行列 <i>H</i> の次数 ($n \geq 0$)。
<i>ilo, ihi</i>	INTEGER。 <i>H</i> は、行と列が <i>ihi</i> +1: <i>n</i> の上準三角になっているとする。また、 $H(i_{lo}, i_{lo}-1) = 0$ ($i_{lo} = 1$ でない場合) であるとする。ルーチン ?lahqr は基本的に行と列が <i>ilo</i> から <i>ihi</i> の Hessenberg 部分行列を対象とするが、 <i>wantt</i> = .TRUE. の場合は <i>H</i> のすべてに変換が適用される。 次の制約がある。 $1 \leq i_{lo} \leq \max(1, i_{hi}); i_{hi} \leq n$.

<i>h, z</i>	<p>REAL (slahqr の場合) DOUBLE PRECISION (dlahqr の場合) COMPLEX clahqr の場合) COMPLEX*16 (zlahqr の場合) 配列: <i>h(ldh,*)</i> には上 Hessenberg 行列 H が格納される。 <i>h</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。 <i>z(ldz,*)</i> <i>wantz = .TRUE.</i> の場合、<i>z</i> には ?hseqr で蓄積された変換の現在の行列 Z を格納しなければならない。 <i>wantz = .FALSE.</i> の場合、<i>z</i> は参照されない。 <i>z</i> の第 2 次元は、$\max(1, n)$ 以上でなければならない。</p>
<i>ldh</i>	INTEGER。 <i>h</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldz</i>	INTEGER。 <i>z</i> の第 1 次元。 $\max(1, n)$ 以上であること。
<i>iloz, ihiz</i>	<p>INTEGER。 <i>wantz = .TRUE.</i> の場合、変換が適用されなければならない Z の行を指定する。 $1 \leq iloz \leq ilo; ihi \leq ihiz \leq n$。</p>

出力パラメーター

<i>h</i>	<p><i>wantt = .TRUE.</i> の場合、H は行と列が <i>ilo:ihiz</i> にある上準三角 (複素型では上三角) で、標準形式の 2×2 対角ブロックを持つ。 <i>wantt = .FALSE.</i> の場合、H の内容は規定されない。</p>
<i>wr, wi</i>	<p>REAL (slahqr の場合) DOUBLE PRECISION (dlahqr の場合) 配列、次元はそれぞれ $\max(1, n)$ 以上。実数型でのみ使用される。 計算された固有値 <i>ilo</i> から <i>ihiz</i> の実数部分と虚数部分はそれぞれ <i>wr</i> と <i>wi</i> の対応する成分に格納される。2 個の固有値が複素共役ペアとして計算された場合は、それらは <i>wr</i> と <i>wi</i> の連続する成分に格納され、すなわち <i>i</i> 番目と (<i>i</i>+1) 番目で $wi(i) > 0$ かつ $wi(i+1) < 0$ である。<i>wantt = .TRUE.</i> の場合、固有値は H に返された Schur 形式の対角と同じ順序で格納され、$wr(i) = H(i, i)$ また 2×2 対角ブロックでは $H(i:i+1, i:i+1)$ で、$wi(i) = \sqrt{H(i+1, i) * H(i, i+1)}$ および $wi(i+1) = -wi(i)$ である。</p>
<i>w</i>	<p>COMPLEX (clahqr の場合) COMPLEX*16 (zlahqr の場合) 配列、次元は $\max(1, n)$ 以上。複素型でのみ使用される。 計算された固有値 <i>ilo</i> から <i>ihiz</i> は対応する w の成分に格納される。 <i>wantt = .TRUE.</i> の場合、固有値は H に返された Schur 形式の対角と同じ順序で格納され、$w(i) = H(i, i)$。</p>
<i>z</i>	<p><i>wantz = .TRUE.</i> の場合、<i>z</i> は更新されている。変換は部分行列 $Z(iloz:ihiz, ilo:ihiz)$ にのみ適用されている。</p>
<i>info</i>	<p>INTEGER。 <i>info = 0</i> の場合、正常に終了したことを示す。 <i>info = i > 0</i> の場合、?lahqr はすべての固有値 <i>ilo</i> から <i>ihiz</i> を、</p>

合計 $30*(ihi-ilo+1)$ 回以内の反復で計算できなかったことを示す。 wr と wi (slahqr/dlahqr の場合) または w (clahqr/zlahqr の場合) の成分 $i+1:ihi$ には、正常に終了したそれら固有値が格納される。

?lahrd

k 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の nb 列を縮退させ、 A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

構文

```
call slahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call dlahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call clahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
call zlahrd( n, k, nb, a, lda, tau, t, ldt, y, ldy )
```

説明

このルーチンは、 k 番目の劣対角よりも下の成分がゼロになるように、実数 / 複素一般 $n \times (n-k+1)$ 行列 A の最初の nb 列を縮退させる。この縮退は直交 / ユニタリー相似変換 $Q'AQ$ によって実行される。ルーチンは、ブロック・リフレクター $I - VTV'$ として Q を定義する行列 V と T 、および行列 $Y = AVT$ を返す。

行列 Q は nb 個の基本リフレクターの積として表現される。
 $Q = H(1)H(2) \dots H(nb)$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

τ は実数 / 複素スカラー、 v は実数 / 複素の $(n-1)$ 成分で構成されるベクトルである。

このルーチンは、[?gehrd](#) から呼び出される補助ルーチンである。

入力パラメーター

n	INTEGER。行列 A の次数 ($n \geq 0$)。
k	INTEGER。縮退のオフセット。最初の nb 列にある k 番目の劣対角よりも下の成分がゼロに縮退される。
nb	INTEGER。縮退させる列数。
a	REAL (slahrd の場合) DOUBLE PRECISION (dlahrd の場合) COMPLEX (clahrd の場合) COMPLEX*16 (zlahrd の場合) $a(lda, n-k+1)$ には縮退対象の $n \times (n-k+1)$ 一般行列 A を格納する。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $\max(1, n)$ 以上。
<i>ldt</i>	INTEGER。 出力配列 <i>t</i> の第 1 次元のサイズ。 $\max(1, nb)$ 以上でなければならない。
<i>ldy</i>	INTEGER。 出力配列 <i>y</i> の第 1 次元のサイズ。 $\max(1, n)$ 以上でなければならない。

出力パラメーター

<i>a</i>	最初の <i>nb</i> 列にある <i>k</i> 番目の劣対角成分とその上の成分は縮退された行列の対応する成分で上書きされる。 <i>k</i> 番目の劣対角よりも下の成分は配列 <i>tau</i> で上書きされ、基本リフレクターの積として行列 <i>Q</i> を表現する。そのほかの列は変更されない。次の「アプリケーション・ノート」を参照。
<i>tau</i>	REAL (slahrd の場合) DOUBLE PRECISION (dlahrd の場合) COMPLEX (clahrd の場合) COMPLEX*16 (zlahrd の場合) 配列、次元は (<i>nb</i>)。 基本リフレクターのスカラー係数が入る。
<i>t, y</i>	REAL (slahrd の場合) DOUBLE PRECISION (dlahrd の場合) COMPLEX (clahrd の場合) COMPLEX*16 (zlahrd の場合) 配列、次元は <i>t</i> (<i>ldt, nb</i>), <i>y</i> (<i>ldy, nb</i>)。 配列 <i>t</i> には上三角行列 <i>T</i> が格納される。 配列 <i>y</i> には $n \times nb$ の行列 <i>Y</i> が格納される。

アプリケーション・ノート

基本リフレクター $H(i)$ に対して、

ルーチン終了時に $v(1:i+k-1) = 0$, $v(i+k) = 1$; $v(i+k+1:n)$ は $a(i+k+1:n, i)$ に格納され、*tau* は $\tau(i)$ に格納される。

ベクトル *v* の成分は、行列の縮退されていない部分に変換を適用するために、次の形式の更新を用いて、*T* と *Y* とともに必要な $(n-k+1) \times nb$ の行列 *V* を形成する。

$$A := (I - VT V') * (A - Y V')$$

$n=7, k=3, nb=2$ におけるルーチン終了時の *A* の内容を以下に示す。

$$\begin{bmatrix} a & h & a & a & a \\ a & h & a & a & a \\ a & h & a & a & a \\ h & h & a & a & a \\ v_1 & h & a & a & a \\ v_1 & v_2 & a & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

a は元の行列 A の成分、 h は上 Hessenberg 行列 H の変更された成分、 v_i は $H(i)$ を定義するベクトルの成分を示す。

?laic1

増加条件推定を 1 ステップ適用する。

構文

```
call slaic1( job, j, x, sest, w, gamma, sestpr, s, c )
call dlaic1( job, j, x, sest, w, gamma, sestpr, s, c )
call claic1( job, j, x, sest, w, gamma, sestpr, s, c )
call zlaic1( job, j, x, sest, w, gamma, sestpr, s, c )
```

説明

ルーチン ?laic1 は最も簡単な形で増加条件推定を 1 ステップ適用する。

$\|x\|_2 = 1$ ($\|a\|_2$ とは a の 2- ノルムを示す) の x を、次を満たすような $j \times j$ の下三角行列 L のおおよその特異ベクトルとする。

$$\|L*x\|_2 = sest$$

次に ?laic1 は、おおよその特異ベクトルを次の意味で満たす $sestpr$ 、 s 、 c を計算する。

$$xhat = \begin{bmatrix} s*x \\ c \end{bmatrix}$$

$$Lhat = \begin{bmatrix} L & 0 \\ w' & gamma \end{bmatrix}$$

$$\|Lhat * xhat\|_2 = sestpr$$

job によって最大または最小特異値に対する推定が計算される。

$[s \ c]'$ と $sestpr^2$ は、次の連立方程式の固有ペアである点に注意する (slaic1/claic の場合)

$$\text{diag}(sest*sest, 0) + [alpha \ gamma] * \begin{bmatrix} alpha \\ gamma \end{bmatrix},$$

ここで、 $alpha = x'*w$ 、

または、次の連立方程式の固有ペアである (claic1/zlaic の場合)

$$\text{diag}(sest*sest, 0) + [\alpha \quad \gamma] * \begin{bmatrix} \text{conjg}(\alpha) \\ \text{conjg}(\gamma) \end{bmatrix},$$

ここで、 $\alpha = \text{conjg}(x)' * w$

入力パラメーター

<i>job</i>	INTEGER。 <i>job</i> =1 の場合、最大特異値に対する推定が計算される。 <i>job</i> =2 の場合、最小特異値に対する推定が計算される。
<i>j</i>	INTEGER。 <i>x</i> と <i>w</i> の長さ。
<i>x, w</i>	REAL (slaic1 の場合) DOUBLE PRECISION (dlaic1 の場合) COMPLEX (claic1 の場合) COMPLEX*16 (zlaic1 の場合) 配列、次元はそれぞれ、(<i>j</i>) それぞれ、ベクトル <i>x</i> と <i>w</i> を格納する。
<i>sest</i>	REAL (slaic1/claic1 の場合) DOUBLE PRECISION (dlaic1/zlaic1 の場合) <i>j</i> × <i>j</i> の行列 <i>L</i> の推定された特異値。
<i>gamma</i>	REAL (slaic1 の場合) DOUBLE PRECISION (dlaic1 の場合) COMPLEX (claic1 の場合) COMPLEX*16 (zlaic1 の場合) 対角成分 <i>gamma</i> 。

出力パラメーター

<i>sestpr</i>	REAL (slaic1/claic1 の場合) DOUBLE PRECISION (dlaic1/zlaic1 の場合) (<i>j</i> +1) × (<i>j</i> +1) の行列 <i>Lhat</i> の推定された特異値。
<i>s, c</i>	REAL (slaic1 の場合) DOUBLE PRECISION (dlaic1 の場合) COMPLEX (claic1 の場合) COMPLEX*16 (zlaic1 の場合) <i>xhat</i> の構成に必要な正弦と余弦。

?laln2

指定された形式の 1×1 または 2×2 の連立線形方程式を解く。

構文

```
call slaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi, x, ldx,
            scale, xnorm, info )
```

```
call dlaln2( ltrans, na, nw, smin, ca, a, lda, d1, d2, b, ldb, wr, wi, x, ldx,
            scale, xnorm, info )
```

説明

このルーチンは次の形式の連立方程式を、

$$(ca A - w D) X = s B \quad \text{または} \quad (ca A' - w D) X = s B$$

可能なスケーリング (s) と A の摂動 (A' は A の転置) を使って解く。

A は $na \times na$ の実数行列、 ca は実数スカラー、 D は $na \times na$ の実数対角行列、 w は実数または複素値、 X と B は $na \times 1$ の行列で w が実数の場合は実数、 w が複素の場合は複素である。パラメーター na は 1 か 2 である。

w が複素の場合、 X と B は $na \times 2$ の行列として表現され、それぞれの第 1 の列が実数部、第 2 の列が虚数部となる。

このルーチンは X をオーバーフローせずに計算できるようにスケール係数 $s (\leq 1)$ を選択する。 X は、 $\text{norm}(ca A - w D) * \text{norm}(X)$ がオーバーフロー未満になるように、必要に応じてさらにスケールされる。

$(ca A - w D)$ の両方の特異値が $smin$ よりも小さい場合、 $smin * I$ (ここで I は単位を表わす) が $(ca A - w D)$ の代わりに使用される。1 つの固有値が $smin$ よりも小さい場合は、最小特異値がおおよそ $smin$ になるように $(ca A - w D)$ の 1 つの成分が摂動される。両方の特異値が $smin$ 以上ならば、 $(ca A - w D)$ は摂動されない。

どのような場合でも、摂動は、大きくとも $\max(smin, ulp * \text{norm}(ca A - w D))$ の小さな倍数となる。

特異値は無限ノルム近似によって計算され、そのため、2 程度の係数に対してのみ正確となる。



注: 入力の大きさは、妥当な係数によってオーバーフローよりも小さいと仮定される (*bignum* を参照)。

入力パラメーター

<i>trans</i>	LOGICAL。 <i>trans</i> = .TRUE. の場合、 A - 転置が使用される。 <i>trans</i> = .FALSE. の場合、 A が使用される (転置なし)。
<i>na</i>	INTEGER。行列 A のサイズである。1 か 2 のみを取り得る。

<i>nw</i>	INTEGER。 <i>w</i> が実数の場合、このパラメーターは 1、 <i>w</i> が複素数の場合、パラメーターは 2 でなければならない。 1 か 2 のみを取り得る。
<i>smin</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) <i>A</i> の特異値における目的の下限。これはアンダーフローまたはオーバーフローに対して安全な距離を持っていなければならない。例えば、 (<i>underflow/machine_precision</i>) と (<i>machine_precision * overflow</i>) の間とする。 (<i>bignum</i> と <i>ulp</i> を参照)
<i>ca</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) <i>A</i> に乗算される係数。
<i>a</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) 配列、次元は (<i>lda, na</i>)。 $na \times na$ の行列 <i>A</i>
<i>lda</i>	INTEGER。 <i>a</i> のリーディング・ディメンジョン。 na 以上でなければならない。
<i>d1, d2</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) それぞれ対角行列 <i>D</i> にある (1, 1) と (2, 2) の成分。 $nw = 1$ の場合は <i>d2</i> は使用されない。
<i>b</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) 配列、次元は (<i>ldb, nw</i>)。 $na \times nw$ の行列 <i>B</i> (右辺)。 $nw = 2$ (<i>w</i> が複素) の場合、列 1 には <i>B</i> の実数部分を格納し、列 2 には虚数部分を格納する。
<i>ldb</i>	INTEGER。 <i>b</i> のリーディング・ディメンジョン。 na 以上でなければならない。
<i>wr, wi</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) それぞれスカラー <i>w</i> の実数部と虚数部。 $nw = 1$ の場合は <i>wi</i> は参照されない。
<i>ldx</i>	INTEGER。出力配列 <i>x</i> のリーディング・ディメンジョン。 na 以上でなければならない。

出力パラメーター

<i>x</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) 配列、次元は (<i>ldx, nw</i>)。このルーチンによって計算された $na \times nw$ の行列 <i>X</i> (未知)。 $nw = 2$ の場合 (<i>w</i> が複素)、列 1 には <i>X</i> の実数部が格納され、列 2 には虚数部が格納される。
----------	---

<i>scale</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) <i>X</i> の計算でオーバーフローを起こさないように <i>B</i> に乗算されるべきスケール係数。そのため、 $(caA - wD)X$ は <i>B</i> ではなく $scale * B$ となる (<i>A</i> の摂動は無視)。大きくとも 1 である。
<i>xnorm</i>	REAL (slaln2 の場合) DOUBLE PRECISION (dlaln2 の場合) <i>X</i> が $na \times nw$ の実数行列としてみなされる場合、 <i>X</i> の無限ノルム。
<i>info</i>	INTEGER。 エラーフラグ。エラーが生じなかった場合はゼロに、引数にエラーがあった場合は負に、 $(caA - wD)$ を摂動しなければならない場合は正になる。 取り得る値は次のとおりである。 <i>info</i> = 0 の場合、正常に終了したことを示し、 $(caA - wD)$ を摂動する必要がなかった。 <i>info</i> = 1 の場合、最小 (または唯一の) 特異値を <i>smin</i> よりも大きくするために、 $(caA - wD)$ を摂動する必要があったことを示す。



注: 高速化するために、このルーチンでは入力のエラーをチェックしない。

?lals0

最小二乗問題を分割統治SVD法を使用して解く過程で乗率を逆に適用する。?gelsd で使用される。

構文

```
call slals0(  icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, work, info )
call dlals0(  icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, work, info )
call clals0(  icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, rwork, info )
call zlals0(  icompq, nl, nr, sqre, nrhs, b, ldb, bx, ldbx, perm,
             givptr, givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z,
             k, c, s, rwork, info )
```

説明

このルーチンは、分割統治 SVD 法を使用して最小二乗問題を解く過程で、右辺行列 B に行を追加した対角行列の左または右特異ベクトル行列のいずれかの乗率を逆に適用する。

左特異ベクトル行列において、3 タイプの直交行列が関係する。

(1L) Givens 回転。回転の回数は *givptr* に格納する。回転が適用された列 / 行のペアは *givcol* に格納する。これら回転の c 値と s 値は *givnum* に格納する。

(2L) 置換。 B の $(n1+1)$ 番目の行が第 1 の行に移動される行であり、 $j = 2:n$ において、 B の *perm(j)* 番目の行が j 番目の行に移動される行である。

(3L) 残りの行列の左特異ベクトル行列。

右特異ベクトル行列において、4 タイプの直交行列が関係する。

(1R) 残りの行列の右特異ベクトル行列。

(2R) $sqre = 1$ の場合、右ヌル空間を生成するために 1 回の追加 Givens 回転。

(3R) (2L) の逆変換。

(4R) (1L) の逆変換。

入力パラメーター

<i>icompq</i>	INTEGER。特異ベクトルを因子分解された形式で計算するかを指定する。 <i>icompq</i> = 0 の場合、左特異ベクトル行列。 <i>icompq</i> = 1 の場合、右特異ベクトル行列。
<i>n1</i>	INTEGER。上ブロックの行次元。 $n1 \geq 1$
<i>nr</i>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<i>sqre</i>	INTEGER。 <i>sqre</i> = 0 の場合、下ブロックは $nr \times nr$ の正方行列。 <i>sqre</i> = 1 の場合、下ブロックは $nr \times (nr+1)$ の矩形行列。二重対角行列は行次元 $n = n1 + nr + 1$ 、列次元 $m = n + sqre$ を持つ。
<i>nrhs</i>	INTEGER。 b と bx の列の数。 1 以上でなければならない。
b	REAL (<i>slals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> の場合) COMPLEX (<i>clals0</i> の場合) COMPLEX*16 (<i>zlals0</i> の場合) 配列、次元は $(ldb, nrhs)$ 。最小二乗問題の右辺を行 1 から m に格納する。
<i>ldb</i>	INTEGER。 b のリーディング・ディメンジョン。 $\max(1, \max(m, n))$ 以上でなければならない。

<i>bx</i>	REAL (<i>slals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> の場合) COMPLEX (<i>clals0</i> の場合) COMPLEX*16 (<i>zlals0</i> の場合) ワークスペース配列、次元は (<i>ldbx, nrhs</i>)。
<i>ldbx</i>	INTEGER。 <i>bx</i> のリーディング・ディメンジョン。
<i>perm</i>	INTEGER。 配列、次元は (<i>n</i>)。2 個のブロックに適用される (収縮とソートによる) 置換。
<i>givptr</i>	INTEGER。 この部分問題で実行された Givens 回転の回数が格納される。
<i>givcol</i>	INTEGER。 配列、次元は (<i>ldgcol, 2</i>)。 Givens 回転に関与した行 / 列のペアを示す数値のペア。
<i>ldgcol</i>	INTEGER。 <i>givcol</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>givnum</i>	REAL (<i>slals0</i> / <i>clals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> / <i>zlals0</i> の場合) 配列、次元は (<i>ldgnum, 2</i>)。対応する Givens 回転で使用された <i>c</i> 値または <i>s</i> 値を示す数。
<i>ldgnum</i>	INTEGER。 配列 <i>difr</i> 、 <i>poles</i> 、 <i>givnum</i> のリーディング・ディメンジョン。 <i>k</i> 以上でなければならない。
<i>poles</i>	REAL (<i>slals0</i> / <i>clals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> / <i>zlals0</i> の場合) 配列、次元は (<i>ldgnum, 2</i>)。 <i>poles</i> (1: <i>k, 1</i>) には永年方程式を解いて得られた新しい特異値を格納し、 <i>poles</i> (1: <i>k, 2</i>) には永年方程式内の極を含む配列を格納する。
<i>difl</i>	REAL (<i>slals0</i> / <i>clals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> / <i>zlals0</i> の場合) 配列、次元は (<i>k</i>)。 <i>difl</i> (<i>i</i>) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i> 番目の (収縮されていない) 古い特異値の距離を格納する。
<i>difr</i>	REAL (<i>slals0</i> / <i>clals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> / <i>zlals0</i> の場合) 配列、次元は (<i>ldgnum, 2</i>)。 <i>difr</i> (<i>i, 1</i>) には、 <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i+1</i> 番目の (収縮されていない) 古い特異値の距離を格納する。 <i>difr</i> (<i>i, 2</i>) には <i>i</i> 番目の右特異ベクトルに対する正規化係数を格納する。
<i>z</i>	REAL (<i>slals0</i> / <i>clals0</i> の場合) DOUBLE PRECISION (<i>dlals0</i> / <i>zlals0</i> の場合) 配列、次元は (<i>k</i>)。収縮調整された更新行ベクトルの成分を格納する。
<i>k</i>	INTEGER。非収縮行列の次元を格納する。これは関連する永年方程式の次数である。 $1 \leq k \leq n$

<i>c</i>	REAL (slals0 /clals0 の場合) DOUBLE PRECISION (dlals0/zlals0 の場合) <i>sqre</i> =0 の場合はガベージを格納し、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>c</i> 値を格納する。
<i>s</i>	REAL (slals0 /clals0 の場合) DOUBLE PRECISION (dlals0/zlals0 の場合) <i>sqre</i> =0 の場合はガベージを格納し、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>s</i> 値を格納する。
<i>work</i>	REAL (slals0 の場合) DOUBLE PRECISION (dlals0 の場合) ワークスペース配列、次元は (<i>k</i>)。実数型でのみ使用される。
<i>rwork</i>	REAL (clals0 の場合) DOUBLE PRECISION (zlals0 の場合) ワークスペース配列、次元は ($k*(1+nrhs) + 2*nrhs$)。複素型でのみ使用される。

出力パラメーター

<i>b</i>	行 1 から <i>n</i> は解 <i>X</i> で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。

?lalsa

コンパクト形式で係数行列の SVD を計算する。
?gelsd で使用される。

構文

```
call slalsa(  icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork,
             info )
call dlalsa(  icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork,
             info )
call clalsa(  icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, rwork, iwork,
             info )
call zlalsa(  icompq, smlsiz, n, nrhs, b, ldb, bx, ldbx, u, ldu, vt, k, difl,
             difr, z, poles, givptr, givcol, ldgcol, perm, givnum, c, s, rwork, iwork,
             info )
```

説明

このルーチンは最小二乗問題を解く中間の過程であり、コンパクト形式で係数行列の SVD を計算する。特異ベクトルは単純な直交行列の積として計算される。

ルーチン ?lalsa は、 $icompg = 0$ の場合は上二重対角行列の左特異ベクトル行列の逆を右辺に適用する。

$icompg = 1$ の場合は右特異ベクトル行列を右辺に適用する。特異ベクトル行列は ?lalsa によってコンパクト形式で生成される。

入力パラメーター

<i>icompg</i>	INTEGER。左または右のどちらの特異ベクトル行列を対象とするか指定する。 <i>icompg</i> = 0 の場合、左特異ベクトル行列を使用する <i>icompg</i> = 1 の場合、右特異ベクトル行列を使用する。
<i>smlsiz</i>	INTEGER。計算ツリーの一番下の部分問題の最大サイズ。
<i>n</i>	INTEGER。上二重対角行列の行と列の次元。
<i>nrhs</i>	INTEGER。 <i>b</i> と <i>bx</i> の列の数。1 以上でなければならない。
<i>b</i>	REAL (<i>slalsa</i> の場合) DOUBLE PRECISION (<i>dlalsa</i> の場合) COMPLEX (<i>clalsa</i> の場合) COMPLEX*16 (<i>zlalsa</i> の場合) 配列、次元は (<i>ldb</i> , <i>nrhs</i>)。最小二乗問題の右辺を行 1 から <i>m</i> に格納する。
<i>ldb</i>	INTEGER。呼び出し元のサブプログラムの <i>b</i> のリーディング・ディメンジョン。max(1, max(<i>m</i> , <i>n</i>)) 以上でなければならない。
<i>ldb_x</i>	INTEGER。出力配列 <i>bx</i> のリーディング・ディメンジョン。
<i>u</i>	REAL (<i>slalsa/clalsa</i> の場合) DOUBLE PRECISION (<i>dlalsa/zlalsa</i> の場合) 配列、次元は (<i>ldu</i> , <i>smlsiz</i>)。 <i>u</i> には、一番下のレベルにあるすべての部分問題の左特異ベクトル行列を格納する。
<i>ldu</i>	INTEGER、 $ldu \geq n$ 。配列 <i>u</i> 、 <i>vt</i> 、 <i>difl</i> 、 <i>difr</i> 、 <i>poles</i> 、 <i>givnum</i> 、 <i>z</i> のリーディング・ディメンジョン。
<i>vt</i>	REAL (<i>slalsa/clalsa</i> の場合) DOUBLE PRECISION (<i>dlalsa/zlalsa</i> の場合) 配列、次元は (<i>ldu</i> , <i>smlsiz</i> + 1)。一番下のレベルにあるすべての部分問題の右特異ベクトル行列を格納する。
<i>k</i>	INTEGER 配列、次元は (<i>n</i>)。
<i>difl</i>	REAL (<i>slalsa/clalsa</i> の場合) DOUBLE PRECISION (<i>dlalsa/zlalsa</i> の場合) 配列、次元は (<i>ldu</i> , <i>nlv1</i>)。 $nlv1 = \text{int}(\log_2(n/(smlsiz+1))) + 1$ 。
<i>difr</i>	REAL (<i>slalsa/clalsa</i> の場合) DOUBLE PRECISION (<i>dlalsa/zlalsa</i> の場合) 配列、次元は (<i>ldu</i> , $2 * nlv1$)。 <i>difl</i> (*, <i>i</i>) と <i>difr</i> (*, $2i - 1$) には、 <i>i</i> 番目のレベルの特異値と (<i>i</i> -1) 番目のレベルの特異値の距離を格納する。 <i>difr</i> (*, $2i$) には <i>i</i> 番目のレベルの部分問題の右特異ベクトル行列に対する正規化係数を格納する。

<i>z</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は (<i>ldu, nlv1</i>)。 <i>z</i> (<i>l, i</i>) には <i>i</i> 番目レベルの部分問題に対する収縮調整された更新行ベクトルの成分を格納する。
<i>poles</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は (<i>ldu, 2*nlv1</i>)。 <i>poles</i> (*, <i>2i-1:2i</i>) には <i>i</i> 番目レベルの永年方程式に関する新しい特異値と古い特異値を格納する。
<i>givptr</i>	INTEGER。 配列、次元は (<i>n</i>)。 <i>givptr</i> (<i>i</i>) には計算ツリーの <i>i</i> 番目問題で実行された Givens 回転の回数を格納する。
<i>givcol</i>	INTEGER。 配列、次元は (<i>ldgcol, 2*nlv1</i>)。それぞれの <i>i</i> で、 <i>iGIVCOL</i> (*, <i>2i-1:2i</i>) には計算ツリーの <i>i</i> 番目レベルで実行された Givens 回転の位置を格納する。
<i>ldgcol</i>	INTEGER、 <i>ldgcol</i> ≥ <i>n</i> 。配列 <i>givcol</i> 、 <i>perm</i> のリーディング・ディメンジョン。
<i>perm</i>	INTEGER。 配列、次元は (<i>ldgcol, nlv1</i>)。 <i>perm</i> (*, <i>i</i>) には計算ツリーの <i>i</i> 番目レベルで実行された置換を格納する。
<i>givnum</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は (<i>ldu, 2*nlv1</i>)。 <i>givnum</i> (*, <i>2i-1:2i</i>) には計算ツリーの <i>i</i> 番目レベルで実行された Givens 回転の <i>c</i> 値と <i>s</i> 値を格納する。
<i>c</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は (<i>n</i>)。 <i>i</i> 番目の部分問題が正方ではない場合、 <i>c</i> (<i>i</i>) には <i>i</i> 番目の部分問題の右ヌル空間に関する Givens 回転の <i>c</i> 値を格納する。
<i>s</i>	REAL (slalsa/clalsa の場合) DOUBLE PRECISION (dlalsa/zlalsa の場合) 配列、次元は (<i>n</i>)。 <i>i</i> 番目の部分問題が正方ではない場合、 <i>s</i> (<i>i</i>) には <i>i</i> 番目の部分問題の右ヌル空間に関する Givens 回転の <i>s</i> 値を格納する。
<i>work</i>	REAL (slalsa の場合) DOUBLE PRECISION (dlalsa の場合) ワークスペース配列、次元は (<i>n</i>) 以上。実数型でのみ使用される。
<i>rwork</i>	REAL (clalsa の場合) DOUBLE PRECISION (zlalsa の場合) ワークスペース配列、次元は $\max(n, (smlsz+1)*nrhs*3)$ 以上。複素型でのみ使用される。

iwork INTEGER。
ワークスペース配列、次元は $(3n)$ 以上。

出力パラメーター

b 行 1 から n は解 X で上書きされる。

bx REAL (*slalsa* の場合)
DOUBLE PRECISION (*dlalsa* の場合) COMPLEX (*clalsa* の場合)
COMPLEX*16 (*zlalsa* の場合)
配列、次元は $(ldb, nrhs)$ 。*b* に対する左または右の特異ベクトル行列の適用結果で上書きされる。

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = $-i < 0$ の場合、 i 番目の引数の値が不正だったことを示す。

?lalsd

最小二乗問題を解くために A の特異値分解を使用する。

構文

```
call slalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, iwork,
            info )
call dlalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, iwork,
            info )
call clalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, rwork,
            iwork, info )
call zlalsd( uplo, smlsiz, n, nrhs, d, e, b, ldb, rcond, rank, work, rwork,
            iwork, info )
```

説明

このルーチンは、 $AX=B$ の各列のユークリッド・ノルムを最小化するために X を求める最小二乗問題を A の特異値分解を使用して解き、ここで A は $n \times n$ の上二重対角、 X と B は $n \times nrhs$ である。解 X は B を上書きする。

最大特異値の *rcond* 倍より小さな A の特異値は、最小二乗問題を解く過程でゼロとして扱われる。この場合は最小ノルム解が返される。実際の特異値は d に昇順で格納され返される。

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリーマシンで動作する。

ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 <i>uplo</i> = 'U' の場合、 <i>d</i> と <i>e</i> は上二重対角行列を定義する。 <i>uplo</i> = 'L' の場合、 <i>d</i> と <i>e</i> は下二重対角行列を定義する。
<i>smlsiz</i>	INTEGER。計算ツリーの一番下の部分問題の最大サイズ。
<i>n</i>	INTEGER。二重対角行列の次元。 $n \geq 0$ 。
<i>nrhs</i>	INTEGER。 <i>B</i> の列の数。1 以上でなければならない。
<i>d</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 配列、次元は (<i>n</i>)。 <i>d</i> には二重対角行列の主対角を格納する。
<i>e</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 配列、次元は (<i>n</i> -1)。入力パラメーターとして、二重対角行列の優対角成分を格納する。出力において <i>e</i> の内容は壊される。
<i>b</i>	REAL (slalsd の場合) DOUBLE PRECISION (dlalsd の場合) COMPLEX (clalsd の場合) COMPLEX*16 (zlalsd の場合) 配列、次元は、(<i>ldb</i> , <i>nrhs</i>)。入力パラメーターとして <i>b</i> には最小二乗問題の右辺を格納する。出力パラメーターとして <i>b</i> には解 <i>X</i> が格納される。
<i>ldb</i>	INTEGER。呼び出し元のサブプログラムの <i>b</i> のリーディング・ディメンジョン。 $\max(1, n)$ 以上でなければならない。
<i>rcond</i>	REAL (slalsd/clalsd の場合) DOUBLE PRECISION (dlalsd/zlalsd の場合) 最大特異値の <i>rcond</i> 倍より小さいか等しい <i>A</i> の特異値は、最小二乗問題を解く過程でゼロとして扱われる。 <i>rcond</i> が負の場合はマシン精度が代わりに使用される。 例えば、 $\text{diag}(S) * X = B$ が最小二乗問題で $\text{diag}(S)$ が特異値の二重対角行列の場合、解は、 $S(i)$ が $rcond * \max(S)$ より大きい場合は $X(i) = B(i) / S(i)$ に、 $S(i)$ が $rcond * \max(S)$ より小さいか等しい場合は $X(i) = 0$ になる。
<i>rank</i>	INTEGER。最大特異値の <i>rcond</i> 倍より大きい <i>A</i> の特異値の個数。
<i>work</i>	REAL (slalsd の場合) DOUBLE PRECISION (dlalsd の場合) COMPLEX (clalsd の場合) COMPLEX*16 (zlalsd の場合) ワークスペース配列。 実数型の場合、次元は $(9n+2n*smlsiz+8n*nlvl+n*nrhs+(smlsiz+1)^2)$ 以上。 ここで、 $nlvl = \max(0, \text{int}(\log_2(n / (smlsiz+1))) + 1)$ 複素型の場合、次元は ($n*nrhs$) 以上。

<i>rwork</i>	REAL (clalsd の場合) DOUBLE PRECISION (zlalsd の場合) ワークスペース配列、複素型でのみ使用される。次元は $(9n + 2n * smlsiz + 8n * nlv1 + 3 * smlsiz * nrhs + (smlsiz + 1)^2)$ 以上。 ここで、 $nlv1 = \max(0, \text{int}(\log_2(\min(m, n) / (smlsiz + 1))) + 1)$
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(3n * nlv1 + 11n)$ 以上。

出力パラメーター

<i>d</i>	<i>info</i> = 0 の場合、 <i>d</i> には二重対角行列の特異値が格納される。
<i>b</i>	<i>b</i> には解 <i>X</i> が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正であったことを示す。 <i>info</i> = <i>i</i> > 0 の場合、 <i>info</i> / (<i>n</i> + 1) から $\text{mod}(\text{info}, n + 1)$ までの行と列にある部分行列の操作中に、アルゴリズムが特異値の計算に失敗したことを示す。

?lamrg

2 つの独立するソートされたセットの成分を単一のソートされたセットに昇順で併合する置換リストを生成する。

構文

```
call slamrg( n1, n2, a, strd1, strd2, index )
call dlamrg( n1, n2, a, strd1, strd2, index )
```

説明

このルーチンは、*a* (2 つの独立したソートされたセットで構成) の成分を単一のソートされたセットに昇順で併合する置換リストを生成する。

入力パラメーター

<i>n1</i> , <i>n2</i>	INTEGER。 これら引数には、併合対象となる 2 つのソートされたリストのそれぞれの長さを格納する。
<i>a</i>	REAL (slamrg の場合) DOUBLE PRECISION (dlamrg の場合) 配列、次元は (<i>n1</i> + <i>n2</i>) <i>a</i> の最初の <i>n1</i> 個の成分には、昇順または降順のどちらかでソートされる数値のリストを格納する。最後の <i>n2</i> 個の成分も同様である。

`strd1, strd2` INTEGER。
配列 `a` を介して取られるストライドである。可能なストライドは 1 と -1 である。これらは `a` の部分集合が昇順 (`strdx = 1`)、または降順 (`strdx = -1`) でソートされることを示す。

出力パラメーター

`index` INTEGER。
配列、次元は $(n1+n2)$ 。
 $i = 1, n1+n2$ に対して $b(i) = a(index(i))$ ならば、この配列には置換が格納され、`b` は昇順でソートされる。

?langb

一般帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slangb( norm, n, kl, ku, ab, ldab, work )
val = dlangb( norm, n, kl, ku, ab, ldab, work )
val = clangb( norm, n, kl, ku, ab, ldab, work )
val = zlangb( norm, n, kl, ku, ab, ldab, work )
```

説明

この関数は、 kl 個の劣対角と ku 個の優対角を持つ $n \times n$ 帯行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F' 'f', 'E' または 'e'
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチンが返す値を上述のように指定する。
`n` INTEGER。行列 A の次数。 $n \geq 0$ 。 $n = 0$ と指定した場合、`?langb` はゼロに設定される。
`kl` INTEGER。行列 A の劣対角成分の数。 $kl \geq 0$

<i>ku</i>	INTEGER。行列 <i>A</i> の優対角成分の数。 $ku \geq 0$
<i>ab</i>	REAL (<i>slangb</i> の場合) DOUBLE PRECISION (<i>dlangb</i> の場合) COMPLEX (<i>clangb</i> の場合) COMPLEX*16 (<i>zlangb</i> の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。帯行列 <i>A</i> で、行 1 から $k1+ku+1$ に格納する。 <i>A</i> の <i>j</i> 番目の列を配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(n, j+k1)$ に対して $ab(ku+1+i-j, j) = a(i, j)$ の場合。
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq k1+ku+1$
<i>work</i>	REAL (<i>slangb/clangb</i> の場合) DOUBLE PRECISION (<i>dlangb/zlangb</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。 <i>norm</i> = 'I' のとき $lwork \geq n$ 。それ以外では <i>work</i> は参照されない。

出力パラメーター

<i>val</i>	REAL (<i>slangb/clangb</i> の場合) DOUBLE PRECISION (<i>dlangb/zlangb</i> の場合) 関数の戻り値。
------------	---

?lange

一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slange( norm, m, n, a, lda, work )
val = dlange( norm, m, n, a, lda, work )
val = clangb( norm, m, n, a, lda, work )
val = zlangb( norm, m, n, a, lda, work )
```

説明

関数 ?lange は、実数 / 複素行列 *A* の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
```

= normF(A)、norm = 'F', 'f', 'E' または 'e' の場合

norm1 は行列の 1- ノルム (最大列合計)、normI は行列の無限ノルム (最大行合計)、normF は行列の Frobenius ノルム (二乗和の平方根) を示す。max(abs(A_{ij})) は行列ノルムではない点に注意する。

入力パラメーター

norm	CHARACTER*1。ルーチン ?lange が返す値を上述のように指定する。
m	INTEGER。行列 A の行数。m ≥ 0。m = 0 と指定した場合、?lange はゼロに設定される。
n	INTEGER。行列 A の列数。n ≥ 0。n = 0 と指定した場合、?lange はゼロに設定される。
a	REAL (slange の場合) DOUBLE PRECISION (dlange の場合) COMPLEX (clange の場合) COMPLEX*16 (zlange の場合) 配列、次元は (lda, n)。m × n の行列 A
lda	INTEGER。配列 a のリーディング・ディメンジョン。 lda ≥ max(m, 1)
work	REAL (slange と clange の場合) DOUBLE PRECISION (dlange と zlange の場合) ワークスペース配列、次元は (lwork)。 norm = 'I' のとき lwork ≥ m。それ以外では work は参照されない。

出力パラメーター

val	REAL (slange/clange の場合) DOUBLE PRECISION (dlange/zlange の場合) 関数の戻り値。
-----	---

?langt

一般三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slangt( norm, n, dl, d, du )
val = dlangt( norm, n, dl, d, du )
val = clangt( norm, n, dl, d, du )
val = zlangt( norm, n, dl, d, du )
```

説明

このルーチンは、実数 / 複素三重対角行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	CHARACTER*1。ルーチン <code>?langt</code> が返す値を上述のように指定する。
<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$ 。 $n = 0$ と指定した場合、 <code>?langt</code> はゼロに設定される。
<code>d1, d, du</code>	REAL (<code>slangt</code> の場合) DOUBLE PRECISION (<code>dlangt</code> の場合) COMPLEX (<code>clangt</code> の場合) COMPLEX*16 (<code>zlangt</code> の場合) 配列 : $d1(n-1), d(n), du(n-1)$ 。 配列 $d1$ には A の $(n-1)$ 個の劣対角成分を格納する。 配列 d には A の対角成分を格納する。 配列 du には A の $(n-1)$ 個の優対角成分を格納する。

出力パラメーター

<code>val</code>	REAL (<code>slangt/clangt</code> の場合) DOUBLE PRECISION (<code>dlangt/zlangt</code> の場合) 関数の戻り値。
------------------	---

?lanhs

上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slanh( norm, n, a, lda, work )
val = dlanh( norm, n, a, lda, work )
val = clanh( norm, n, a, lda, work )
```

```
val = zlanhs( norm, n, a, lda, work )
```

説明

関数 `?lanhs` は、Hessenberg 行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチン `?lanhs` が返す値を上述のように指定する。

`n` INTEGER。行列 A の次数。 $n \geq 0$ 。 $n = 0$ と指定した場合、`?lanhs` はゼロに設定される。

`a` REAL (`slanhs` の場合)
DOUBLE PRECISION (`dlanhs` の場合)
COMPLEX (`clanhs` の場合)
COMPLEX*16 (`zlanhs` の場合)
配列、次元は (lda, n) 。 $n \times n$ の上 Hessenberg 行列 A 。 A のうち最初の劣対角から下の部分は参照されない。

`lda` INTEGER。配列 `a` のリーディング・ディメンジョン。
 $lda \geq \max(n, 1)$

`work` REAL (`slanhs` と `clanhs` の場合)
DOUBLE PRECISION (`dlanhs` と `zlanhs` の場合)
ワークスペース配列、次元は $(lwork)$ 。
`norm = 'I'` のとき $lwork \geq n$ 。 それ以外では `work` は参照されない。

出力パラメーター

`val` REAL (`slanhs/clanhs` の場合)
DOUBLE PRECISION (`dlanhs/zlanhs` の場合)
関数の戻り値。

?lansb

対称帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slansb( norm, uplo, n, k, ab, ldab, work )
val = dlansb( norm, uplo, n, k, ab, ldab, work )
val = clansb( norm, uplo, n, k, ab, ldab, work )
val = zlansb( norm, uplo, n, k, ab, ldab, work )
```

説明

関数 ?lansb は、 k 個の優対角を持つ $n \times n$ 実数 / 複素対称帯行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。 `max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	CHARACTER*1。ルーチン ?lansb が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。帯行列 A の上三角部分または下三角部分のどちらを与えるか指定する。 <code>uplo = 'U'</code> の場合、上三角部分を与える。 <code>uplo = 'L'</code> の場合、下三角部分を与える。
<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lansb はゼロに設定される。
<code>k</code>	INTEGER。帯行列 A の優対角または劣対角の個数。 $k \geq 0$
<code>ab</code>	REAL (slansb の場合) DOUBLE PRECISION (dlansb の場合) COMPLEX (clansb の場合) COMPLEX*16 (zlansb の場合) 配列、次元は (<code>ldab</code> , <code>n</code>)。対称帯行列 A の上三角または下三角で、 <code>ab</code> の最初の $k+1$ 行に格納する。 A の j 番目の列を配列 <code>ab</code> の j 番目の列に次のように格納する。

$uplo = 'U'$ の場合、 $\max(1, j-k) \leq i \leq j$ に対して
 $ab(k+1+i-j, j) = a(i, j)$

$uplo = 'L'$ の場合、 $j \leq i \leq \min(n, j+k)$ に対して
 $ab(1+i-j, j) = a(i, j)$

ldab INTEGER。配列 *ab* のリーディング・ディメンジョン。
 $ldab \geq k+1$

work REAL (*slansb* と *clansb* の場合)
 DOUBLE PRECISION (*dlansb* と *zlansb* の場合)
 ワークスペース配列、次元は (*lwork*)。
 $norm = 'I', '1', 'O'$ のいずれかであるとき $lwork \geq n$ 。それ以外では *work* は参照されない。

出力パラメーター

val REAL (*slansb/clansb* の場合)
 DOUBLE PRECISION (*dlansb/zlansb* の場合)
 関数の戻り値。

?lanhb

エルミート帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = clanhb( norm, uplo, n, k, ab, ldab, work )
val = zlanhb( norm, uplo, n, k, ab, ldab, work )
```

説明

このルーチンは、 k 個の優対角を持つ $n \times n$ エルミート帯行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は *val* に格納される。

$val = \max(\text{abs}(A_{ij}))$ 、 $norm = 'M'$ または $'m'$ の場合
 $= \text{norm}_1(A)$ 、 $norm = '1'$ または $'O'$ または $'o'$ の場合
 $= \text{norm}_I(A)$ 、 $norm = 'I'$ または $'i'$ の場合
 $= \text{norm}_F(A)$ 、 $norm = 'F', 'f', 'E'$ または $'e'$ の場合

norm_1 は行列の 1- ノルム (最大列合計)、 norm_I は行列の無限ノルム (最大行合計)、 norm_F は行列の Frobenius ノルム (二乗和の平方根) を示す。 $\max(\text{abs}(A_{ij}))$ は行列ノルムではない点に注意する。

入力パラメーター

<i>norm</i>	CHARACTER*1。ルーチン ?lanhb が返す値を上述のように指定する。
<i>uplo</i>	CHARACTER*1。帯行列 <i>A</i> の上三角部分または下三角部分のどちらを与えるか指定する。 <i>uplo</i> = 'U' の場合、上三角部分を与える。 <i>uplo</i> = 'L' の場合、下三角部分を与える。
<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lanhb はゼロに設定される。
<i>k</i>	INTEGER。帯行列 <i>A</i> の優対角または劣対角の個数。 $k \geq 0$
<i>ab</i>	COMPLEX (clanhb の場合) COMPLEX*16 (zlanhb の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。対称帯行列 <i>A</i> の上三角または下三角で、 <i>ab</i> の最初の <i>k</i> +1 行に格納する。 <i>A</i> の <i>j</i> 番目の列を配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 <i>uplo</i> = 'U' の場合、 $\max(1, j-k) \leq i \leq j$ に対して $ab(k+1+i-j, j) = a(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq \min(n, j+k)$ に対して $ab(1+i-j, j) = a(i, j)$ 対角成分の虚数部分は設定する必要はなくゼロとして仮定される点に注意する。
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq k+1$
<i>work</i>	REAL (clanhb の場合) DOUBLE PRECISION (zlanhb の場合) ワークスペース配列、次元は (<i>lwork</i>)。 <i>norm</i> = 'I'、'L'、'O' のいずれかであるとき $lwork \geq n$ 。それ以外では <i>work</i> は参照されない。

出力パラメーター

<i>val</i>	REAL (slanhb/clanhb の場合) DOUBLE PRECISION (dlanhb/zlanhb の場合) 関数の戻り値。
------------	---

?lansp

圧縮形式で与えられる対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slansp( norm, uplo, n, ap, work )
val = dlansp( norm, uplo, n, ap, work )
val = clansp( norm, uplo, n, ap, work )
val = zlansp( norm, uplo, n, ap, work )
```

説明

関数 ?lansp は、圧縮形式で与えられる実数 / 複素対称行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。 `max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	CHARACTER*1。ルーチン ?lansp が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。対称行列 A の上三角部分または下三角部分のどちらを与えるか指定する。 <code>uplo = 'U'</code> の場合、 A の上三角部分を与える。 <code>uplo = 'L'</code> の場合、 A の下三角部分を与える。
<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lansp はゼロに設定される。
<code>ap</code>	REAL (slansp の場合) DOUBLE PRECISION (dlansp の場合) COMPLEX (clansp の場合) COMPLEX*16 (zlansp の場合) 配列、次元は $(n(n+1)/2)$ 。線形配列に列方向に圧縮された対称行列 A の上三角または下三角。 A の j 番目の列は配列 <code>ap</code> に次のように格納する。 <code>uplo = 'U'</code> の場合、 $1 \leq i \leq j$ に対して <code>ap(i + (j-1)j/2) = A(i,j)</code> <code>uplo = 'L'</code> の場合、 $j \leq i \leq n$ に対して <code>ap(i + (j-1)(2n-j)/2) = A(i,j)</code>

`work` REAL (slansp と clansp の場合)
 DOUBLE PRECISION (dlansp と zlansp の場合)
 ワークスペース配列、次元は (`lwork`)。
`norm = 'I', '1', 'O'` のいずれかであるとき $lwork \geq n$ 。それ以外では `work` は参照されない。

出力パラメーター

`val` REAL (slansp/clansp の場合)
 DOUBLE PRECISION (dlansp/zlansp の場合)
 関数の戻り値。

?lanhp

圧縮形式で与えられる複素エルミート行列の
 1- ノルムの値、Frobenius ノルムの値、無限ノル
 ムの値、あるいは最大絶対値の成分を返す。

構文

```
val = clanhp( norm, uplo, n, ap, work )
val = zlanhp( norm, uplo, n, ap, work )
```

説明

関数 ?lanhp は、圧縮形式で与えられる複素エルミート行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F' 'f', 'E' または 'e'
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチン ?lanhp が返す値を上述のように指定する。

`uplo` CHARACTER*1。エルミート行列 A の上三角部分または下三角部分のどちらを与えるか指定する。
`uplo = 'U'` の場合、 A の上三角部分を与える。
`uplo = 'L'` の場合、 A の下三角部分を与える。

<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lanhp</code> はゼロに設定される。
<code>ap</code>	COMPLEX (<code>clanhp</code> の場合) COMPLEX*16 (<code>zlanhp</code> の場合) 配列、次元は $(n(n+1)/2)$ 。線形配列に列方向に圧縮されたエルミート行列 A の上三角または下三角。 A の j 番目の列は配列 <code>ap</code> に次のように格納する。 <code>uplo = 'U'</code> の場合、 $1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i, j)$ <code>uplo = 'L'</code> の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i, j)$
<code>work</code>	REAL (<code>clanhp</code> の場合) DOUBLE PRECISION (<code>zlanhp</code> の場合) ワークスペース配列、次元は (<code>lwork</code>)。 <code>norm = 'I', '1', 'O'</code> のいずれかであるとき $lwork \geq n$ 。それ以外では <code>work</code> は参照されない。

出力パラメーター

<code>val</code>	REAL (<code>clanhp</code> の場合) DOUBLE PRECISION (<code>zlanhp</code> の場合) 関数の戻り値。
------------------	---

?lanst/?lanht

実数対称または複素エルミート三重対角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slanst( norm, n, d, e )
val = dlanst( norm, n, d, e )
val = clanht( norm, n, d, e )
val = zlanht( norm, n, d, e )
```

説明

関数 `?lanst/?lanht` は、実数対称または複素エルミート三重対角行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	CHARACTER*1。ルーチン <code>?lanst/?lanht</code> が返す値を上述のように指定する。
<code>n</code>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$ $n = 0$ と指定した場合、 <code>?lanst/?lanht</code> はゼロに設定される。
<code>d</code>	REAL (<code>slanst/clanht</code> の場合) DOUBLE PRECISION (<code>dlanst/zlanht</code> の場合) 配列、次元は (<i>n</i>)。 <i>A</i> の対角成分。
<code>e</code>	REAL (<code>slanst</code> の場合) DOUBLE PRECISION (<code>dlanst</code> の場合) COMPLEX (<code>clanht</code> の場合) COMPLEX*16 (<code>zlanht</code> の場合) 配列、次元は (<i>n</i> -1)。 <i>A</i> の (<i>n</i> -1) 個の劣対角成分または優対角成分。

出力パラメーター

<code>val</code>	REAL (<code>slanst/clanht</code> の場合) DOUBLE PRECISION (<code>dlanst/zlanht</code> の場合) 関数の戻り値。
------------------	---

?lansy

実数 / 複素対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slansy( norm, uplo, n, a, lda, work )
val = dlansy( norm, uplo, n, a, lda, work )
val = clansy( norm, uplo, n, a, lda, work )
val = zlansy( norm, uplo, n, a, lda, work )
```

説明

関数 `?lansy` は、実数 / 複素対称行列 *A* の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

`val = max(abs(Aij))`、`norm = 'M'` または `'m'` の場合

= $\text{norm}_1(A)$ 、 $\text{norm} = '1'$ または $'O'$ または $'o'$ の場合

= $\text{norm}_I(A)$ 、 $\text{norm} = 'I'$ または $'i'$ の場合

= $\text{norm}_F(A)$ 、 $\text{norm} = 'F', 'f', 'E'$ または $'e'$ の場合

norm_1 は行列の 1- ノルム (最大列合計)、 norm_I は行列の無限ノルム (最大行合計)、 norm_F は行列の Frobenius ノルム (二乗和の平方根) を示す。 $\max(\text{abs}(A_{ij}))$ は行列ノルムではない点に注意する。

入力パラメーター

<i>norm</i>	CHARACTER*1。ルーチン ?lansy が返す値を上述のように指定する。
<i>uplo</i>	CHARACTER*1。対称行列 A の上三角部分と下三角部分のどちらを参照させるか指定する。 = 'U' の場合、 A の上三角部分を参照させる。 = 'L' の場合、 A の下三角部分を参照させる。
<i>n</i>	INTEGER。行列 A の次数。 $n \geq 0$ $n=0$ と指定した場合、?lansy はゼロに設定される。
<i>a</i>	REAL (slansy の場合) DOUBLE PRECISION (dlansy の場合) COMPLEX (clansy の場合) COMPLEX*16 (zlansy の場合) 配列、次元は (lda, n)。対称行列 A 。 $uplo = 'U'$ の場合、 a の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 a の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 a の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。配列 a のリーディング・ディメンジョン。 $lda \geq \max(n, 1)$
<i>work</i>	REAL (slansy と clansy の場合) DOUBLE PRECISION (dlansy と zlansy の場合) ワークスペース配列、次元は ($lwork$)。 $norm = 'I', '1', 'O'$ のいずれかであるとき $lwork \geq n$ 。それ以外では $work$ は参照されない。

出力パラメーター

<i>val</i>	REAL (slansy/clansy の場合) DOUBLE PRECISION (dlansy/zlansy の場合) 関数の戻り値。
------------	---

?lanhe

複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = clanhe( norm, uplo, n, a, lda, work )
val = zlanhe( norm, uplo, n, a, lda, work )
```

説明

関数 ?lanhe は、複素エルミート行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	CHARACTER*1。ルーチン ?lanhe が返す値を上述のように指定する。
<code>uplo</code>	CHARACTER*1。エルミート行列 A の上三角部分と下三角部分のどちらを参照させるか指定する。 = 'U' の場合、 A の上三角部分を参照させる。 = 'L' の場合、 A の下三角部分を参照させる。
<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$ $n = 0$ と指定した場合、?lanhe はゼロに設定される。
<code>a</code>	COMPLEX (clanhe の場合) COMPLEX*16 (zlanhe の場合) 配列、次元は (lda, n) 。エルミート行列 A 。 <code>uplo = 'U'</code> の場合、 a の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 <code>uplo = 'L'</code> の場合、 a の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 a の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。配列 a のリーディング・ディメンジョン。 $lda \geq \max(n, 1)$

`work` REAL (`clanhe` の場合)
 DOUBLE PRECISION (`zlanhe` の場合)
 ワークスペース配列、次元は (`lwork`)。
`norm = 'I', '1', 'O'` のいずれかであるとき $lwork \geq n$ 。それ以外では `work` は参照されない。

出力パラメーター

`val` REAL (`clanhe` の場合)
 DOUBLE PRECISION (`zlanhe` の場合)
 関数の戻り値。

?lantb

三角帯行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slantb( norm, uplo, diag, n, k, ab, ldab, work )
val = dlantb( norm, uplo, diag, n, k, ab, ldab, work )
val = clantb( norm, uplo, diag, n, k, ab, ldab, work )
val = zlantb( norm, uplo, diag, n, k, ab, ldab, work )
```

説明

関数 ?lantb は、 $(k+1)$ 個の対角を持つ $n \times n$ 三角帯行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチン ?lantb が返す値を上述のように指定する。

`uplo` CHARACTER*1。行列 A が上三角か下三角かを指定する。
 = 'U' の場合、上三角
 = 'L' の場合、下三角。

<i>diag</i>	CHARACTER*1。行列 <i>A</i> が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$ 。 $n = 0$ と指定した場合、 ?lantb はゼロに設定される。
<i>k</i>	INTEGER。 <i>uplo</i> = 'U' の場合は行列 <i>A</i> の優対角成分の数、 <i>uplo</i> = 'L' の場合は行列 <i>A</i> の劣対角成分の数。 $k \geq 0$
<i>ab</i>	REAL (<i>slantb</i> の場合) DOUBLE PRECISION (<i>dlantb</i> の場合) COMPLEX (<i>clantb</i> の場合) COMPLEX*16 (<i>zlantb</i> の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。上または下三角帯行列 <i>A</i> で、 <i>ab</i> の最初の <i>k</i> +1 行に格納する。 <i>A</i> の <i>j</i> 番目の列を配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 <i>uplo</i> = 'U' の場合、 $\max(1, j-k) \leq i \leq j$ に対して $ab(k+1+i-j, j) = a(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq \min(n, j+k)$ に対して $ab(1+i-j, j) = a(i, j)$ <i>diag</i> = 'U' の場合、行列 <i>A</i> の対角成分に対応する配列 <i>ab</i> の成分は参照されないが、1 として仮定される。
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq k+1$
<i>work</i>	REAL (<i>slantb</i> と <i>clantb</i> の場合) DOUBLE PRECISION (<i>dlantb</i> と <i>zlantb</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。 <i>norm</i> = 'I' のとき $lwork \geq n$ 。それ以外では <i>work</i> は参照されない。

出力パラメーター

<i>val</i>	REAL (<i>slantb</i> / <i>clantb</i> の場合) DOUBLE PRECISION (<i>dlantb</i> / <i>zlantb</i> の場合) 関数の戻り値。
------------	---

?lantp

圧縮形式で与えられる対称行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slantp( norm, uplo, diag, n, ap, work )
val = dlantp( norm, uplo, diag, n, ap, work )
val = clantp( norm, uplo, diag, n, ap, work )
```

```
val = zlantp( norm, uplo, diag, n, ap, work )
```

説明

関数 ?lantp は、圧縮形式で与えられる三角行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F', 'f', 'E' または 'e' の場合
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチン ?lantp が返す値を上述のように指定する。

`uplo` CHARACTER*1。行列 A が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角。

`diag` CHARACTER*1。行列 A が単位三角かどうかを指定する。
= 'N' の場合、単位三角ではない。
= 'U' の場合、単位三角。

`n` INTEGER。行列 A の次数。
 $n \geq 0$ 。 $n = 0$ と指定した場合、?lantp はゼロに設定される。

`ap` REAL (slantp の場合)
DOUBLE PRECISION (dlantp の場合)
COMPLEX (clantp の場合)
COMPLEX*16 (zlantp の場合)
配列、次元は $(n(n+1)/2)$ 。上三角または下三角帯行列 A で、線形配列にカラム方向に圧縮されている。 A の j 番目の列は配列 `ap` に次のように格納する。
`uplo = 'U'` の場合、 $1 \leq i \leq j$ に対して $AP(i + (j-1)j/2) = a(i, j)$
`uplo = 'L'` の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = a(i, j)$
`diag = 'U'` の場合、行列 A の対角成分に対応する配列 `ap` の成分は参照されないが、1 として仮定される。

`work` REAL (slantp と clantp の場合)
DOUBLE PRECISION (dlantp と zlantp の場合)
ワークスペース配列、次元は (`lwork`)。
`norm = 'I'` のとき $lwork \geq n$ 。それ以外では `work` は参照されない。

出力パラメーター

`val` REAL (slantr/clantr の場合)
DOUBLE PRECISION (dlantr/zlantr の場合)
関数の戻り値。

?lantr

台形または三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = slantr( norm, uplo, diag, m, n, a, lda, work )
val = dlantr( norm, uplo, diag, m, n, a, lda, work )
val = clantr( norm, uplo, diag, m, n, a, lda, work )
val = zlantr( norm, uplo, diag, m, n, a, lda, work )
```

説明

関数 ?lantr は、台形または三角行列 A の、1- ノルムの値、または Frobenius ノルムの値、または無限ノルムの値、または最大絶対値の成分を返す。

関数の戻り値は `val` に格納される。

```
val = max(abs(Aij)), norm = 'M' または 'm' の場合
      = norm1(A), norm = '1' または 'O' または 'o' の場合
      = normI(A), norm = 'I' または 'i' の場合
      = normF(A), norm = 'F' 'f', 'E' または 'e'
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(Aij))` は行列ノルムではない点に注意する。

入力パラメーター

`norm` CHARACTER*1。ルーチン ?lantr が返す値を上述のように指定する。

`uplo` CHARACTER*1。行列 A が上三角か下三角かを指定する。
'U' の場合、上台形。
'L' の場合、下台形。
 $m = n$ の場合、 A は台形ではなく三角であることに注意する。

`diag` CHARACTER*1。行列 A が単位三角かどうかを指定する。
'N' の場合、単位対角ではない。
'U' の場合、単位対角。

<i>m</i>	INTEGER。行列 <i>A</i> の行数。 $m \geq 0$ 、 <i>uplo</i> = 'U' の場合はさらに $m \leq n$ 、 $m = 0$ と指定した場合、 <i>?lantr</i> はゼロに設定される。
<i>n</i>	INTEGER。行列 <i>A</i> の列数。 $n \geq 0$ 、 <i>uplo</i> = 'L' の場合はさらに $n \leq m$ 、 $n = 0$ と指定した場合、 <i>?lantr</i> はゼロに設定される。
<i>a</i>	REAL (<i>slantr</i> の場合) DOUBLE PRECISION (<i>dlantr</i> の場合) COMPLEX (<i>clantr</i> の場合) COMPLEX*16 (<i>zlantr</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 台形行列 <i>A</i> ($m = n$ の場合 <i>A</i> は三角)。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> の先頭の $m \times n$ 上台形部分には上台形行列を格納する。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> の先頭の $m \times n$ 上台形部分には上台形行列を格納する。厳密な下三角部分は参照されない。 <i>diag</i> = 'U' の場合、 <i>a</i> の対角成分は参照されず 1 と仮定される点に注意する。
<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(m, 1)$
<i>work</i>	REAL (<i>slantr/clantrp</i> の場合) DOUBLE PRECISION (<i>dlantr/zlantr</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。 <i>norm</i> = 'I' のとき $lwork \geq m$ 。それ以外では <i>work</i> は参照されない。

出力パラメーター

<i>val</i>	REAL (<i>slantr/clantrp</i> の場合) DOUBLE PRECISION (<i>dlantr/zlantr</i> の場合) 関数の戻り値。
------------	--

?lanv2

2×2 実数非対称行列の Schur 因子分解を標準形式で計算する。

構文

```
call slanv2( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
call dlanv2( a, b, c, d, rt1r, rt1i, rt2r, rt2i, cs, sn )
```

説明

このルーチンは 2×2 実数非対称行列の Schur 因子分解を標準形式で計算する。

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} cs & -sn \\ sn & cs \end{bmatrix} \begin{bmatrix} aa & bb \\ cc & dd \end{bmatrix} \begin{bmatrix} cs & sn \\ -sn & cs \end{bmatrix}$$

ここで、次のいずれかである。

1. aa と dd が行列の実数固有値となるように $cc = 0$ 、または、
2. $aa \pm \sqrt{bb*cc}$ が複素共役固有値となるように $aa = dd$ と $bb*cc < 0$

このルーチンは、実数固有値の計算において桁落ち誤差のリスクを低減するために、また可能ならば $\text{abs}(rt1r) \geq \text{abs}(rt2r)$ を保証するために調整されている。

入力パラメーター

a, b, c, d REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
入力行列の成分。

出力パラメーター

a, b, c, d 標準化された Schur 形式の成分で上書きされる。
 $rt1r, rt1i, rt2r, rt2i$ REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
固有値の実数部と虚数部。固有値が複素共役ペアの場合、 $rt1i > 0$
 cs, sn REAL (slanv2 の場合)
DOUBLE PRECISION (dlanv2 の場合)
回転行列のパラメーター

?lapll

2 つのベクトルの線形従属性を測定する。

構文

```
call slapll( n, x, incx, y, incy, ssmin )
call dlapll( n, x, incx, y, incy, ssmin )
call clapll( n, x, incx, y, incy, ssmin )
call zlapll( n, x, incx, y, incy, ssmin )
```

説明

長さ n の 2 つの列ベクトル x と y が与えられ、

$A = (x \ y)$ を $n \times 2$ 行列とする。

ルーチン ?lapll は最初に A の QR 因子分解を $A = QR$ として計算し、次に 2×2 上三角行列 R の SVD を計算する。 R の小さい方の特異値は $ssmin$ で返され、ベクトル x と y の線形従属性の測定結果として使用される。

入力パラメーター

n INTEGER。ベクトル x およびベクトル y の長さ。

<i>x</i>	REAL (slap11 の場合) DOUBLE PRECISION (dlap11 の場合) COMPLEX (clap11 の場合) COMPLEX*16 (zlap11 の場合) 配列、次元は $(1+(n-1)incx)$ 。 <i>x</i> には <i>n</i> -ベクトル <i>x</i> を格納する。
<i>y</i>	REAL (slap11 の場合) DOUBLE PRECISION (dlap11 の場合) COMPLEX (clap11 の場合) COMPLEX*16 (zlap11 の場合) 配列、次元は $(1+(n-1)incy)$ 。 <i>y</i> には <i>n</i> -ベクトル <i>y</i> を格納する。
<i>incx</i>	INTEGER。 <i>x</i> の連続する成分間の増分で、 $incx > 0$
<i>incy</i>	INTEGER。 <i>y</i> の連続する成分間の増分で、 $incy > 0$

出力パラメーター

<i>x</i>	<i>x</i> は上書きされる。
<i>y</i>	<i>y</i> は上書きされる。
<i>ssmin</i>	REAL (slap11/clap11 の場合) DOUBLE PRECISION (dlap11/zlap11 の場合) $n \times 2$ の行列 $A = (x \ y)$ の最小固有値。

?lapmt

行列の列に対して順方向または逆方向置換を実行する。

構文

```
call slapmt( forwrd, m, n, x, ldx, k )
call dlapmt( forwrd, m, n, x, ldx, k )
call clapmt( forwrd, m, n, x, ldx, k )
call zlapmt( forwrd, m, n, x, ldx, k )
```

説明

ルーチン ?lapmt は、整数 $1, \dots, n$ の置換 $k(1), k(2), \dots, k(n)$ で指定されたとおりに、 $m \times n$ 行列 *X* の列を再配置する。

forwrd = .TRUE. の場合、順方向置換である。

$j = 1, 2, \dots, n$ では、 $X(*, k(j))$ は $X(*, j)$ に移動される。

forwrd = .FALSE. の場合、逆方向置換である。

$j = 1, 2, \dots, n$ では、 $X(*, j)$ は $X(*, k(j))$ に移動される。

入力パラメーター

<i>forwrd</i>	LOGICAL。 <i>forwrd</i> = .TRUE. の場合、順方向置換である。 <i>forwrd</i> = .FALSE. の場合、逆方向置換である。
<i>m</i>	INTEGER。行列 <i>X</i> の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。行列 <i>X</i> の列数。 $n \geq 0$ 。
<i>x</i>	REAL (<i>slapmt</i> の場合) DOUBLE PRECISION (<i>dlapmt</i> の場合) COMPLEX (<i>clapmt</i> の場合) COMPLEX*16 (<i>zlapmt</i> の場合) 配列、次元は (<i>ldx</i> , <i>n</i>)。 $m \times n$ 行列 <i>X</i> を格納する。
<i>ldx</i>	INTEGER。配列 <i>x</i> のリーディング・ディメンション。 $ldx \geq \max(1, m)$ 。
<i>k</i>	INTEGER。 配列、次元は (<i>n</i>)。 <i>k</i> には置換ベクトルを格納する。

出力パラメーター

<i>x</i>	<i>x</i> は置換された行列 <i>X</i> で上書きされる。
----------	-------------------------------------

?lapy2

$\text{sqrt}(x^2+y^2)$ を返す。

構文

```
val = slapy2( x, y )
val = dlapy2( x, y )
```

説明

関数 ?lapy2 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、 $\text{sqrt}(x^2+y^2)$ を返す。

入力パラメーター

<i>x, y</i>	REAL (<i>slapy2</i> の場合) DOUBLE PRECISION (<i>dlapy2</i> の場合) 入力値と <i>x</i> と <i>y</i> を指定する。
-------------	---

出力パラメーター

<i>val</i>	REAL (<i>slapy2</i> の場合) DOUBLE PRECISION (<i>dlapy2</i> の場合) 関数の戻り値。
------------	---

?lapy3

$\text{sqrt}(x^2+y^2+z^2)$ を返す。

構文

```
val = slapy3( x, y, z )
```

```
val = dlapy3( x, y, z )
```

説明

関数 ?lapy3 は、無用なオーバーフローや有害なアンダーフローを発生させないようにしながら、 $\text{sqrt}(x^2+y^2+z^2)$ を返す。

入力パラメーター

x, *y*, *z* REAL (slapy3 の場合)
 DOUBLE PRECISION (dlapy3 の場合)
 入力値 *x*、*y*、*z* を指定する。

出力パラメーター

val REAL (slapy3 の場合)
 DOUBLE PRECISION (dlapy3 の場合)
 関数の戻り値。

?laqgb

?gbequ で計算された行と列のスケール係数を使って一般帯行列をスケールリングする。

構文

```
call slaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )
```

```
call dlaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )
```

```
call claqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )
```

```
call zlaqgb( m, n, kl, ku, ab, ldab, r, c, rowcnd, colcnd, amax, equed )
```

説明

このルーチンは、*kl* 個の劣対角と *ku* 個の優対角を持つ一般 $m \times n$ 帯行列 *A* を、ベクトル *r* と *c* に格納されている行と列のスケール係数を用いて平衡化する。

入力パラメーター

m INTEGER。行列 *A* の行数。 $m \geq 0$ 。
n INTEGER。行列 *A* の列数。 $n \geq 0$
kl INTEGER。 *A* の帯内にある劣対角の数。 $kl \geq 0$

<i>ku</i>	INTEGER。A の帯内にある優対角の数。 $ku \geq 0$
<i>ab</i>	REAL (<i>slaqgb</i> の場合) DOUBLE PRECISION (<i>dlaqgb</i> の場合) COMPLEX (<i>claqgb</i> の場合) COMPLEX*16 (<i>zlaqgb</i> の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。行列 A を帯格納形式で行 1 から $k1+ku+1$ に格納する。A の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(m, j+k1)$ に対して $ab(ku+1+i-j, j) = A(i, j)$
<i>ldab</i>	INTEGER。配列 <i>ab</i> のリーディング・ディメンジョン。 $lda \geq k1+ku+1$
<i>amax</i>	REAL (<i>slaqgb/claqgb</i> の場合) DOUBLE PRECISION (<i>dlaqgb/zlaqgb</i> の場合) 最大行列成分の絶対値。

出力パラメーター

<i>ab</i>	A と同じ格納形式の平衡化行列で上書きされる。 平衡化された行列の形式は <i>equed</i> を参照のこと。
<i>r, c</i>	REAL (<i>slaqgb/claqgb</i> の場合) DOUBLE PRECISION (<i>dlaqgb/zlaqgb</i> の場合) 配列: <i>r</i> (<i>m</i>), <i>c</i> (<i>n</i>)。それぞれ、A に対する行と列のスケール係数。
<i>rowcnd</i>	REAL (<i>slaqgb/claqgb</i> の場合) DOUBLE PRECISION (<i>dlaqgb/zlaqgb</i> の場合) 最小の $r(i)$ を最大の $r(i)$ で割った値。
<i>colcnd</i>	REAL (<i>slaqgb/claqgb</i> の場合) DOUBLE PRECISION (<i>dlaqgb/zlaqgb</i> の場合) 最小の $c(i)$ を最大の $c(i)$ で割った値。
<i>equed</i>	CHARACTER*1。 実行された平衡化の形式を表す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'R' の場合、行の平衡化、すなわち、A は <i>diag(r)</i> で事前乗算された。 <i>equed</i> = 'C' の場合、列の平衡化、すなわち、A は <i>diag(c)</i> で事後乗算された。 <i>equed</i> = 'B' の場合、行と列の平衡化、すなわち、A は $diag(r)*A*diag(c)$ で置き換えられた。

アプリケーション・ノート

このルーチンは内部パラメーター *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケールリングの実行判定に使用される。 $rowcnd < thresh$ の場合は行スケールリングが実行され、 $colcnd < thresh$ の場合は列スケールリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいた行スケールリングの実行判定に使用される。 $amax > large$ または $amax < small$ の場合に行スケールリングが実行される。

?laqge

?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。

構文

```
call slaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call dlaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call claqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
call zlaqge( m, n, a, lda, r, c, rowcnd, colcnd, amax, equed )
```

説明

このルーチンは、一般 $m \times n$ 行列 A を、ベクトル r と c に格納されている行と列のスケール係数を用いて平衡化する。

入力パラメーター

m	INTEGER。行列 A の行数。 $m \geq 0$ 。
n	INTEGER。行列 A の列数。 $n \geq 0$ 。
a	REAL (slaqge の場合) DOUBLE PRECISION (dlaqge の場合) COMPLEX (claqge の場合) COMPLEX*16 (zlaqge の場合) 配列、次元は (lda, n) 。 $m \times n$ の行列 A を格納する。
lda	INTEGER。配列 a のリーディング・ディメンジョン。 $lda \geq \max(m, 1)$
r	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 配列、次元は (m) 。 A に対する行スケール係数。
c	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 配列、次元は (n) 。 A に対する列スケール係数。
$rowcnd$	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 最小の $r(i)$ を最大の $r(i)$ で割った値。
$colcnd$	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 最小の $c(i)$ を最大の $c(i)$ で割った値。
$amax$	REAL (slanqge/claqge の場合) DOUBLE PRECISION (dlaqge/zlaqge の場合) 最大行列成分の絶対値。

出力パラメーター

<i>a</i>	平衡化された行列で上書きされる。 平衡化された行列の形式は <i>equed</i> を参照のこと。
<i>equed</i>	CHARACTER*1。 実行された平衡化の形式を表す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'R' の場合、行の平衡化、すなわち、 <i>A</i> は <i>diag(r)</i> で事前乗算された。 <i>equed</i> = 'C' の場合、列の平衡化、すなわち、 <i>A</i> は <i>diag(c)</i> で事後乗算された。 <i>equed</i> = 'B' の場合、行と列の平衡化、すなわち、 <i>A</i> は <i>diag(r)*A*diag(c)</i> で置き換えられた。

アプリケーション・ノート

このルーチンは内部パラメーター *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、行または列のスケール係数の比率にもとづいた行または列のスケールリングの実行判定に使用される。*rowcnd* < *thresh* の場合は行スケールリングが実行され、*colcnd* < *thresh* の場合は列スケールリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいた行スケールリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合に行スケールリングが実行される。

?laqp2

行列ブロックに対して列ピボット演算を用いた QR 因子分解を計算する。

構文

```
call slaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call dlaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call claqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
call zlaqp2( m, n, offset, a, lda, jpvt, tau, vn1, vn2, work )
```

説明

このルーチンはブロック *A*(*offset*+1:*m*, 1:*n*) に対して列ピボット演算を用いた QR 因子分解を計算する。ブロック *A*(1:*offset*, 1:*n*) はピボットされるが因子分解はされない。

入力パラメーター

<i>m</i>	INTEGER。行列 <i>A</i> の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。行列 <i>A</i> の列数。 $n \geq 0$ 。
<i>offset</i>	INTEGER。因子分解はしないがピボット演算すべき行列 <i>A</i> の行数。 $offset \geq 0$ 。

<i>a</i>	REAL (slaqp2 の場合) DOUBLE PRECISION (dlaqp2 の場合) COMPLEX (claqp2 の場合) COMPLEX*16 (zlaqp2 の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 $m \times n$ の行列 <i>A</i> を格納する。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
<i>jpvt</i>	INTEGER。 配列、次元は (<i>n</i>)。 呼び出し時に、 $jpvt(i) \neq 0$ の場合、 <i>A</i> の <i>i</i> 番目の列が $A*P$ の先頭 (先頭の列) に置換される。 $jpvt(i) = 0$ の場合、 <i>A</i> の <i>i</i> 番目の列はフリー列になる。
<i>vn1</i> , <i>vn2</i>	REAL (slaqp2/claqp2 の場合) DOUBLE PRECISION (dlaqp2/zlaqp2 の場合) 配列、次元はそれぞれ、(<i>n</i>)。 それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。
<i>work</i>	REAL (slaqp2 の場合) DOUBLE PRECISION (dlaqp2 の場合) COMPLEX (claqp2 の場合) COMPLEX*16 (zlaqp2 の場合) ワークスペース配列、次元は (<i>n</i>)。

出力パラメーター

<i>a</i>	ブロック <i>A</i> (<i>offset</i> +1: <i>m</i> , 1: <i>n</i>) の上三角には得られた三角係数が上書きされる。 対角より下のブロック <i>A</i> (<i>offset</i> +1: <i>m</i> , 1: <i>n</i>) の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交行列 <i>Q</i> を表現する。 ブロック <i>A</i> (1: <i>offset</i> , 1: <i>n</i>) はピボットされるが因子分解はされない。
<i>jpvt</i>	$jpvt(i) = k$ の場合、 <i>A</i> の <i>k</i> 番目の列が $A*P$ の <i>i</i> 番目の列になる。
<i>tau</i>	REAL (slaqp2 の場合) DOUBLE PRECISION (dlaqp2 の場合) COMPLEX (claqp2 の場合) COMPLEX*16 (zlaqp2 の場合) 配列、次元は ($\min(m,n)$)。 基本リフレクターのスカラー係数。
<i>vn1</i> , <i>vn2</i>	それぞれ部分的な列ノルムと完全な列ノルムを持つベクトルが格納される。

?laqps

BLAS レベル3 を使って実数 $m \times n$ 行列 A に対して列ピボット演算を用いた QR 因子分解のステップを計算する。

構文

```
call slaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call dlaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call claqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
call zlaqps( m, n, offset, nb, kb, a, lda, jpvt, tau, vn1, vn2, auxv, f, ldf )
```

説明

このルーチンは、BLAS レベル3 を使って、実数 $m \times n$ 行列 A に対して列ピボット演算を用いた QR 因子分解を計算する。このルーチンは、行 $offset+1$ から始めて A の nb 列の因子分解を試み、BLAS レベル3 ルーチン ?gemm を使って行列全体を更新する。

大幅な桁落ちの発生によって ?laqps が nb 列の因子分解を実行できない場合がある。実際に因子分解された列数は kb に返される。

ブロック $A(1:offset, 1:n)$ はピボットされるが因子分解はされない。

入力パラメーター

<i>m</i>	INTEGER。行列 A の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。行列 A の列数。 $n \geq 0$ 。
<i>offset</i>	INTEGER。前のステップですでに因子分解された A の行数。
<i>nb</i>	INTEGER。因子分解する列数。
<i>a</i>	REAL (slaqps の場合) DOUBLE PRECISION (dlaqps の場合) COMPLEX (claqps の場合) COMPLEX*16 (zlaqps の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 $m \times n$ の行列 A を格納する。
<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
<i>jpvt</i>	INTEGER。 配列、次元は (<i>n</i>)。 $jpvt(i) = k$ の場合、フル行列 A の列 k は AP の位置 i に置換されている。
<i>vn1</i> , <i>vn2</i>	REAL (slaqps/claqps の場合) DOUBLE PRECISION (dlaqps/zlaqps の場合) 配列、次元はそれぞれ、(<i>n</i>)。それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトルを格納する。

<i>auxv</i>	REAL (<i>slaqps</i> の場合) DOUBLE PRECISION (<i>dlaqps</i> の場合) COMPLEX (<i>claqps</i> の場合) COMPLEX*16 (<i>zlaqps</i> の場合) 配列、次元は (<i>nb</i>)。補助ベクトル。
<i>f</i>	REAL (<i>slaqps</i> の場合) DOUBLE PRECISION (<i>dlaqps</i> の場合) COMPLEX (<i>claqps</i> の場合) COMPLEX*16 (<i>zlaqps</i> の場合) 配列、次元は (<i>ldf, nb</i>)。行列 $F' = L * Y' * A$ 。
<i>ldf</i>	INTEGER。配列 <i>f</i> のリーディング・ディメンジョン。 $ldf \geq \max(1, n)$ 。

出力パラメーター

<i>kb</i>	INTEGER。実際に因子分解された列数。
<i>a</i>	ブロック A (<i>offset+1:m, 1:kb</i>) は得られた三角係数が上書きされる。また、ブロック A (<i>1:offset, 1:n</i>) はピボットされているが因子分解はされていない。 行列の残りの部分ブロック A (<i>offset+1:m, kb+1:n</i>) は更新されている。
<i>jpvt</i>	INTEGER 配列、次元は (<i>n</i>)。 $jpvt(i) = k$ の場合、フル行列 A の列 k は AP の位置 i に置換されている。
<i>tau</i>	REAL (<i>slaqps</i> の場合) DOUBLE PRECISION (<i>dlaqps</i> の場合) COMPLEX (<i>claqps</i> の場合) COMPLEX*16 (<i>zlaqps</i> の場合) 配列、次元は (<i>kb</i>)。基本リフレクターのスカラー係数。
<i>vn1, vn2</i>	それぞれ、部分的な列ノルムと完全な列ノルムを持つベクトル。
<i>auxv</i>	補助ベクトル。
<i>f</i>	行列 $F' = L * Y' * A$ 。

?laqsb

?pbequ で計算されたスケール係数を用いて対称/エルミート帯行列をスケールリングする。

構文

```
call slaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call dlaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call claqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
call zlaqsb( uplo, n, kd, ab, ldab, s, scond, amax, equed )
```

説明

このルーチンはベクトル s に格納されているスケール係数を使って対称帯行列 A を平衡化する。

入力パラメーター

<i>uplo</i>	CHARACTER*1。対称行列 A の上三角部分または下三角部分のどちらが格納されているかを指定する。 <i>uplo</i> = 'U' の場合は、上三角。 <i>uplo</i> = 'L' の場合は、下三角。
<i>n</i>	INTEGER。行列 A の次数。 $n \geq 0$
<i>kd</i>	INTEGER。 <i>uplo</i> = 'U' の場合は行列 A の優対角成分の数、 <i>uplo</i> = 'L' の場合は行列 A の劣対角成分の数。 $kd \geq 0$
<i>ab</i>	REAL (<i>slaqsb</i> の場合) DOUBLE PRECISION (<i>dlaqsb</i> の場合) COMPLEX (<i>claqsb</i> の場合) COMPLEX*16 (<i>zlaqsb</i> の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。対称帯行列 A の上三角または下三角で、配列の最初の $kd+1$ 行に格納する。 A の j 番目の列を配列 ab の j 番目の列に次のように格納する。 <i>uplo</i> = 'U' の場合、 $\max(1, j-kd) \leq i \leq j$ に対して $ab(kd+1+i-j, j) = A(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq \min(n, j+kd)$ に対して $ab(1+i-j, j) = A(i, j)$
<i>ldab</i>	INTEGER。配列 ab のリーディング・ディメンジョン。 $ldab \geq kd+1$ 。
<i>scond</i>	REAL (<i>slaqsb/claqsb</i> の場合) DOUBLE PRECISION (<i>dlaqsb/zlaqsb</i> の場合) 最小の $s(i)$ を最大の $s(i)$ で割った値。
<i>amax</i>	REAL (<i>slaqsb/claqsb</i> の場合) DOUBLE PRECISION (<i>dlaqsb/zlaqsb</i> の場合) 最大行列成分の絶対値。

出力パラメーター

<i>ab</i>	<i>info</i> = 0 の場合、帯行列 A のコレスキー因子分解 $A = U'U$ または $A = LL'$ で得られた三角係数 U または L が、 A と同じ格納形式で格納される。
<i>s</i>	REAL (<i>slaqsb/claqsb</i> の場合) DOUBLE PRECISION (<i>dlaqsb/zlaqsb</i> の場合) 配列、次元は (n)。 A のスケール係数。
<i>equed</i>	CHARACTER*1。 平衡化が行われたかどうかを示す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'Y' の場合は、平衡化が行われ、 A は $diag(s)*A*diag(s)$ で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメーター *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqsp

?ppequ で計算されたスケール係数を用いて圧縮格納形式にある対称/エルミート行列をスケーリングする。

構文

```
call slaqsp( uplo, n, ap, s, scnd, amax, equed )
call dlaqsp( uplo, n, ap, s, scnd, amax, equed )
call claqsp( uplo, n, ap, s, scnd, amax, equed )
call zlaqsp( uplo, n, ap, s, scnd, amax, equed )
```

説明

ルーチン ?laqsp は、ベクトル *s* に格納されているスケール係数を使って対称行列 *A* を平衡化する。

入力パラメーター

<i>uplo</i>	CHARACTER*1。対称行列 <i>A</i> の上三角部分または下三角部分のどちらが格納されているかを指定する。 <i>uplo</i> = 'U' の場合は、上三角。 <i>uplo</i> = 'L' の場合は、下三角。
<i>n</i>	INTEGER。行列 <i>A</i> の次数。 $n \geq 0$
<i>ap</i>	REAL (slaqsp の場合) DOUBLE PRECISION (dlaqsp の場合) COMPLEX (claqsp の場合) COMPLEX*16 (zlaqsp の場合) 配列、次元は $(n(n+1)/2)$ 。線形配列に列方向に圧縮された対称行列 <i>A</i> の上三角または下三角を格納する。 <i>A</i> の <i>j</i> 番目の列は配列 <i>ap</i> に次のように格納する。 <i>uplo</i> = 'U' の場合、 $1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i, j)$
<i>s</i>	REAL (slaqsp/claqsp の場合) DOUBLE PRECISION (dlaqsp/zlaqsp の場合) 配列、次元は (<i>n</i>)。 <i>A</i> のスケール係数。
<i>scnd</i>	REAL (slaqsp/claqsp の場合) DOUBLE PRECISION (dlaqsp/zlaqsp の場合) 最小の <i>s</i> (<i>i</i>) を最大の <i>s</i> (<i>i</i>) で割った値。

amax REAL (*slaqsp/claqsp* の場合)
DOUBLE PRECISION (*dlaqsp/zlaqsp* の場合)
最大行列成分の絶対値。

出力パラメーター

ap A と同じ形式で平衡化された行列 $\text{diag}(s)*A*\text{diag}(s)$ で上書きされる。

equed CHARACTER*1。
平衡化が行われたかどうかを示す。
equed = 'N' の場合は、平衡化は行われていない。
equed = 'Y' の場合は、平衡化が行われ、 A は $\text{diag}(s)*A*\text{diag}(s)$ で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメーター *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqsy

?poequ で計算されたスケール係数を用いて対称/エルミート行列をスケーリングする。

構文

```
call slaqsy( uplo, n, a, lda, s, scnd, amax, equed )
call dlaqsy( uplo, n, a, lda, s, scnd, amax, equed )
call claqsy( uplo, n, a, lda, s, scnd, amax, equed )
call zlaqsy( uplo, n, a, lda, s, scnd, amax, equed )
```

説明

このルーチンは、ベクトル *s* に格納されているスケール係数を使って対称行列 A を平衡化する。

入力パラメーター

uplo CHARACTER*1。対称行列 A の上三角部分または下三角部分のどちらが格納されているかを指定する。
uplo = 'U' の場合は、上三角。
uplo = 'L' の場合は、下三角。

n INTEGER。行列 A の次数。
 $n \geq 0$

<i>a</i>	REAL (slaqsy の場合) DOUBLE PRECISION (dlaqsy の場合) COMPLEX (claqsy の場合) COMPLEX*16 (zlaqsy の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。対称行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、 <i>a</i> の先頭の $n \times n$ 上三角部分に行列 <i>A</i> の上三角部分を格納する。 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>a</i> の先頭の $n \times n$ 下三角部分に行列 <i>A</i> の下三角部分を格納する。 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(n, 1)$
<i>s</i>	REAL (slaqsy/claqsy の場合) DOUBLE PRECISION (dlaqsy/zlaqsy の場合) 配列、次元は (<i>n</i>)。 <i>A</i> のスケール係数。
<i>scond</i>	REAL (slaqsy/claqsy の場合) DOUBLE PRECISION (dlaqsy/zlaqsy の場合) 最小の $s(i)$ を最大の $s(i)$ で割った値。
<i>amax</i>	REAL (slaqsy/claqsy の場合) DOUBLE PRECISION (dlaqsy/zlaqsy の場合) 最大行列成分の絶対値。

出力パラメーター

<i>a</i>	<i>equed</i> = 'Y' の場合、平衡化された行列、 $\text{diag}(s) * A * \text{diag}(s)$ が格納される。
<i>equed</i>	CHARACTER*1。 平衡化が行われたかどうかを示す。 <i>equed</i> = 'N' の場合は、平衡化は行われていない。 <i>equed</i> = 'Y' の場合は、平衡化が行われ、 <i>A</i> は $\text{diag}(s) * A * \text{diag}(s)$ で置き換えられている。

アプリケーション・ノート

このルーチンは内部パラメーター *thresh*、*large*、*small* を使用する。*thresh* はしきい値で、スケール係数の比率にもとづいたスケーリングの実行判定に使用される。*scond* < *thresh* の場合にスケーリングが実行される。*large* と *small* はそれぞれしきい値で、最大行列成分の絶対サイズにもとづいたスケーリングの実行判定に使用される。*amax* > *large* または *amax* < *small* の場合にスケーリングが実行される。

?laqtr

実数準三角連立方程式または特殊な形式の複素準三角連立方程式を実数演算で解く。

構文

```
call slaqtr( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
call dlaqtr( ltran, lreal, n, t, ldt, b, w, scale, x, work, info )
```

説明

ルーチン ?laqtr は実数準三角連立方程式、

$$\text{op}(T) * p = \text{scale} * c, \quad \text{lreal} = \text{.TRUE.} \text{ の場合}$$

または複素準三角連立方程式、

$$\text{op}(T + iB) * (p + iq) = \text{scale} * (c + id), \quad \text{lreal} = \text{.FALSE.} \text{ の場合を実数演算で解き、}$$

ここで T は上準三角である。

$\text{lreal} = \text{.FALSE.}$ の場合、 T の最初の対角ブロックは 1×1 でなければならない。
また、 B は次のような特殊な構造の行列である。

$$B = \begin{bmatrix} b_1 & b_2 & \dots & \dots & b_n \\ & w & & & \\ & & w & & \\ & & & \dots & \\ & & & & w \end{bmatrix}$$

$\text{op}(A) = A$ または A' 。ここで A' は行列 A の共役転置である。

入力では、

$$x = \begin{bmatrix} c \\ d \end{bmatrix}, \text{ 出力では、 } x = \begin{bmatrix} p \\ q \end{bmatrix}$$

このルーチンはルーチン [?trsna](#) における条件数推定に用いられる。

入力パラメーター

ltran LOGICAL。
ltran は共役転置のオプションを指定する。
= .FALSE. の場合、 $\text{op}(T + iB) = T + iB$ 、
= .TRUE. の場合、 $\text{op}(T + iB) = (T + iB)'$ 。

<i>lreal</i>	LOGICAL。 <i>lreal</i> は入力行列の構造を指定する。 = .FALSE. の場合、入力は複素、 = .TRUE. の場合、入力は実数。
<i>n</i>	INTEGER。 <i>n</i> は $T + iB$ の次数を指定する。 $n \geq 0$
<i>t</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) 配列、次元は (<i>ldt</i> , <i>n</i>)。 <i>t</i> には行列を Schur 標準形で格納する。 <i>lreal</i> = .FALSE. の場合、 <i>t</i> の最初の対角ブロックは 1×1 でなければならない。
<i>ldt</i>	INTEGER。 行列 T のリーディング・ディメンジョン。 $ldt \geq \max(1,n)$
<i>b</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) 配列、次元は (<i>n</i>)。 <i>b</i> には上述のように行列 B を形成するための成分を格納する。 <i>lreal</i> = .TRUE. の場合、 <i>b</i> は参照されない。
<i>w</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) <i>w</i> には行列 B の対角成分を格納する。 <i>lreal</i> = .TRUE. の場合、 <i>w</i> は参照されない。
<i>x</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) 配列、次元は ($2n$)。 <i>x</i> には連立方程式の右辺を格納する。
<i>work</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) ワークスペース配列、次元は (<i>n</i>)。

出力パラメーター

<i>scale</i>	REAL (slaqtr の場合) DOUBLE PRECISION (dlaqtr の場合) <i>scale</i> にはスケール係数が格納される。
<i>x</i>	<i>x</i> は解で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = 1 の場合、非特異点を維持するために、ある対角 1×1 ブロックが小さい数 <i>smin</i> によって摂動されたことを示す。 <i>info</i> = 2 の場合、非特異点を維持するために、ある対角 2×2 ブロックが ?1aln2 に含まれる小さい数によって摂動されたことを示す。



注: 高速化するために、このルーチンでは入力のエラーをチェックしない。

?lar1v

三重対角行列 $LDL^T - \sigma I$ の行 $b1$ から bn にある部分行列に対して逆行列の (スケールされた) r 番目の列を計算する。

構文

```
call slar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call dlar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call clar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
call zlar1v( n, b1, bn, sigma, d, l, ld, lld, gersch, z, ztz, mingma, r, isuppz,
            work )
```

説明

ルーチン ?lar1v は、三重対角行列 $LDL^T - \sigma I$ の行 $b1$ から bn にある部分行列に対して、逆行列の (スケールされた) r 番目の列を計算する。
この計算は次のようなステップで実行される。

1. 定常 qd 変換、 $LDL^T - \sigma I = L(+) D(+) L(+)^T$
2. 進行 qd 変換、 $LDL^T - \sigma I = U(-) D(-) U(-)^T$ 、
3. 記変換の組み合わせと、大きさが最大 (のうちの 1 つ) となる逆行列の対角成分の場所を示すインデックスとして r を選択すれば、 $LDL^T - \sigma I$ の逆行列の対角成分を計算する。
4. 逆行列の (スケールされた) r 番目の列を、定常変換の上部分と進行変換の下部分との組み合わせで得られるツイスト分解を使用して計算する。

入力パラメーター

n	INTEGER。行列 LDL^T の次数。
$b1$	INTEGER。 LDL^T の部分行列の最初のインデックス。
bn	INTEGER。 LDL^T の部分行列の最後のインデックス。
$sigma$	REAL (slar1v/clar1v の場合) DOUBLE PRECISION (dlar1v/zlar1v の場合) シフト。初期時点の $r=0$ の場合、 $sigma$ は LDL^T の固有値の良好な近似でなければならない。
l	REAL (slar1v/clar1v の場合) DOUBLE PRECISION (dlar1v/zlar1v の場合) 配列、次元は $(n-1)$ 。単位二重対角行列 L の $(n-1)$ 個の劣対角成分を成分 1 から $n-1$ に格納する。
d	REAL (slar1v/clar1v の場合) DOUBLE PRECISION (dlar1v/zlar1v の場合) 配列、次元は (n) 。対角行列 D の n 個の対角成分。

<i>ld</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) 配列、次元は (n-1)。n-1 個の成分 $L_i * D_i$
<i>lld</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) 配列、次元は (n-1)。n-1 個の成分 $L_i * L_i * D_i$
<i>gersch</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) 配列、次元は (2n)。n 個の Gerschgorin 間隔。r がゼロとして入力された場合、これらは r の初期検索を制限するために使用される。
<i>r</i>	INTEGER。 最初の入力では r はゼロとする。出力では大きさにおいて最大の逆行列の対角成分の場所を示すインデックスとなる。その後の反復では出力と同じ値を入力に与えなければならない。
<i>work</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) ワークスペース配列、次元は (4n)。

出力パラメーター

<i>z</i>	REAL (slarlv の場合) DOUBLE PRECISION (dlarlv の場合) COMPLEX (clarlv の場合) COMPLEX*16 (zlarlv の場合) 配列、次元は (n)。逆行列の (スケールされた) r 番目の列。 $z(r)$ は 1 として返される。
<i>ztz</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) z のノルムの平方。
<i>mingma</i>	REAL (slarlv/clarlv の場合) DOUBLE PRECISION (dlarlv/zlarlv の場合) $LDL^T - \sigma * I$ の逆行列の (大きさにおいて) 最大の対角成分の逆数。
<i>r</i>	大きさでは最大となる逆行列の対角成分の場所を示すインデックス。
<i>isuppz</i>	INTEGER。 配列、次元は (2)。z にあるベクトルのサポート情報、すなわち、ベクトル z は成分 <i>isuppz</i> (1) から <i>isuppz</i> (2) のみ非ゼロ。

?lar2v

一連の 2×2 対称/エルミート行列に、実数余弦と実数/複素正弦を持つ面回転のベクトルを両側から適用する。

構文

```
call slar2v( n, x, y, z, incx, c, s, incc )
call dlar2v( n, x, y, z, incx, c, s, incc )
call clar2v( n, x, y, z, incx, c, s, incc )
call zlar2v( n, x, y, z, incx, c, s, incc )
```

説明

ルーチン ?lar2v は、ベクトル x 、 y 、 z の成分で定義される一連の 2×2 実数対称または複素エルミート行列に、実数余弦を持つ実数/複素面回転のベクトルを両側から適用する。 $i = 1, 2, \dots, n$ では、

$$\begin{bmatrix} x_i & z_i \\ \text{conjg}(z_i) & y_i \end{bmatrix} = \begin{bmatrix} c(i) & \text{conjg}(s(i)) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i & z_i \\ \text{conjg}(z_i) & y_i \end{bmatrix} \begin{bmatrix} c(i) & -\text{conjg}(s(i)) \\ s(i) & c(i) \end{bmatrix}$$

入力パラメーター

n	INTEGER。適用する面回転の回数。
x, y, z	REAL (slar2v の場合) DOUBLE PRECISION (dlar2v の場合) COMPLEX (clar2v の場合) COMPLEX*16 (zlar2v の場合) 配列、次元はそれぞれ、 $(1+(n-1)*incx)$ 。それぞれ、ベクトル x 、 y 、 z を格納する。?lar2v のすべての型で x と y は実数とする。
$incx$	INTEGER。 x 、 y 、 z の成分間の増分。 $incx > 0$
c	REAL (slar2v/clar2v の場合) DOUBLE PRECISION (dlar2v/zlar2v の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。
s	REAL (slar2v の場合) DOUBLE PRECISION (dlar2v の場合) COMPLEX (clar2v の場合) COMPLEX*16 (zlar2v の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の正弦。
$incc$	INTEGER。 c と s の成分間の増分。 $incc > 0$

出力パラメーター

x, y, z ベクトル x 、 y 、 z は変換結果で書き換えられる。

?larf

一般矩形行列に基本リフレクターを適用する。

構文

```
call slarf( side, m, n, v, incv, tau, c, ldc, work )
call dlarf( side, m, n, v, incv, tau, c, ldc, work )
call clarf( side, m, n, v, incv, tau, c, ldc, work )
call zlarf( side, m, n, v, incv, tau, c, ldc, work )
```

説明

このルーチンは、実数/複素基本リフレクター H を、実数/複素 $m \times n$ 行列 C に、左または右のいずれかから適用する。 H は次の形式で表現される。

$$H = I - \tau * v * v',$$

τ は実/複素スカラー、 v は実/複素ベクトルである。

$\tau = 0$ の場合、 H は単位行列をとる。

clarf/zlarf で H' (H の共役転置) を適用するには τ の代わりに $\text{conjg}(\tau)$ を与える。

入力パラメーター

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、形式 $H * C$ <i>side</i> = 'R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 C の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>v</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) 配列、次元は (1 + (m-1)*abs(incv)) (side = 'L' の場合) (1 + (n-1)*abs(incv)) (side = 'R' の場合) H の表現におけるベクトル v 。 $\tau = 0$ の場合、 v は使用されない。
<i>incv</i>	INTEGER。 v の成分間の増分。 <i>incv</i> $\neq 0$
<i>tau</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) H の表現における値 τ 。

<i>c</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 $m \times n$ の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (slarf の場合) DOUBLE PRECISION (dlarf の場合) COMPLEX (clarf の場合) COMPLEX*16 (zlarf の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>side</i> = 'L' の場合は行列 $H * C$ で、 <i>side</i> = 'R' の場合は行列 $C * H$ で上書きされる。
----------	---

?larfb

ブロック・リフレクターまたはその転置/共役転置を一般矩形行列に適用する。

構文

```
call slarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,
            ldwork )
call dlarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,
            ldwork )
call clarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,
            ldwork )
call zlarfb( side, trans, direct, storev, m, n, k, v, ldv, t, ldt, c, ldc, work,
            ldwork )
```

説明

ルーチン ?larfb は、複素ブロック・リフレクター H またはその転置 H' を、複素 $m \times n$ 行列 C に、左または右のいずれかから適用する。

入力パラメーター

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、 H または H' を左から適用する。 <i>side</i> = 'R' の場合、 H または H' を右から適用する。
-------------	--

<i>trans</i>	CHARACTER*1。 <i>trans</i> = 'N' の場合、 <i>H</i> を適用する (転置なし)。 <i>trans</i> = 'C' の場合、 <i>H'</i> を適用する (転置 / 共役転置)
<i>direct</i>	CHARACTER*1。基本リフレクターの積からどのように <i>H</i> が表現されているかを示す。 <i>direct</i> = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ (順方向) <i>direct</i> = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	CHARACTER*1。リフレクターを定義するベクトルがどのように格納されているかを示す。 <i>storev</i> = 'C' の場合、列方向 <i>storev</i> = 'R' の場合、行方向。
<i>m</i>	INTEGER。行列 <i>C</i> の行数。
<i>n</i>	INTEGER。行列 <i>C</i> の列数。
<i>k</i>	INTEGER。行列 <i>T</i> の次数 (積がブロック・リフレクターを定義する基本リフレクターの個数に等しい)。
<i>v</i>	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列、次元は <i>storev</i> = 'C' の場合では (<i>ldv</i> , <i>k</i>) <i>storev</i> = 'R' および <i>side</i> = 'L' の場合では (<i>ldv</i> , <i>m</i>) <i>storev</i> = 'R' および <i>side</i> = 'R' の場合では (<i>ldv</i> , <i>n</i>) 行列 <i>V</i> 。
<i>ldv</i>	INTEGER。 配列 <i>v</i> のリーディング・ディメンジョン。 <i>storev</i> = 'C' および <i>side</i> = 'L' の場合、 $ldv \geq \max(1, m)$ 、 <i>storev</i> = 'C' および <i>side</i> = 'R' の場合、 $ldv \geq \max(1, n)$ 、 <i>storev</i> = 'R' の場合、 $ldv \geq k$ 。
<i>t</i>	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列、次元は (<i>ldt</i> , <i>k</i>)。 ブロック・リフレクターの表現にある $k \times k$ 行列 <i>T</i> の三角を格納する。
<i>ldt</i>	INTEGER。配列 <i>t</i> のリーディング・ディメンジョン。 $ldt \geq k$
<i>c</i>	REAL (slarfb の場合) DOUBLE PRECISION (dlarfb の場合) COMPLEX (clarfb の場合) COMPLEX*16 (zlarfb の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 $m \times n$ の行列 <i>C</i> を格納する。

<code>ldc</code>	INTEGER。配列 <code>c</code> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<code>work</code>	REAL (<code>slarfb</code> の場合) DOUBLE PRECISION (<code>dlarfb</code> の場合) COMPLEX (<code>clarfb</code> の場合) COMPLEX*16 (<code>zlarfb</code> の場合) ワークスペース配列、次元は (<code>ldwork, k</code>)。
<code>ldwork</code>	INTEGER。配列 <code>work</code> のリーディング・ディメンジョン。 <code>side = 'L'</code> の場合、 $ldwork \geq \max(1, n)$ 、 <code>side = 'R'</code> の場合、 $ldwork \geq \max(1, m)$

出力パラメーター

<code>c</code>	終了時に、 <code>c</code> は $H * C$ または $H' * C$ あるいは $C * H$ または $C * H'$ によって上書きされる。
----------------	---

?larfg

基本リフレクター (Householder 行列) を生成する

構文

```
call slarfg( n, alpha, x, incx, tau )
call dlarfg( n, alpha, x, incx, tau )
call clarfg( n, alpha, x, incx, tau )
call zlarfg( n, alpha, x, incx, tau )
```

説明

ルーチン ?larfg は、次を満たすような次数 n の実数 / 複素基本リフレクター H を生成する。

$$H' * \begin{bmatrix} \alpha \\ x \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \end{bmatrix} \quad H' * H = I$$

α と β は、スカラー (β はすべての型で実数)、 x は $(n-1)$ 成分で構成される実数 / 複素ベクトルである。 H は次の形式で表現される。

$$H = I - \tau * \begin{bmatrix} 1 \\ v \end{bmatrix} * \begin{bmatrix} 1 & v' \end{bmatrix},$$

τ は実数 / 複素スカラー、 v は実数 / 複素の $(n-1)$ 成分で構成されるベクトルである。`clarfg/zlarfg` では、 H はエルミートではない点に注意する。

x の成分がすべてゼロの場合 (かつ、複素型で $alpha$ が実数)、 $tau = 0$ となり、 H は単位行列をとる。

それ以外では、 $1 \leq tau \leq 2$ (実数型の場合)、または
 $1 \leq \text{Re}(tau) \leq 2$ かつ $\text{abs}(tau-1) \leq 1$ (複素型の場合)

入力パラメーター

n	INTEGER。基本リフレクターの次数。
$alpha$	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) $alpha$ の値を格納する。
x	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) 配列、次元は $(1+(n-2)*\text{abs}(incx))$ 。 ベクトル x を格納する。
$incx$	INTEGER。 x の成分間の増分。 $incx > 0$

出力パラメーター

$alpha$	$beta$ の値で上書きされる。
x	ベクトル v で上書きされる。
tau	REAL (slarfg の場合) DOUBLE PRECISION (dlarfg の場合) COMPLEX (clarfg の場合) COMPLEX*16 (zlarfg の場合) tau の値を格納する。

?larft

ブロック・リフレクター $H = I - VTV^H$ の三角係数
 T を生成する。

構文

```
call slarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarft( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarft( direct, storev, n, k, v, ldv, tau, t, ldt )
```

説明

ルーチン `?larft` は、次数 n の実/複素ブロック・リフレクター H の三角係数 T を生成する。ブロック・リフレクター H は k 個の基本リフレクターの積として定義される

$direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ と T は上三角である。

$direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ と T は下三角である。

$storev = 'C'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは配列 v の i 番目の列に格納され、 $H = I - V^* T^* V$ である。

$storev = 'R'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは配列 v の i 番目の行に格納され、 $H = I - V^* T^* V$ である。

入力パラメーター

<code>direct</code>	CHARACTER*1。ブロック・リフレクターを生成するために基本リフレクターを乗算する次数を指定する。 = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ (順方向) = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<code>storev</code>	CHARACTER*1。基本リフレクターを定義するベクトルがどのように格納されているかを指定する。(次のアプリケーション・ノートも参照): = 'C' の場合、列方向 = 'R' の場合、行方向。
<code>n</code>	INTEGER。ブロック・リフレクター H の次数。 $n \geq 0$
<code>k</code>	INTEGER。三角係数 T の次数 (基本リフレクターの個数に等しい)。 $k \geq 1$
<code>v</code>	REAL (slarft の場合) DOUBLE PRECISION (dlarft の場合) COMPLEX (clarft の場合) COMPLEX*16 (zlarft の場合) 配列、次元は $storev = 'C'$ の場合は (ldv, k) 、または $storev = 'R'$ の場合は (ldv, n) 。 行列 V 。
<code>ldv</code>	INTEGER。配列 v のリーディング・ディメンジョン。 $storev = 'C'$ の場合、 $ldv \geq \max(1, n)$ 、 $storev = 'R'$ の場合、 $ldv \geq k$ 。
<code>tau</code>	REAL (slarft の場合) DOUBLE PRECISION (dlarft の場合) COMPLEX (clarft の場合) COMPLEX*16 (zlarft の場合) 配列、次元は (k) 。 $tau(i)$ には、基本リフレクター $H(i)$ のスカラー係数が入っていないなければならない。
<code>ldt</code>	INTEGER。出力配列 t のリーディング・ディメンジョン。 $ldt \geq k$

出力パラメーター

t	REAL (slarft の場合) DOUBLE PRECISION (dlarft の場合) COMPLEX (clarft の場合) COMPLEX*16 (zlarft の場合) 配列、次元は (ldt,k) 。ブロック・リフレクターの $k \times k$ 三角係数 T 。 $direct = 'F'$ の場合、 T は上三角。 $direct = 'B'$ の場合、 T は下三角。配列の残りの部分は使用されない。
v	行列 V 。

アプリケーション・ノート

行列 V の形状と $H(i)$ を定義するベクトルの格納形式を、 $n=5$ 、 $k=3$ において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、出力で元に戻される。配列の残りの部分は使用されない。

$direct = 'F'$ および $storev = 'C'$ $direct = 'F'$ および $storev = 'R'$

$$\begin{bmatrix} 1 \\ v_1 & 1 \\ v_1 & v_2 & 1 \\ v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \end{bmatrix} \qquad \begin{bmatrix} 1 & v_1 & v_1 & v_1 & v_1 \\ & 1 & v_2 & v_2 & v_2 \\ & & 1 & v_3 & v_3 \end{bmatrix}$$

$direct = 'B'$ および $storev = 'C'$ $direct = 'B'$ および $storev = 'R'$

$$\begin{bmatrix} v_1 & v_2 & v_3 \\ v_1 & v_2 & v_3 \\ 1 & v_2 & v_3 \\ & 1 & v_3 \\ & & 1 \end{bmatrix} \qquad \begin{bmatrix} v_1 & v_1 & 1 \\ v_2 & v_2 & v_2 & 1 \\ v_3 & v_3 & v_3 & v_3 & 1 \end{bmatrix}$$

?larfx

リフレクターが次数 ≤ 10 を持つ場合、ループの繰り返しを避けながら、基本リフレクターを一般矩形行列に適用する。

構文

```
call slarfx( side, m, n, v, tau, c, ldc, work )
call dlarfx( side, m, n, v, tau, c, ldc, work )
call clarfx( side, m, n, v, tau, c, ldc, work )
call zlarfx( side, m, n, v, tau, c, ldc, work )
```

説明

ルーチン ?larfx は、実数 / 複素基本リフレクター H を、実数 / 複素 $m \times n$ 行列 C に、左辺または右辺から適用する。

H は次の形式で表現される。

$H = I - \tau * v * v'$ 、 τ は実数 / 複素スカラー、 v は実数 / 複素ベクトルである。

$\tau = 0$ の場合、 H は単位行列をとる。

入力パラメーター

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、形式 $H * C$ <i>side</i> = 'R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 C の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>v</i>	REAL (slarfx の場合) DOUBLE PRECISION (dlarfx の場合) COMPLEX (clarfx の場合) COMPLEX*16 (zlarfx の場合) 配列、次元は <i>side</i> = 'L' の場合 (<i>m</i>) <i>side</i> = 'R' の場合 (<i>n</i>) H の表現におけるベクトル v 。
<i>tau</i>	REAL (slarfx の場合) DOUBLE PRECISION (dlarfx の場合) COMPLEX (clarfx の場合) COMPLEX*16 (zlarfx の場合) H の表現における値 τ 。
<i>c</i>	REAL (slarfx の場合) DOUBLE PRECISION (dlarfx の場合) COMPLEX (clarfx の場合) COMPLEX*16 (zlarfx の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 $m \times n$ の行列 c を格納する。

<i>ldc</i>	INTEGER。配列 <i>c</i> のリーディング・ディメンジョン。 $lda \geq (1, m)$
<i>work</i>	REAL (<i>slarfx</i> の場合) DOUBLE PRECISION (<i>dlarfx</i> の場合) COMPLEX (<i>clarfx</i> の場合) COMPLEX*16 (<i>zlarfx</i> の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>) <i>H</i> が次数 < 11 を持つ場合は <i>work</i> は参照されない。

出力パラメーター

<i>c</i>	<i>side</i> = 'L' の場合は行列 $H * C$ で、 <i>side</i> = 'R' の場合は行列 $C * H$ で上書きされる。
----------	---

?largv

実数余弦および実数 / 複素正弦を持つ面回転のベクトルを生成する。

構文

```
call slargv( n, x, incx, y, incy, c, incc )
call dlargv( n, x, incx, y, incy, c, incc )
call clargv( n, x, incx, y, incy, c, incc )
call zlargv( n, x, incx, y, incy, c, incc )
```

説明

このルーチンは、実数 / 複素ベクトル x と y の成分で決定される、実数余弦を持った実数 / 複素面回転のベクトルを生成する。

slargv/dlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -s(i) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a_i \\ 0 \end{bmatrix}, \quad i = 1, 2, \dots, n \text{ の場合}$$

clargv/zlargv の場合

$$\begin{bmatrix} c(i) & s(i) \\ -\text{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix}, \quad i = 1, 2, \dots, n \text{ の場合}$$

ここで $c(i)^2 + \text{abs}(s(i))^2 = 1$ 。また、次の規則が使用される (clartg/zlartg と同じだが、BLAS レベル 1 ルーチン crotg/zrotg とは異なる)。

$y_i = 0$ の場合、 $c(i) = 1$ かつ $s(i) = 0$ 、

$x_i = 0$ の場合、 $c(i) = 0$ かつ r_i が実数となるように $s(i)$ が選択される。

入力パラメーター

<i>n</i>	INTEGER。適用する面回転の回数。
<i>x</i> , <i>y</i>	REAL (slargv の場合) DOUBLE PRECISION (dlargv の場合) COMPLEX (clargv の場合) COMPLEX*16 (zlargv の場合) 配列、次元はそれぞれ $(1 + (n-1)*incx)$ と $(1 + (n-1)*incy)$ 。 ベクトル <i>x</i> 、 <i>y</i> を格納する。
<i>incx</i>	INTEGER。 <i>x</i> の成分間の増分。 <i>incx</i> > 0
<i>incy</i>	INTEGER。 <i>y</i> の成分間の増分。 <i>incy</i> > 0
<i>incc</i>	INTEGER。出力配列 <i>c</i> の分間の増分。 <i>incc</i> > 0

出力パラメーター

<i>x</i>	$i = 1, \dots, n$ に対して、 <i>x</i> (<i>i</i>) は a_i (実数型) または r_i (複素型) で上書きされる。
<i>y</i>	面回転の正弦 <i>s</i> (<i>i</i>) で上書きされる。
<i>c</i>	REAL (slargv/clargv の場合) DOUBLE PRECISION (dlargv/zlargv の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。

?larnv

乱数ベクトルを一様分布または正規分布で返す。

構文

```
call slarnv( idist, iseed, n, x )
call dlarnv( idist, iseed, n, x )
call clarnv( idist, iseed, n, x )
call zlarnv( idist, iseed, n, x )
```

説明

ルーチン ?larnv は、*n* 個の実数 / 複素乱数のベクトルを一様分布または正規分布で返す。

このルーチンは、一様分布 $(0, 1)$ の実数乱数を生成するために、ベクトル化可能なコードを使った最大 128 のバッチ処理中に補助ルーチン [?laruv](#) を呼び出す。一様分布から正規分布への数値変換は Box-Muller 法を使用する。

入力パラメーター

<i>idist</i>	INTEGER。乱数の分布を指定する。 slarnv と dlanrv の場合。 = 1 の場合、一様分布 $(0, 1)$ = 2 の場合、一様分布 $(-1, 1)$ = 3 の場合、正規分布 $(0, 1)$ 。 clarv と zlanrv の場合。 = 1 の場合、実数部分と虚数部分がそれぞれ一様分布 $(0, 1)$ = 2 の場合、実数部分と虚数部分がそれぞれ一様分布 $(-1, 1)$ = 3 の場合、実数部分と虚数部分がそれぞれ正規分布 $(0, 1)$ = 4 の場合、円盤 $abs(z) < 1$ 上で一様分布 = 5 の場合、円 $abs(z) = 1$ 上で一様分布。
<i>iseed</i>	INTEGER。 配列、次元は (4)。 乱数生成器の種 (シード) を与える。配列成分は 0 から 4095 の間でなければならない。また <i>iseed</i> (4) は奇数でなければならない。
<i>n</i>	INTEGER。生成する乱数の個数。

出力パラメーター

<i>x</i>	REAL (slarnv の場合) DOUBLE PRECISION (dlarnv の場合) COMPLEX (clarv の場合) COMPLEX*16 (zlanrv の場合) 配列、次元は (<i>n</i>)。生成された乱数。
<i>iseed</i>	更新された種。

?larrb

より高い精度で固有値を見つけるための限定二分法を与える。

構文

```
call slarrb( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap, werr,
            work, iwork, info )
call dlarrb( n, d, l, ld, lld, ifirst, ilast, sigma, reltol, w, wgap, werr,
            work, iwork, info )
```

説明

このルーチンは、比較的安定な表現 (RRR) の LDL^T が与えられた状態で、より高い精度で LDL^T の固有値 $w(ifirst)$ から $w(ilast)$ を見つけるために、「限定された」二分法を実行する。

間隔 [左, 右] は、それらの中点と部分幅を配列 w と $werr$ にそれぞれ格納することによって維持される。

入力パラメーター

n	INTEGER。行列の次数。
d	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は (n) 。対角行列 D の n 個の対角成分。
l	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は $(n-1)$ 。単位二重対角行列 L の $n-1$ 個の劣対角成分。
ld	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は $(n-1)$ 。 $n-1$ 個の成分 $L_i * D_i$
lld	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は $(n-1)$ 。 $n-1$ 個の成分 $L_i * L_i * D_i$ 。
$ifirst$	INTEGER。クラスター内の最初の固有値のインデックス。
$ilast$	INTEGER。クラスター内の最後の固有値のインデックス。
$sigma$	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) LDL^T を形成するために使用されたシフト。 (?slarrf を参照)
$reltol$	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 相対許容値
w	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は (n) 。 $w(ifirst)$ から $w(ilast)$ には LDL^T の固有値に対応した推定を格納する。
$wgap$	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は (n) 。 LDL^T の固有値間の隔たり。
$werr$	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) 配列、次元は (n) 。 $werr(ifirst)$ から $werr(ilast)$ には推定 $w(ifirst)$ から $w(ilast)$ の誤差を格納する。
$work$	REAL (slarrb の場合) DOUBLE PRECISION (dlarrb の場合) ワークスペース配列。このパラメーターはルーチン内では使用されない。

<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(2n)$
出力パラメーター	
<i>w</i>	固有値の推定の精度を改善した結果が上書きされる。
<i>wgap</i>	微小な隔たりが変更される。
<i>werr</i>	推定 $w(\text{ifirst})$ から $w(\text{ilast})$ の誤差に対して精度を改善した結果が上書きされる。
<i>info</i>	INTEGER。 エラーフラグ。ルーチンではこのパラメーターは設定されない。

?larre

三重対角行列 T が与えられたとき、小さい非対角成分をゼロに設定し、縮退されていないブロック T_i に対して基本表現と固有値を探す。

構文

```
call slarre( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work, info )
call dlarre( n, d, e, tol, nsplit, isplit, m, w, woff, gersch, work, info )
```

説明

三重対角行列 T が与えられたとき、このルーチンは「小さい」非対角成分をゼロに設定し、縮退されていないブロック T_i に対して次を探す。

- 数 σ_i
- 基本表現 $T_i - \sigma_i I = L_i D_i L_i^T$
- 各 $L_i D_i L_i^T$ の固有値

求められた表現と固有値は対称三重対角行列の固有値を計算するために ?stegr で使用される。現時点で基本表現は正定値または負定値として制限されており、定値行列の固有値は *dqds* アルゴリズムによって求められる (サブルーチン ?lasq2)。また ?larre は、 $L_i D_i L_i^T$ に対する n 個の Gerschgorin 間隔を出力する利点を持つ。

入力パラメーター

<i>n</i>	INTEGER。行列の次数。
<i>d</i>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列、次元は (n) 。三重対角行列 T の n 個の対角成分を格納する。
<i>e</i>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列、次元は (n) 。三重対角行列 T の $(n-1)$ 個の劣対角成分を格納する。 $e(n)$ は設定する必要はない。

<code>tol</code>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 分割のしきい値。入力において $ e(i) < tol$ の場合、行列 T は小さいブロックに分割される。
<code>nsplit</code>	INTEGER。ブロック T を分割する数。 $1 \leq nsplit \leq n$ 。
<code>work</code>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) ワークスペース配列、次元は $(4*n)$

出力パラメーター

<code>d</code>	対角行列 D_i の n 個の対角成分で上書きされる。
<code>e</code>	単位二重対角行列 L_i の劣対角成分で上書きされる。
<code>isplit</code>	INTEGER。 配列、次元は $(2n)$ 。 T を部分行列に分割した分割点。第 1 の部分行列は 1 から <code>isplit(1)</code> の行 / 列で構成され、第 2 の部分行列は <code>isplit(1)+1</code> から <code>isplit(2)</code> の行 / 列で構成され、以下同様であり、 <code>nsplit</code> 番目は <code>isplit(nsplit-1)+1</code> から <code>isplit(nsplit)=n</code> の行 / 列で構成される。
<code>m</code>	INTEGER。求められた (すべての $L_i D_i L_i^T$) 固有値の総数。
<code>w</code>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列、次元は (n) 。最初の m 個の成分には固有値が格納される。各ブロック $L_i D_i L_i^T$ の各固有値は昇順でソートされる。
<code>woff</code>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列、次元は (n) 。 <code>nsplit</code> 基本点 σ_i
<code>gersch</code>	REAL (slarre の場合) DOUBLE PRECISION (dlarre の場合) 配列、次元は $(2n)$ 。 n 個の Gerschgorin 間隔。
<code>info</code>	INTEGER。?lasq2 のエラーコード。

?larrf

少なくとも 1 つの固有値が相対的に分離されているような新しい比較的安定な表現を探す。

構文

```
call slarrf( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work, iwork,
            info )
call dlarrf( n, d, l, ld, lld, ifirst, ilast, w, dplus, lplus, work, iwork,
            info )
```

説明

初期表現 LDL^T と (相対的に) 値が近いそれらの固有値のクラスター $w(i\text{first})$, $w(i\text{first}+1), \dots, w(i\text{last})$ が与えられたとき、ルーチン `?larrf` は、新しい比較的安定な表現を探す。

$$LDL^T - \sigma_i I = L(+D(+L(+)^T$$

$L(+D(+L(+)^T$ の少なくとも 1 つの固有値が相対的に分離されている。

入カパラメーター

<code>n</code>	INTEGER。行列の次数。
<code>d</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n</code>)。対角行列 D の n 個の対角成分。
<code>l</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n-1</code>)。単位二重対角行列 L の (<code>n-1</code>) 個の劣対角成分。
<code>ld</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n-1</code>)。 $n-1$ 個の成分 $L_i * D_i$
<code>lld</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n-1</code>)。 $n-1$ 個の成分 $L_i * L_i * D_i$
<code>ifirst</code>	INTEGER。クラスター内の最初の固有値のインデックス。
<code>ilast</code>	INTEGER。クラスター内の最後の固有値のインデックス。
<code>w</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n</code>)。 LDL^T の固有値を昇順で格納する。 $w(i\text{first})$ から $w(i\text{last})$ が比較的安定な表現のクラスターを形成する。
<code>sigma</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) $L(+D(+L(+)^T$ を形成するために使用されたシフト。
<code>work</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) ワークスペース配列。

出カパラメーター

<code>w</code>	$w(i\text{first})$ から $w(i\text{last})$ は、 $L(+D(+L(+)^T$ の対応する固有値の推定で上書きされる。
<code>dplus</code>	REAL (<code>slarrf</code> の場合) DOUBLE PRECISION (<code>dlarrf</code> の場合) 配列、次元は (<code>n</code>)。対角行列 $D(+)$ の n 個の対角成分。

lplus REAL (slarrf の場合)
 DOUBLE PRECISION (dlarrf の場合)
 配列、次元は (n)。lplus の最初の (n-1) 個の成分には単位二重対角行列 $L(+)$ の劣対角成分が格納される。lplus(n) は *sigma* に設定される。

?larrv

L 、 D 、 LDL^T の固有値が与えられたとき、三重対角行列 $T = LDL^T$ の固有ベクトルを計算する。

構文

```
call slarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call dlarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call clarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
call zlarrv( n, d, l, isplit, m, w, iblock, gersch, tol, z, ldz, isuppz, work,
            iwork, info )
```

説明

ルーチン ?larrv は、 L 、 D 、 LDL^T の固有値が与えられたとき、三重対角行列 $T = LDL^T$ の固有ベクトルを計算する。
 入力の固有値は L と D の成分に対して相対的に高い精度を持っていなければならない。
 目的とする出力精度は入力パラメーター *tol* で指定できる。

入力パラメーター

n INTEGER。行列の次数。 $n \geq 0$

d REAL (slarrv/clarrv の場合)
 DOUBLE PRECISION (dlarrv/zlarrv の場合)
 配列、次元は (n)。対角行列 D の n 個の対角成分を格納する。

l REAL (slarrv/clarrv の場合)
 DOUBLE PRECISION (dlarrv/zlarrv の場合)
 配列、次元は (n-1)。単位対角行列 L の (n-1) 個の劣対角成分を l の成分 1 から n-1 に格納する。l(n) を設定する必要はない。

isplit INTEGER。
 配列、次元は (n)。T を部分行列に分割した分割点。第 1 の部分行列は 1 から isplit(1) の行 / 列で構成され、第 2 の部分行列は isplit(1)+1 から isplit(2) の行 / 列で構成され、以下同様。

tol REAL (slarrv/clarrv の場合)
 DOUBLE PRECISION (dlarrv/zlarrv の場合)
 固有値 / 固有ベクトルに対する絶対誤差許容値。
 入力固有値の誤差は tol を上限としなければならない。固有ベクトル出力は tol を上限とする誤差ノルムを持ち、また、異

なる固有ベクトル間のドット積は tol に制限される。 tol は $n * eps * |T|$ 以上でなければならない、ここで eps はマシン精度、 $|T|$ は三重対角行列の 1- ノルムである。

m	INTEGER。求められた固有値の個数。 $0 \leq m \leq n$ 。 $range = 'A'$ の場合、 $m = n$ 、 $range = 'I'$ の場合、 $m = iu - il + 1$ 。
w	REAL (slarrv/clarrv の場合) DOUBLE PRECISION (dlarrv/zlarrv の場合) 配列、次元は (n) 。 w の最初の m 個の成分には固有ベクトルを計算する固有値を格納する。固有値は分割されたブロックごとにグループ分けし、ブロック内で最小から最大に並べられていなければならない (zlarre の出力配列 w がここで要求される)。 w の誤差は tol を上限としなければならない。
$iblock$	INTEGER。 配列、次元は (n) 。 w 内の対応する固有値に関係した部分行列インデックス。固有値 $w(i)$ が第 1 の部分行列の先頭から属する場合は $iblock(i) = 1$ 、 $w(i)$ が第 2 の行列に属する場合は $iblock(i) = 2$ 、以下同様。
ldz	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq 1$ 、 $jobz = 'V'$ の場合は、 $ldz \geq \max(1, n)$ 。
$work$	REAL (slarrv/clarrv の場合) DOUBLE PRECISION (dlarrv/zlarrv の場合) ワークスペース配列、次元は $(13n)$
$iwork$	INTEGER。 ワークスペース配列、次元は $(6n)$ 。

出力パラメーター

d	d は上書きされることがある。
l	l は上書きされる。
z	REAL (slarrv の場合) DOUBLE PRECISION (dlarrv の場合) COMPLEX (clarrv の場合) COMPLEX*16 (zlarrv の場合) 配列、次元は $(ldz, \max(1, m))$ 。 $jobz = 'V'$ の場合、 $info = 0$ ならば、行列 T の正規直交固有ベクトルのうち、選択された固有値に対応するものが z の最初の m 列に格納される。すなわち、 $w(i)$ に対応する固有ベクトルが z の i 番目の列に入る。 $jobz = 'N'$ の場合、 z は参照されない。



注: 配列 z には、 $\max(1, m)$ 以上の列が提供されなければならない。
 $range = 'V'$ の場合は、 m の正確な値が事前にわからないため、上限値を使用する必要がある。

<i>isuppz</i>	INTEGER。 配列、次元は $(2 * \max(1, m))$ 。z に入っている固有ベクトルのサポート情報、すなわち z に入っている非ゼロの成分を示すインデックス。i 番目の固有ベクトルは <i>isuppz</i> (2i-1) から <i>isuppz</i> (2i) までの成分のみ非ゼロである。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = -i < 0 の場合、i 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = 1 ならば ?larrb で内部エラーが発生したことを示す。 <i>info</i> = 2 ならば ?stein で内部エラーが発生したことを示す。

?lartg

実数余弦と実数/複素正弦を持つ面回転を生成する。

構文

```
call slartg( f, g, cs, sn, r )
call dlartg( f, g, cs, sn, r )
call clartg( f, g, cs, sn, r )
call zlartg( f, g, cs, sn, r )
```

説明

このルーチンは、次のように面回転を生成する。

$$\begin{bmatrix} cs & sn \\ -\text{conj}(sn) & cs \end{bmatrix} \cdot \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

ここで $cs^2 + |sn|^2 = 1$

このルーチンは BLAS レベル 1 ルーチン [?rotg](#) の実行速度が遅く精度が高いバージョンであるが、次の点で相違がある。

slartg/dlartg の場合。

f と *g* は出力時に変更されない。

g = 0 の場合、*cs* = 1 と *sn* = 0

f = 0 かつ *g* ≠ 0 の場合、浮動小数点演算は実行されず *cs* = 0 と *sn* = 1 (対角にゼロがある場合、?bdsqr の動作を省く)。

f が大きさを *g* を超えた場合、*cs* は正となる。

clartg/zlartg の場合。

f と g は出力時に変更されない。

$g = 0$ の場合、 $cs = 1$ と $sn = 0$

$f = 0$ の場合、 $cs = 0$ 、と r が実数になるように sn が選択される。

入力パラメーター

f, g REAL (slartg の場合)
 DOUBLE PRECISION (dlartg の場合)
 COMPLEX (clartg の場合)
 COMPLEX*16 (zlartg の場合)
 回転対象のベクトルの第 1 および第 2 成分。

出力パラメーター

cs REAL (slartg/clartg の場合)
 DOUBLE PRECISION (dlartg/zlartg の場合)
 回転の余弦。

sn REAL (slartg の場合)
 DOUBLE PRECISION (dlartg の場合)
 COMPLEX (clartg の場合)
 COMPLEX*16 (zlartg の場合)
 回転の正弦。

r REAL (slartg の場合)
 DOUBLE PRECISION (dlartg の場合)
 COMPLEX (clartg の場合)
 COMPLEX*16 (zlartg の場合)
 回転されたベクトルの非ゼロの成分。

?lartv

実数余弦と実数/複素制限を持つ面回転のベクトルをベクトル対の成分に適用する。

構文

```
call slartv( n, x, incx, y, incy, c, s, incc )
call dlartv( n, x, incx, y, incy, c, s, incc )
call clartv( n, x, incx, y, incy, c, s, incc )
call zlartv( n, x, incx, y, incy, c, s, incc )
```

説明

このルーチンは、実数余弦を持つ実数/複素面回転を実数/複素ベクトル x と y の成分に適用する。 $i = 1, 2, \dots, n$ では、

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} c(i) & s(i) \\ -\text{conjg}(s(i)) & c(i) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

入力パラメーター

<i>n</i>	INTEGER。適用する面回転の回数。
<i>x, y</i>	REAL (slartv の場合) DOUBLE PRECISION (dlartv の場合) COMPLEX (clartv の場合) COMPLEX*16 (zartv の場合) 配列、次元はそれぞれ $(1+(n-1)*incx)$ と $(1+(n-1)*incy)$ 。入力ベクトル <i>x</i> と <i>y</i>
<i>incx</i>	INTEGER。 <i>x</i> の成分間の増分。 <i>incx</i> > 0
<i>incy</i>	INTEGER。 <i>y</i> の成分間の増分。 <i>incy</i> > 0
<i>c</i>	REAL (slartv/clartv の場合) DOUBLE PRECISION (dlartv/zartv の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の余弦。
<i>s</i>	REAL (slartv の場合) DOUBLE PRECISION (dlartv の場合) COMPLEX (clartv の場合) COMPLEX*16 (zartv の場合) 配列、次元は $(1+(n-1)*incc)$ 。面回転の正弦。
<i>incc</i>	INTEGER。 <i>c</i> と <i>s</i> の成分間の増分。 <i>incc</i> > 0

出力パラメーター

<i>x, y</i>	回転されたベクトル <i>x</i> と <i>y</i>
-------------	-------------------------------

?laruv

n 個の実数乱数のベクトルを一樣分布で返す。

構文

```
call slaruv( iseed, n, x )
call dlaruv( iseed, n, x )
```

説明

ルーチン ?laruv は *n* 個の実数乱数のベクトルを一樣分布 (0, 1) で返す ($n \leq 128$)。

このルーチンは、[?larnv](#) から呼び出される補助ルーチンである。

入力パラメーター

<i>iseed</i>	INTEGER。 配列、次元は (4)。乱数生成器の種 (シード) を与える。配列成分は 0 から 4095 の間でなければならない。また <i>iseed</i> (4) は奇数でなければならない。
<i>n</i>	INTEGER。生成する乱数の個数。 $n \leq 128$

出力パラメーター

<i>x</i>	REAL (<i>slaruv</i> の場合) DOUBLE PRECISION (<i>dlaruv</i> の場合) 配列、次元は (<i>n</i>)。生成された乱数。
<i>seed</i>	種は更新される。

?larz

(?tzrzf が返したとおりの) 基本リフレクターを一般行列に適用する。

構文

```
call slarz( side, m, n, l, v, incv, tau, c, ldc, work )
call dlarz( side, m, n, l, v, incv, tau, c, ldc, work )
call clarz( side, m, n, l, v, incv, tau, c, ldc, work )
call zlarz( side, m, n, l, v, incv, tau, c, ldc, work )
```

説明

ルーチン ?larz は、実数 / 複素基本リフレクター H を、実数 / 複素 $m \times n$ 行列 C に、左または右のいずれかから適用する。 H は次の形式で表現される。

$$H = I - \tau * v * v',$$

τ は実 / 複素スカラー、 v は実 / 複素ベクトルである。

$\tau = 0$ の場合、 H は単位行列をとる。

複素型で H' (H の共役転置) を適用するには τ の代わりに $\text{conjg}(\tau)$ を与える。

H は [?tzrzf](#) から返されたとおり、 k 個の基本リフレクターの積である。

入力パラメーター

<i>side</i>	CHARACTER*1。 <i>side</i> = 'L' の場合、形式 $H * C$ <i>side</i> = 'R' の場合、形式 $C * H$ 。
<i>m</i>	INTEGER。行列 C の行数。
<i>n</i>	INTEGER。行列 C の列数。
<i>l</i>	INTEGER。Householder ベクトルとして意味を持った部分が格納されているベクトル v の成分数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$

<i>v</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列、次元は $(1+(l-1)*abs(incv))$ 。?tzrzf から返されたとおり、 <i>H</i> の表現中にあるベクトル <i>v</i> 。 <i>tau</i> = 0 の場合、 <i>v</i> は使用されない。
<i>incv</i>	INTEGER。 <i>v</i> の成分間の増分。 <i>incv</i> ≠ 0
<i>tau</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) <i>H</i> の表現における値 <i>tau</i> 。
<i>c</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 <i>m</i> × <i>n</i> の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>c</i> のリーディング・ディメンジョン。 <i>ldc</i> ≥ max(1, <i>m</i>)
<i>work</i>	REAL (slarz の場合) DOUBLE PRECISION (dlarz の場合) COMPLEX (clarz の場合) COMPLEX*16 (zlarz の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>side</i> = 'L' の場合は行列 <i>H</i> * <i>C</i> で、 <i>side</i> = 'R' の場合は行列 <i>C</i> * <i>H</i> で上書きされる。
----------	---

?larzb

ブロック・リフレクターまたはその転置/共役転置を一般矩形行列に適用する。

```
call slarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
call dlarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
call clarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
```

```
call zlarzb( side, trans, direct, storev, m, n, k, l, v, ldv, t, ldt, c, ldc,
            work, ldwork )
```

説明

このルーチンは実数/複素ブロック・リフレクター H またはその転置 H^T (または複素型の場合 H^H) を、実数/複素分布 $m \times n$ 行列 C に左または右のいずれかから適用する。現時点で、`storev = 'R'` と `direct = 'B'` のみがサポートされている。

入力パラメーター

<code>side</code>	CHARACTER*1。 <code>side = 'L'</code> の場合、 H または H' を左から適用する。 <code>side = 'R'</code> の場合、 H または H' を右から適用する。
<code>trans</code>	CHARACTER*1。 <code>trans = 'N'</code> の場合、 H を適用する (転置なし)。 <code>trans = 'C'</code> の場合、 H' を適用する (転置/共役転置)
<code>direct</code>	CHARACTER*1。基本リフレクターの積からどのように H が表現されているかを示す。 = 'F' の場合、 $H = H(1)H(2)\dots H(k)$ 順方向、現時点でこの機能はサポートされていない)。 = 'B' の場合、 $H = H(k)\dots H(2)H(1)$ (逆方向)。
<code>storev</code>	CHARACTER*1。リフレクターを定義するベクトルがどのように格納されているかを示す。 = 'C' の場合、列方向 (現時点でこの機能はサポートされていない)。 = 'R' の場合、行方向。
<code>m</code>	INTEGER。行列 C の行数。
<code>n</code>	INTEGER。行列 C の列数。
<code>k</code>	INTEGER。行列 T の次数 (積がブロック・リフレクターを定義する基本リフレクターの個数に等しい)。
<code>l</code>	INTEGER。Householder ベクトルとして意味を持った部分が格納されている行列 V の列数。 <code>side = 'L'</code> の場合、 $m \geq l \geq 0$ 、 <code>side = 'R'</code> の場合、 $n \geq l \geq 0$
<code>v</code>	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合) COMPLEX*16 (zlarzb の場合) 配列、次元は (ldv, nv)。 <code>storev = 'C'</code> の場合は $nv = k$ 、 <code>storev = 'R'</code> の場合は $nv = l$
<code>ldv</code>	INTEGER。配列 v のリーディング・ディメンジョン。 <code>storev = 'C'</code> の場合 $ldv \geq 1$ 、 <code>storev = 'R'</code> の場合 $ldv \geq k$
<code>t</code>	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合)

	COMPLEX*16 (zlarzb の場合) 配列、次元は (ldt,k)。ブロック・リフレクターの表現中にある三角 $k \times k$ 行列 T
ldt	INTEGER。配列 t のリーディング・ディメンジョン。 $ldt \geq k$
c	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合) COMPLEX*16 (zlarzb の場合) 配列、次元は (ldc,n)。 $m \times n$ の行列 C を格納する。
ldc	INTEGER。配列 c のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
work	REAL (slarzb の場合) DOUBLE PRECISION (dlarzb の場合) COMPLEX (clarzb の場合) COMPLEX*16 (zlarzb の場合) ワークスペース配列、次元は (ldwork, k)。
ldwork	INTEGER。配列 $work$ のリーディング・ディメンジョン。 $side = 'L'$ の場合、 $ldwork \geq \max(1, n)$ 、 $side = 'R'$ の場合、 $ldwork \geq \max(1, m)$

出力パラメーター

c	c は、 $H \cdot C$ 、または $H' \cdot C$ 、または $C \cdot H$ 、または $C \cdot H'$ で上書きされる。
---	--

?larzt

ブロック・リフレクター $H = I - VTV^H$ の三角係数 T を生成する。

構文

```
call slarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call dlarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call clarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
call zlarzt( direct, storev, n, k, v, ldv, tau, t, ldt )
```

説明

このルーチンは、 k 個の基本リフレクターの積として定義されている次数 $> n$ の実 / 複素ブロック・リフレクター H の三角係数 T を生成する。

$direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ と T は上三角である。

$direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ と T は下三角である。

$storev = 'c'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは、配列 v の i 番目の列に次のように格納される。

$$H = I - V \cdot T \cdot V'$$

$storev = 'c'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは、配列 v の i 番目の行に次のように格納される。

$$H = I - V^* T V$$

現時点で、 $storev = 'r'$ と $direct = 'b'$ のみがサポートされている。

入力パラメーター

$direct$	CHARACTER*1。ブロック・リフレクターを生成するために基本リフレクターを乗算する次数を指定する。 $direct = 'f'$ の場合、 $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない) $direct = 'b'$ の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
$storev$	CHARACTER*1。基本リフレクターを定義するベクトルがどのように格納されているかを指定する。(次のアプリケーション・ノートも参照): $storev = 'c'$ の場合、列方向 (現時点でこの機能はサポートされていない)。 $storev = 'r'$ の場合、行方向。
n	INTEGER。ブロック・リフレクター H の次数。 $n \geq 0$
k	INTEGER。三角係数 T の次数 (基本リフレクターの個数に等しい)。 $k \geq 1$
v	REAL (slarzt の場合) DOUBLE PRECISION (dlarzt の場合) COMPLEX (clarzt の場合) COMPLEX*16 (zlarzt の場合) 配列、次元は $storev = 'c'$ の場合では (ldv, k) $storev = 'r'$ の場合は (ldv, n) 。 行列 V 。
ldv	INTEGER。配列 v のリーディング・ディメンジョン。 $storev = 'c'$ の場合、 $ldv \geq \max(1, n)$ 、 $storev = 'r'$ の場合、 $ldv \geq k$ 。
tau	REAL (slarzt の場合) DOUBLE PRECISION (dlarzt の場合) COMPLEX (clarzt の場合) COMPLEX*16 (zlarzt の場合) 配列、次元は (k) 。 $tau(i)$ には、基本リフレクター $H(i)$ のスカラー係数が入っていないなければならない。
ldt	INTEGER。出力配列 t のリーディング・ディメンジョン。 $ldt \geq k$

出力パラメーター

t	REAL (slarzt の場合) DOUBLE PRECISION (dlarzt の場合) COMPLEX (clarzt の場合) COMPLEX*16 (zlarzt の場合)
-----	---

配列、次元は (ldt,k) 。ブロック・リフレクターの $k \times k$ 三角係数 T 。 $direct = 'F'$ の場合、 T は上三角。 $direct = 'B'$ の場合、 T は下三角。配列の残りの部分は使用されない。

v 行列 V 。次の「アプリケーション・ノート」を参照。

アプリケーション・ノート

行列 V の形状と $H(i)$ を定義するベクトルの格納形式の例を、 $n=5$ 、 $k=3$ において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復元される。配列の残りの部分は使用されない。

$direct = 'F'$ および $storev = 'C'$: $direct = 'F'$ および $storev = 'R'$:

$$V = \begin{bmatrix} v_1 & v_2 & v_3 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot \\ \cdot & 1 & \cdot \\ \cdot & \cdot & 1 \end{bmatrix} \qquad \begin{array}{c} \text{---}V\text{---} \\ / \qquad \qquad \backslash \\ \left[\begin{array}{cccccc} v_1 & v_1 & v_1 & v_1 & v_1 & \dots & 1 \\ v_2 & v_2 & v_2 & v_2 & v_2 & \dots & 1 \\ v_3 & v_3 & v_3 & v_3 & v_3 & \dots & 1 \end{array} \right] \end{array}$$

$direct = 'B'$ および $storev = 'C'$: $direct = 'B'$ および $storev = 'R'$:

$$V = \begin{bmatrix} 1 \\ \cdot & 1 \\ \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ v_1 & v_2 & v_3 \end{bmatrix} \qquad \begin{array}{c} \text{---}V\text{---} \\ / \qquad \qquad \backslash \\ \left[\begin{array}{cccccc} 1 & \dots & v_1 & v_1 & v_1 & v_1 & v_1 \\ \cdot & 1 & \dots & v_2 & v_2 & v_2 & v_2 \\ \cdot & \cdot & 1 & \dots & v_3 & v_3 & v_3 & v_3 \end{array} \right] \end{array}$$

?las2

2 × 2 三角行列の特異値を計算する。

構文

```
call slas2( f, g, h, ssmín, ssmáx )
call dlas2( f, g, h, ssmín, ssmáx )
```

説明

ルーチン ?las2 は 2 × 2 の行列、の特異値を計算する。

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

出力では、*ssmin* に小さい方の特異値、*ssmax* に大きい方の特異値が格納される。

入力パラメーター

f, g, h REAL (slas2 の場合)
 DOUBLE PRECISION (dlas2 の場合)
それぞれ、2 × 2 行列の (1,1)、(1,2)、(2,2) の成分。

出力パラメーター

ssmin, ssmáx REAL (slas2 の場合)
 DOUBLE PRECISION (dlas2 の場合)
それぞれ、小さい方の特異値と大きい方の特異値。

アプリケーション・ノート

オーバーフロー / アンダーフローの場合を除いて、加算 / 減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (*ulp*) で正確である。

IEEE 演算で、1 つの行列成分が無限でもコードは正しく動作する。

最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな *ulp* 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して 2 の因子以内にあるとオーバーフローが起こることがある。)

アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、結果は、アンダーフローしきい値に近いサイズの摂動によって変更された行列に一致することがある。

?lascl

一般矩形行列に c_{to}/c_{from} として定義される実数スカラーを乗算する。

構文

```
call slascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call dlascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call clascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
call zlascl( type, kl, ku, cfrom, cto, m, n, a, lda, info )
```

説明

ルーチン ?lascl は $m \times n$ 実数/複素行列 A に実数スカラー $cto/cfrom$ を乗算する。最終結果 $cto * A(i,j) / cfrom$ がオーバーフロー/アンダーフローしない限り、演算はオーバーフロー/アンダーフローなしで実行される。

A の形式は $type$ によって、フル、上三角、下三角、上 Hessenberg、または帯から指定される。

入力パラメーター

$type$	CHARACTER*1。 $type$ は入力行列の格納形式を示す。 = 'G' の場合、 A はフル行列。 = 'L' の場合、 A は下三角行列。 = 'U' の場合、 A は上三角行列。 = 'H' の場合、 A は上 Hessenberg 行列。 = 'B' の場合、 A は下バンド幅 kl と上バンド幅 ku を持つ対称帯行列で、下半分のみ格納される。 = 'Q' の場合、 A は下バンド幅 kl と上バンド幅 ku を持つ対称帯行列で、上半分のみ格納される。 = 'Z' の場合、 A は下バンド幅 kl と上バンド幅 ku を持つ対称帯行列。
kl	INTEGER。 A の下バンド幅。 $type = 'B', 'Q',$ または $'Z'$ の場合のみ参照される。
ku	INTEGER。 A の上バンド幅。 $type = 'B', 'Q',$ または $'Z'$ の場合のみ参照される。
$cfrom, cto$	REAL (slascl/clascl の場合) DOUBLE PRECISION (dlascl/zlascl の場合) 行列 A に乗算する $cto/cfrom$ 。最終結果 $cto * A(i, j) / cfrom$ がオーバーフロー/アンダーフローなしで表現される場合、 $A(i, j)$ はオーバーフロー/アンダーフローなしで計算される。 $cfrom$ は非ゼロでなければならない。
m	INTEGER。行列 A の行数。 $m \geq 0$ 。
n	INTEGER。行列 A の列数。 $n \geq 0$ 。

<i>a</i>	REAL (slascl の場合) DOUBLE PRECISION (dlascl の場合) COMPLEX (clascl の場合) COMPLEX*16 (zlascl の場合) 配列、次元は (<i>lda</i> , <i>m</i>)。 <i>cto/cfrom</i> が乗算される行列。格納形式については <i>type</i> を参照。
<i>lda</i>	INTEGER。配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

出力パラメーター

<i>a</i>	乗算された行列 <i>A</i>
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。

?lasd0

対角 *d* と非対角 *e* を持つ実数上二重対角 $n \times m$ 行列 *B* の特異値を計算する。
?bdsdc で使用される。

構文

```
call slasd0( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work, info )
call dlasd0( n, sqre, d, e, u, ldu, vt, ldvt, smlsiz, iwork, work, info )
```

説明

ルーチン ?lasd0 は、分割統治法を使って、対角 *d* と対角外 *e* を持つ実数上二重対角 $n \times m$ 行列 *B* の特異値分解 (SVD) を計算する。ここで $m = n + sqre$

このアルゴリズムは、 $B = U * S * VT$ であるような直交行列 *U* と *VT* を計算する。
特異値 *S* は *d* に上書きされる。

関連するサブルーチン [?lasda](#) は、特異値のみを計算し、オプションで特異ベクトルをコンパクト形式で計算する。

入力パラメーター

<i>n</i>	INTEGER。上二重対角行列の行次元を格納する。主対角配列 <i>d</i> の次元でもある。
<i>sqre</i>	INTEGER。二重対角行列の列のサイズを指定する。 <i>sqre</i> = 0 の場合、二重対角行列は列次元 $m = n$ を持つ。 <i>sqre</i> = 1 の場合、二重対角行列は列次元 $m = n + 1$ を持つ。
<i>d</i>	REAL (slasd0 の場合) DOUBLE PRECISION (dlasd0 の場合) 配列、次元は (<i>n</i>)。 <i>d</i> には二重対角行列の主対角を格納する。

<i>e</i>	REAL (slasd0 の場合) DOUBLE PRECISION (dlsasd0 の場合) 配列、次元は $(m-1)$ 。二重対角行列の劣対角成分を格納する。出力において <i>e</i> の内容は壊される。
<i>ldu</i>	INTEGER。出力配列 <i>u</i> のリーディング・ディメンジョンを格納する。
<i>ldvt</i>	INTEGER。出力配列 <i>vt</i> のリーディング・ディメンジョンを格納する。
<i>smlsiz</i>	INTEGER。計算ツリーが一番下の部分問題の最大サイズを格納する。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は $(8n)$ 以上でなければならない。
<i>work</i>	REAL (slasd0 の場合) DOUBLE PRECISION (dlsasd0 の場合) ワークスペース配列、次元は $(3m^2 + 2m)$ 以上でなければならない。

出力パラメーター

<i>d</i>	<i>info</i> = 0 の場合、 <i>d</i> には二重対角行列の特異値が上書きされる。
<i>u</i>	REAL (slasd0 の場合) DOUBLE PRECISION (dlsasd0 の場合) 配列、次元は (ldq, n) 以上。 <i>u</i> には左特異値が格納される。
<i>vt</i>	REAL (slasd0 の場合) DOUBLE PRECISION (dlsasd0 の場合) 配列、次元は $(ldvt, m)$ 以上。 <i>vt'</i> には右特異値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = $-i < 0$ の場合、 <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> = 1 の場合、特異値が収束しなかったことを示す。

?lasd1

指定サイズの上二重対角行列 *B* の SVD を計算する。*?bdsdc* で使用される。

構文

```
call slasd1( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq, iwork, work,
            info )
call dlsasd1( nl, nr, sqre, d, alpha, beta, u, ldu, vt, ldvt, idxq, iwork, work,
             info )
```

説明

このルーチンは上二重対角 $n \times m$ 行列 B の SVD を計算する。ここで $n = n1 + nr + 1$ および $m = n + sqre$ 。ルーチン `?lasd1` は `?lasd0` から呼び出される。

関連サブルーチン `?lasd7` は特異値 (および因子分解形式の特異ベクトル) を目的とする場合を扱う。

`?lasd1` は次のように SVD を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ Z1' & a & Z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \ 0) * VT(out)$$

$Z = (Z1' \ a \ Z2' \ b) = u' \ VT'$ 、 u は次元 m のベクトルで、 $n1+1$ 番目と $n1+2$ 番目の成分に $alpha$ と $beta$ を持ち他はゼロである。 $sqre = 0$ の場合、成分 b は空である。

元の行列の左特異ベクトルは u に格納され、右特異ベクトルの転置は vt に格納され、特異値は d に格納される。アルゴリズムは次の 3 過程で構成される。

1. 第 1 の過程では、複数の特異値がある場合、または Z ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに、永年方程式問題の次元は 1 だけ縮小される。この過程はルーチン `?lasd2` によって実行される。
2. 第 2 の過程では更新された特異値を計算する。これは、ルーチン `?lasd4` (`?lasd3` として呼び出される) を介して永年方程式の根の平方根を見つけることによって行われる。このルーチンは現在の問題の特異ベクトルも計算する。
3. 最後の過程では、更新された特異値を直接使用して更新された特異ベクトルを計算する。現在の問題に対する特異ベクトルは、全体問題から得られる特異ベクトルで乗算される。

入力パラメーター

<code>n1</code>	INTEGER。上ブロックの行次元。 $n1 \geq 1$
<code>nr</code>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<code>sqre</code>	INTEGER。 $sqre = 0$ の場合、下ブロックは $nr \times nr$ の正方行列。 $sqre = 1$ の場合、下ブロックは $nr \times (nr+1)$ の矩形行列。二重対角行列は行次元 $n = n1 + nr + 1$ 、列次元 $m = n + sqre$ を持つ。
<code>d</code>	REAL (<code>slasd1</code> の場合) DOUBLE PRECISION (<code>dlasd1</code> の場合) 配列、次元は $(n = n1 + nr + 1)$ 。 $d(1:n1, 1:n1)$ には上ブロックの特異値を格納し、 $d(n1+2:n)$ には下ブロックの特異値を格納する。
<code>alpha</code>	REAL (<code>slasd1</code> の場合) DOUBLE PRECISION (<code>dlasd1</code> の場合) 追加した行に関連する対角成分。

<i>beta</i>	REAL (<i>slasd1</i> の場合) DOUBLE PRECISION (<i>dlasd1</i> の場合) 追加した行に関連する非対角成分。
<i>u</i>	REAL (<i>slasd1</i> の場合) DOUBLE PRECISION (<i>dlasd1</i> の場合) 配列、次元は (<i>ldu</i> , <i>n</i>)。 <i>u</i> (1: <i>n1</i> , 1: <i>n1</i>) には上ブロックの左特異ベクトルを格納し、 <i>u</i> (<i>n1</i> +2: <i>n</i> , <i>n1</i> +2: <i>n</i>) には下ブロックの左特異ベクトルを格納する。
<i>ldu</i>	INTEGER。配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq \max(1, n)$
<i>vt</i>	REAL (<i>slasd1</i> の場合) DOUBLE PRECISION (<i>dlasd1</i> の場合) 配列、次元は (<i>ldvt</i> , <i>m</i>)、ここで $m = n + sqre$ 。 <i>vt</i> (1: <i>n1</i> +1, 1: <i>n1</i> +1)' には上ブロックの右特異ベクトルを格納し、 <i>vt</i> (<i>n1</i> +2: <i>m</i> , <i>n1</i> +2: <i>m</i>)' には下ブロックの右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq \max(1, m)$
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (4 <i>n</i>)
<i>work</i>	REAL (<i>slasd1</i> の場合) DOUBLE PRECISION (<i>dlasd1</i> の場合) ワークスペース配列、次元は ($3m^2 + 2m$)。

出力パラメーター

<i>d</i>	<i>d</i> (1: <i>n</i>) には、変更された行列の特異値が格納される。
<i>u</i>	<i>u</i> には二重対角行列の左特異ベクトルが格納される。
<i>vt</i>	<i>vt</i> ' には二重対角行列の右特異ベクトルが格納される。
<i>idxq</i>	INTEGER 配列、次元は (<i>n</i>)。解かれたばかりの部分問題をソートされた順に再統合する置換で上書きされる。すなわち、 <i>d</i> (<i>idxq</i> (<i>i</i> = 1, <i>n</i>)) は昇順となる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = - <i>i</i> < 0 の場合、 <i>i</i> 番目の引数の値が不正であったことを示す。 <i>info</i> = 1 の場合、特異値が収束しなかったことを示す。

?lasd2

2 つの特異値のセットをソートされた単一のセットに併合する。

?bdsdc で使用される。

構文

```
call slasd2( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt, dsigma, u2,
            ldu2, vt2, ldvt2, idxp, idx, idxc, idxq, coltyp, info )
```

```
call dlasd2( nl, nr, sqre, k, d, z, alpha, beta, u, ldu, vt, ldvt, dsigma, u2,
            ldu2, vt2, ldvt2, idxp, idx, idxc, idxq, coltyp, info )
```

説明

ルーチン ?lasd2 は 2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こるには 2 つの場合がある。2 個以上の特異値が互いに近い場合、または、Z ベクトルに微小成分がある場合である。そのような場合ごとに、関連する永年方程式問題の次元は 1 だけ縮小される。

ルーチン ?lasd2 は [?lasd1](#) から呼び出される。

入力パラメーター

<i>nl</i>	INTEGER。上ブロックの行次元。 $nl \geq 1$
<i>nr</i>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<i>sqre</i>	INTEGER。 <i>sqre</i> = 0 の場合、下ブロックは $nr \times nr$ の正方行列。 <i>sqre</i> = 1 の場合、下ブロックは $nr \times (nr+1)$ の矩形行列。二重対角行列は行次元 $n = nl + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<i>d</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (<i>n</i>)。 <i>d</i> には結合を行う 2 つの部分行列の特異値を格納する。
<i>alpha</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 追加した行に関連する非対角成分。
<i>u</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (<i>ldu</i> , <i>n</i>)。 <i>u</i> には、(1, 1), (<i>n</i> 1, <i>n</i> 1) と (<i>n</i> 1+2, <i>n</i> 1+2), (<i>n</i> , <i>n</i>) の隅にある 2 つの平方ブロック内の 2 個の部分行列の左特異ベクトルを格納する。
<i>ldu</i>	INTEGER。配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq n$

<i>ldu2</i>	INTEGER。出力配列 <i>u2</i> のリーディング・ディメンジョン。 $ldu2 \geq n$
<i>vt</i>	REAL (<i>slasd2</i> の場合) DOUBLE PRECISION (<i>dlsd2</i> の場合) 配列、次元は (<i>ldvt</i> , <i>m</i>)。 <i>vt'</i> には、(1, 1), (<i>n</i> +1, <i>n</i> +1) と (<i>n</i> +2, <i>n</i> +2), (<i>m</i> , <i>m</i>) の隅にある 2 つの平方ブロック内の 2 個の部分行列の右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq m$
<i>ldvt2</i>	INTEGER。出力配列 <i>vt2</i> のリーディング・ディメンジョン。 $ldvt2 \geq m$
<i>idxp</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。この配列には、収縮された <i>d</i> の値を配列の終わりに配置した置換を格納する。出力で、 <i>idxp</i> (2: <i>k</i>) は収縮されていない <i>d</i> 値を指し、 <i>idxp</i> (<i>k</i> +1: <i>n</i>) は収縮された特異値を指す。
<i>idx</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。 <i>d</i> の内容を昇順でソートしたときに使用した置換を格納する。
<i>coltyp</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。ワークスペースとして、 <i>u2</i> 行列内の列、または <i>vt2</i> 行列内の行を示すラベルを次のタイプから格納する。 1: 上半分のみゼロ 2: 下半分のみ非ゼロ 3: 密 4: 収縮
<i>idxq</i>	INTEGER。 配列、次元は (<i>n</i>)。 <i>d</i> に入っている 2 つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1 つの位置だけ後ろに動かされなければならない。また、後半にある成分は、最初にそれら値に <i>n</i> +1 を加算しなければならない。

出力パラメーター

<i>k</i>	INTEGER。収縮されていない行列の次元が格納される。 $1 \leq k \leq n$
<i>d</i>	<i>d</i> は、後続の (<i>n</i> - <i>k</i>) 個の更新された特異値 (収縮された) の昇順で上書きされる。
<i>u</i>	<i>u</i> の最後の <i>n</i> - <i>k</i> 個の列は、後続の (<i>n</i> - <i>k</i>) 個の更新された左特異値 (収縮された) で上書きされる。
<i>z</i>	REAL (<i>slasd2</i> の場合) DOUBLE PRECISION (<i>dlsd2</i> の場合) 配列、次元は (<i>n</i>)。 <i>z</i> は永年方程式内の更新列ベクトルで上書きされる。

<i>dsigma</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (n)。永年方程式内の対角成分 (k-1 個の特異値と 1 個のゼロ) のコピーが格納される。
<i>u2</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (1du2, n)。最初の k-1 個の左特異ベクトルのコピーが格納される。これは新しい左特異ベクトルを解くために ?lasd3 での行列乗算 (?gemm) で使用される。u2 は 4 つのブロックに配置される。第 1 のブロックは列 n1+1 が 1 で他はゼロである。第 2 のブロックには n1 とその上のみ非ゼロの成分が格納される。第 3 のブロックには n1+1 よりも下のみ非ゼロの成分が格納される。第 4 のブロックは密。
<i>vt</i>	<i>vt'</i> の最後の n-k 個の列は、後続の (n-k) 個の更新された右特異値 (収縮された) で上書きされる。sqre = 1 の場合、 <i>vt</i> の最後の行は右ヌル空間にかかる。
<i>vt2</i>	REAL (slasd2 の場合) DOUBLE PRECISION (dlasd2 の場合) 配列、次元は (1dvt2, n)。vt2' には最初の k 個の右特異ベクトルのコピーが格納される。これは新しい右特異ベクトルを解くために ?lasd3 での行列乗算 (?gemm) で使用される。vt2 は 3 つのブロックに配置される。第 1 のブロックには sigma 内の特殊 0 対角成分に対応する行が格納される。第 2 のブロックには n1+1 とその前のみ非ゼロが格納される。第 3 のブロックには n1+2 とその後ろのみ非ゼロが格納される。
<i>idxc</i>	INTEGER。 配列、次元は (n)。収縮された U 行列の列を 3 つのグループに配置する置換が格納される。第 1 のグループには n1 とその上に非ゼロの成分が格納され、第 2 のグループには n1+2 よりも下のみ非ゼロの成分が格納され、第 3 は密である。
<i>coltyp</i>	<i>coltyp(i)</i> が i 番目のタイプ列の次元となる次元 4 の配列で上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = -i < 0 の場合、i 番目の引数の値が不正であったことを示す。

?lasd3

D と Z 内の値によって定義されている永年方程式に対して根のすべての平方根を求め、次に行列乗算によって特異ベクトルを更新する。`?bdsdc` で使用される。

構文

```
call slasd3( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2, vt, ldvt,
            vt2, ldvt2, idxc, ctot, z, info )
call dlasd3( nl, nr, sqre, k, d, q, ldq, dsigma, u, ldu, u2, ldu2, vt, ldvt,
            vt2, ldvt2, idxc, ctot, z, info )
```

説明

ルーチン `?lasd3` は、 D と Z 内の値によって定義されている永年方程式に対して根のすべての平方根を求める。`?plasd4` に対する適切な呼び出しを実行し、続いて行列乗算によって特異ベクトルを更新する。

ルーチン `?lasd3` は `?plasd1` から呼び出される。

入力パラメーター

<code>nl</code>	INTEGER。上ブロックの行次元。 $nl \geq 1$
<code>nr</code>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<code>sqre</code>	INTEGER。 $sqre = 0$ の場合、下ブロックは $nr \times nr$ の正方行列。 $sqre = 1$ の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。二重対角行列は行次元 $n = nl + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<code>k</code>	INTEGER。永年方程式の大きさで、 $1 \leq k \leq n$
<code>q</code>	REAL (<code>slasd3</code> の場合) DOUBLE PRECISION (<code>dlasd3</code> の場合) ワークスペース配列、次元は (ldq, k) 以上。
<code>ldq</code>	INTEGER。配列 <code>q</code> のリーディング・ディメンジョン。 $ldq \geq k$
<code>dsigma</code>	REAL (<code>slasd3</code> の場合) DOUBLE PRECISION (<code>dlasd3</code> の場合) 配列、次元は (k) 。この配列の先頭の k 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<code>u</code>	REAL (<code>slasd3</code> の場合) DOUBLE PRECISION (<code>dlasd3</code> の場合) 配列、次元は (ldu, n) 。この行列の最後の $n-k$ 個の列には収縮された左特異ベクトルを格納する。

<i>ldu</i>	INTEGER。配列 <i>u</i> のリーディング・ディメンジョン。 $ldu \geq n$
<i>u2</i>	REAL (<i>slasd3</i> の場合) DOUBLE PRECISION (<i>dlasd3</i> の場合) 配列、次元は (<i>ldu2</i> , <i>n</i>)。この行列の先頭の <i>k</i> 個の列には分割問題に対する収縮されていない左特異ベクトルを格納する。
<i>ldu2</i>	INTEGER。配列 <i>u2</i> のリーディング・ディメンジョン。 $ldu2 \geq n$
<i>vt</i>	REAL (<i>slasd3</i> の場合) DOUBLE PRECISION (<i>dlasd3</i> の場合) 配列、次元は (<i>ldvt</i> , <i>m</i>)。vt' の最後の <i>m-k</i> 個の列には収縮された右特異ベクトルを格納する。
<i>ldvt</i>	INTEGER。配列 <i>vt</i> のリーディング・ディメンジョン。 $ldvt \geq n$
<i>vt2</i>	REAL (<i>slasd3</i> の場合) DOUBLE PRECISION (<i>dlasd3</i> の場合) 配列、次元は (<i>ldvt2</i> , <i>n</i>)。vt2' の先頭の <i>k</i> 個の列には分割問題に対する収縮されていない右特異ベクトルを格納する。
<i>ldvt2</i>	INTEGER。配列 <i>vt2</i> のリーディング・ディメンジョン。 $ldvt2 \geq n$
<i>idxc</i>	INTEGER。 配列、次元は (<i>n</i>)。u の列 (および vt の行) を 3 つのグループに配置する置換。第 1 のグループには <i>n1+1</i> とその上 (または前) のみに非ゼロの成分が格納される。第 2 のグループには <i>n1+2</i> とその下 (または後ろ) にのみ非ゼロの成分が格納される。第 3 のグループは密である。ただし、u の最初の列と vt の最初の行は別々に扱われる。?lasd4 で求められた特異ベクトルの行は、行列乗算が実行される前に、同様に置換されなければならない。
<i>ctot</i>	INTEGER。 配列、次元は (4)。idxc に記述されているとおり、u に含まれる列 (または vt に含まれる行) のタイプのそれぞれの合計。4 番目の列タイプは収縮された任意の列である。
<i>z</i>	REAL (<i>slasd3</i> の場合) DOUBLE PRECISION (<i>dlasd3</i> の場合) 配列、次元は (<i>k</i>)。この配列の先頭の <i>k</i> 個の成分には収縮調整された更新行ベクトルの成分を格納する。

出力パラメーター

<i>d</i>	REAL (<i>slasd3</i> の場合) DOUBLE PRECISION (<i>dlasd3</i> の場合) 配列、次元は (<i>k</i>)。永年方程式の根の平方根が昇順で上書きされる。
----------	--

info INTEGER。
info = 0 の場合、正常に終了したことを示す。
info = -*i* < 0 の場合、*i* 番目の引数の値が不正だったことを示す。
info = 1 の場合、特異値が収束しなかったことを示す。

アプリケーション・ノート

このコードは浮動小数点演算に対してきわめて緩い仮定を行う。加減算においてガード桁を持つマシン、または Cray X-MP、Cray Y-MP、Cray C-90、または Cray-2 のような減算のガード桁を持たないバイナリーマシンで動作する。ガード桁を持たない 16 進マシンや 10 進マシンでは異常終了するのも考えられるが、そのような前例はない。

?lasd4

正値対角行列に対する正値対称階数 1 更新の、
i 番目の更新された固有値の平方根を計算する。
 ?bdsdc で使用される。

構文

```
call slasd4( n, i, d, z, delta, rho, sigma, work, info )
call dlasd4( n, i, d, z, delta, rho, sigma, work, info )
```

説明

このルーチンは、正値対角行列に対する正値対称階数 1 の更新の、*i* 番目の更新された固有値の平方根を計算する。正値対角行列の成分は、配列 *d* 中の対応する成分の平方として与えられ、*i* < *j* では $0 \leq d(i) < d(j)$ 、および $\rho > 0$ である。これは呼び出しルーチンによって配置され、一般性が失われることはない。階数 1 の更新された連立方程式はゆえに、

$$\text{diag}(d) * \text{diag}(d) + \rho * Z * Z_{\text{transpose}}$$

Z のユークリッド・ノルムを 1 とする。この方法には、単純補間の有利関数を用いた永久方程式内の有利関数の近似が含まれる。

入力パラメーター

n INTEGER。全配列の長さ。
i INTEGER。計算する固有値のインデックス。 $1 \leq i \leq n$
d REAL (slasd4 の場合)
 DOUBLE PRECISION (dlasd4 の場合)
 配列、次元は (*n*)。
 元の固有値。 *i* < *j* では $0 \leq d(i) < d(j)$ のように順序どおりに並べられていると仮定される。
z REAL (slasd4 の場合)
 DOUBLE PRECISION (dlasd4 の場合)
 配列、次元は (*n*)。
 更新ベクトルの成分。

<i>rho</i>	REAL (slasd4 の場合) DOUBLE PRECISION (dlasd4 の場合) 対称更新式におけるスカラー。
<i>work</i>	REAL (slasd4 の場合) DOUBLE PRECISION (dlasd4 の場合) ワークスペース配列、次元は (<i>n</i>)。 <i>n</i> ≠ 1 の場合、 <i>work</i> の <i>j</i> 番目の成分に (<i>d</i> (<i>j</i>) + <i>sigma</i> _{<i>i</i>}) を格納する。 <i>n</i> = 1 の場合、 <i>work</i> (1) = 1

出力パラメーター

<i>delta</i>	REAL (slasd4 の場合) DOUBLE PRECISION (dlasd4 の場合) 配列、次元は (<i>n</i>)。 <i>n</i> ≠ 1 の場合、 <i>delta</i> の <i>j</i> 番目の成分に (<i>d</i> (<i>j</i>) - <i>sigma</i> _{<i>i</i>}) が格納される。 <i>n</i> = 1 の場合、 <i>delta</i> (1) = 1。ベクトル <i>delta</i> には (特異) 固有ベクトルを構成するために必要な情報が格納される。
<i>sigma</i>	REAL (slasd4 の場合) DOUBLE PRECISION (dlasd4 の場合) 計算された λ_i 、 <i>i</i> 番目の更新された固有値。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 > 0 の場合で <i>info</i> = 1 ならば、更新処理に失敗したことを示す。

?lasd5

2 × 2 対角行列の正値対称階数 1 更新から、*i* 番目の固有値の平方根を計算する。*?bdsdc* で使用される。

構文

```
call slasd5( i, d, z, delta, rho, dsigma, work )
call dlasd5( i, d, z, delta, rho, dsigma, work )
```

説明

このルーチンは、2 × 2 対角行列の正値対称階数 1 更新から、*i* 番目の固有値の平方根を計算する。

$$\text{diag}(d) * \text{diag}(d) + \text{rho} * Z * Z_{\text{transpose}}$$

配列 *d* の対角成分は *i* < *j* において $0 \leq d(i) < d(j)$ を満たすものとする。合わせて、*rho* > 0、ベクトル *Z* のユークリッド・ノルムは 1 であるとする。

入力パラメーター

i INTEGER。計算する固有値のインデックス。*i* = 1 または *i* = 2

<i>d</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) 配列、次元は (2)。 元の固有値。 $0 \leq d(1) < d(2)$ とする。
<i>z</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) 配列、次元は (2)。 更新ベクトルの成分。
<i>rho</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) 対称更新式におけるスカラー。
<i>work</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) ワークスペース配列、次元は (2) <i>work</i> の <i>j</i> 番目の成分には ($d(j) + \text{sigma}_i$) を格納する。

出力パラメーター

<i>delta</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) 配列、次元は (2)。 <i>delta</i> の <i>j</i> 番目の成分に ($d(j) - \lambda_j$) が格納される。ベクトル <i>delta</i> には固有ベクトルを構成するために必要な情報が格納される。
<i>dsigma</i>	REAL (slasd5 の場合) DOUBLE PRECISION (dlsd5 の場合) 計算された λ_j 、 <i>i</i> 番目の更新された固有値。

?lasd6

2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列の SVD を計算する。
?bdsdc で使用される。

構文

```
call slasd6(  icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq, perm, givptr,
             givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z, k, c, s, work, iwork,
             info )
call dlsd6(  icompq, nl, nr, sqre, d, vf, vl, alpha, beta, idxq, perm, givptr,
             givcol, ldgcol, givnum, ldgnum, poles, difl, difr, z, k, c, s, work, iwork,
             info )
```

説明

ルーチン ?lasd6 は、2 つの小さい行列を行追加によって併合して得た、更新された上二重対角行列 *B* の SVD を計算する。このルーチンは、特異値すべてと、オプションで因子分解形式の特異ベクトル行列を必要とする問題でのみ使用される。*B* は $n \times m$ の行

列で、 $n = n1 + nr + 1$ かつ $m = n + sqre$ である。関連サブルーチン [?lasd1](#) は、二重対角行列のすべての特異値と特異ベクトルが必要な場合を扱う。[?lasd6](#) は次のように *SVD* を計算する。

$$B = U(in) * \begin{bmatrix} D1(in) & 0 & 0 & 0 \\ z1' & a & z2' & b \\ 0 & 0 & D2(in) & 0 \end{bmatrix} * VT(in)$$

$$= U(out) * (D(out) \ 0) * VT(out)$$

$Z = (z1' \ a \ z2' \ b) = u' \ VT$ 、 u は次元 m のベクトルで、 $n1+1$ 番目と $n1+2$ 番目の成分に α と β を持ち他はゼロである。 $sqre = 0$ の場合、成分 b は空である。

B の特異値は、下ブロックの全右特異ベクトルの最初の成分 $D1$ と、上ブロックの全右特異ベクトルの最後の成分 $D2$ を使って計算できる。これら成分は [?lasd6](#) の中で、それぞれ vf と $v1$ に格納され更新される。そのため U と VT は明示的には参照されない。特異値は D に格納される。アルゴリズムは2つの過程で構成される。

1. 第1の過程では、複数の特異値がある場合、または Z ベクトルにゼロがある場合、問題の大きさの収縮を行う。そのような場合ごとに、永年方程式問題の次元は1だけ縮小される。この過程はルーチン [?lasd7](#) によって実行される。
2. 第2の過程では更新された特異値を計算する。これは、ルーチン [?lasd4](#) ([?lasd8](#) として呼び出される) を介して永年方程式の根を見つけることによつて行われる。また、このルーチンは、 vf と $v1$ を更新し、更新された特異値と古い特異値の距離を計算する。[?lasd6](#) は [?lasda](#) から呼び出される。

入力パラメーター

<i>icompg</i>	INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。 = 0 の場合、特異値のみ計算する。 = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。
<i>n1</i>	INTEGER。上ブロックの行次元。 $n1 \geq 1$
<i>nr</i>	INTEGER。下ブロックの行次元。 $nr \geq 1$
<i>sqre</i>	INTEGER。 = 0 の場合、下ブロックは $nr \times nr$ の正方行列。 = 1 の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。 二重対角行列は行次元 $n = n1 + nr + 1$ 、列次元 $m = n + sqre$ を持つ。
<i>d</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) 配列、次元は $(n1+nr+1)$ 。 $d(1:n1, 1:n1)$ には上ブロックの特異値を格納し、 $d(n1+2:n)$ には下ブロックの特異値を格納する。

<i>vf</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) 配列、次元は (m)。 <i>vf</i> (1:n1+1) には上ブロックの全右特異ベクトルの最初の成分を格納し、 <i>vf</i> (n1+2:m) には下ブロックの全右左特異ベクトルの最初の成分を格納する。
<i>v1</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) 配列、次元は (m)。 <i>v1</i> (1:n1+1) には上ブロックの全右特異ベクトルの最後の成分を格納し、 <i>v1</i> (n1+2:m) には下ブロックの全右左特異ベクトルの最後の成分を格納する。
<i>alpha</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) 追加した行に関連する非対角成分。
<i>ldgcol</i>	INTEGER。出力配列 <i>givcol</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>ldgnum</i>	INTEGER。出力配列 <i>givnum</i> 、 <i>poles</i> のリーディング・ディメンジョン。 <i>n</i> 以上でなければならない。
<i>work</i>	REAL (slasd6 の場合) DOUBLE PRECISION (dlasd6 の場合) ワークスペース配列、次元は (4m)。
<i>iwork</i>	INTEGER ワークスペース 配列、次元は (3n)

出力パラメーター

<i>d</i>	<i>d</i> (1:n) は更新された行列の固有値で上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異ベクトルの最初の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分で上書きされる。
<i>idxq</i>	INTEGER。 配列、次元は (n)。解かれたばかりの部分問題をソートされた順に再統合する置換で上書きされる。すなわち、 <i>d</i> (<i>idxq</i> (<i>i</i> = 1, n)) は昇順となる。
<i>perm</i>	INTEGER。 配列、次元は (n)。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。 <i>icompq</i> = 0 の場合は参照されない。

<i>givcol</i>	INTEGER。 配列、次元は (<i>ldgcol</i> , 2)。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。 <i>icompq</i> = 0 の場合は参照されない。
<i>givnum</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) 配列、次元は (<i>ldgnum</i> , 2)。それぞれの値は対応する Givens 回転で使用される <i>C</i> 値または <i>S</i> 値を示す。 <i>icompq</i> = 0 の場合は参照されない。
<i>poles</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) 配列、次元は (<i>ldgnum</i> , 2)。配列 <i>poles</i> (1, *) には永年方程式を解いて得られた新しい特異値が格納される。配列 <i>poles</i> (2, *) には永年方程式内の極が格納される。 <i>icompq</i> = 0 の場合は参照されない。
<i>difl</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) 配列、次元は (<i>n</i>)。 <i>difl</i> (<i>i</i>) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i> 番目の (収縮されていない) 古い特異値との距離が格納される。
<i>difr</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) 配列、 次元は、(<i>ldgnum</i> , 2) (<i>icompq</i> = 1 の場合)、 次元は、(<i>n</i>) (<i>icompq</i> = 0 の場合)。 <i>difr</i> (<i>i</i> , 1) には <i>i</i> 番目の更新された (収縮されていない) 特異値と <i>i</i> +1 番目の (収縮されていない) 古い特異値との距離が格納される。 <i>icompq</i> = 1 の場合、配列 <i>difr</i> (1: <i>k</i> , 2) には右特異ベクトル行列に対する正規化係数が格納される。 <i>difl</i> と <i>difr</i> の詳細は ? <i>lasd8</i> を参照のこと。
<i>z</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) 配列、次元は (<i>m</i>)。 この配列の先頭成分には収縮調整された更新行ベクトルが格納される。
<i>k</i>	INTEGER。非収縮行列の次元を格納する。これは関連する永年方程式の次数である。 $1 \leq k \leq n$ 。
<i>c</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) <i>sqre</i> = 0 の場合はガベージが格納され、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>C</i> 値が <i>c</i> に格納される。
<i>S</i>	REAL (<i>slasd6</i> の場合) DOUBLE PRECISION (<i>dlsasd6</i> の場合) <i>sqre</i> = 0 の場合はガベージが格納され、 <i>sqre</i> = 1 の場合は右ヌル空間に関する Givens 回転の <i>S</i> 値が <i>s</i> に格納される。

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
 > 0 の場合、*info* = 1 は固有値が収束しなかったことを示す。

?lasd7

2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。*?bdsdc* で使用される。

構文

```
call slasd7( icompg, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw, alpha, beta,
            dsigma, idx, idxp, idxq, perm, givptr, givcol, ldgcol, givnum, ldgnum, c, s,
            info )
```

```
call dlasd7( icompg, nl, nr, sqre, k, d, z, zw, vf, vfw, vl, vlw, alpha, beta,
            dsigma, idx, idxp, idxq, perm, givptr, givcol, ldgcol, givnum, ldgnum, c, s,
            info )
```

説明

ルーチン *?lasd7* は 2 つの特異値のセットをソートされた単一のセットに併合する。続いて、問題の大きさの収縮を試みる。収縮が起こり得るには 2 つの場合がある。2 個以上の特異値が互いに近い場合、または、Z ベクトルに微小成分がある場合である。そのような条件が発生するごとに、関連する永年方程式問題の次元は 1 だけ縮小される。*?lasd7* は [?lasd6](#) から呼び出される。

入力パラメーター

icompg INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。
 = 0 の場合、特異値のみ計算する。
 = 1 の場合、上二重対角行列の特異ベクトルをコンパクト形式で計算する。

nl INTEGER。上ブロックの行次元。
 $nl \geq 1$

nr INTEGER。下ブロックの行次元。
 $nr \geq 1$

sqre INTEGER。
 = 0 の場合、下ブロックは $nr \times nr$ 平方行列である。
 = 1 の場合、下ブロックは $nr \times (nr+1)$ 矩形行列である。二重対角行列は行次元 $n = nl + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。

d REAL (*slasd7* の場合)
 DOUBLE PRECISION (*dlasd7* の場合)
 配列、次元は (*n*)。 *d* には結合を行う 2 つの部分行列の特異値を格納する。

<i>zw</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (<i>m</i>)。z 用のワークスペース。
<i>vf</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (<i>m</i>)。vf(1:n1+1)には上ブロックの全右特異ベクトルの最初の成分を格納し、vf(n1+2:m)には下ブロックの全右左特異ベクトルの最初の成分を格納する。
<i>vfw</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (<i>m</i>)。vf 用のワークスペース。
<i>v1</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (<i>m</i>)。v1(1:n1+1)には上ブロックの全右特異ベクトルの最後の成分を格納し、v1(n1+2:m)には下ブロックの全右左特異ベクトルの最後の成分を格納する。
<i>v1w</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (<i>m</i>)。v1 用のワークスペース。
<i>alpha</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 追加した行に関連する対角成分。
<i>beta</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 追加した行に関連する非対角成分。
<i>idx</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。d の内容を昇順でソートしたときに使用した置換を格納する。
<i>idxp</i>	INTEGER。 ワークスペース配列、次元は (<i>n</i>)。収縮した d の値を配列の終わりに配置した置換を格納する。
<i>idxq</i>	INTEGER。 配列、次元は (<i>n</i>)。d に入っている 2つの部分問題を昇順で別々にソートする置換。この置換の前半にある成分は、最初に 1つの位置だけ後ろに動かされなければならない。また、この置換の後半にある成分は、最初にそれら値に n1+1 を加算しなければならない。
<i>ldgcol</i>	INTEGER。出力配列 <i>givcol</i> のリーディング・ディメンジョン。 n 以上でなければならない。
<i>ldgnum</i>	INTEGER。出力配列 <i>givnum</i> のリーディング・ディメンジョン。 n 以上でなければならない。

出力パラメーター

<i>k</i>	INTEGER。収縮されていない行列の次元が格納される。これは関連する永年方程式の次数である。 $1 \leq k \leq n$
<i>d</i>	<i>d</i> は、後続の $(n-k)$ 個の更新された特異値 (収縮された) の昇順で上書きされる。
<i>z</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (m) 。 <i>z</i> は永年方程式内の更新列ベクトルで上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異値の最初の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分で上書きされる。
<i>dsigma</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は (n) 。永年方程式内の対角成分 ($k-1$ 個の特異値と 1 個のゼロ) のコピーが格納される。
<i>idxp</i>	<i>idxp(2:k)</i> は収縮されていない <i>d</i> 値を指し、 <i>idxp(k+1:n)</i> は収縮された特異値を指す。
<i>perm</i>	INTEGER。 配列、次元は (n) 。各固有ブロックに適用される (収縮とソートによる) 置換が格納される。 <i>icompg = 0</i> の場合は参照されない。
<i>givptr</i>	INTEGER。この部分問題で実行された Givens 回転の回数が格納される。 <i>icompg = 0</i> の場合は参照されない。
<i>givcol</i>	INTEGER。 配列、次元は $(ldgcol, 2)$ 。それぞれの数のペアは Givens 回転で対象となる列ペアを示す。 <i>icompg = 0</i> の場合は参照されない。
<i>givnum</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) 配列、次元は $(ldgnum, 2)$ 。それぞれの値は対応する Givens 回転で使用される <i>C</i> 値または <i>S</i> 値を示す。 <i>icompg = 0</i> の場合は参照されない。
<i>c</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) <i>sqre = 0</i> の場合はガベージが格納され、 <i>sqre = 1</i> の場合は右ヌル空間に関する Givens 回転の <i>C</i> 値が <i>c</i> に格納される。
<i>s</i>	REAL (slasd7 の場合) DOUBLE PRECISION (dlasd7 の場合) <i>sqre = 0</i> の場合はガベージが格納され、 <i>sqre = 1</i> の場合は右ヌル空間に関する Givens 回転の <i>S</i> 値が <i>s</i> に格納される。

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?lasd8

永年方程式の根の平方根を求め、*D* の各成分に対して最も近い 2 つの極までの距離を格納する。
 ?bdsdc で使用される。

構文

```
call slasd8( icompg, k, d, z, vf, vl, difl, difr, lddifr, dsigma, work, info )
call dlasd8( icompg, k, d, z, vf, vl, difl, difr, lddifr, dsigma, work, info )
```

説明

ルーチン ?lasd8 は、*dsigma* と *z* の値によって定義されている永年方程式に対して根の平方根を求める。[?lasd4](#) に対する適切な呼び出しを実行し、続いて *d* の各成分に対して最も近い 2 つの極までの距離 (*dsigma* 内の成分) を格納する。また、元の二重対角行列の全右特異ベクトルの最初と最後の成分である配列 *vf* と *vl* を更新する。?lasd8 は [?lasd6](#) から呼び出される。

入力パラメーター

icompg INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。
 = 0 の場合、特異値のみ計算する。
 = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。

k INTEGER。?lasd4 で解かれる有利関数の項目数。 $k \geq 1$

z REAL (slasd8 の場合)
 DOUBLE PRECISION (dlasd8 の場合)
 配列、次元は (*k*)。この配列の先頭の *k* 個の成分には収縮調整された更新行ベクトルの成分を格納する。

vf REAL (slasd8 の場合)
 DOUBLE PRECISION (dlasd8 の場合)
 配列、次元は (*k*)。 *vf* には *dbede8* から渡された情報を格納する。

vl REAL (slasd8 の場合)
 DOUBLE PRECISION (dlasd8 の場合)
 配列、次元は (*k*)。 *vl* には *dbede8* から渡された情報を格納する。

lddifr INTEGER。出力配列 *diffr* のリーディング・ディメンジョン。
k 以上でなければならない。

<i>dsigma</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlsasd8 の場合) 配列、次元は (k)。この配列の先頭の k 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<i>work</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlsasd8 の場合) ワークスペース配列、次元は $(3k)$ 以上。

出力パラメーター

<i>d</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlsasd8 の場合) 配列、次元は (k)。 <i>d</i> は更新された特異ベクトルで上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異ベクトルの最初の成分の先頭の k 個の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分の先頭の k 個の成分で上書きされる。
<i>difl</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlsasd8 の場合) 配列、次元は (k)。 $difl(i) = d(i) - dsigma(i)$ が格納される。
<i>difr</i>	REAL (slasd8 の場合) DOUBLE PRECISION (dlsasd8 の場合) 配列、 次元は、 $(lddifr, 2)$ ($icompg = 1$ の場合) または (k) ($icompg = 0$ の場合) $difr(i, 1) = d(i) - dsigma(i+1)$ が格納される。 $difr(k, 1)$ は定義されてなく参照されない。 $icompg = 1$ の場合、配列 $difr(1:k, 2)$ には右特異ベクトル行列に対する正規化係数が格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は i 番目の引数の値が不正だったことを示す。 > 0 の場合、 $info = 1$ は固有値が収束しなかったことを示す。

?lasd9

永年方程式の根の平方根を求め、 D の各成分に対して最も近い2つの極までの距離を格納する。
?bdsdc で使用される。

構文

```
call slasd9( icompg, ldu, k, d, z, vf, vl, difl, difr, dsigma, work, info )
call dlasd9( icompg, ldu, k, d, z, vf, vl, difl, difr, dsigma, work, info )
```

説明

ルーチン ?lasd9 は、 $dsigma$ と z の値によって定義されている永年方程式に対して根の平方根を求める。このルーチンは [?lasd4](#) に対する適切な呼び出しを実行し、続いて d の各成分に対して最も近い2つの極までの距離 ($dsigma$ 内の成分) を格納する。また、元の二重対角行列の全右特異ベクトルの最初と最後の成分である配列 vf と vl を更新する。?lasd9 は [?lasd7](#) から呼び出される。

入力パラメーター

<i>icompg</i>	INTEGER。呼び出されたルーチン内で、因子分解形式で特異ベクトルを計算するかどうかを指定する。 <i>icompg</i> = 0 の場合、特異値のみ計算する。 <i>icompg</i> = 1 の場合、合わせて特異ベクトルを因子分解形式で計算する。
<i>k</i>	INTEGER。slasd4 で解かれる有利関数の項目数。 $k \geq 1$
<i>dsigma</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (k)。この配列の先頭の k 個の成分には収縮された更新問題の古い根を格納する。これらは永年方程式の極である。
<i>z</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (k)。この配列の先頭の k 個の成分には収縮調整された更新行ベクトルの成分を格納する。
<i>vf</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (k)。 <i>vf</i> には sbede8 から渡された情報を格納する。
<i>vl</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (k)。 <i>vl</i> には sbede8 から渡された情報を格納する。
<i>work</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) ワークスペース配列、次元は ($3k$) 以上。

出力パラメーター

<i>d</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (<i>k</i>)。 <i>d</i> (<i>i</i>) は更新された特異ベクトルで上書きされる。
<i>vf</i>	<i>vf</i> は二重対角行列の全右特異ベクトルの最初の成分の先頭の <i>k</i> 個の成分で上書きされる。
<i>v1</i>	<i>v1</i> は二重対角行列の全右特異ベクトルの最後の成分の先頭の <i>k</i> 個の成分で上書きされる。
<i>difl</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、次元は (<i>k</i>)。 <i>difl</i> (<i>i</i>) = <i>d</i> (<i>i</i>) - <i>dsigma</i> (<i>i</i>) が格納される。
<i>difr</i>	REAL (slasd9 の場合) DOUBLE PRECISION (dlasd9 の場合) 配列、 次元は、(<i>ldu</i> , 2) (<i>icompq</i> = 1 の場合) または (<i>k</i>) (<i>icompq</i> = 0 の場合) <i>difr</i> (<i>i</i> , 1) = <i>d</i> (<i>i</i>) - <i>dsigma</i> (<i>i</i> +1) が格納される。 <i>difr</i> (<i>k</i> , 1) は定義されてなく参照されない。 <i>icompq</i> = 1 の場合、配列 <i>difr</i> (1: <i>k</i> , 2) には右特異ベクトル行列に対する正規化係数が格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 > 0 の場合、 <i>info</i> = 1 は固有値が収束しなかったことを示す。

?lasda

対角 *d* と非対角 *e* を持つ実数上二重対角行列の特異値分解 (SVD) を計算する。?bdsdc で使用される。

構文

```
call slasda( icompq, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl, difr, z, poles,
            givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork, info )
call dlasda( icompq, smlsiz, n, sqre, d, e, u, ldu, vt, k, difl, difr, z, poles,
            givptr, givcol, ldgcol, perm, givnum, c, s, work, iwork, info )
```

説明

ルーチン `?lasda` は、分割統治法を使って、対角 d と非対角 e を持つ実数上二重対角 $n \times m$ 行列 B の特異値分解 (SVD) を計算する。

ここで $m = n + sqre$ 。このアルゴリズムは SVD で特異値 $B = U * S * VT$ を計算する。直交行列 U と VT の計算はオプションであり、コンパクト形式で計算される。関連するサブルーチン `?lasd0` は特異値と特異ベクトルを黙示的な形式で計算する。

入力パラメーター

<code>icompg</code>	INTEGER。コンパクト形式で特異ベクトルを計算するかどうかを指定する。 = 0 の場合、特異値のみ計算する。 = 1 の場合、上二重対角行列の特異ベクトルをコンパクト形式で計算する。
<code>smlsiz</code>	INTEGER。計算ツリーの一番下の部分問題の最大サイズ。
<code>n</code>	INTEGER。上二重対角行列の行次元。主対角配列 d の次元でもある。
<code>sqre</code>	INTEGER。二重対角行列の列のサイズを指定する。 $sqre = 0$ の場合、二重対角行列は列次元 $m = n$ を持つ。 $sqre = 1$ の場合、二重対角行列は列次元 $m = n + 1$ を持つ。
<code>d</code>	REAL (<code>slasda</code> の場合) DOUBLE PRECISION (<code>dlasda</code> の場合) 配列、次元は (n) 。 d には二重対角行列の主対角を格納する。
<code>e</code>	REAL (<code>slasda</code> の場合) DOUBLE PRECISION (<code>dlasda</code> の場合) 配列、次元は $(m - 1)$ 。二重対角行列の劣対角成分を格納する。出力において e の内容は壊される。
<code>ldu</code>	INTEGER。配列 u 、 vt 、 $difl$ 、 $difr$ 、 $poles$ 、 $givnum$ 、 z のリーディング・ディメンジョン。 $ldu \geq n$ 。
<code>ldgcol</code>	INTEGER。配列 $givcol$ 、 $perm$ のリーディング・ディメンジョン。 $ldgcol \geq n$
<code>work</code>	REAL (<code>slasda</code> の場合) DOUBLE PRECISION (<code>dlasda</code> の場合) ワークスペース配列、次元は $(6n + (smlsiz + 1)^2)$ 。
<code>iwork</code>	INTEGER。 ワークスペース配列、次元は $(7n)$ 以上でなければならない。

出力パラメーター

<code>d</code>	$info = 0$ の場合、二重対角行列の特異値が格納される。
<code>u</code>	REAL (<code>slasda</code> の場合) DOUBLE PRECISION (<code>dlasda</code> の場合) 配列、次元は $(ldu, smlsiz)$ ($icompg = 1$ の場合)

	<p>$icompq = 0$ の場合は参照されない。 $icompq = 1$ の場合、u には一番下のレベルの全部分問題の左特異ベクトル行列が格納される。</p>
<i>vt</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、次元は ($ldu, smlsiz+1$) ($icompq = 1$ の場合)。 $icompq = 0$ の場合は参照されない。 $icompq = 1$ の場合、vt には一番下のレベルの全部分問題の右特異ベクトル行列が格納される。</p>
<i>k</i>	<p>INTEGER。 配列、 次元は (n) ($icompq = 1$ の場合) または 次元は (1) ($icompq = 0$ の場合)。 $icompq = 1$ の場合、$k(i)$ には計算ツリーにおける i 番目の永年方程式の次元が格納される。</p>
<i>difl</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、次元は ($ldu, nlv1$)、 ここで $nlv1 = \text{floor}(\log_2(n/smlsiz))$。</p>
<i>difr</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、 次元は ($ldu, 2 nlv1$) ($icompq = 1$ の場合) または (n) ($icompq = 0$ の場合)。 $icompq = 1$ の場合、$difl(1:n, i)$ と $difr(1:n, 2i-1)$ には、i 番目のレベルの特異値と ($i-1$) 番目のレベルの特異値との距離が記録され、$difr(1:n, 2i)$ には右特異ベクトル行列に対する正規化係数が格納される。詳細は ?lasd8 を参照のこと。</p>
<i>z</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、 次元は ($ldu, nlv1$) ($icompq = 1$ の場合) または (n) ($icompq = 0$ の場合)。 $z(1, i)$ の最初の k 個の成分には、i 番目のレベルの部分問題に対する収縮調整された更新行ベクトルの成分が格納される。</p>
<i>poles</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合) 配列、次元は ($ldu, 2*nlv1$) ($icompq = 1$ の場合)。 $icompq = 0$ の場合は参照されない。 $icompq = 1$ の場合、$poles(1, 2i-1)$ と $poles(1, 2i)$ には i 番目レベルの永年方程式に関する新しい特異値と古い特異値が格納される。</p>
<i>givptr</i>	<p>INTEGER。 配列、次元は (n) ($icompq = 1$ の場合)。 $icompq = 0$ の場合は参照されない。 $icompq = 1$ の場合、$givptr(i)$ には計算ツリーの i 番目問題で実行された Givens 回転の回数が格納される。</p>

<i>givcol</i>	<p>INTEGER。</p> <p>配列、次元は $(ldgcol, 2*nlv1)$ ($icompq=1$ の場合)。 $icompq=0$ の場合は参照されない。 $icompq=1$ の場合、$givcol(1, 2i-1)$ と $givcol(1, 2i)$ には計算ツリーの i 番目問題で実行された Givens 回転の位置が格納される。</p>
<i>perm</i>	<p>INTEGER</p> <p>配列、次元は $(ldgcol, nlv1)$ ($icompq=1$ の場合)。 $icompq=0$ の場合は参照されない。 $icompq=1$ の場合、$perm(1, i)$ には計算ツリーの i 番目問題で実行された置換が格納される。</p>
<i>givnum</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合)</p> <p>配列、次元は $(ldu, 2*nlv1)$ ($icompq=1$ の場合)。 $icompq=0$ の場合は参照されない。$icompq=1$ の場合、$givnum(1, 2i-1)$ と $givnum(1, 2i)$ には計算ツリーの i 番目問題で実行された Givens 回転の C 値と S 値が格納される。</p>
<i>c</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合)</p> <p>配列、 次元は (n) ($icompq=1$ の場合) または 次元は (1) ($icompq=0$ の場合)。 $icompq=1$ で i 番目の部分問題が正方ではない場合、$c(i)$ には i 番目の部分問題の右ヌル空間に関する Givens 回転の C 値が格納される。</p>
<i>s</i>	<p>REAL (slasda の場合) DOUBLE PRECISION (dlasda の場合)</p> <p>配列、 次元は (n) ($icompq=1$ の場合)。 (1) ($icompq=0$ の場合)。 $icompq=1$ で i 番目の部分問題が正方ではない場合、$s(i)$ には i 番目の部分問題の右ヌル空間に関する Givens 回転の S 値が格納される。</p>
<i>info</i>	<p>INTEGER。</p> <p>$=0$ の場合、正常に終了したことを示す。 <0 の場合、$info=-i$ は i 番目の引数の値が不正だったことを示す。 $=1$ の場合、特異値が収束しなかったことを示す。</p>

?lasdq

対角 d と非対角 e を持つ実二重対角行列を計算する。

?bdsdc で使用される。

構文

```
call slasdq( uplo, sqre, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
            work, info )
```

```
call dlasdq( uplo, sqre, n, ncv, nru, ncc, d, e, vt, ldvt, u, ldu, c, ldc,
            work, info )
```

説明

ルーチン ?lasdq は、対角 d と非対角 e を持つ実数 (上または下) 二重対角行列の特異値分解 (svd) を、必要に応じて変換の蓄積を行いながら計算する。 B を入力の実二重対角行列としたとき、このアルゴリズムは $B = QSP'$ であるような直交行列 Q と P を計算する (P' は P の転置)。特異値 S は d に上書きされる。

必要に応じて入力行列 U は UQ に変更される。

必要に応じて入力行列 VT は $P'VT$ に変更される。

必要に応じて入力行列 C は $Q'C$ に変更される。

入力パラメーター

<i>uplo</i>	CHARACTER*1. <i>uplo</i> は入力二重対角行列が上二重対角か下二重対角かを指定する。 <i>uplo</i> = 'U' または 'u' の場合、 B は上二重対角。 <i>uplo</i> = 'L' または 'l' の場合、 B は下二重対角。
<i>sqre</i>	INTEGER。 = 0 の場合、入力行列は $n \times n$ = 1 の場合、 <i>uplu</i> = 'U' ならば入力行列は $n \times (n+1)$ 、 <i>uplu</i> = 'L' ならば $(n+1) \times n$ 。 二重対角行列は行次元 $n = n1 + nr + 1$ と列次元 $m = n + sqre \geq n$ を持つ。
<i>n</i>	INTEGER. <i>n</i> は行列内の行数と列数を指定する。 <i>n</i> は 0 以上でなければならない。
<i>ncvt</i>	INTEGER. <i>ncvt</i> は行列 VT の列数を指定する。 <i>ncvt</i> は 0 以上でなければならない。
<i>nru</i>	INTEGER. <i>nru</i> は行列 U の行数を指定する。 <i>nru</i> は 0 以上でなければならない。
<i>ncc</i>	INTEGER. <i>ncc</i> は行列 C の列数を指定する。 <i>ncc</i> は 0 以上でなければならない。
<i>d</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は (<i>n</i>)。 d には、SVD の実行対象である二重対角行列の対角成分を格納する。

<i>e</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は、 <i>sqre</i> = 0 の場合は (<i>n</i> -1)、 <i>sqre</i> = 1 の場合は (<i>n</i>)。 <i>e</i> の成分には、SVD の実行対象である二重対角行列の非対角成分を格納する。
<i>vt</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は (<i>ldvt</i> , <i>ncvt</i>)。出力時に <i>P'</i> によって事前乗算される行列を格納する。 <i>sqre</i> = 0 ならば次元は <i>n</i> × <i>ncvt</i> 、 <i>sqre</i> = 1 ならば次元は (<i>n</i> +1) × <i>ncvt</i> (<i>ncvt</i> = 0 の場合は参照されない)。
<i>ldvt</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldvt</i> には <i>vt</i> のリーディング・ディメンジョンを格納する。 <i>ldvt</i> の値は 1 以上でなければならない。さらに <i>ncvt</i> が非ゼロならば、 <i>ldvt</i> は <i>n</i> 以上でなければならない。
<i>u</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は (<i>ldu</i> , <i>n</i>)。出力時に <i>Q</i> によって事後乗算される行列を格納する。 <i>sqre</i> = 0 ならば次元は <i>nru</i> × <i>n</i> 、 <i>sqre</i> = 1 ならば次元は <i>nru</i> × (<i>n</i> +1) (<i>nru</i> = 0 の場合は参照されない)。
<i>ldu</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldu</i> には <i>u</i> のリーディング・ディメンジョンを格納する。 <i>ldu</i> は $\max(1, nru)$ 以上でなければならない。
<i>c</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は (<i>ldc</i> , <i>ncc</i>)。出力時に <i>Q'</i> によって事前乗算される <i>n</i> × <i>ncc</i> の行列を格納する。 <i>sqre</i> = 0 ならば次元は <i>n</i> × <i>ncc</i> 、 <i>sqre</i> = 1 ならば次元は (<i>n</i> +1) × <i>ncc</i> (<i>ncc</i> = 0 の場合は参照されない)。
<i>ldc</i>	INTEGER。呼び出し (サブ) プログラムで宣言されたとおり、 <i>ldc</i> には <i>c</i> のリーディング・ディメンジョンを格納する。 <i>ldc</i> は 1 以上でなければならない。さらに <i>ncc</i> が非ゼロならば、 <i>ldc</i> は <i>n</i> 以上でなければならない。
<i>work</i>	REAL (slasdq の場合) DOUBLE PRECISION (dlasdq の場合) 配列、次元は (<i>4n</i>)。ワークスペース配列。 <i>ncvt</i> 、 <i>nru</i> 、または <i>ncc</i> のうち 1 つが非ゼロで、かつ、 <i>n</i> が 2 以上の場合にのみ参照される。

出力パラメーター

<i>d</i>	<i>d</i> には特異値が昇順で上書きされる。
<i>e</i>	通常終了時は <i>e</i> には 0 が格納される。アルゴリズムが収束しない場合、 <i>d</i> と <i>e</i> には、入力として与えられた行列と直交的に等価な二重対角行列の対角と劣対角成分が格納される。
<i>vt</i>	終了時に、行列は <i>P'</i> によって事前に乗算されている。
<i>u</i>	終了時に、行列は <i>Q</i> によって事後に乗算されている。

<i>c</i>	終了時に、行列は <i>Q'</i> によって事前に乗算されている。
<i>info</i>	INTEGER。ゼロは正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>-info</i> 番目の引数の値が不正であったことを示す。 <i>info</i> > 0 の場合、アルゴリズムが収束しなかったことを示し、 <i>info</i> は収束しなかった優対角の個数を表わす。

?lasdt

二重対角分割統治に対する部分問題のツリーを生成する。

?bdsdc で使用される。

構文

```
call slasdt( n, lvl, nd, inode, ndiml, ndimr, msub )
call dlasdt( n, lvl, nd, inode, ndiml, ndimr, msub )
```

説明

このルーチンは二重対角分割統治に対する部分問題のツリーを生成する。

入力パラメーター

<i>n</i>	INTEGER。二重対角行列の対角成分の個数を格納する。
<i>msub</i>	INTEGER。一番下のツリーにある各部分問題が取り得る最大行次元を格納する。

出力パラメーター

<i>lvl</i>	INTEGER。計算ツリーのレベル数が格納される。
<i>nd</i>	INTEGER。ツリーのノード数が格納される。
<i>inode</i>	INTEGER。 配列、次元は (<i>n</i>)。部分問題の中央が格納される。
<i>ndiml</i>	INTEGER。 配列、次元は (<i>n</i>)。左の子の列次元が格納される。
<i>ndimr</i>	INTEGER。 配列、次元は (<i>n</i>)。右の子の列次元が格納される。

?laset

行列の非対角成分と対角成分を**指定された値**に初期化する。

構文

```
call slaset( uplo, m, n, alpha, beta, a, lda )
call dlaset( uplo, m, n, alpha, beta, a, lda )
call claset( uplo, m, n, alpha, beta, a, lda )
call zlaset( uplo, m, n, alpha, beta, a, lda )
```

説明

このルーチンは、 $m \times n$ 行列 A で、対角を β に、非対角を α に初期化する。

入力パラメーター

<code>uplo</code>	CHARACTER*1。設定対象の行列の部分を指定する。 <code>uplo = 'U'</code> の場合、上三角部分が設定される。 A の厳密な下三角部分は変更されない。 <code>uplo = 'L'</code> の場合、下三角部分が設定される。 A の厳密な上三角部分は変更されない。 それ以外では行列 A のすべての部分が設定される。
<code>m</code>	INTEGER。行列 A の行数。 $m \geq 0$ 。
<code>n</code>	INTEGER。行列 A の列数。 $n \geq 0$ 。
<code>alpha, beta</code>	REAL (<code>slaset</code> の場合) DOUBLE PRECISION (<code>dlaset</code> の場合) COMPLEX (<code>claset</code> の場合) COMPLEX*16 (<code>zlaset</code> の場合) それぞれ、非対角成分と対角成分に設定する定数。
<code>a</code>	REAL (<code>slaset</code> の場合) DOUBLE PRECISION (<code>dlaset</code> の場合) COMPLEX (<code>claset</code> の場合) COMPLEX*16 (<code>zlaset</code> の場合) 配列、次元は (lda, n) 。 $m \times n$ の行列 A を格納する。
<code>lda</code>	INTEGER。配列 A のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

出力パラメーター

<code>a</code>	A の先頭の $m \times n$ 部分行列は次のように設定される。 <code>uplo = 'U'</code> の場合、 $A(i, j) = \alpha$ 、 $1 \leq i \leq j-1$ 、 $1 \leq j \leq n$ 、 <code>uplo = 'L'</code> の場合、 $A(i, j) = \alpha$ 、 $j+1 \leq i \leq m$ 、 $1 \leq j \leq n$ 、
----------------	--

それ以外の場合、 $A(i,j) = \alpha$ 、 $1 \leq i \leq m$ 、 $1 \leq j \leq n$ 、 $i \neq j$ 、

また、すべての $uplo$ で $A(i,i) = \beta$ 、 $1 \leq i \leq \min(m, n)$ 。

?lasq1

実数平方二重対角行列の特異値を計算する。
?bdsqr で使用される。

構文

```
call slasq1( n, d, e, work, info )
call dlasq1( n, d, e, work, info )
```

説明

ルーチン ?lasq1 は、対角 d と非対角 e を持つ実数 $n \times n$ 二重対角行列の特異値を計算する。特異値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバーフローは発生しない。

入力パラメーター

n	INTEGER。行列内の行と列の数。 $n \geq 0$ 。
d	REAL (slasq1 の場合) DOUBLE PRECISION (dlasq1 の場合) 配列、次元は (n) 。 d には、SVD の実行対象である二重対角行列の対角成分を格納する。
e	REAL (slasq1 の場合) DOUBLE PRECISION (dlasq1 の場合) 配列、次元は (n) 。成分 $e(1:n-1)$ には、SVD の実行対象である二重対角行列の非対角成分を格納する。
$work$	REAL (slasq1 の場合) DOUBLE PRECISION (dlasq1 の場合) ワークスペース配列、次元は $(4n)$

出力パラメーター

d	d には特異値が降順で上書きされる。
e	e は上書きされる。
$info$	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は i 番目の引数の値が不正だったことを示す。 > 0 の場合、アルゴリズムが失敗したことを示す。 = 1 の場合、分割は e に格納されている正の値でマークされた。 = 2 の場合、 z の現在のブロックは $30 * n$ 回の反復 (内側の

while ループ) によって対角化されなかった。
 = 3 の場合、外側の while ループの終了条件を満たさなかった (プログラムは n を超える縮退されないブロックを生成した)。

?lasq2

相対的に高い精度で、 qd 配列 z に関する対称正定値三重対角行列のすべての固有値を計算する。

?bdsqr と ?stegr で使用される。

構文

```
call slasq2( n, z, info )
call dlasq2( n, z, info )
```

説明

ルーチン ?lasq2 は、 qd 配列 z に関する対称正値三重対角行列のすべての固有値を計算する。固有値は相対的に高い精度で計算され、非正規化、アンダーフロー、オーバーフローは発生しない。

三重対角行列に対する z の関係を理解するには、 L を劣対角 $z(2, 4, 6, \dots)$ を持つ単位下二重対角行列とし、 U を対角 $z(1, 3, 5, \dots)$ とその上に 1 を持つ上二重対角行列とする。三重対角は LU となる。または、類似した対称三重対角としてもよい。

入力パラメーター

n INTEGER。行列内の行と列の数。 $n \geq 0$ 。
 z REAL (slasq2 の場合)
 DOUBLE PRECISION (dlasq2 の場合)
 配列、次元は $(4n)$ 。 z には qd 配列を格納する。

出力パラメーター

z 成分 1 から n には固有値が降順で、 $z(2n+1)$ には対角和が、 $z(2n+2)$ には固有値の合計が格納される。 $n > 2$ の場合、 $z(2n+3)$ には反復回数が、 $z(2n+4)$ には $n\text{div}s/n\text{in}^2$ が、 $z(2n+5)$ には失敗したシフトのパーセントが格納される。

$info$ INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、 i 番目の引数がスカラーで値が不正だった場合は $info = -i$ 、 i 番目の引数が配列で j 番目の成分の値が不正だった場合は $info = -(i * 100 + j)$;
 > 0 の場合、アルゴリズムが失敗したことを示す。
 = 1 の場合、分割は e に格納されている正の値でマークされた。
 = 2 の場合、 z の現在のブロックは $30 * n$ 回の反復 (内側の while

ループ)によって対角化されなかった。
 =3 の場合、外側の while ループの終了条件を満たさなかった
 (プログラムは n を超える縮退されないブロックを生成した)。

アプリケーション・ノート

ルーチン `?lasq2` は論理変数 `ieee` を定義する。マシンが IEEE-754 浮動小数点スタンダードに従って無限および NaN を取り扱う場合は `.TRUE.` となり、そうでない場合は `.FALSE.` となる。変数は [?lasq3](#) に渡される。

?lasq3

収縮をチェックし、シフトを計算し、`dqds` を呼び出す。`?bdsqr` で使用される。

構文

```
call slasq3( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter, ndiv, ieee )
call dlasq3( i0, n0, z, pp, dmin, sigma, desig, qmax, nfail, iter, ndiv, ieee )
```

説明

ルーチン `?lasq3` は、収縮をチェックし、シフトを計算し、`dqds` を呼び出す。失敗した場合はシフトを変更して出力が正になるまで繰り返す。

入力パラメーター

<code>i0</code>	INTEGER。最初のインデックス。
<code>n0</code>	INTEGER。最後のインデックス。
<code>z</code>	REAL (<code>slasq3</code> の場合) DOUBLE PRECISION (<code>dlasq3</code> の場合) 配列、次元は $(4n)$ 。 z には <code>qd</code> 配列を格納する。
<code>pp</code>	INTEGER。 <code>ping</code> のとき <code>pp = 0</code> 、 <code>pong</code> のとき <code>pp = 1</code>
<code>desig</code>	REAL (<code>slasq3</code> の場合) DOUBLE PRECISION (<code>dlasq3</code> の場合) <code>sigma</code> の下位次数部分。
<code>qmax</code>	REAL (<code>slasq3</code> の場合) DOUBLE PRECISION (<code>dlasq3</code> の場合) <code>q</code> の最大値。
<code>ieee</code>	LOGICAL。IEEE または非 IEEE 演算を示すフラグ (?lasq5 に渡される)。

出力パラメーター

<code>dmin</code>	REAL (<code>slasq3</code> の場合) DOUBLE PRECISION (<code>dlasq3</code> の場合) <code>d</code> の最小値。
-------------------	--

<i>sigma</i>	REAL (slasq3 の場合) DOUBLE PRECISION (dlasq3 の場合) 現在のセグメントで使用されるシフトの合計。
<i>desig</i>	<i>sigma</i> の下位次数部分。
<i>nfail</i>	INTEGER。シフト回数が大きすぎたことを示す。
<i>iter</i>	INTEGER。反復の回数。
<i>ndiv</i>	INTEGER。除算の回数。

?lasq4

以前の変換で得られた d の値を用いて最小固有値に対する近似を計算する。
?bdsqr で使用される。

構文

```
call slasq4( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2, tau,
           ttype )
call dlasq4( i0, n0, z, pp, n0in, dmin, dmin1, dmin2, dn, dn1, dn2, tau,
           ttype )
```

説明

このルーチンは、以前の変換で得られた d の値を用いて、最小固有値に対する近似 τ を計算する。

入力パラメーター

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) 配列、次元は $(4n)$ 。 z には qd 配列を格納する。
<i>pp</i>	INTEGER。ping のとき $pp=0$ 、pong のとき $pp=1$
<i>noin</i>	INTEGER。eigtest の開始にある $n0$ の値。
<i>dmin</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) d の最小値。
<i>dmin1</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) $d(n0)$ を除く d の最小値。
<i>dmin2</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) $d(n0)$ と $d(n0-1)$ を除く d の最小値。

<i>dn</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) <i>d</i> (<i>n</i>) が格納される。
<i>dn1</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) <i>d</i> (<i>n</i> -1) が格納される。
<i>dn2</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) <i>d</i> (<i>n</i> -2) が格納される。

出力パラメーター

<i>tau</i>	REAL (slasq4 の場合) DOUBLE PRECISION (dlasq4 の場合) シフト。
<i>ttype</i>	INTEGER。シフトのタイプ。

?lasq5

ping-pong 形式で *dqds* 変換を 1 つ計算する。
?bdsqr と ?stegr で使用される。

構文

```
call slasq5(i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2, ieee)
call dlasq5(i0, n0, z, pp, tau, dmin, dmin1, dmin2, dn, dnm1, dnm2, ieee)
```

説明

このルーチンは *ping-pong* 形式で *dqds* 変換を 1 つ計算する。IEEE マシンと非 IEEE マシンに対応する。

入力パラメーター

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) 配列、次元は (4 <i>n</i>)。 <i>z</i> には <i>qd</i> 配列を格納する。余分な引数を防ぐため <i>z</i> (4* <i>n0</i>) に <i>emin</i> を格納する。
<i>pp</i>	INTEGER。 <i>ping</i> のとき <i>pp</i> = 0、 <i>pong</i> のとき <i>pp</i> = 1
<i>tau</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) シフトである。
<i>ieeee</i>	LOGICAL。 IEEE または非 IEEE 演算を示すフラグ。

出力パラメーター

<i>dmin</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> の最小値。
<i>dmin1</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> (<i>n0</i>) を除く <i>d</i> の最小値。
<i>dmin2</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> (<i>n0</i>) と <i>d</i> (<i>n0</i> -1) を除く <i>d</i> の最小値。
<i>dn</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> の最後の値である <i>d</i> (<i>n0</i>) が格納される。
<i>drn1</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> (<i>n0</i> -1) が格納される。
<i>drn2</i>	REAL (slasq5 の場合) DOUBLE PRECISION (dlasq5 の場合) <i>d</i> (<i>n0</i> -2) が格納される。

?lasq6

ping-pong 形式で *dqd* 変換を 1 つ計算する。
?bdsqr と ?stegr で使用される。

構文

```
call slasq6( i0, n0, z, pp, dmin, dmin1, dmin2, dn, drn1, drn2 )
call dlasq6( i0, n0, z, pp, dmin, dmin1, dmin2, dn, drn1, drn2 )
```

説明

ルーチン ?lasq6 は、アンダーフローとオーバーフローに対する保護を行いながら、*ping-pong* 形式で *dqd* (ゼロに等しいシフト) 変換を 1 つ計算する。

入力パラメーター

<i>i0</i>	INTEGER。最初のインデックス。
<i>n0</i>	INTEGER。最後のインデックス。
<i>z</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) 配列、次元は (4 <i>n</i>)。 <i>z</i> には <i>qd</i> 配列を格納する。余分な引数を防ぐため <i>z</i> (4* <i>n0</i>) に <i>emin</i> を格納する。
<i>pp</i>	INTEGER。ping のとき <i>pp</i> = 0、pong のとき <i>pp</i> = 1

出力パラメーター

<i>dmin</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> の最小値。
<i>dmin1</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> (<i>n0</i>) を除く <i>d</i> の最小値。
<i>dmin2</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> (<i>n0</i>) と <i>d</i> (<i>n0</i> -1) を除く <i>d</i> の最小値。
<i>dn</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> の最後の値である <i>d</i> (<i>n0</i>) が格納される。
<i>dnm1</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> (<i>n0</i> -1) が格納される。
<i>dnm2</i>	REAL (slasq6 の場合) DOUBLE PRECISION (dlasq6 の場合) <i>d</i> (<i>n0</i> -2) が格納される。

?lasr

一般矩形行列に一連の面回転を適用する。

構文

```
call slasr( side, pivot, direct, m, n, c, s, a, lda )
call dlasr( side, pivot, direct, m, n, c, s, a, lda )
call clasr( side, pivot, direct, m, n, c, s, a, lda )
call zlasr( side, pivot, direct, m, n, c, s, a, lda )
```

説明

このルーチンは次の変換を実行する。

side = 'L' または 'l' のとき $A := PA$ (左辺側)

side = 'R' または 'r' のとき $A := AP$ (右辺側)

A は $m \times n$ の実数行列、*P* は直交行列で、パラメーター *pivot* と *direct* によって次のように定義される一連の面回転で構成される (*side* = 'L' または 'l' のとき $z = m$ 、*side* = 'R' または 'r' のとき $z = n$)。

direct = 'F' または 'f' のとき (順方向順)、

$$P = P(z-1) \dots P(2) P(1),$$

direct = 'B' または 'b' のとき (逆方向順)、

$$P = P(1) P(2) \dots P(z-1),$$

P(*k*) は次の面に対する面回転行列である。

$pivot = 'v'$ または $'v'$ (可変ピボット) のとき、面 $(k, k+1)$

$pivot = 't'$ または $'t'$ (上ピボット) のとき、面 $(1, k+1)$

$pivot = 'b'$ または $'b'$ (下ピボット) のとき、面 (k, z)

$c(k)$ と $s(k)$ には $P(k)$ を定義する余弦と正弦が格納されていなければならない。行列 $P(k)$ の 2×2 面回転部分 $R(k)$ は、次の形式とみなす。

$$R(k) = \begin{bmatrix} c(k) & s(k) \\ -s(k) & c(k) \end{bmatrix}.$$

入力パラメーター

<i>side</i>	CHARACTER*1。面回転行列 P を A の左または右に適用するかを指定する。 = 'L' の場合、左、 $A := PA$ を計算する。 = 'R' の場合、右、 $A := AP$ を計算する。
<i>direct</i>	CHARACTER*1。 P が面回転の順方向順または逆方向順かを指定する。 = 'F' の場合、順方向、 $P = P(z-1) \dots P(2) P(1)$ = 'B' の場合、逆方向、 $P = P(1) P(2) \dots P(z-1)$
<i>pivot</i>	CHARACTER*1。面回転行列 $P(k)$ の面を指定する。 = 'v' の場合、可変ピボット、面 $(k, k+1)$ = 't' の場合、上ピボット、面 $(1, k+1)$ = 'b' の場合、下ピボット、面 (k, z)
<i>m</i>	INTEGER。行列 A の行数。 $m \leq 1$ の場合、イミディエイト・リターンが有効。
<i>n</i>	INTEGER。行列 A の列数。 $n \leq 1$ の場合、イミディエイト・リターンが有効。
<i>c, s</i>	REAL (slasr/clasr の場合) DOUBLE PRECISION (dlasr/zlasr の場合) 配列、次元は、 $side = 'L'$ の場合 $(m-1)$ $side = 'R'$ の場合 $(n-1)$ 。 $c(k)$ と $s(k)$ には上述のように行列 $P(k)$ を定義する余弦と正弦を格納する。
<i>a</i>	REAL (slasr の場合) DOUBLE PRECISION (dlasr の場合) COMPLEX (clasr の場合) COMPLEX*16 (zlasr の場合) 配列、次元は (lda, n) 。 $m \times n$ の行列 A
<i>lda</i>	INTEGER。配列 A のリーディング・ディメンジョン。 $lda \geq \max(1, m)$

出力パラメーター

<i>a</i>	$side = 'R'$ の場合は pa 、 $side = 'L'$ の場合は AP で上書きされる。
----------	--

?lasrt

数を昇順または降順でソートする。

構文

```
call slasrt( id, n, d, info )
call dlasrt( id, n, d, info )
```

説明

?lasrt は、*d* に格納されている数を昇順 (*id* = 'I' の場合) または降順 (*id* = 'D' の場合) でソートする。このルーチンはクイックソートを使用するが、*size* ≤ 20 の配列では挿入ソートが使われる。スタックの次元によって *n* はおよそ 2^{32} に制限される。

入力パラメーター

<i>id</i>	CHARACTER*1。 = 'I' の場合、 <i>d</i> を昇順でソートする。 = 'D' の場合、 <i>d</i> を降順でソートする。
<i>n</i>	INTEGER。配列 <i>d</i> の長さ。
<i>d</i>	REAL (slasrt の場合) DOUBLE PRECISION (dlasrt の場合) ソートする配列を格納する。

出力パラメーター

<i>d</i>	終了時に、 <i>id</i> の設定によって、 <i>d</i> は昇順 ($d(1) \leq \dots \leq d(n)$) または降順 ($d(1) \geq \dots \geq d(n)$) となる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?lassq

スケーリング形式で表現された二乗和を更新する。

構文

```
call slassq( n, x, incx, scale, sumsq )
call dlassq( n, x, incx, scale, sumsq )
call classq( n, x, incx, scale, sumsq )
call zlassq( n, x, incx, scale, sumsq )
```

説明

実数ルーチン `slassq/dlassq` は、次のように値 `scl` と `smsq` を返す。

$$scl^2 * smsq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで $x(i) = x(1 + (i - 1) incx)$ 。

`sumsq` の値は非負であるとし、`scl` は次の値を返す。

$$scl = \max(scale, abs(x(i)))$$

`scale` と `sumsq` は `scale` と `sumsq` によって与えられなければならない。また `scl` と `smsq` は `scale` と `sumsq` にそれぞれ上書きされる。

複素ルーチン `classq/zlassq` は、次のように値 `scl` と `ssq` を返す。

$$scl^2 * ssq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで $x(i) = \text{abs}(x(1 + (i - 1) incx))$ 。

`sumsq` の値は少なくともユニティーであるとみなされ、`ssq` の値は、次のように満たされる。

$$1.0 \leq ssq \leq sumsq + 2n$$

`scale` の値は非負であるとし、`scl` は次の値を返す。

$$scl = \max_i(scale, \text{abs}(\text{real}(x(i))), \text{abs}(\text{aimag}(x(i))))$$

`scale` と `sumsq` は `scale` と `sumsq` によって与えられなければならない。また `scl` と `ssq` は `scale` と `sumsq` にそれぞれ上書きされる。

?lassq の全ルーチンは唯一、ベクトル `x` を出力に渡す。

入力パラメーター

<code>n</code>	INTEGER。ベクトル <code>x</code> で使用する成分の個数。
<code>x</code>	REAL (<code>slassq</code> の場合) DOUBLE PRECISION (<code>dlassq</code> の場合) COMPLEX (<code>classq</code> の場合) COMPLEX*16 (<code>zlassq</code> の場合) スケーリングされた二乗和を計算するベクトル。 $x(i) = x(1 + (i - 1) incx)$, $1 \leq i \leq n$
<code>incx</code>	INTEGER。ベクトル <code>x</code> の連続する値の増分。 $incx > 0$
<code>scale</code>	REAL (<code>slassq/classq</code> の場合) DOUBLE PRECISION (<code>dlassq/zlassq</code> の場合) 上の式で示した <code>scale</code> の値を格納する。
<code>sumsq</code>	REAL (<code>slassq/classq</code> の場合) DOUBLE PRECISION (<code>dlassq/zlassq</code> の場合) 上の式で示した <code>sumsq</code> の値を格納する。

出力パラメーター

`scale` は、二乗和に対するスケール係数 `scl` で上書きされる。

`sumsq` 実数型の場合。
`sumsq` は上の式で示した `sumsq` の値で上書きされる。
 複素数型の場合:
`sumsq` は上の式で示した `ssq` の値で上書きされる。

?lasv2

2 × 2 三角行列の特異値分解を計算する。

構文

```
call slasv2( f, g, h, ssmín, ssmáx, snr, csr, snl, csl )
call dlasv2( f, g, h, ssmín, ssmáx, snr, csr, snl, csl )
```

説明

ルーチン ?lasv2 は 2 × 2 三角行列の特異値分解を計算する。

$$\begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

出力において、`abs(ssmax)` には大きいほうの特異値が、`abs(ssmin)` には小さいほうの特異値が、`(csl, snl)` と `(csr, snr)` には `abs(ssmax)` に対する左と右の特異ベクトルが格納され、分解を与える。

$$\begin{bmatrix} csl & snl \\ -snl & csl \end{bmatrix} \begin{bmatrix} f & g \\ 0 & h \end{bmatrix} \begin{bmatrix} csr & -snr \\ snr & csr \end{bmatrix} = \begin{bmatrix} ssmáx & 0 \\ 0 & ssmín \end{bmatrix}$$

入力パラメーター

`f, g, h` REAL (slasv2 の場合)
 DOUBLE PRECISION (dlasv2 の場合)
 それぞれ、2 × 2 行列の (1,1)、(1,2)、(2,2) の成分。

出力パラメーター

`ssmin, ssmáx` REAL (slasv2 の場合)
 DOUBLE PRECISION (dlasv2 の場合)
 それぞれ、`abs(ssmin)` と `abs(ssmax)` は小さいほうの固有値および大きいほうの特異値。

`snl, csl` REAL (slasv2 の場合)
 DOUBLE PRECISION (dlasv2 の場合)
 ベクトル `(csl, snl)` は特異値 `abs(ssmax)` に対する単位左特異ベクトル。

snr, csr REAL (slasv2 の場合)
 DOUBLE PRECISION (dlasv2 の場合)
 ベクトル (*csr, sn*) は特異値 $\text{abs}(ssmax)$ に対する単位右特異ベクトル。

アプリケーション・ノート

入力パラメーターは、出力パラメーターにエイリアスされることがある。オーバーフロー/アンダーフローの場合を除いて、減算にガード桁がなくとも、すべての出力の大きさは最後の位置からわずかな単位以内 (ulp) で正確である。

IEEE 演算で、1つの行列成分が無限でもコードは正しく動作する。最大特異値自身がオーバーフローしない限り、あるいは最大特異値がオーバーフローからわずかな ulp 範囲内にある限り、オーバーフローは起こらない。(Cray のように部分オーバーフローを持つマシンでは、最大特異値がオーバーフローに対して2の因子以内にあるとオーバーフローが起こることがある。)

アンダーフローは緩やかであればアンダーフローは無害である。そうでない場合、結果は、アンダーフローしきい値に近いサイズの摂動によって変更された行列に一致することがある。

?laswp

一般矩形行列に対して一連の行交換を実行する。

構文

```
call slaswp( n, a, lda, k1, k2, ipiv, incx )
call dlaswp( n, a, lda, k1, k2, ipiv, incx )
call claswp( n, a, lda, k1, k2, ipiv, incx )
call zlaswp( n, a, lda, k1, k2, ipiv, incx )
```

説明

このルーチンは行列 *A* に一連の行交換を実行する。行交換は *a* の行 *k1* から *K2* のそれぞれに対して開始される、

入力パラメーター

n INTEGER。行列 *A* の列数。

a REAL (slaswp の場合)
 DOUBLE PRECISION (dlaswp の場合)
 COMPLEX (claswp の場合)
 COMPLEX*16 (zlaswp の場合)
 配列、次元は (*lda, n*)。
 行交換を適用する列次元 *n* の行列を格納する。

lda INTEGER。配列 *a* のリーディング・ディメンジョン。

k1 INTEGER。 *ipiv* の行交換を適用する最初の成分。

k2 INTEGER。 *ipiv* の行交換を適用する最後の成分。

<i>ipiv</i>	INTEGER。 配列、次元は $(m * \text{abs}(\text{incx}))$ ピボットのインデックスのベクトル。 <i>ipiv</i> の位置 <i>k1</i> から <i>k2</i> にある成分のみがアクセスされる。 <i>ipiv(k) = l</i> には行 <i>k</i> と <i>l</i> が交換される意味がある。
<i>incx</i>	INTEGER。 <i>ipiv</i> の連続する値の増分。 <i>ipiv</i> が負の場合、ピボットは逆順に適用される。

出力パラメーター

<i>a</i>	置換された行列で上書きされる。
----------	-----------------

?lasy2

行列の次数が 1 または 2 のシルベスター行列式を解く。

構文

```
call slasy2( ltranl, ltranr, isgn, n1, n2, t1, ldt1, tr, ldtr, b, ldb, scale, x,
            ldx, xnorm, info )
call dlasy2( ltranl, ltranr, isgn, n1, n2, t1, ldt1, tr, ldtr, b, ldb, scale, x,
            ldx, xnorm, info )
```

説明

このルーチンは、次式で $n1 \times n2$ 行列 X を解く。 $1 \leq n1, n2 \leq 2$ 、

$$\text{op}(TL) * X + \text{isgn} * X * \text{op}(TR) = \text{scale} * B$$

ここで、

TL は $n1 \times n1$ 、

TR は $n2 \times n2$ 、

B は $n1 \times n2$ 、

または、 $\text{isgn} = 1$ または -1 。 $\text{op}(T) = T$ または T' 、ここで T は T の転置を表す。

入力パラメーター

<i>ltranl</i>	LOGICAL。 <i>ltranl</i> は $\text{op}(TL)$ を指定する。 = .FALSE. の場合、 $\text{op}(TL) = TL$ 、 = .TRUE. の場合、 $\text{op}(TL) = TL'$ 。
<i>ltranr</i>	LOGICAL。 <i>ltranr</i> は $\text{op}(TR)$ を指定する。 = .FALSE. の場合、 $\text{op}(TR) = TR$ 、 = .TRUE. の場合、 $\text{op}(TR) = TR'$ 。
<i>isgn</i>	INTEGER。 <i>isgn</i> は前述のとおり式の符号を指定する。 <i>isgn</i> は 1 または -1 を取り得る。

<i>n1</i>	INTEGER。 <i>n1</i> は行列 <i>TL</i> の次数を指定する。 <i>n1</i> は、0、1、2 を取り得る。
<i>n2</i>	INTEGER。 <i>n2</i> は行列 <i>TR</i> の次数を指定する。 <i>n2</i> は、0、1、2 を取り得る。
<i>t1</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) 配列、次元は (<i>ldt1</i> , 2)。 <i>t1</i> には $n1 \times n1$ の行列 <i>TL</i> を格納する。
<i>ldt1</i>	INTEGER。 行列 <i>t1</i> のリーディング・ディメンション。 $ldt1 \geq \max(1, n1)$
<i>tr</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) 配列、次元は (<i>ldtr</i> , 2)。 <i>tr</i> には $n2 \times n2$ の行列 <i>TR</i> を格納する。
<i>ldtr</i>	INTEGER。 行列 <i>tr</i> のリーディング・ディメンション。 $ldtr \geq \max(1, n2)$
<i>b</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) 配列、次元は (<i>ldb</i> , 2)。 $n1 \times n2$ の行列 <i>b</i> には式の右辺を格納する。
<i>ldb</i>	INTEGER。 行列 <i>b</i> のリーディング・ディメンション。 $ldb \geq \max(1, n1)$
<i>ldx</i>	INTEGER。 出力行列 <i>x</i> のリーディング・ディメンション。 $ldx \geq \max(1, n1)$
出力パラメーター	
<i>scale</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) 終了時に、 <i>scale</i> にはスケール係数が格納される。 <i>scale</i> は 1 以下に選択され解のオーバーフローを防ぐ。
<i>x</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) 配列、次元は (<i>ldx</i> , 2)。 <i>x</i> には $n1 \times n2$ の解が格納される。
<i>xnorm</i>	REAL (<i>slasy2</i> の場合) DOUBLE PRECISION (<i>dlasy2</i> の場合) <i>xnorm</i> には解の無限ノルムが格納される。
<i>info</i>	INTEGER。 <i>info</i> には次の値が設定される。 0 の場合、正常に終了したことを示す。 1 の場合、 <i>TL</i> と <i>TR</i> は近すぎる固有値を持つため、 <i>TL</i> と <i>TR</i> は非特異方程式を得るために摂動された。



注: 高速化するために、このルーチンでは入力のエラーをチェックしない。

?lasyf

対角ピボット演算法を使って実数 / 複素対称行列の部分因子分解を計算する。

構文

```
call slasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call dlasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call clasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call zlasyf( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
```

説明

ルーチン ?lasyf は、**Bunch-Kaufman** 対角ピボット演算法を使って、実数 / 複素対称行列 A の部分因子分解を計算する。部分因子分解は次の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix} \quad \text{uplo = 'U' の場合、または}$$

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix} \quad \text{uplo = 'L' の場合、}$$

ここで D の次数は大きくとも nb である。実際の次数は引数 kb で返され、 nb か $nb-1$ 、あるいは $n \leq nb$ の場合は n である。

このルーチンは、[?sytrf](#) から呼び出される補助ルーチンである。

部分行列 A_{11} ($uplo = 'U'$ の場合) または A_{22} ($uplo = 'L'$ の場合) を更新するためにブロック化コード (レベル 3 BLAS の呼び出し) が使用されている。

入力パラメーター

<code>uplo</code>	CHARACTER*1。 対称行列 A の上三角部分または下三角部分のどちらが格納されるかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>n</code>	INTEGER。 行列 A の次数。 $n \geq 0$ 。
<code>nb</code>	INTEGER。 因子分解すべき行列 A の最大列。 2×2 ピボットブロックを許すために、 <code>nb</code> は小さくとも 2 でなければならない。
<code>a</code>	REAL (<code>slasyf</code> の場合) DOUBLE PRECISION (<code>dlasyf</code> の場合) COMPLEX (<code>clasyf</code> の場合) COMPLEX*16 (<code>zlasyf</code> の場合) 配列、次元は (lda, n) 。対称行列 A を格納する。 <code>uplo</code> = 'U' の場合、 <code>a</code> の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を格納する。 <code>a</code> の厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、 <code>a</code> の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 <code>a</code> の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 <code>a</code> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<code>w</code>	REAL (<code>slasyf</code> の場合) DOUBLE PRECISION (<code>dlasyf</code> の場合) COMPLEX (<code>clasyf</code> の場合) COMPLEX*16 (<code>zlasyf</code> の場合) ワークスペース配列、次元は (ldw, nb)
<code>ldw</code>	INTEGER。 配列 <code>a</code> のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

出力パラメーター

<code>kb</code>	INTEGER。 実際に因子分解された A の列数。 <code>kb</code> は <code>nb-1</code> か <code>nb</code> 、あるいは $n \leq nb$ の場合は n である。
<code>a</code>	<code>a</code> は部分因子分解の詳細で上書きされる。
<code>ipiv</code>	INTEGER。 配列、次元は (n) 。交換の結果と D のブロック構造の各成分が格納される。 <code>uplo</code> = 'U' の場合、 <code>ipiv</code> の最後の <code>kb</code> 個の成分のみが設定される。 <code>uplo</code> = 'L' の場合、最初の <code>kb</code> 個の成分のみが設定される。 <code>ipiv(k) > 0</code> の場合、 k 番目の行と列と <code>ipiv(k)</code> 番目の行と列は交換された。 $D(k, k)$ は 1×1 の対角ブロックである。 <code>uplo</code> = 'U' かつ <code>ipiv(k) = ipiv(k-1) < 0</code> の場合、 $k-1$ 番目の行と列と <code>-ipiv(k)</code> 番目の行と列は交換された。 $D(k-1:k, k-1:k)$ は 2×2 の対角ブロックで

ある。

$uplo = 'L'$ かつ $ipiv(k) = ipiv(k+1) < 0$ の場合、 $k+1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k:k+1, k:k+1)$ は 2×2 の対角ブロックである。

info

INTEGER。

= 0 の場合、正常に終了したことを示す。

> 0 の場合、 $info = k$ ならば $D(k, k)$ は完全にゼロである。因子分解は完了したが、ブロック対角行列 D は完全に特異である。

?lahef

対角ピボット演算法を使って複素エルミート無限行列の部分因子分解を計算する。

構文

```
call clahef( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
call zlahef( uplo, n, nb, kb, a, lda, ipiv, w, ldw, info )
```

説明

ルーチン `?lahef` は、Bunch-Kaufman 対角ピボット演算法を使って、複素エルミート行列 A の部分因子分解を計算する。部分因子分解は次の形式を持つ。

$$A = \begin{bmatrix} I & U_{12} \\ 0 & U_{22} \end{bmatrix} \begin{bmatrix} A_{11} & 0 \\ 0 & D \end{bmatrix} \begin{bmatrix} I & 0 \\ U_{12}' & U_{22}' \end{bmatrix} \quad uplo = 'U' \text{ の場合、または}$$

$$A = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} L_{11}' & L_{21}' \\ 0 & I \end{bmatrix} \quad uplo = 'L' \text{ の場合、}$$

ここで D の次数は大きくとも nb である。実際の次数は引数 kb で返され、 nb か $nb-1$ 、あるいは $n \leq nb$ の場合は n である。
なお、 U' は U の共役転置を表わす。

このルーチンは、[?hetrf](#) から呼び出される補助ルーチンである。部分行列 A_{11} ($uplo = 'U'$ の場合) または A_{22} ($uplo = 'L'$ の場合) を更新するためにブロック化コード (レベル 3 BLAS の呼び出し) が使用されている。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 エルミート行列 A の上三角部分と下三角部分のどちらが格納されるかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 行列 A の次数。 $n \geq 0$ 。
<i>nb</i>	INTEGER。 因子分解すべき行列 A の最大列。 2×2 ピボットブロックを許すために、 <i>nb</i> は小さくとも 2 でなければならない。
<i>a</i>	COMPLEX (clahef の場合) COMPLEX*16 (zlahef の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 エルミート行列 A を格納する。 <i>uplo</i> = 'U' の場合、 <i>a</i> の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を含む。 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>a</i> の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<i>w</i>	COMPLEX (clahef の場合) COMPLEX*16 (zlahef の場合) ワークスペース配列、次元は (<i>ldw</i> , <i>nb</i>)
<i>ldw</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

出力パラメーター

<i>kb</i>	INTEGER。 実際に因子分解された A の列数。 <i>kb</i> は <i>nb</i> -1 か <i>nb</i> 、あるいは $n \leq nb$ の場合は n である。
<i>a</i>	<i>a</i> は部分因子分解の詳細で上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は (<i>n</i>)。交換の結果と D のブロック構造の各成分が格納される。 <i>uplo</i> = 'U' の場合、 <i>ipiv</i> の最後の <i>kb</i> 個の成分のみが設定される。 <i>uplo</i> = 'L' の場合、最初の <i>kb</i> 個の成分のみが設定される。 <i>ipiv</i> (<i>k</i>) > 0 の場合、 <i>k</i> 番目の行と列と <i>ipiv</i> (<i>k</i>) 番目の行と列は交換された。 $D(k, k)$ は 1×1 の対角ブロックである。 <i>uplo</i> = 'U' かつ <i>ipiv</i> (<i>k</i>) = <i>ipiv</i> (<i>k</i> -1) < 0 の場合、 <i>k</i> -1 番目の行と列と - <i>ipiv</i> (<i>k</i>) 番目の行と列は交換された。 $D(k-1:k, k-1:k)$ は 2×2 の対角ブロックである。

$uplo = 'L'$ かつ $ipiv(k) = ipiv(k+1) < 0$ の場合、 $k+1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k:k+1, k:k+1)$ は 2×2 の対角ブロックである。

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 > 0 の場合、 $info = k$ ならば $D(k, k)$ は完全にゼロである。因子分解は完了したが、ブロック対角行列 D は完全に特異である。

?latbs

三角帯連立方程式を解く。

構文

```
call slatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call dlatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call clatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
call zlatbs( uplo, trans, diag, normin, n, kd, ab, ldab, x, scale, cnorm, info )
```

説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケーリングして解く。

$AX = sb$ 、または $A^T x = SB$ 、または $A^H x = sb$ (複素型の場合)

A は上または下三角帯行列である。 A^T は A の転置を表わし、 A^H は A の共役転置を表わし、 x と b は n 成分のベクトル、 s はスケール係数で、 x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケーリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン [?tbsv](#) が呼び出される。行列 A が特異 (任意の j に対して $A(j, j) = 0$) の場合、 s は 0 に設定され $Ax = 0$ に対するゼロでない解が返される

入力パラメーター

uplo CHARACTER*1。
 行列 A が上三角か下三角かを指定する。
 = 'U' の場合、上三角
 = 'L' の場合、下三角。

trans CHARACTER*1。
 A に適用する演算を指定する。
 = 'N' の場合、 $Ax = sb$ を解く (転置なし)
 = 'T' の場合、 $A^T x = sb$ を解く (転置)
 = 'C' の場合、 $A^H x = sb$ を解く (共役転置)

diag CHARACTER*1。
 行列 A が単位三角かどうかを指定する。
 = 'N' の場合、単位三角ではない。
 = 'U' の場合、単位三角。

<i>normin</i>	CHARACTER*1。 <i>cnorm</i> を設定したかどうかを指定する。 = 'Y' の場合、 <i>cnorm</i> には列ノルムを格納した。 = 'N' の場合、 <i>cnorm</i> は設定していない。終了時に、ノルムが計算され、 <i>cnorm</i> に格納される。
<i>n</i>	INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$ 。
<i>kd</i>	INTEGER。 三角行列 <i>A</i> の劣対角または優対角の個数。 $kd \geq 0$
<i>ab</i>	REAL (<i>slatbs</i> の場合) DOUBLE PRECISION (<i>dlatbs</i> の場合) COMPLEX (<i>clatbs</i> の場合) COMPLEX*16 (<i>zlatbs</i> の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。上三角または下三角帯行列 <i>A</i> で、配列の最初の <i>kd</i> +1 行に格納する。 <i>A</i> の <i>j</i> 番目の列を配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 <i>uplo</i> = 'U' の場合、 $\max(1, j-kd) \leq i \leq j$ に対して $ab(kd+1+i-j, j) = A(i, j)$ <i>uplo</i> = 'L' の場合、 $j \leq i \leq \min(n, j+kd)$ に対して $ab(1+i-j, j) = A(i, j)$
<i>ldab</i>	INTEGER。 配列 <i>ab</i> のリーディング・ディメンジョン。 $ldab \geq kd+1$ 。
<i>x</i>	REAL (<i>slatbs</i> の場合) DOUBLE PRECISION (<i>dlatbs</i> の場合) COMPLEX (<i>clatbs</i> の場合) COMPLEX*16 (<i>zlatbs</i> の場合) 配列、次元は (<i>n</i>)。 三角法の右辺 <i>b</i> を格納する。
<i>cnorm</i>	REAL (<i>slatbs/clatbs</i> の場合) DOUBLE PRECISION (<i>dlatbs/zlatbs</i> の場合) 配列、次元は (<i>n</i>)。 <i>normin</i> = 'Y' の場合、 <i>cnorm</i> は入力引数となり <i>cnorm</i> (<i>j</i>) には <i>A</i> の <i>j</i> 番目列の非対角部分を格納する。 <i>trans</i> = 'N' の場合、 <i>cnorm</i> (<i>j</i>) は無限ノルムに等しいか大きくなければならない。 <i>trans</i> = 'T' または 'C' の場合、 <i>cnorm</i> (<i>j</i>) は 1- ノルムに等しいか大きくなければならない。

出力パラメーター

<i>scale</i>	REAL (<i>slatbs/clatbs</i> の場合) DOUBLE PRECISION (<i>dlatbs/zlatbs</i> の場合) 上述のとおり三角法に対するスケール係数 <i>s</i> 。 <i>scale</i> = 0 は行列 <i>A</i> が特異であるか不適切にスケールされたことを示し、ベクトル <i>x</i> は $Ax = 0$ に対する完全または近似解となる。
<i>cnorm</i>	<i>normin</i> = 'N' の場合、 <i>cnorm</i> は出力引数となり <i>cnorm</i> (<i>j</i>) は <i>A</i> の <i>J</i> 番目列の非対角部分の 1- ノルムを返す。

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*info* = -*k* は *k* 番目の引数の値が不正だったことを示す。

?latdf

?getc2 による $n \times n$ 行列の LU 因子分解を使い、
 Dif 推定値の逆数に対する影響を計算する。

構文

```
call slatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call dlatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call clatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
call zlatdf( ijob, n, z, ldz, rhs, rdsum, rdscal, ipiv, jpiv )
```

説明

ルーチン ?latdf は、[?getc2](#) で計算された $n \times n$ 行列 Z の LU 因子分解を使い、 x に対して $Zx = b$ を解き、さらに x のノルムが可能な限り大きくなるような右辺 b を選択して、Dif 推定値の逆数に対する影響を計算する。入力において、 $rhs = b$ にはこれまでに部分式から解いた影響を格納し、出力において $rhs = x$ となる。

?getc2 で返される Z の因子分解は形式 $Z = PLUQ$ を持っており、 P と Q は置換行列である。 L は単位対角成分を持つ下三角、 U は上三角である。

入力パラメーター

ijob INTEGER。
ijob = 2 の場合、最初に ?gecon を使って Z のおおよそのヌルベクトル e を計算し、続いて e は正規化され、最後に 2-ノルム (x) に大きな値を与える符号を使って $Zx = \pm e - f$ を解く。このオプションはデフォルトよりも 5 倍程度負荷が高い。*ijob* ≠ 2 (デフォルト) の場合、右辺 b の全成分が +1 または -1 のいずれかとして選択される局所的先読み法。

n INTEGER。
 行列 Z の列数。

z REAL (slatdf/clatdf の場合)
 DOUBLE PRECISION (dlatdf/zlatdf の場合)
 配列、次元は (*ldz*, *n*)
 $n \times n$ 行列 Z を ?getc2 で計算した因子分解の LU 部分を格納する。 $Z = PLUQ$

ldz INTEGER。
 配列 z のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

<i>rhs</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) 配列、次元は (<i>n</i>)。 <i>rhs</i> には他の部分式で得られた影響を格納する。
<i>rdsum</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) ?tgsy1 で得られた <i>Dif</i> 推定に対する、計算された影響の二乗和を格納する。ここでスケール係数 <i>rdscal</i> は因子分解されている。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdsum</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>rdscal</i>	REAL (slatdf/clatdf の場合) DOUBLE PRECISION (dlatdf/zlatdf の場合) <i>rdsum</i> のオーバーフローを防ぐために使用されるスケール係数を格納する。 <i>trans</i> = 'T' の場合、 <i>rdscal</i> は使われない。 <i>rdscal</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>ipiv</i>	INTEGER。 配列、次元は (<i>n</i>)。 ピボットのインデックス。 $1 \leq i \leq n$ で、行列の行 <i>i</i> は行 <i>ipiv</i> (<i>i</i>) と交換されている。
<i>jpiv</i>	INTEGER。 配列、次元は (<i>n</i>)。 ピボットのインデックス。 $1 \leq j \leq n$ で、行列の列 <i>j</i> は列 <i>jpiv</i> (<i>j</i>) と交換されている。

出力パラメーター

<i>rhs</i>	<i>rhs</i> には、 <i>ijob</i> の値に対応する成分で構成される、部分式の解が上書きされる
<i>rdsum</i>	対応する二乗和は、現在の部分式から得られた影響によって更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。
<i>rdscal</i>	<i>rdscal</i> は、 <i>rdsum</i> に格納されている現在の影響に関して更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。

?latps

圧縮形式で格納されている行列を持った三角連立方程式を解く。

構文

```
call slatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call dlatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call clatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
call zlatps( uplo, trans, diag, normin, n, ap, x, scale, cnorm, info )
```

説明

ルーチン ?latps は、次の三角法のうち 1 つをオーバーフローを防ぐためにスケールリングして解く。

$Ax = sb$ 、または $A^T x = SB$ 、または $A^H x = sb$ (複素型の場合)

A は圧縮形式で格納されている上または下三角行列である。 A^T は A の転置を表わし、 A^H は A の共役転置を表わし、 x と b は n 成分のベクトル、 s はスケール係数で、 x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケールリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン [?tpsv](#) が呼び出される。行列 A が特異 (任意の j に対して $A(j, j) = 0$) の場合、 s は 0 に設定され $Ax = 0$ に対するゼロでない解が返される

入力パラメーター

<i>uplo</i>	CHARACTER*1。 行列 A が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>trans</i>	CHARACTER*1。 A に適用する演算を指定する。 = 'N' の場合、 $Ax = sb$ を解く (転置なし) = 'T' の場合、 $A^T x = sb$ を解く (転置) = 'C' の場合、 $A^H x = sb$ を解く (共役転置)
<i>diag</i>	CHARACTER*1。 行列 A が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<i>normin</i>	CHARACTER*1。 <i>cnorm</i> を設定したかどうかを指定する。 = 'Y' の場合、 <i>cnorm</i> には列ノルムを格納した。 = 'N' の場合、 <i>cnorm</i> は設定していない。ノルムが計算され、出力時に <i>cnorm</i> に格納される。
<i>n</i>	INTEGER。 行列 A の次数。 $n \geq 0$ 。

<i>ap</i>	<p>REAL (slatps の場合) DOUBLE PRECISION (dlatps の場合) COMPLEX (clatps の場合) COMPLEX*16 (zlatps の場合)。 配列、次元は $(n(n+1)/2)$。上三角または下三角帯行列 A で、線形配列にカラム方向に圧縮されている。A の j 番目の列は配列 ap に次のように格納する。 <i>uplo</i> = 'U' の場合、$1 \leq i \leq j$ に対して $ap(i + (j-1)j/2) = A(i, j)$ <i>uplo</i> = 'L' の場合、$j \leq i \leq n$ に対して $ap(i + (j-1)(2n-j)/2) = A(i, j)$</p>
<i>x</i>	<p>REAL (slatps の場合) DOUBLE PRECISION (dlatps の場合) COMPLEX (clatps の場合) COMPLEX*16 (zlatps の場合)。 配列、次元は (n) 三角法の右辺 b を格納する。</p>
<i>cnorm</i>	<p>REAL (slatps/clatps の場合) DOUBLE PRECISION (dlatps/zlatps の場合) 配列、次元は (n)。 <i>normin</i> = 'Y' の場合、<i>cnorm</i> は入力引数となり <i>cnorm</i>(j) には A の j 番目列の非対角部分を格納する。<i>trans</i> = 'N' の場合、<i>cnorm</i>(j) は無限ノルムに等しいか大きくなければならない。 <i>trans</i> = 'T' または 'C' の場合、<i>cnorm</i>(j) は 1- ノルムに等しいか大きくなければならない。</p>

出力パラメーター

<i>x</i>	終了時に、 x は解のベクトル x によって上書きされる。
<i>scale</i>	<p>REAL (slatps/clatps の場合) DOUBLE PRECISION (dlatps/zlatps の場合) 上述のとおり三角法に対するスケール係数 s。 <i>scale</i> = 0 は行列 A が特異であるか不適切にスケールされたことを示し、ベクトル x は $Ax = 0$ に対する完全または近似解となる。</p>
<i>cnorm</i>	<i>normin</i> = 'N' の場合、 <i>cnorm</i> は出力引数となり <i>cnorm</i> (j) は A の J 番目列の非対角部分の 1- ノルムを返す。
<i>info</i>	<p>INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、<i>info</i> = $-k$ は k 番目の引数の値が不正だったことを示す。</p>

?latrd

直交/ユニタリー相似変換を用いて、対称/エルミート行列 A の最初の nb 行/列を実数三重対角形式に縮退する。

構文

```
call slatrd( uplo, n, nb, a, lda, e, tau, w, ldw )
call dlatrd( uplo, n, nb, a, lda, e, tau, w, ldw )
call clatrd( uplo, n, nb, a, lda, e, tau, w, ldw )
call zlatrd( uplo, n, nb, a, lda, e, tau, w, ldw )
```

説明

ルーチン ?latrd は、実数対称または複素エルミート行列 a の NB 行と nb 列を、直交/ユニタリー相似変換 $Q^T A Q$ によって対称/エルミート実数三重対角形式に縮退し、 A の縮退されていない部分に変換を適用するために必要となる V と W を返す。

$uplo = 'U'$ の場合、?latrd は、上三角が与えられる行列に対して最後の nb 行と nb 列の縮退を実行する。

$uplo = 'L'$ の場合、?latrd は、下三角が与えられる行列に対して最初の nb 行と nb 列の縮退を実行する。

このルーチンは、[?sytrd](#)/[?hetrd](#) から呼び出される補助ルーチンである。

入力パラメーター

$uplo$	CHARACTER 対称/エルミート行列 A の上三角または下三角部分のどちらを格納するか指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
n	INTEGER。 行列 A の次数。
nb	INTEGER。 縮退する行数と列数。
a	REAL (slatrd の場合) DOUBLE PRECISION (dlatrd の場合) COMPLEX (clatrd の場合) COMPLEX*16 (zlatrd の場合) 配列、次元は (lda, n) 。 対称/エルミート行列 A を格納する。 $uplo = 'U'$ の場合、 a の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 a の先頭の $n \times n$ 下三角部分は行列 A の下三角部分を含む。 a の厳密な上三角部分は参照されない。
lda	INTEGER。 配列 a のリーディング・ディメンジョン。 $lda \geq (1, n)$ 。

ldw INTEGER。
出力配列 *w* のリーディング・ディメンジョン。 $ldw \geq \max(1, n)$ 。

出力パラメーター

a *uplo* = 'U' の場合、最後の *nb* 列が三重対角形式に縮退され、*a* の対角成分は三重対角形式の対角成分で上書きされる。対角よりも上の成分は配列 *tau* とともに基本リフレクターの積として直交/ユニタリーの行列 *Q* を表現する。
uplo = 'L' の場合、最初の *nb* 列が三重対角形式に縮退され、*a* の対角成分は三重対角形式の対角成分で上書きされる。対角よりも下の成分は配列 *tau* とともに基本リフレクターの積として直交/ユニタリーの行列 *Q* を表現する。

e REAL (slatrd/clatrd の場合)
DOUBLE PRECISION (dlatrd/zlatrd の場合)
uplo = 'U' の場合、*e*(*n-nb:n-1*) には縮退された行列の最後の *nb* 列の優対角成分が格納される。
uplo = 'L' の場合、*e*(*1:nb*) には縮退された行列の最初の *nb* 列の劣対角成分が格納される。

tau REAL (slatrd の場合)
DOUBLE PRECISION (dlatrd の場合)
COMPLEX (clatrd の場合)
COMPLEX*16 (zlatrd の場合)
配列、次元は (*lda*, *n*)。
基本リフレクターのスケール係数で、*uplo* = 'U' の場合は *tau*(*n-nb:n-1*) に、*uplo* = 'L' の場合は *tau*(*1:nb*) に格納される。

w REAL (slatrd の場合)
DOUBLE PRECISION (dlatrd の場合)
COMPLEX (clatrd の場合)
COMPLEX*16 (zlatrd の場合)
配列、次元は (*lda*, *n*)。
A の縮退されていない部分の更新に必要な $n \times nb$ 行列 *W*。

アプリケーション・ノート

uplo = 'U' の場合、行列 *Q* は基本リフレクターの積として表現される。

$$Q = H(n) H(n-1) \dots H(n-nb+1)$$

それぞれの *H*(*i*) は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで $v(i:n) = 0$ と $v(i-1) = 1$ 。出力において、*v*(*1:i-1*) は *a*(*1:i-1*, *i*) に格納され、*tau* は *tau*(*i-1*) に格納される。

uplo = 'L' の場合、行列 *Q* は基本リフレクターの積として表現される。

$$Q = H(1) H(2) \dots H(nb)$$

それぞれの *H*(*i*) は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

tau は実数 / 複素スカラー、*v* は実数 / 複素ベクトルで $v(1:i) = 0$ と $v(i+1) = 1$ 。出力において、*v*(*i+1:n*) は *a*(*i+1:n*, *i*) に格納され、*tau* は *tau*(*i*) に格納される。

ベクトル v の成分は、行列の縮退されていない部分に変換を適用するために、行列 W とともに必要となる $n \times nb$ の行列 V を、形式の対称 / エルミート階数 $-2k$ 更新を使用して形成する。

$$A := A - VW' - WV'$$

終了時の a の内容を次に示す ($n=5$ および $nb=2$ の場合)。

$uplo = 'U'$ の場合

$uplo = 'L'$ の場合

$$\begin{array}{c} \left[\begin{array}{ccccc} a & a & a & v_4 & v_5 \\ & a & a & v_4 & v_5 \\ & & a & 1 & v_5 \\ & & & d & 1 \\ & & & & d \end{array} \right] \end{array} \quad \begin{array}{c} \left[\begin{array}{ccccc} d & & & & \\ & 1 & d & & \\ & v_1 & 1 & a & \\ & v_1 & v_2 & a & a \\ & v_1 & v_2 & a & a & a \end{array} \right] \end{array}$$

d は縮退された行列の対角成分、 a は変更されていない元の行列の成分、 v_i は $H(i)$ を定義するベクトルの成分を表わす。

?latrs

オーバーフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。

構文

```
call slatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call dlatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call clatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
call zlatrs( uplo, trans, diag, normin, n, a, lda, x, scale, cnorm, info )
```

説明

このルーチンは、次の三角法をオーバーフローを防ぐためにスケールリングして解く。

$$AX = sb, \text{ または } A^T x = SB, \text{ または } A^H x = sb \text{ (複素型の場合)}$$

A は上または下三角行列である。 A^T は A の転置を表わし、 A^H は A の共役転置を表わし、 x と b は n 成分のベクトル、 s はスケール係数で、 x の成分がオーバーフローしきい値未満になるように通常は 1 以下に選択される。スケールリングされていない問題がオーバーフローを起こさない場合は、レベル 2 の BLAS ルーチン [?trsv](#) が呼び出される。行列 A が特異 (任意の j に対して $A(j, j) = 0$) の場合、 s は 0 に設定され $Ax = 0$ に対するゼロでない解が返される

入力パラメーター

<i>uplo</i>	CHARACTER*1。 行列 <i>A</i> が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>trans</i>	CHARACTER*1。 <i>A</i> に適用する演算を指定する。 = 'N' の場合、 $Ax = sb$ を解く (転置なし) = 'T' の場合、 $A^T x = sb$ を解く (転置) = 'C' の場合、 $A^H x = sb$ を解く (共役転置)
<i>diag</i>	CHARACTER*1。 行列 <i>A</i> が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<i>normin</i>	CHARACTER*1。 <i>cnorm</i> を設定したかどうかを指定する。 = 'Y' の場合、 <i>cnorm</i> には列ノルムを格納した。 = 'N' の場合、 <i>cnorm</i> は設定していない。終了時に、ノルムが計算され、 <i>cnorm</i> に格納される。
<i>n</i>	INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$
<i>a</i>	REAL (<i>slatrs</i> の場合) DOUBLE PRECISION (<i>dlatrs</i> の場合) COMPLEX (<i>clatrs</i> の場合) COMPLEX*16 (<i>zlatrs</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。三角行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> の先頭の $n \times n$ の上三角部分に上三角行列を格納する。 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> の先頭の $n \times n$ の下三角部分に下三角行列を格納する。 <i>a</i> の厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合も <i>a</i> の対角成分は参照されず 1 とみなされる。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。
<i>x</i>	REAL (<i>slatrs</i> の場合) DOUBLE PRECISION (<i>dlatrs</i> の場合) COMPLEX (<i>clatrs</i> の場合) COMPLEX*16 (<i>zlatrs</i> の場合) 配列、次元は (<i>n</i>)。三角法の右辺 <i>b</i> を格納する。
<i>cnorm</i>	REAL (<i>slatrs/clatrs</i> の場合) DOUBLE PRECISION (<i>dlatrs/zlatrs</i> の場合) 配列、次元は (<i>n</i>)。 <i>normin</i> = 'Y' の場合、 <i>cnorm</i> は入力引数となり、 <i>cnorm</i> (<i>j</i>) には <i>A</i> の <i>j</i> 番目の列の非対角部分を格納する。 <i>trans</i> = 'N' の場合、 <i>cnorm</i> (<i>j</i>) は無限ノルムに等しいか大きくなければならない。 <i>trans</i> = 'T' または 'C' の場合、 <i>cnorm</i> (<i>j</i>) は 1-ノルムに等しいか大きくなければならない。

出力パラメーター

x	終了時に、 x は解のベクトル x によって上書きされる。
$scale$	REAL (slatrs/clatrs の場合) DOUBLE PRECISION (dlatrs/zlatrs の場合) 配列、次元は (lda, n) 。上述のとおり三角法に対するスケール係数 s 。 $scale = 0$ は行列 A が特異であるか不適切にスケーリングされたことを示し、ベクトル x は $Ax = 0$ に対する完全または近似解となる。
$cnorm$	$normin = 'n'$ の場合、 $cnorm$ は出力引数となり $cnorm(j)$ は A の j 番目列の非対角部分の 1- ノルムを返す。
$info$	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -k$ は k 番目の引数の値が不正だったことを示す。

アプリケーション・ノート

x に対するおおまかな制限が計算される。これがオーバーフロー未満の場合は [?trsv](#) が呼び出され、そうでなければ演算ごとにオーバーフローまたはゼロ除算をチェックする特定のコードが使われる。

$Ax = b$ を解くうえで列方向の格納形式が使用される。 A が下三角の場合、基本アルゴリズムは次のようになる。

```

x[1:n] := b[1:n]
j = 1, ..., n の場合、
x(j) := x(j) / A(j,j)
x[j+1:n] := x[j+1:n] - x(j)*A[j+1:n,j]
end

```

ループを j 回反復した後、 x の成分に対する制限を定義する。

$M(j) = x[1:j]$ に対する制限
 $G(j) = x[j+1:n]$ に対する制限
初期時は、 $M(0) = 0$ and $G(0) = \max\{x(i), i=1, \dots, n\}$ とする。

$j+1$ 回目の反復で、以下を得る。

$$M(j+1) \leq G(j) / |A(j+1, j+1)|$$

$$G(j+1) \leq G(j) + M(j+1) * |A[j+2:n, j+1]|$$

$$\leq G(j) (1 + cnorm(j+1) / |A(j+1, j+1)|)$$

$cnorm(j+1)$ は、対角を含めない状態で A の列 $J+1$ の無限ノルムに等しいか大きい。ゆえに、

$$G(j) \leq G(0) \prod_{1 \leq i \leq j} (1 + cnorm(i) / |A(i, i)|)$$

および

$$|x(j)| \leq (G(0)/|A(j,j)|) \prod_{1 \leq i \leq j} (1 + c_{norm}(i)/|A(i,i)|).$$

$|x(j)| \leq M(j)$ であるから、 $j=1, \dots, n$ における最大 $M(j)$ の逆数が $\max(\text{underflow}, 1/\text{overflow})$ よりも大きい場合はレベル 2 の BLAS ルーチン `?trsv` を使用する。

$x(j)$ に対する制限もまた、オーバーフローの可能性のない、列方向の格納形式でのステップの実行判断に使用される。計算された制限が大きな定数よりも大きい場合、オーバーフローを防ぐために x はスケールされる。しかし、制限がオーバーフローする場合、 x は 0 に、 $x(j)$ は 1 に、`scale` は 0 にそれぞれ設定され、 $Ax=0$ に対するゼロでない解が返される。

同様に、列方向の格納形式が $A^T x = b$ または $A^H x = b$ を解くために使用される。 A が上三角の場合、基本アルゴリズムは次のようになる。

```

j = 1, ..., n の場合、
x(j) := ( b(j) - A[1:j-1, j]' x[1:j-1] ) / A(j, j)
end

```

2 つの制限が同時に計算される。

$G(j) = (b(i) - A[1:i-1, i]' x[1:i-1])$ に対する制限、 $1 \leq i \leq j$

$M(j) = x(i)$ に対する制限、 $1 \leq i \leq j$

初期値は $G(0) = 0$ 、 $M(0) = \max\{b(i), i=1, \dots, n\}$ で、ここで $j \geq 1$ に対し $G(j) \geq G(j-1)$ と $M(j) \geq M(j-1)$ の制約を追加する。

よって $x(j)$ に対する制限は、

$$\begin{aligned}
M(j) &\leq M(j-1) * (1 + c_{norm}(j)) / |A(j, j)| \\
&\leq M(0) \prod_{1 \leq i \leq j} (1 + c_{norm}(i) / |A(i, i)|)
\end{aligned}$$

これによって、 $1/M(n)$ と $1/G(n)$ の両方が $\max(\text{underflow}, 1/\text{overflow})$ を超えている場合でも安全に `?trsv` を呼び出すことができる。

?latrz

上台形行列を直交/ユニタリ変換によって因子分解する。

構文

```

call slatz( m, n, l, a, lda, tau, work )
call dlatrz( m, n, l, a, lda, tau, work )
call clatz( m, n, l, a, lda, tau, work )
call zlatrz( m, n, l, a, lda, tau, work )

```

説明

ルーチン `?latrz` は、次の $m \times (m+1)$ の実数 / 複素上台形行列を、
 $[A1 \ A2] = [A(1:m, 1:m) \ A(1:m, n-l+1:n)]$

$(R \ 0) * Z$ として、直交 / ユニタリー変換を使用して因子分解する。Z は $(m+1) \times (m+1)$ 直交 / ユニタリー行列、R と A1 は $m \times m$ の上三角行列である。

入力パラメーター

<code>m</code>	INTEGER。 行列 A の行数。 $m \geq 0$ 。
<code>n</code>	INTEGER。 行列 A の列数。 $n \geq 0$ 。
<code>l</code>	INTEGER。 Householder ベクトルとして意味を持った部分が格納されている行列 A の列数。 $n-m \geq l \geq 0$
<code>a</code>	REAL (<code>slatz</code> の場合) DOUBLE PRECISION (<code>dlatz</code> の場合) COMPLEX (<code>clatz</code> の場合) COMPLEX*16 (<code>zlatrz</code> の場合) 配列、次元は (lda, n) 。 配列 <code>a</code> 先頭の $m \times n$ 上台形部分には因子分解する行列を格納しておかなければならない。
<code>lda</code>	INTEGER。 配列 <code>a</code> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$
<code>work</code>	REAL (<code>slatz</code> の場合) DOUBLE PRECISION (<code>dlatz</code> の場合) COMPLEX (<code>clatz</code> の場合) COMPLEX*16 (<code>zlatrz</code> の場合) ワークスペース配列、次元は (m)

出力パラメーター

<code>a</code>	<code>a</code> の先頭の $m \times m$ 上三角部分には上三角行列 R が格納される。 <code>a</code> の最初の m 行の $n-l+1$ から n までの成分は、配列 <code>tau</code> とともに、基本リフレクター m の積として直交 / ユニタリー行列 Z を表現する。
<code>tau</code>	REAL (<code>slatz</code> の場合) DOUBLE PRECISION (<code>dlatz</code> の場合) COMPLEX (<code>clatz</code> の場合) COMPLEX*16 (<code>zlatrz</code> の場合) 配列、次元は (m) 。基本リフレクターのスカラー係数。

アプリケーション・ノート

因子分解は Householder 法を用いて得る。A の $(m - k + 1)$ 番目の行にゼロを導入するために使用される k 番目の変換行列 $Z(k)$ は次の形式で与えられる。

$$Z(k) = \begin{bmatrix} I & 0 \\ 0 & T(k) \end{bmatrix},$$

ここで、

$$T(k) = I - \tau * u(k) * u(k)' \quad u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

τ はスカラー、 $z(k)$ は l 成分のベクトルである。 τ と $z(k)$ は、 $A2$ の k 番目の行の成分をゼロにするために選択される。

スカラー τ は τ の k 番目の成分で返され、ベクトル $u(k)$ は、 $z(k)$ の成分が $a(k, l+1), \dots, a(k, n)$ に格納されるような $A2$ の k 番目の行で返される。 R の成分は $A1$ の上三角部分で返される。

Z は次の式で与えられる。

$$Z = Z(1)Z(2) \dots Z(m)$$

?lauu2

積 UU^H または $L^H L$ を計算する。ここで U と L は上三角または下三角行列(非ブロック化アルゴリズム)。

構文

```
call slauu2( uplo, n, a, lda, info )
call dlauu2( uplo, n, a, lda, info )
call clauu2( uplo, n, a, lda, info )
call zlauu2( uplo, n, a, lda, info )
```

説明

ルーチン ?lauu2 は積 UU' または LL' を計算する。ここで、三角係数 U または L は、それぞれ配列 a の上三角または下三角部分に格納される。

$uplo = 'U'$ または $'u'$ の場合、 a の係数 U は結果の上三角で上書きされる。

$uplo = 'L'$ または $'l'$ の場合、 a の係数 L は結果の下三角で上書きされる。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル2のルーチン](#) を呼び出す。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 配列 <i>a</i> に格納されている三角係数が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 三角係数 <i>U</i> または <i>L</i> の次数。 $n \geq 0$ 。
<i>a</i>	REAL (slauu2 の場合) DOUBLE PRECISION (dlauu2 の場合) COMPLEX (clauu2 の場合) COMPLEX*16 (zlauu2 の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。三角係数 <i>U</i> または <i>L</i> を格納する。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	<i>uplo</i> = 'U' の場合、 <i>a</i> の上三角は積 <i>UU'</i> の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、 <i>a</i> の下三角は積 <i>LL'</i> の下三角で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。

?lauum

積 UU^H または $L^H L$ を計算する。ここで *U* と *L* は上三角または下三角行列(ブロック化アルゴリズム)。

構文

```
call slauum( uplo, n, a, lda, info )
call dlauum( uplo, n, a, lda, info )
call clauum( uplo, n, a, lda, info )
call zlauum( uplo, n, a, lda, info )
```

説明

ルーチン ?lauum は積 *UU'* または *LL'* を計算する。三角係数 *U* または *L* は、配列 *a* の上三角または下三角部分に格納される。

uplo = 'U' または 'u' の場合、*a* の係数 *U* は結果の上三角で上書きされる。
uplo = 'L' または 'l' の場合、*a* の係数 *L* は結果の下三角で上書きされる。

このルーチンはアルゴリズムのブロック化形式で、[BLAS レベル 3 のルーチン](#)を呼び出す。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 配列 <i>a</i> に格納されている三角係数が上三角か下三角かを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 三角係数 <i>U</i> または <i>L</i> の次数。 $n \geq 0$ 。
<i>a</i>	REAL (<i>slauum</i> の場合) DOUBLE PRECISION (<i>dlauum</i> の場合) COMPLEX (<i>clauum</i> の場合) COMPLEX*16 (<i>zlauum</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。三角係数 <i>U</i> または <i>L</i> を格納する。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	<i>uplo</i> = 'U' の場合、 <i>a</i> の上三角は積 <i>UU'</i> の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、 <i>a</i> の下三角は積 <i>LL'</i> の下三角で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。

?org2l/?ung2l

?geqlf で求めた *QL* 因子分解から、全部または一部の直交/ユニタリー行列 *Q* を生成する (非ブロック化アルゴリズム)。

構文

```
call sorg2l( m, n, k, a, lda, tau, work, info )
call dorg2l( m, n, k, a, lda, tau, work, info )
call corg2l( m, n, k, a, lda, tau, work, info )
call zorg2l( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org2l/?ung2l は、直交列を持つ $m \times n$ の実数/複素行列 *Q* を生成する。これは、次数 *m* の *k* 個の基本リフレクターの積の最後の *n* 列として定義される。

$Q = H(k) \dots H(2) H(1)$ (?geqlf によって返される)

入力パラメーター

<i>m</i>	INTEGER。 行列 <i>Q</i> の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 <i>Q</i> の列数。 $m \geq n \geq 0$ 。
<i>k</i>	INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクターの個数。 $n \geq k \geq 0$ 。
<i>a</i>	REAL (sorg21 の場合) DOUBLE PRECISION (dorg21 の場合) COMPLEX (cung21 の場合) COMPLEX*16 (zung21 の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 <i>a</i> の (<i>n-k+i</i>) 番目の列には、?geqlf によって配列引数 <i>a</i> の最後の <i>k</i> 列に返されたとおりに、基本リフレクター <i>H(i)</i> を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$
<i>lda</i>	INTEGER。 配列 <i>a</i> の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorg21 の場合) DOUBLE PRECISION (dorg21 の場合) COMPLEX (cung21 の場合) COMPLEX*16 (zung21 の場合) 配列、次元は (<i>k</i>)。 <i>tau(i)</i> には、?geqlf から返されたとおりに、基本リフレクター <i>H(i)</i> のスカラー係数を格納しておかなければならない。
<i>work</i>	REAL (sorg21 の場合) DOUBLE PRECISION (dorg21 の場合) COMPLEX (cung21 の場合) COMPLEX*16 (zung21 の場合) ワークスペース配列、次元は (<i>n</i>)。

出力パラメーター

<i>a</i>	$m \times n$ の行列 <i>Q</i> で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?org2r/?ung2r

?geqrf で求めた QR 因子分解から、全部または一部の直交/ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。

構文

```
call sorg2r( m, n, k, a, lda, tau, work, info )
call dorg2r( m, n, k, a, lda, tau, work, info )
call cung2r( m, n, k, a, lda, tau, work, info )
call zung2r( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org2r/?ung2r は、直交列を持つ $m \times n$ の実数/複素行列 Q を生成する。これは、次数 m の k 個の基本リフレクターの積の最初の n 列として定義される。

$$Q = H(1)H(2) \dots H(k)$$

[?geqrf](#) によって返される。

入力パラメーター

<i>m</i>	INTEGER。 行列 Q の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 Q の列数。 $m \geq n \geq 0$ 。
<i>k</i>	INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $n \geq k \geq 0$ 。
<i>a</i>	REAL (sorg2r の場合) DOUBLE PRECISION (dorg2r の場合) COMPLEX (cung2r の場合) COMPLEX*16 (zung2r の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 <i>a</i> の <i>i</i> 番目の列には、?geqrf によって配列引数 <i>a</i> の最初の <i>k</i> 列に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$
<i>lda</i>	INTEGER。 配列 <i>a</i> の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorg2r の場合) DOUBLE PRECISION (dorg2r の場合) COMPLEX (cung2r の場合) COMPLEX*16 (zung2r の場合) 配列、次元は (<i>k</i>)。 <i>tau</i> (<i>i</i>) には、?geqrf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数を格納しておかなければならない。

work REAL (sorg2r の場合)
 DOUBLE PRECISION (dorg2r の場合)
 COMPLEX (cung2r の場合)
 COMPLEX*16 (zung2r の場合)
 ワークスペース配列、次元は (n)。

出力パラメーター

a $m \times n$ の行列 Q で上書きされる。

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?orgl2/?ungl2

?gelqf で求めた LQ 因子分解から、全部または一部の直交/ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。

構文

```
call sorgl2( m, n, k, a, lda, tau, work, info )
call dorgl2( m, n, k, a, lda, tau, work, info )
call cungl2( m, n, k, a, lda, tau, work, info )
call zungl2( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?org21/?ung21 は、直交行を持つ $m \times n$ の実数/複素行列 Q を生成する。これは、次数 n の k 個の基本リフレクターの積の最初の m 行として定義される。

$$Q = H(k) \dots H(2) H(1) \text{ または } Q = H(k)' \dots H(2)' H(1)'$$

[?gelqf](#) によって返される。

入力パラメーター

m INTEGER。
 行列 Q の行数。 $m \geq 0$ 。

n INTEGER。
 行列 Q の列数。 $n \geq m$

k INTEGER。
 その積が行列 Q を定義する基本リフレクターの個数。
 $m \geq k \geq 0$

a REAL (sorgl2 の場合)
 DOUBLE PRECISION (dorgl2 の場合)
 COMPLEX (cungl2 の場合)
 COMPLEX*16 (zungl2 の場合)

配列、次元は (lda, n) 。 a の i 番目の行には、?gelqf によって配列引数 a の最初の k 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$

<i>lda</i>	INTEGER。 配列 a の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorgl2 の場合) DOUBLE PRECISION (dorgl2 の場合) COMPLEX (cungl2 の場合) COMPLEX*16 (zungl2 の場合) 配列、次元は (k) 。 $tau(i)$ には、?gelqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数を格納しておかなければならない。
<i>work</i>	REAL (sorgl2 の場合) DOUBLE PRECISION (dorgl2 の場合) COMPLEX (cungl2 の場合) COMPLEX*16 (zungl2 の場合) ワークスペース配列、次元は (m)

出力パラメーター

<i>a</i>	$m \times n$ の行列 Q で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は i 番目の引数の値が不正だったことを示す。

?orgr2/?ungr2

?gerqf で求めた RQ 因子分解から、全部または一部の直交/ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。

構文

```
call sorgr2( m, n, k, a, lda, tau, work, info )
call dorgr2( m, n, k, a, lda, tau, work, info )
call cungr2( m, n, k, a, lda, tau, work, info )
call zungr2( m, n, k, a, lda, tau, work, info )
```

説明

ルーチン ?orgr2/?ungr2 は、直交行を持つ $m \times n$ の実数の行列 Q を生成する。これは、から返されたとおりに、次数 n の k 個の基本リフレクターの積の最後の m 行として定義される。

$Q = H(1)H(2) \dots H(k)$ または $Q = H(1)'H(2)' \dots H(k)'$

?gerqf によって返される。

入力パラメーター

<i>m</i>	INTEGER。行列 <i>Q</i> の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 <i>Q</i> の列数。 $n \geq m$
<i>k</i>	INTEGER。 その積が行列 <i>Q</i> を定義する基本リフレクターの個数。 $m \geq k \geq 0$
<i>a</i>	REAL (sorgr2 の場合) DOUBLE PRECISION (dorgr2 の場合) COMPLEX (cungr2 の場合) COMPLEX*16 (zungr2 の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 <i>a</i> の (<i>m-k+i</i>) 番目の行には、?gerqf に よって配列引数 <i>a</i> の最後の <i>k</i> 行に返されたとおりに、基本リフ レクター <i>H(i)</i> を定義するベクトルを格納しておかなければなら ない。ここで、 $i = 1, 2, \dots, k$
<i>lda</i>	INTEGER。 配列 <i>a</i> の第 1 次元。 $lda \geq \max(1, m)$
<i>tau</i>	REAL (sorgr2 の場合) DOUBLE PRECISION (dorgr2 の場合) COMPLEX (cungr2 の場合) COMPLEX*16 (zungr2 の場合) 配列、次元は (<i>k</i>)。 <i>tau(i)</i> には、?gerqf から返されたとおりに、 基本リフレクター <i>H(i)</i> のスカラー係数を格納しておかなければ ならない。
<i>work</i>	REAL (sorgr2 の場合) DOUBLE PRECISION (dorgr2 の場合) COMPLEX (cungr2 の場合) COMPLEX*16 (zungr2 の場合) ワークスペース配列、次元は (<i>m</i>)

出力パラメーター

<i>a</i>	$m \times n$ の行列 <i>Q</i> で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示 す。

?orm2l/?unm2l

一般行列と `?geqlf` で求めた QL 因子分解の直交/ユニタリー行列を乗算する (非ブロック化アルゴリズム)。

構文

```
call sorm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2l( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン `?orm2l/?unm2l` は一般実数 / 複素 $m \times n$ 行列 C を以下で上書きする。

$side = 'L'$ かつ $trans = 'N'$ の場合、 Q^*C 、
 $side = 'L'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 Q^*C 、

$side = 'R'$ かつ $trans = 'N'$ の場合、 C^*Q 、
 $side = 'R'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 C^*Q

Q は、 k 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(k) \dots H(2) H(1)$$

`?geqlf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

side CHARACTER*1。
 = 'L' の場合、 Q または Q' を左から適用する。
 = 'R' の場合、 Q または Q' を右から適用する。

trans CHARACTER*1。
 = 'N' の場合、 Q を適用する (転置なし)
 = 'T' の場合、 Q' を適用する (転置、実数型の場合)
 = 'c' の場合、 Q' (共役転置、複素数型の場合)

m INTEGER。
 行列 C の行数。 $m \geq 0$ 。

n INTEGER。
 行列 C の列数。 $n \geq 0$ 。

k INTEGER。
 積が行列 Q を定義する基本リフレクターの個数。
 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。
 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。

<i>a</i>	REAL (sorm21 の場合) DOUBLE PRECISION (dorm21 の場合) COMPLEX (cunm21 の場合) COMPLEX*16 (zunm21 の場合) 配列、次元は (<i>lda</i> , <i>k</i>)。 <i>a</i> の <i>i</i> 番目の列には、?geqlf によって配列引数 <i>a</i> の最後の <i>k</i> 列に返されたとおりに、基本リフレクター <i>H</i> (<i>i</i>) を定義するベクトルを格納しておかなければならない。ここで、 <i>i</i> = 1, 2, ..., <i>k</i> 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 <i>side</i> = 'L' の場合、 <i>lda</i> ≥ max(1, <i>m</i>)、 <i>side</i> = 'R' の場合、 <i>lda</i> ≥ max(1, <i>n</i>)。
<i>tau</i>	REAL (sorm21 の場合) DOUBLE PRECISION (dorm21 の場合) COMPLEX (cunm21 の場合) COMPLEX*16 (zunm21 の場合) 配列、次元は (<i>k</i>)。 <i>tau</i> (<i>i</i>) には、?geqlf から返されたとおりに、基本リフレクター <i>h</i> (<i>i</i>) のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sorm21 の場合) DOUBLE PRECISION (dorm21 の場合) COMPLEX (cunm21 の場合) COMPLEX*16 (zunm21 の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 <i>m</i> × <i>n</i> の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>C</i> のリーディング・ディメンジョン。 <i>ldc</i> ≥ max(1, <i>m</i>)
<i>work</i>	REAL (sorm21 の場合) DOUBLE PRECISION (dorm21 の場合) COMPLEX (cunm21 の場合) COMPLEX*16 (zunm21 の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>c</i> は <i>QC</i> または <i>Q'C</i> 、あるいは <i>CQ</i> または <i>CQ'</i> によって上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?orm2r/?unm2r

一般行列と `?geqrf` で求めた *QR* 因子分解の直交/ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。

構文

```
call sorm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunm2r( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン `?orm2l/?unm2l` は一般実数 / 複素 $m \times n$ 行列 C を以下で上書きする。

$side = 'L'$ かつ $trans = 'N'$ の場合、 Q^*C 、
 $side = 'L'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 Q^*C 、

$side = 'R'$ かつ $trans = 'N'$ の場合、 C^*Q 、
 $side = 'R'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 C^*Q

Q は、 k 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(1) H(2) \dots H(k)$$

`?geqrf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

side CHARACTER*1。
 = 'L' の場合、 Q または Q' を左から適用する。
 = 'R' の場合、 Q または Q' を右から適用する。

trans CHARACTER*1。
 = 'N' の場合、 Q を適用する (転置なし)
 = 'T' の場合、 Q' を適用する (転置、実数型の場合)
 = 'c' の場合、 Q' (共役転置、複素数型の場合)

m INTEGER。
 行列 C の行数。 $m \geq 0$ 。

n INTEGER。
 行列 C の列数。 $n \geq 0$ 。

k INTEGER。
 その積が行列 Q を定義する基本リフレクターの個数。
 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。
 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。

<i>a</i>	REAL (sorm2r の場合) DOUBLE PRECISION (dorm2r の場合) COMPLEX (cunm2r の場合) COMPLEX*16 (zunm2r の場合) 配列、次元は (<i>lda</i> , <i>k</i>)。 <i>a</i> の <i>i</i> 番目の列には、?geqrf によって配列引数 <i>a</i> の最初の <i>k</i> 列に返されたとおりに、基本リフレクター <i>H</i> (<i>i</i>) を定義するベクトルを格納しておかなければならない。ここで、 <i>i</i> = 1, 2, ..., <i>k</i> 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 <i>side</i> = 'L' の場合、 <i>lda</i> ≥ max(1, <i>m</i>)、 <i>side</i> = 'R' の場合、 <i>lda</i> ≥ max(1, <i>n</i>)。
<i>tau</i>	REAL (sorm2r の場合) DOUBLE PRECISION (dorm2r の場合) COMPLEX (cunm2r の場合) COMPLEX*16 (zunm2r の場合) 配列、次元は (<i>k</i>)。 <i>tau</i> (<i>i</i>) には、?geqrf から返されたとおりに、基本リフレクター <i>H</i> (<i>i</i>) のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sorm2r の場合) DOUBLE PRECISION (dorm2r の場合) COMPLEX (cunm2r の場合) COMPLEX*16 (zunm2r の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 <i>m</i> × <i>n</i> の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>C</i> のリーディング・ディメンジョン。 <i>ldc</i> ≥ max(1, <i>m</i>)
<i>work</i>	REAL (sorm2r の場合) DOUBLE PRECISION (dorm2r の場合) COMPLEX (cunm2r の場合) COMPLEX*16 (zunm2r の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>c</i> は <i>QC</i> または <i>Q'C</i> 、あるいは <i>CQ'</i> または <i>CQ</i> によって上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?orml2/?unml2

一般行列と `?gelqf` で求めた LQ 因子分解の直交/ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。

構文

```
call sorml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dorml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunml2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン `?orml2/?unml2` は一般実数/複素 $m \times n$ 行列 C を以下で上書きする。

$side = 'L'$ かつ $trans = 'N'$ の場合、 Q^*C 、
 $side = 'L'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 Q^*C 、

$side = 'R'$ かつ $trans = 'N'$ の場合、 C^*Q 、
 $side = 'R'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 C^*Q

Q は、 k 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(k) \dots H(2) H(1) \text{ または } Q = H(k)' \dots H(2)' H(1)'$$

`?gelqf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	CHARACTER*1。 = 'L' の場合、 Q または Q' を左から適用する。 = 'R' の場合、 Q または Q' を右から適用する。
<i>trans</i>	CHARACTER*1。 = 'N' の場合、 Q を適用する (転置なし) = 'T' の場合、 Q' を適用する (転置、実数型の場合) = 'c' の場合、 Q' (共役転置、複素数型の場合)
<i>m</i>	INTEGER。 行列 C の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 C の列数。 $n \geq 0$ 。
<i>k</i>	INTEGER。 基本リフレクター (その積で行列 Q を定義) の個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。

<i>a</i>	REAL (sorml2 の場合) DOUBLE PRECISION (dorml2 の場合) COMPLEX (cunml2 の場合) COMPLEX*16 (zunml2 の場合) 配列、次元は <i>side</i> = 'L' の場合 (<i>lda</i> , <i>m</i>)、 <i>side</i> = 'R' の場合は (<i>lda</i> , <i>n</i>)。 <i>a</i> の <i>i</i> 番目の行には、?gelqf によって配列引数 <i>a</i> の最初の <i>k</i> 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $i=1, 2, \dots, k$ 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, k)$
<i>tau</i>	REAL (sorml2 の場合) DOUBLE PRECISION (dorml2 の場合) COMPLEX (cunml2 の場合) COMPLEX*16 (zunml2 の場合) 配列、次元は (<i>k</i>)。 <i>tau</i> (<i>i</i>) には、?gelqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (sorml2 の場合) DOUBLE PRECISION (dorml2 の場合) COMPLEX (cunml2 の場合) COMPLEX*16 (zunml2 の場合) 配列、次元は (<i>ldc</i> , <i>n</i>) $m \times n$ の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (sorml2 の場合) DOUBLE PRECISION (dorml2 の場合) COMPLEX (cunml2 の場合) COMPLEX*16 (zunml2 の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>c</i> は QC または $Q'C$ 、あるいは CQ' または CQ によって上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?ormr2/?unmr2

一般行列と `?gerqf` で求めた RQ 因子分解の直交/ユニタリー行列とを乗算する(非ブロック化アルゴリズム)。

構文

```
call sormr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call dormr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call cunmr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
call zunmr2( side, trans, m, n, k, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン `?ormr2/?unmr2` は一般実数/複素 $m \times n$ 行列 C を以下で上書きする。

$side = 'L'$ かつ $trans = 'N'$ の場合、 $Q * C$ 、
 $side = 'L'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 $Q' * C$ 、

$side = 'R'$ かつ $trans = 'N'$ の場合、 $C * Q$ 、
 $side = 'R'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 $C * Q'$

Q は、 k 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(1)H(2) \dots H(k) \text{ または } Q = H(1)'H(2)' \dots H(k)'$$

`?gerqf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	CHARACTER*1。 = 'L' の場合、 Q または Q' を左から適用する。 = 'R' の場合、 Q または Q' を右から適用する。
<i>trans</i>	CHARACTER*1。 = 'N' の場合、 Q を適用する (転置なし) = 'T' の場合、 Q' を適用する (転置、実数型の場合) = 'c' の場合、 Q' (共役転置、複素数型の場合)
<i>m</i>	INTEGER。 行列 C の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 C の列数。 $n \geq 0$ 。
<i>k</i>	INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。

<i>a</i>	<p>REAL (sormr2 の場合) DOUBLE PRECISION (dormr2 の場合) COMPLEX (cunmr2 の場合) COMPLEX*16 (zunmr2 の場合) 配列、次元は <i>side</i> = 'L' の場合 (<i>lda</i>, <i>m</i>)、 <i>side</i> = 'R' の場合は (<i>lda</i>, <i>n</i>)。 <i>a</i> の <i>i</i> 番目の行には、?gerqf によって配列引数 <i>A</i> の最後の <i>k</i> 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、$i = 1, 2, \dots, k$。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。</p>
<i>lda</i>	<p>INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, k)$</p>
<i>tau</i>	<p>REAL (sormr2 の場合) DOUBLE PRECISION (dormr2 の場合) COMPLEX (cunmr2 の場合) COMPLEX*16 (zunmr2 の場合) 配列、次元は (<i>k</i>)。 <i>tau</i>(<i>i</i>) には、?gerqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数を格納しておかなければならない。</p>
<i>c</i>	<p>REAL (sormr2 の場合) DOUBLE PRECISION (dormr2 の場合) COMPLEX (cunmr2 の場合) COMPLEX*16 (zunmr2 の場合) 配列、次元は (<i>ldc</i>, <i>n</i>)。 <i>m</i> × <i>n</i> の行列 <i>C</i> を格納する。</p>
<i>ldc</i>	<p>INTEGER。 配列 <i>C</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$</p>
<i>work</i>	<p>REAL (sormr2 の場合) DOUBLE PRECISION (dormr2 の場合) COMPLEX (cunmr2 の場合) COMPLEX*16 (zunmr2 の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)</p>

出力パラメーター

<i>c</i>	<i>c</i> は QC または $Q'C$ 、あるいは CQ' または CQ によって上書きされる。
<i>info</i>	<p>INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、<i>info</i> = -<i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。</p>

?ormr3/?unmr3

一般行列と `?tzzrf` で求めた RZ 因子分解の直交/ユニタリー行列とを乗算する (非ブロック化アルゴリズム)。

構文

```
call sormr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call dormr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call cunmr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
call zunmr3( side, trans, m, n, k, l, a, lda, tau, c, ldc, work, info )
```

説明

ルーチン `?ormr3/?unmr3` は一般実数 / 複素 $m \times n$ 行列 C を以下で上書きする。

$side = 'L'$ かつ $trans = 'N'$ の場合、 Q^*C 、
 $side = 'L'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 Q^*C 、

$side = 'R'$ かつ $trans = 'N'$ の場合、 CQ 、
 $side = 'R'$ かつ、 $trans = 'T'$ (実数型) または $trans = 'c'$ (複素型) の場合、 CQ

Q は、 k 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(1)H(2)\dots H(k)$$

`?tzzrf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	CHARACTER*1。 = 'L' の場合、 Q または Q' を左から適用する。 = 'R' の場合、 Q または Q' を右から適用する。
<i>trans</i>	CHARACTER*1。 = 'N' の場合、 Q を適用する (転置なし) = 'T' の場合、 Q' を適用する (転置、実数型の場合) = 'c' の場合、 Q' (共役転置、複素数型の場合)
<i>m</i>	INTEGER。 行列 C の行数。 $m \geq 0$ 。
<i>n</i>	INTEGER。 行列 C の列数。 $n \geq 0$ 。
<i>k</i>	INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。

<i>l</i>	INTEGER。 Householder リフレクターとして意味を持った部分が格納されている行列 <i>A</i> の列数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$ 。
<i>a</i>	REAL (<i>sormr3</i> の場合) DOUBLE PRECISION (<i>dormr3</i> の場合) COMPLEX (<i>cunmr3</i> の場合) COMPLEX*16 (<i>zunmr3</i> の場合) 配列、次元は <i>side</i> = 'L' の場合 (<i>lda</i> , <i>m</i>)、 <i>side</i> = 'R' の場合は (<i>lda</i> , <i>n</i>)。 <i>a</i> の <i>i</i> 番目の行には、 <i>?tzrzf</i> によって配列引数 <i>a</i> の最後の <i>k</i> 行に返されたとおりに、基 { リフレクター <i>H(i)</i> } を定義するベクトルを格納しておかなければならない。ここで、 $i = 1, 2, \dots, k$ 。配列 <i>a</i> はルーチンにより変更されるが終了時に復元される。
<i>lda</i>	INTEGER。 配列 <i>A</i> のリーディング・ディメンジョン。 $lda \geq \max(1, k)$
<i>tau</i>	REAL (<i>sormr3</i> の場合) DOUBLE PRECISION (<i>dormr3</i> の場合) COMPLEX (<i>cunmr3</i> の場合) COMPLEX*16 (<i>zunmr3</i> の場合) 配列、次元は (<i>k</i>)。 <i>tau(i)</i> には、 <i>?tzrzf</i> から返されたとおりに、基本リフレクター <i>H(i)</i> のスカラー係数を格納しておかなければならない。
<i>c</i>	REAL (<i>sormr3</i> の場合) DOUBLE PRECISION (<i>dormr3</i> の場合) COMPLEX (<i>cunmr3</i> の場合) COMPLEX*16 (<i>zunmr3</i> の場合) 配列、次元は (<i>ldc</i> , <i>n</i>)。 <i>m</i> × <i>n</i> の行列 <i>C</i> を格納する。
<i>ldc</i>	INTEGER。 配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$
<i>work</i>	REAL (<i>sormr3</i> の場合) DOUBLE PRECISION (<i>dormr3</i> の場合) COMPLEX (<i>cunmr3</i> の場合) COMPLEX*16 (<i>zunmr3</i> の場合) ワークスペース配列、次元は <i>side</i> = 'L' の場合 (<i>n</i>) <i>side</i> = 'R' の場合 (<i>m</i>)

出力パラメーター

<i>c</i>	<i>c</i> は <i>QC</i> または <i>Q'C</i> 、あるいは <i>CQ'</i> または <i>CQ</i> によって上書きされる。
----------	--

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。

?pbt2f2

対称 / エルミート 正定値帯行列のコレスキー因子分解を計算する (非ブロック化アルゴリズム)。

構文

```
call spbt2f2( uplo, n, kd, ab, ldab, info )
call dpbt2f2( uplo, n, kd, ab, ldab, info )
call cpbt2f2( uplo, n, kd, ab, ldab, info )
call zpbt2f2( uplo, n, kd, ab, ldab, info )
```

説明

このルーチンは、実対称 / 複素エルミート正定値帯行列 A に対してコレスキー因子分解を行う。因子分解の形式は次のとおりである。

$A = U^T U$ 、*uplo* = 'U' の場合、または
 $A = L L^T$ 、*uplo* = 'L' の場合

U は上三角行列、 U^T は U の転置、 L は下三角である。
 このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。

入力パラメーター

uplo CHARACTER*1。
 対称 / エルミート行列 A の上三角または下三角部分のどちらを格納するか指定する。
 = 'U' の場合、上三角
 = 'L' の場合、下三角。

n INTEGER。
 行列 A の次数。 $n \geq 0$ 。

kd INTEGER。
uplo = 'U' の場合は行列 A の優対角成分の個数、*uplo* = 'L' の場合は行列 A の劣対角成分の個数。
 $kd \geq 0$

ab REAL (spbt2f2 の場合)
 DOUBLE PRECISION (dpbt2f2 の場合)
 COMPLEX (cpbt2f2 の場合)
 COMPLEX*16 (zpbt2f2 の場合)
 配列、次元は (*ldab*, *n*)。
 対称 / エルミート帯行列 A の上三角または下三角を、配列の最初の $kd+1$ 行に格納する。 A の j 番目の列を配列 *ab* の j 番目の列

入力パラメーター

<i>uplo</i>	CHARACTER*1。 対称/エルミート行列 A の上三角または下三角部分のどちらを格納するか指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 行列 A の次数。 $n \geq 0$ 。
<i>a</i>	REAL (spotf2 の場合) DOUBLE PRECISION (dpotf2 の場合) COMPLEX (cpotf2 の場合) COMPLEX*16 (zpotf2 の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 対称/エルミート行列 A を格納する。 <i>uplo</i> = 'U' の場合、 <i>a</i> の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を格納する。 <i>a</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>a</i> の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 <i>a</i> の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	<i>info</i> = 0 の場合、コレスキー因子分解 $A = U^H U$ または $A = L L^H$ により得られた係数 U または L で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。 > 0 の場合、 <i>info</i> = <i>k</i> は次数 <i>k</i> の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

?ptts2

?pttrf で計算した L 、 D 、 L^H 因子分解を用いて、形式 $AX = B$ の三重対角連立方程式を解く。

構文

```
call sptts2( n, nrhs, d, e, b, ldb )
call dptts2( n, nrhs, d, e, b, ldb )
call cptts2( iuplo, n, nrhs, d, e, b, ldb )
call zptts2( iuplo, n, nrhs, d, e, b, ldb )
```

説明

ルーチン `?ptts2` は次の形式の三重対角連立方程式を解く。

$$AX = B$$

実数型の `sptts2/dptts2` は [spttrf/dpttrf](#) で計算された A の LDL' 因子分解を使用する。複素数型の `cptts2/zptts2` は [cpttrf/zpttrf](#) で計算された A の $U'DU$ または LDL' 因子分解を使用する。

D はベクトル d で指定される対角行列、 U (または L) はその優対角成分 (劣対角成分) がベクトル e で指定されている単位二重対角行列、 X と B は $n \times nrhs$ の行列である。

入力パラメーター

<code>iuplo</code>	INTEGER。複素型でのみ使用される。 因子分解の形式と、ベクトル e が上二重単位対角係数 U の優対角成分か下二重単位対角係数 L の劣対角成分のどちらであるかを指定する。 = 1 の場合、 $A = U'DU$ 、 e は U の優対角成分。 = 0 の場合、 $A = LDL'$ 、 e は L の劣対角成分。
<code>n</code>	INTEGER。 三重対角行列 A の次数。 $n \geq 0$ 。
<code>nrhs</code>	INTEGER。 右辺の数、すなわち、行列 B の列数。 $nrhs \geq 0$ 。
<code>d</code>	REAL (<code>sptts2/cptts2</code> の場合) DOUBLE PRECISION (<code>dptts2/zptts2</code> の場合) 配列、次元は (n) 。 A の因子分解で得られる対角行列 D の n 対角成分
<code>e</code>	REAL (<code>sptts2</code> の場合) DOUBLE PRECISION (<code>dptts2</code> の場合) COMPLEX (<code>cptts2</code> の場合) COMPLEX*16 (<code>zptts2</code> の場合) 配列、次元は $(n-1)$ 。 A の LDL' 因子分解から得られる単位二重対角係数 L の $(n-1)$ 個の劣対角成分を格納する (実数型、また <code>iuplo = 0</code> の場合は複素型)。 複素型において <code>iuplo = 1</code> の場合、 e には因子分解 $A = U'DU$ から得られる単位二重対角係数 U の $(n-1)$ 個の優対角成分を格納する。
<code>b</code>	REAL (<code>sptts2/cptts2</code> の場合) DOUBLE PRECISION (<code>dptts2/zptts2</code> の場合) 配列、次元は $(ldb, nrhs)$ 。 連立線形方程式に対するベクトル B の右辺。
<code>ldb</code>	INTEGER。 配列 B のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<code>b</code>	解ベクトル X で上書きされる。
----------------	--------------------

?rscl

ベクトルに実スカラーの逆数を掛ける。

構文

```
call srscl( n, sa, sx, incx )
call drscl( n, sa, sx, incx )
call csrscl( n, SA, SX, INCX )
call zdrscl( n, SA, SX, INCX )
```

説明

ルーチン ?rscl は、 n 成分の実数 / 複素ベクトル x に実数スカラー $1/a$ を乗算する。最終結果 x/a がオーバーフローまたはアンダーフローを起こさない限り、演算はオーバーフローまたはアンダーフローを起こすことなく実行される。

入力パラメーター

n	INTEGER。 ベクトル x の成分の個数。
sa	REAL (srscl/csrscl の場合) DOUBLE PRECISION (drscl/zdrscl の場合) ベクトル x の各成分の除算に使用されるスカラー a 。 sa は ≥ 0 でなければならない。 そうしないとサブルーチンでゼロの除算が発生する。
sx	REAL (srscl の場合) DOUBLE PRECISION (drscl の場合) COMPLEX (csrscl の場合) COMPLEX*16 (zdrscl の場合) 配列、次元は $(1+(n-1)*abs(incx))$ 。 ベクトル x の n 成分。
$incx$	INTEGER。 ベクトル sx の連続する値の間の増加分。 $incx > 0$ の場合、 $sx(1) = x(1)$ 、 $1 < i \leq n$ に対して $sx(1+(i-1)*incx) = x(i)$

出力パラメーター

sx	結果 x/a で上書きされる。
------	-------------------

?sygs2/?hegs2

?potrf で得られた因子分解の結果を用いて、対称/エルミート汎用固有値問題を標準形式に縮退させる(非ブロック化アルゴリズム)。

構文

```
call ssygs2( itype, uplo, n, a, lda, b, ldb, info )
call dsygs2( itype, uplo, n, a, lda, b, ldb, info )
call chs2( itype, uplo, n, a, lda, b, ldb, info )
call zhegs2( itype, uplo, n, a, lda, b, ldb, info )
```

説明

ルーチン ?sygs2/?hegs2 は、実数対称または複素エルミートの汎用固有値問題を標準化形式に縮退させる。

itype = 1 の場合、問題は

$$Ax = \lambda Bx$$

となり、 A は $\text{inv}(U)*A*\text{inv}(U)$ または $\text{inv}(L)*A*\text{inv}(L)$ で上書きされる。

itype = 2 または 3 の場合、問題は

$$ABx = \lambda x \text{ または } B Ax = \lambda x$$

となり、 A は UAU' または $L'AL$ で上書きされる。 B は、 $U'U$ または LL' として、[?potrf](#) によって事前に因子分解されていないなければならない。

入力パラメーター

itype	INTEGER。 = 1 の場合、 $\text{inv}(U)*A*\text{inv}(U)$ または $\text{inv}(L)*A*\text{inv}(L)$ を計算する。 = 2 または 3 の場合、 UAU' または $L'AL$ を計算する。
uplo	CHARACTER 対称/エルミート行列 A の上三角部分または下三角部分のどちらが格納されているか、および B がどのように因子分解されたかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
n	INTEGER。 行列 A と B の次数。 $n \geq 0$ 。
a	REAL (ssygs2 の場合) DOUBLE PRECISION (dsygs2 の場合) COMPLEX (chs2 の場合) COMPLEX*16 (zhegs2 の場合) 配列、次元は (lda, n)。 対称/エルミート行列 A を格納する。 uplo = 'U' の場合、 A の先頭の $n \times n$ 上三角部分に行列 a の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 uplo = 'L' の場合、 a の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 a の厳密な上三角部分は参照されない。

<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1,n)$ 。
<i>b</i>	REAL (<i>ssygs2</i> の場合) DOUBLE PRECISION (<i>dsygs2</i> の場合) COMPLEX (<i>chegs2</i> の場合) COMPLEX*16 (<i>zhegs2</i> の場合) 配列、次元は (<i>ldb</i> , <i>n</i>)。 <i>?potrf</i> によって返された、 <i>B</i> のコレスキー因子分解で得られた三角係数。
<i>ldb</i>	INTEGER。 配列 <i>B</i> のリーディング・ディメンジョン。 $ldb \geq \max(1,n)$ 。

出力パラメーター

<i>a</i>	<i>info</i> = 0 の場合、変換後の行列が <i>A</i> と同じ形式で格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。

?sytd2/?hetd2

直交/ユニタリー相似変換を用いて、対称/エルミート行列を実数対称三重対角形式に縮退させる (非ブロック化アルゴリズム)。

構文

```
call ssytd2( uplo, n, a, lda, d, e, tau, info )
call dsytd2( uplo, n, a, lda, d, e, tau, info )
call chetd2( uplo, n, a, lda, d, e, tau, info )
call zhetd2( uplo, n, a, lda, d, e, tau, info )
```

説明

ルーチン *?sytd2/?hetd2* は、直交/ユニタリー相似変換 $Q' A Q = T$ を用いて、実数対称/複素エルミート行列 *A* を実数対称三重対角形式 *T* に縮退させる。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 対称/エルミート行列 <i>A</i> の上三角または下三角部分のどちらを格納するか指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$ 。

<i>a</i>	<p>REAL (ssytd2 の場合) DOUBLE PRECISION (dsytd2 の場合) COMPLEX (chetd2 の場合) COMPLEX*16 (zhetd2 の場合) 配列、次元は (<i>lda</i>, <i>n</i>)。 対称 / エルミート行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、<i>A</i> の先頭の $n \times n$ 上三角部分に行列 <i>a</i> の上三角部分を格納する。<i>a</i> の厳密な下三角部分は参照されない。<i>uplo</i> = 'L' の場合、<i>a</i> の先頭の $n \times n$ 下三角部分に行列 <i>A</i> の下三角部分を格納する。<i>a</i> の厳密な上三角部分は参照されない。</p>
<i>lda</i>	<p>INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。$lda \geq \max(1, n)$。</p>

出力パラメーター

<i>a</i>	<p><i>uplo</i> = 'U' の場合、<i>a</i> の対角成分と最初の優対角成分は三重対角行列 <i>T</i> の対応する成分で上書きされる。最初の優対角成分より上の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交 / ユニタリー行列 <i>Q</i> を表現する。 <i>uplo</i> = 'L' の場合、<i>a</i> の対角成分と最初の劣対角成分は、三重対角行列 <i>T</i> の対応する成分で上書きされる。最初の劣対角成分より下の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交 / ユニタリー行列 <i>Q</i> を表現する。</p>
<i>d</i>	<p>REAL (ssytd2/chetd2 の場合) DOUBLE PRECISION (dsytd2/zhetd2 の場合) 配列、次元は (<i>n</i>)。 三重対角行列 <i>T</i> の対角成分。 $d(i) = a(i, i)$。</p>
<i>e</i>	<p>REAL (ssytd2/chetd2 の場合) DOUBLE PRECISION (dsytd2/zhetd2 の場合) 配列、次元は (<i>n</i>-1)。 三重対角行列 <i>T</i> の非対角成分。 <i>uplo</i> = 'U' の場合、$e(i) = a(i, i+1)$、 <i>uplo</i> = 'L' の場合、$e(i) = a(i+1, i)$。</p>
<i>tau</i>	<p>REAL (ssytd2 の場合) DOUBLE PRECISION (dsytd2 の場合) COMPLEX (chetd2 の場合) COMPLEX*16 (zhetd2 の場合) 配列、次元は (<i>n</i>-1)。 基本リフレクターのスカラー係数。</p>
<i>info</i>	<p>INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、<i>info</i> = -<i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。</p>

?sytf2

対角ピボット演算法を使用して、実数 / 複素不定値対称行列の因子分解を計算する (非ブロック化アルゴリズム)。

構文

```
call ssytf2( uplo, n, a, lda, ipiv, info )
call dsytf2( uplo, n, a, lda, ipiv, info )
call csytf2( uplo, n, a, lda, ipiv, info )
call zsytf2( uplo, n, a, lda, ipiv, info )
```

説明

ルーチン ?sytf2 は、Bunch-Kaufman 対角ピボット演算法を使用して、実数 / 複素不定値対称行列 A の因子分解を計算する。

$$A = U D U' \text{ または } A = L D L'$$

U (または L) は置換行列と単位上 (下) 三角行列の積、 U' は U の転置、 D は 1×1 と 2×2 の対角ブロックを持つ対称ブロック対角行列である。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。

入力パラメーター

<code>uplo</code>	CHARACTER*1。 対称行列 A の上三角部分または下三角部分のどちらが格納されるかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<code>n</code>	INTEGER。 行列 A の次数。 $n \geq 0$ 。
<code>a</code>	REAL (ssytf2 の場合) DOUBLE PRECISION (dsytf2 の場合) COMPLEX (csytf2 の場合) COMPLEX*16 (zsytf2 の場合) 配列、次元は (lda, n) 。 対称行列 A を格納する。 <code>uplo</code> = 'U' の場合、 A の先頭の $n \times n$ 上三角部分に行列 a の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 <code>uplo</code> = 'L' の場合、 a の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 a の厳密な上三角部分は参照されない。
<code>lda</code>	INTEGER。 配列 a のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	ブロック対角行列 D と、係数 U または L を得るために使用された乗数が上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は (n) 。 交換の結果と D のブロック構造の各成分が格納される。 $ipiv(k) > 0$ の場合、 k 番目の行と列と $ipiv(k)$ 番目の行と列は交換された。 $D(k, k)$ は 1×1 の対角ブロックである。 $uplo = 'U'$ かつ $ipiv(k) = ipiv(k-1) < 0$ の場合、 $k-1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k-1:k, k-1:k)$ は 2×2 の対角ブロックである。 $uplo = 'L'$ かつ $ipiv(k) = ipiv(k+1) < 0$ の場合、 $k+1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k:k+1, k:k+1)$ は 2×2 の対角ブロックである。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -k$ は k 番目の引数の値が不正だったことを示す。 > 0 の場合、 $info = k$ ならば $D(k, k)$ は完全にゼロである。因子分解は完了したが、ブロック対角行列 D は完全に特異で、連立方程式の解の算出に D を使用すると 0 での除算が発生する。

?hetf2

対角ピボット演算法を使用して、複素エルミート行列の因子分解を計算する (非ブロック化アルゴリズム)。

構文

```
call chetf2( uplo, n, a, lda, ipiv, info )
call zhetf2( uplo, n, a, lda, ipiv, info )
```

説明

このルーチンは、**Bunch-Kaufman** 対角ピボット演算法を使用して、複素エルミート行列 A の因子分解を計算する。

$$A = U D U' \text{ または } A = L D L'$$

U (または L) は置換行列と単位上 (下) 三角行列の積、 U' は U の共役転置、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#)を呼び出す。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 エルミート行列 A の上三角部分と下三角部分のどちらが格納されるかを指定する。 = 'U' の場合、上三角 = 'L' の場合、下三角。
<i>n</i>	INTEGER。 行列 A の次数。 $n \geq 0$ 。
<i>a</i>	COMPLEX (chetf2 の場合) COMPLEX*16 (zhetf2 の場合)。 配列、次元は (lda, n)。 エルミート行列 A を格納する。 <i>uplo</i> = 'U' の場合、 A の先頭の $n \times n$ 上三角部分に行列 a の上三角部分を格納する。 a の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 a の先頭の $n \times n$ 下三角部分に行列 A の下三角部分を格納する。 a の厳密な上三角部分は参照されない。
<i>lda</i>	INTEGER。 配列 a のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	ブロック対角行列 D と、係数 U または L を得るために使用された乗数が上書きされる。
<i>ipiv</i>	INTEGER。 配列、次元は (n)。 交換の結果と D のブロック構造の各成分が格納される。 $ipiv(k) > 0$ の場合、 k 番目の行と列と $ipiv(k)$ 番目の行と列は交換された。 $D(k, k)$ は 1×1 の対角ブロックである。 $uplo = 'U'$ かつ $ipiv(k) = ipiv(k-1) < 0$ の場合、 $k-1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k-1:k, k-1:k)$ は 2×2 の対角ブロックである。 $uplo = 'L'$ かつ $ipiv(k) = ipiv(k+1) < 0$ の場合、 $k+1$ 番目の行と列と $-ipiv(k)$ 番目の行と列は交換された。 $D(k:k+1, k:k+1)$ は 2×2 の対角ブロックである。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 $info = -k$ は k 番目の引数の値が不正だったことを示す。 > 0 の場合、 $info = k$ ならば $D(k, k)$ は完全にゼロである。因子分解は完了したが、ブロック対角行列 D は完全に特異で、連立方程式の解の算出に D を使用すると 0 での除算が発生する。

?tgex2

直交/ユニタリー等価変換を使用して、(準)上三角行列のペア中の隣接対角ブロックを交換する。

構文

```
call stgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1, n2, work,
            lwork, info )
call dtgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, n1, n2, work,
            lwork, info )
call ctgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, info )
call ztgex2( wantq, wantz, n, a, lda, b, ldb, q, ldq, z, ldz, j1, info )
```

説明

実数ルーチン `stgex2/dtgex2` は、直交等価変換を用いて、(準)上三角行列のペア (A , B) の中の 1×1 の隣接対角ブロック (A_{11} , B_{11}) と 2×2 の隣接対角ブロック (A_{22} , B_{22}) を交換する。(A , B) は汎用実数 Schur 標準形 ([sqqges/dqgges](#) により返されたとおり) でなければならない。すなわち、 A は 1×1 と 2×2 の対角ブロックを持つ上三角ブロックである。 B は上三角である。

複素ルーチン `ctgex2/ztgex2` は、ユニタリー等価変換を用いて、上三角行列のペア (A , B) の中の 1×1 の隣接対角ブロック (A_{11} , B_{11}) と 2×2 の隣接対角ブロック (A_{22} , B_{22}) を交換する。(A , B) は汎用 Schur 標準形でなければならない。すなわち、 A と B はどちらも上三角である。

すべてのルーチンは、汎用 Schur ベクトルの Q と Z をオプションで更新する。

$$Q(\text{in}) * A(\text{in}) * Z(\text{in})' = Q(\text{out}) * A(\text{out}) * Z(\text{out})'$$

$$Q(\text{in}) * B(\text{in}) * Z(\text{in})' = Q(\text{out}) * B(\text{out}) * Z(\text{out})'$$

入力パラメーター

<code>wantq</code>	LOGICAL. <code>wantq = .TRUE.</code> の場合、左変換行列 Q を更新する。 <code>wantq = .FALSE.</code> の場合、 Q を更新しない。
<code>wantz</code>	LOGICAL。 <code>wantz = .TRUE.</code> の場合、右変換行列 Z を更新する。 <code>wantz = .FALSE.</code> の場合、 Z を更新しない。
<code>n</code>	INTEGER。 行列 A と B の次数。 $n \geq 0$ 。
<code>a, b</code>	REAL (<code>stgex2</code> の場合) DOUBLE PRECISION (<code>dtgex2</code> の場合) COMPLEX (<code>ctgex2</code> の場合) COMPLEX*16 (<code>ztgex2</code> の場合) 配列、次元はそれぞれ、(<code>lda, n</code>) および (<code>ldb, n</code>) ペア (A, B) となっている行列 A と B を格納する。

<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1,n)$ 。
<i>ldb</i>	INTEGER。 配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1,n)$ 。
<i>q, z</i>	REAL (stgex2 の場合) DOUBLE PRECISION (dtgex2 の場合) COMPLEX (ctgex2 の場合) COMPLEX*16 (ztgex2 の場合) 配列、次元はそれぞれ、(<i>ldq, n</i>) および (<i>ldz, n</i>) <i>wantq</i> = .TRUE. の場合は <i>q</i> に直交 / ユニタリー行列 <i>Q</i> を格納し、 <i>wantz</i> = .TRUE. の場合は <i>z</i> に直交 / ユニタリー行列 <i>Z</i> を格納する。
<i>ldq</i>	INTEGER。 配列 <i>q</i> のリーディング・ディメンジョン。 $ldq \geq 1$ <i>wantq</i> = .TRUE. の場合、 $ldq \geq n$
<i>ldz</i>	INTEGER。 配列 <i>z</i> のリーディング・ディメンジョン。 $ldz \geq 1$ <i>wantz</i> = .TRUE. の場合、 $ldz \geq n$
<i>j1</i>	INTEGER 最初のブロック (<i>A11, B11</i>) へのインデックス。 $1 \leq j1 \leq n$
<i>n1</i>	INTEGER。実数型でのみ使用される。 最初のブロック (<i>A11, B11</i>) の次数。 $n1 = 0, 1$ または 2
<i>n2</i>	INTEGER。実数型でのみ使用される。 2 番目のブロック (<i>A22, B22</i>) の次数。 $n2 = 0, 1$ または 2
<i>work</i>	REAL (stgex2 の場合) DOUBLE PRECISION (dtgex2 の場合) ワークスペース配列、次元は (<i>lwork</i>)。実数型でのみ使用される。
<i>lwork</i>	INTEGER 配列 <i>work</i> の次元。 $lwork \geq \max(n*(n2+n1), 2*(n2+n1)^2)$

出力パラメーター

<i>a</i>	更新された行列 <i>A</i> で上書きされる。
<i>b</i>	更新された行列 <i>B</i> で上書きされる。
<i>q</i>	更新された行列 <i>Q</i> で上書きされる。 <i>wantq</i> = .FALSE. の場合は参照されない。
<i>z</i>	更新された行列 <i>Z</i> で上書きされる。 <i>wantz</i> = .FALSE. の場合は参照されない。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 stgex2/dtgex2 の場合、 <i>info</i> = 1 の場合、変換された行列 (<i>A, B</i>) が汎用 Schur 形式からかけ離れていることを示す。ブロックは交

換されず、 (A, B) と (Q, Z) は変更されない。すなわち、この問題は悪条件である。 $info = -16$ の場合、 $lwork$ が小さすぎることを示す。 $lwork$ の適切な値は $work(1)$ に返される。

`ctgex2/ztgex2` の場合。

$info = 1$ の場合、変換後の行列ペア (A, B) が汎用 Schur 形式から遠すぎる。すなわち、この問題は悪条件である。 (A, B) は部分的に順序が変更されている可能性があり、 $ilst$ は移動対象ブロックの現在の位置の最初の行を指している。

?tgsy2

汎用シルベスター式を解く (非ブロック化アルゴリズム)。

構文

```
call stgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
call dtgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
call ctgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
call ztgsy2( trans, ijob, m, n, a, lda, b, ldb, c, ldc, d, ldd, e, lde, f, ldf,
            scale, rdsum, rdscal, iwork, pq, info )
```

説明

ルーチン `?tgsy2` は、レベル 1 BLAS とレベル 2 BLAS を使用して、次の汎用シルベスター式を解く。

$$\begin{aligned} AR - LB &= \text{scale} * C \\ DR - LE &= \text{scale} * F, \end{aligned} \quad (1)$$

R と L は未知の $m \times n$ 行列、 (A, D) 、 (B, E) 、

(C, F) はそれぞれ $m \times m$ 、 $n \times n$ 、および $m \times n$ の与えられた行列のペアである。

`stgsy2/dtgsy2` の場合、ペア (A, D) と (B, E) は汎用 Schur 標準形でなければならない。すなわち、 A と B は上準三角行列で、 D と E は上三角行列でなければならない。

`ctgsy2/ztgsy2` の場合、行列 A と B と D と E は上三角行列である (すなわち、 (A, D) と (B, E) は汎用 Schur 形)。

(C, F) は解 (R, L) で上書きされる。 $0 \leq \text{scale} \leq 1$ はオーバーフローを避けるために選択された出力スケール係数である。

行列表記では、方程式 (1) を解くことは次式を解くことに対応する。

$$Zx = \text{scale} * b$$

ただし、 Z は以下のように定義される。

$$Z = \begin{bmatrix} \text{kron}(I_n, A) & -\text{kron}(B', I_m) \\ \text{kron}(I_n, D) & -\text{kron}(E', I_m) \end{bmatrix} \quad (2)$$

I_k はサイズが k の単位行列、 X' は X の転置である。

$\text{kron}(X, Y)$ は行列 X と Y の Kronecker 積を表わす。

$\text{trans} = 'T'$ の場合、次の転置 (共役転置) 式を y について解く。

$$Zy = \text{scale} * b$$

これは、以下の式を R, L に対して解くことと等価である。

$$\begin{aligned} A'R + D'L &= \text{scale} * C \\ R'B' + L'E' &= \text{scale} * (-F) \end{aligned} \quad (3)$$

上の事例は、[?lacon](#) による逆コミュニケーションを使用した、

$\text{Dif}(A, D), (B, E) = \text{sigma_min}(Z)$ の推定の計算に用いられる。

?tgsy2 はまた ($i\text{job} \geq 1$ に対して)、?tgsy1 における 2 つの行列ペア間の隔たりの上限計算に影響する。入力 $(A, D), (B, e)$ は、?tgsy1 において行列ペアの部分行列束である。詳細は [?tgsyl](#) を参照のこと。

入力パラメーター

<i>trans</i>	CHARACTER <i>trans</i> = 'N' の場合、汎用シルベスター式 (1) を解く。 <i>trans</i> = 'T' の場合、転置された式 (3) を解く。
<i>ijob</i>	INTEGER。 実行する機能を指定する。 <i>ijob</i> = 0 の場合、(1) のみを解く。 <i>ijob</i> = 1 の場合、この部分式を始点として、2 個の行列ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (先読み法が用いられる)。 <i>ijob</i> = 2 の場合、この部分式を始点として、2 個の行列ペア間の隔たりの Frobenius ノルムにもとづく推定までの影響が計算される (部分式に対して ?gecon が用いられる)。 <i>trans</i> = 'T' の場合は参照されない。
<i>m</i>	INTEGER。 <i>m</i> は、 A と D の次数、および C, F, R, L の行次元を指定する。
<i>n</i>	INTEGER。 <i>n</i> は、 B と E の次数、および C, F, R, L の列次元を指定する。
<i>a, b</i>	REAL (<i>stgsy2</i> の場合) DOUBLE PRECISION (<i>dtgsy2</i> の場合) COMPLEX (<i>ctgsy2</i> の場合) COMPLEX*16 (<i>ztgsy2</i> の場合) 配列、次元はそれぞれ、(<i>lda, m</i>) および (<i>ldb, n</i>)。 <i>a</i> には上 (準) 三角行列 A を、 <i>b</i> には上 (準) 三角行列 B を格納する。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, m)$ 。
<i>ldb</i>	INTEGER。 配列 <i>b</i> のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

<i>c, f</i>	REAL (stgsy2 の場合) DOUBLE PRECISION (dtgsy2 の場合) COMPLEX (ctgsy2 の場合) COMPLEX*16 (ztgsy2 の場合) 配列、次元はそれぞれ、(<i>ldc, n</i>) および (<i>ldf, n</i>)。 <i>c</i> には (1) の最初の行列方程式の右辺を、 <i>f</i> には (1) の 2 番目の行列方程式の右辺を格納する。
<i>ldc</i>	INTEGER。 配列 <i>c</i> のリーディング・ディメンジョン。 $ldc \geq \max(1, m)$ 。
<i>d, e</i>	REAL (stgsy2 の場合) DOUBLE PRECISION (dtgsy2 の場合) COMPLEX (ctgsy2 の場合) COMPLEX*16 (ztgsy2 の場合) 配列、次元はそれぞれ、(<i>ldd, m</i>) および (<i>lde, n</i>)。 <i>d</i> には上三角行列 <i>D</i> を、 <i>e</i> には上三角行列 <i>E</i> を格納する。
<i>ldd</i>	INTEGER。 配列 <i>d</i> のリーディング・ディメンジョン。 $ldd \geq \max(1, m)$ 。
<i>lde</i>	INTEGER。 配列 <i>e</i> のリーディング・ディメンジョン。 $lde \geq \max(1, n)$ 。
<i>ldf</i>	INTEGER。 配列 <i>f</i> のリーディング・ディメンジョン。 $ldf \geq \max(1, m)$ 。
<i>rdsum</i>	REAL (stgsy2/ctgsy2 の場合) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合) ?tgsy1 で得られた Dif 推定に対する、計算された影響の二乗和を格納する。ここでスケール係数 <i>rdscal</i> は因子分解されている。
<i>rdscal</i>	REAL (stgsy2/ctgsy2 の場合) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合) <i>rdsum</i> のオーバーフローを防ぐために使用されるスケール係数を格納する。
<i>iwork</i>	INTEGER。実数型でのみ使用される。 ワークスペース配列、次元は ($m+n+2$)

出力パラメーター

<i>c</i>	<i>ijob</i> = 0 の場合、 <i>c</i> は解 <i>R</i> で上書きされる。
<i>f</i>	<i>ijob</i> = 0 の場合、 <i>F</i> は解 <i>L</i> で上書きされる。
<i>scale</i>	REAL (stgsy2/ctgsy2 の場合) DOUBLE PRECISION (dtgsy2/ztgsy2 の場合) $0 \leq scale \leq 1$ が格納される。 $0 < scale < 1$ の場合、解 <i>R</i> と解 <i>L</i> (入力時は <i>C</i> と <i>F</i>) にはそれぞれ、わずかに摂動する系の解が入るが、入力行列 <i>A</i> 、 <i>B</i> 、 <i>D</i> 、 <i>E</i> は変更されない。 <i>scale</i> = 0 の場合、 <i>R</i> と <i>L</i> にはそれぞれ <i>C</i> = <i>F</i> = 0 の均一な系の解が入る。通常は <i>scale</i> = 1 である。

<i>rdsum</i>	対応する二乗和は、現在の部分式から得られた影響によって更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdsum</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>rdscal</i>	<i>rdscal</i> は、 <i>rdsum</i> に格納されている現在の影響に関して更新される。 <i>trans</i> = 'T' の場合、 <i>rdsum</i> は使われない。 <i>rdscal</i> は ?tgsy2 が ?tgsy1 から呼び出されたときのみ意味を持つことに注意する。
<i>pq</i>	INTEGER。実数型でのみ使用される。 ルーチン <i>stgsy2/dtgsy2</i> で解かれた部分式 (2×2 、 4×4 、 8×8 のサイズ) の個数が格納される。
<i>info</i>	INTEGER。 <i>info</i> には次の値が設定される。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 > 0 の場合、行列のペア (<i>A</i> , <i>D</i>) と (<i>B</i> , <i>E</i>) は、同じ固有値か、きわめて近い固有値を持つ。

?trti2

三角行列の逆行列を計算する (非ブロック化アルゴリズム)。

構文

```
call strti2( uplo, diag, n, a, lda, info )
call dtrti2( uplo, diag, n, a, lda, info )
call ctrti2( uplo, diag, n, a, lda, info )
call ztrti2( uplo, diag, n, a, lda, info )
```

説明

ルーチン ?trti2 は実数 / 複素の上三角または下三角行列の逆行列を計算する。

このルーチンは、アルゴリズムの Level 2 BLAS バージョンである。

入力パラメーター

uplo CHARACTER*1。
行列 *A* が上三角か下三角かを指定する。
= 'U' の場合、上三角
= 'L' の場合、下三角。

<i>diag</i>	CHARACTER*1。 行列 <i>A</i> が単位三角かどうかを指定する。 = 'N' の場合、単位三角ではない。 = 'U' の場合、単位三角。
<i>n</i>	INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$ 。
<i>a</i>	REAL (<i>strti2</i> の場合) DOUBLE PRECISION (<i>dtrti2</i> の場合) COMPLEX (<i>ctrti2</i> の場合) COMPLEX*16 (<i>ztrti2</i> の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。 三角行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> の先頭の $n \times n$ 上三角部分に上三角行列を格納する。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> の先頭の $n \times n$ 下三角部分に下三角行列を格納する。厳密な上三角部は参照されない。 <i>diag</i> = 'U' の場合も対角成分は参照されず 1 とみなされる。
<i>lda</i>	INTEGER。 配列 <i>a</i> のリーディング・ディメンジョン。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	終了時に、元の行列の (三角) 逆行列で、同じ格納形式で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>k</i> は <i>k</i> 番目の引数の値が不正だったことを示す。

ユーティリティー関数とルーチン

このセクションでは、LAPACK ユーティリティー関数とルーチンについて説明する。次の表はこれらのルーチンについてまとめたものである。

表 5-2 LAPACK ユーティリティー・ルーチン

ルーチン名	データ型	説明
ilaenv		アルゴリズム性能のチューニング値を戻す環境問い合わせ関数。
ieeeck		無限大演算および NaN 演算が安全かどうかを確認する。ilaenv により呼び出される。
lsame		2 つの文字の同一性を大文字 / 小文字に関わらずテストする。
lsamen		2 つの文字列の同一性を大文字 / 小文字に関わらずテストする。
?labad	s, d	指数範囲がきわめて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。
?lamch	s, d	浮動小数点演算に対するマシン・パラメーターを決定する。
?lamc1	s, d	?lamc2 から呼び出される。beta, t, rnd, ieee1 で与えられるマシン・パラメーターを決定する。
?lamc2	s, d	?lamch で使用される。引数リストで指定されたマシン・パラメーターを決定する。

表 5-2 LAPACK ユーティリティー・ルーチン

ルーチン名	データ型	説明
?lamc3	s,d	?lamc1 ~ ?lamc5 から呼び出される。a と b の加算を実行する前に a と b を格納させる。
?lamc4	s,d	このルーチンは ?lamc2 のサービスルーチンである。
?lamc5	s,d	?lamc2 から呼び出される。オーバーフローしないマシンの最大浮動小数点値の計算を試みる。
second/dsecnd		プロセスのユーザー時間を返す。
xerbla		LAPACK ルーチンから呼び出されるエラー処理ルーチン。

ilaenv

アルゴリズム性能のチューニング値を戻す環境問い合わせ関数。

構文

```
value = ilaenv( ispec, name, opts, n1, n2, n3, n4 )
```

説明

問い合わせ関数 `ilaenv` はローカル環境に対する問題依存パラメーターを選択するために LAPACK ルーチンから呼び出される。パラメーターの説明は `ispec` を参照のこと。

このバージョンは、現在の多くのコンピューター上で最適ではないが、良好な性能を得る一連のパラメーターを与える。ユーザーは、引数中のオプションと問題サイズ情報を使って、使用対象のマシンに対するチューニング・パラメーターを設定するために、このサブルーチンを変更することが奨励される。

このルーチンはすべて小文字にすると正しく機能しない。すべて大文字で使用することは許される。

入力パラメーター

`ispec` INTEGER。ilaenv の戻り値として得たいパラメーターを指定する。

- = 1: 最適ブロックサイズ。戻り値が 1 のとき、非ブロック化アルゴリズムで最適な性能が得られる。
- = 2: ブロックルーチンを使用すべき最小ブロックサイズ。使用できるブロックサイズがこの値未満の場合は、非ブロック化ルーチンを使用すべきである。
- = 3: クロスオーバー点 (ブロックルーチンでは、この値未満の N に対して、非ブロック化ルーチンを使用すべきである)。
- = 4: シフト数。非対称の固有値ルーチンで使用される。

= 5: ブロック化が使用される最小の列次元。

矩形ブロックは $k \times m$ 以上の次元を持っていなければならない、ここで k は `ilaenv(2, ...)` で与えられ、 m は `ilaenv(5, ...)` で与えられる。

= 6: SVD に対するクロスオーバー点 ($m \times n$ の行列を二重対角形式に縮退する場合、 $\max(m, n)/\min(m, n)$ がこの値を超えたなら、行列を三角形式に縮退するために最初に *QR* 因子分解が使用される)。

= 7: プロセッサの個数。

= 8: 非対称固有値問題に対するマルチ。シフト *QR* および *QZ* 法のクロスオーバー点

= 9: 分割統治アルゴリズムにおける計算ツリーの一番下にある部分問題の最大サイズ (`?gelsd` と `?gesdd` で使用する)。

= 10: トラップを生じないと信頼される IEEE NaN 演算。

= 11: トラップを生じないと信頼される無限大演算。

<i>name</i>	CHARACTER*(*)。呼び出しサブルーチンの名称で、大文字でも小文字でもよい。
<i>opts</i>	CHARACTER*(*)。サブルーチン <i>name</i> に対する文字オプションを単一文字列に連結したもの。例えば、 <code>uplo = 'U'</code> 、 <code>trans = 'T'</code> 、 <code>diag = 'N'</code> の三角ルーチンは <code>opts = 'UTN'</code> として指定してもよい。
<i>n1, n2, n3, n4</i>	INTEGER。サブルーチン <i>name</i> に対する問題の次元。すべてのパラメーターが必要でない場合もある。

出力パラメーター

<i>value</i>	INTEGER。 <i>value</i> ≥ 0 の場合、 <i>ispec</i> で指定されたパラメーター値が返される。 <i>value</i> = $-k < 0$ の場合、 k 番目の引数が不正な値であることを示す。
--------------	---

アプリケーション・ノート

LAPACK ルーチンから `ilaenv` を呼び出すときには以下の規則が使用される。

- opts* は、*ispec* で指定したパラメーター値を求めるために使用しない場合でも、*name* の引数リストと同じ順番でサブルーチン *name* のすべての文字オプションを連結したものである。
- 問題次元 *n1, n2, n3, n4* は *name* の引数リストと同じ順番で指定する。*n1* が最初に使用され、次に *n2* が使われ、以下同様に続く。使用されない問題次元には値 -1 が渡される。
- `ilaenv` が戻すパラメーター値は呼び出しサブルーチン内で有効性をチェックする。例えば、`strtri` の最適ブロックサイズを検索するには `ilaenv` を次のように使用する。

```
nb = ilaenv( 1, 'strtri', uplo // diag, n, -1, -1, -1 )
if( nb.le.1 ) nb = max( 1, n )
```

以下は、C コードでの `ilaenv` の使用例である。

例 5-1 C コードでの ILAENV 関数の使用例

```
#include <stdio.h>
#include "mkl.h"

int main(void)
{
    int size = 1000;
    int ispec = 1;
    int dummy = -1;
    int blockSize1 = ilaenv(&ispec, "dsytrd", "U", &size, &dummy, &dummy, &dummy);
    int blockSize2 = ilaenv(&ispec, "dormtr", "LUN", &size, &size, &dummy, &dummy);
    printf("DSYTRD blocksize = %d\n", blockSize1);
    printf("DORMTR blocksize = %d\n", blockSize2);
    return 0;
}
```

ieeack

無限大演算および NaN 演算が安全かどうかを確認する。

`ilaenv` により呼び出される。

構文

```
ival = ieeack( ispec, zero, one )
```

説明

関数の `ieeack` は、[ilaenv](#) により呼び出され、無限大および、おそらく NaN 演算が安全であること (トラップが生じない) を確認する。

入力パラメーター

<code>ispec</code>	INTEGER。無限大演算のみ、または無限大演算と NaN 演算の両方を検証するかどうかを指定する。 <code>ispec = 0</code> の場合、無限大演算のみを確認する。 <code>ispec = 1</code> の場合、無限大演算と NaN 演算を確認する。
<code>zero</code>	REAL。値 0.0 を含まなければならない。 これはこのコードがコンパイラーによって最適化されないように渡される。
<code>one</code>	REAL。値 1.0 を含まなければならない。 これはこのコードがコンパイラーによって最適化されないように渡される。

出力値

<code>ndimr</code>	INTEGER。 <code>ival = 0</code> の場合、演算は失敗し、正確な結果を得られなかったを示す。 <code>ival = 1</code> の場合、演算により正確な結果が得られたことを示す。
--------------------	--

Isame

2つの文字の同一性を大文字/小文字に関わらずテストする。

構文

```
val = lsame( ca, cb )
```

説明

この論理関数は、*ca* と *cb* が大文字/小文字に関わらず同一の文字のときに `.TRUE.` を返す。

入力パラメーター

ca, cb CHARACTER*1。比較する単一文字を指定する。

出力パラメーター

val LOGICAL。比較結果を示す。

Isamen

2つの文字列の同一性を大文字/小文字に関わらずテストする。

構文

```
val = lsamen( n, ca, cb )
```

説明

この論理関数は、文字列 *ca* の最初の *n* 文字と文字列 *cb* の最初の *n* 文字が大文字/小文字に関わらず、等しいかどうかをテストする。`lsamen` 関数は大文字/小文字に関わらず、*ca* と *cb* が等価のときに `.TRUE.` を返し、それ以外では `.FALSE.` を返す。`len(ca)` または `len(cb)` が *n* 未満のときにも、`lsamen` は `.FALSE.` を返す。

入力パラメーター

n INTEGER。 *ca* と *cb* を比較する文字数。

ca, cb CHARACTER*(*)。長さ *n* 以上の2つの比較する文字列を指定する。それぞれの文字列は先頭の *n* 文字のみアクセスされる。

出力パラメーター

val LOGICAL。比較結果を示す。

?labad

指数範囲がきわめて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。

構文

```
call slabad( small, large )  
call dlabad( small, large )
```

説明

このルーチンは、アンダーフローとオーバーフローに対して [?lamch](#) によって計算された値を入力として受け取り、*large* のログが十分に大きい場合にそれら値のそれぞれの平方根を返す。このサブルーチンは、Cray のような大きな指数範囲を持つマシンを判別するために使用し、アンダーフローとオーバーフローのリミット値を [?lamch](#) で計算された値の平方根に再定義する。Cray に見られるような上半分の指数範囲における計算能力が弱い場合でも [?lamch](#) は補正を行わないため、このサブルーチンが必要となる。

入力パラメーター

<i>small</i>	REAL (<i>slabad</i> の場合) DOUBLE PRECISION (<i>dlabad</i> の場合) ?lamch によって計算されたアンダーフローしきい値。
<i>large</i>	REAL (<i>slabad</i> の場合) DOUBLE PRECISION (<i>dlabad</i> の場合) ?lamch によって計算されたオーバーフローしきい値。

出力パラメーター

<i>small</i>	$\log_{10}(\textit{large})$ が十分に大きい場合は <i>small</i> の平方根で上書きされ、そうでない場合は変更されない。
<i>large</i>	$\log_{10}(\textit{large})$ が十分に大きい場合は <i>large</i> の平方根で上書きされ、そうでない場合は変更されない。

?lamch

浮動小数点演算に対するマシン・パラメーターを決定する。

構文

```
val = slamch( cmach )  
val = dlamch( cmach )
```

説明

関数 [?lamch](#) は、単精度および倍精度のマシン・パラメーターを決定する。

入力パラメーター

cmach CHARACTER*1。?lamch が返す値を指定する。

= 'E' または 'e', *val* = *eps*
 = 'S' または 's', *val* = *sfmin*
 = 'B' または 'b', *val* = *base*
 = 'P' または 'p', *val* = *eps*base*
 = 'N' または 'n', *val* = *t*
 = 'R' または 'r', *val* = *rnd*
 = 'M' または 'm', *val* = *emin*
 = 'U' または 'u', *val* = *rmin*
 = 'L' または 'l', *val* = *emax*
 = 'O' または 'o', *val* = *rmax*

ここで、
eps = 相対マシン精度。
sfmin = $1/sfmin$ がオーバーフローしないような安全な最小値。
base = マシンの基数。
prec = $eps*base$
t = 仮数における (基数) 桁数。
rnd = 丸めが追加で発生した場合 1.0、それ以外では 0.0。
emin = (緩やかに) アンダーフローを起こす前の最小指数。
rmin = $underflow_threshold - base^{*(emin-1)}$
emax = オーバーフローを起こす前の最大指数
rmax = $overflow_threshold - (base^{*emax})*(1-eps)$

出力パラメーター

val REAL (*slamch* の場合)
 DOUBLE PRECISION (*diamch* の場合)
 関数の戻り値。

?lamc1

?lamc2 から呼び出される。
beta, *t*, *rnd*, *ieee1* で与えられるマシン・パラメーターを決定する。

構文

```
call slamc1( beta, t, rnd, ieee1 )
call dlamc1( beta, t, rnd, ieee1 )
```

説明

ルーチン ?lamc1 は *beta*, *t*, *rnd*, *ieee1* で与えられるマシン・パラメーターを決定する。

出力パラメーター

beta INTEGER。マシンの基数。

<i>t</i>	INTEGER。仮数における (<i>beta</i>) 桁数。
<i>rnd</i>	LOGICAL。 適切な丸め (<i>rnd</i> = .TRUE.) または切捨て (<i>rnd</i> = .FALSE.) が追加で発生したかを示す。 マシンの演算性能を決める信頼性のある指標ではない。
<i>ieee1</i>	LOGICAL。 <i>ieee</i> の「round to nearest」形式で丸めが発生したかどうかを示す。

?lamc2

?lamch で使用される。
引数リストで指定されたマシン・パラメーターを決定する。

構文

```
call slamc2( beta, t, rnd, eps, emin, rmin, emax, rmax )
call dlamc2( beta, t, rnd, eps, emin, rmin, emax, rmax )
```

説明

ルーチン ?lamc2 は引数リストで指定されたマシン・パラメーターを決定する。

出力パラメーター

<i>beta</i>	INTEGER。マシンの基数。
<i>t</i>	INTEGER。仮数における (<i>beta</i>) 桁数。
<i>rnd</i>	LOGICAL。 適切な丸め (<i>rnd</i> = .TRUE.) または切捨て (<i>rnd</i> = .FALSE.) が追加で発生したかを示す。 マシンの演算性能を決める信頼性のある指標ではない。
<i>eps</i>	REAL (slamc2 の場合) DOUBLE PRECISION (dlamc2 の場合) 以下を満たすような最小の正の値。 $f1(1.0 - eps) < 1.0$ 、 <i>f1</i> は計算された値を示す。
<i>emin</i>	INTEGER。(緩やかに) アンダーフローを起こす前の最小指数。
<i>rmin</i>	REAL (slamc2 の場合) DOUBLE PRECISION (dlamc2 の場合) $base^{emin-1}$ で与えられ、 <i>base</i> は <i>beta</i> の浮動小数点値である。
<i>emax</i>	INTEGER。オーバーフローを起こす前の最大指数。

`rmax` REAL (`slamc2` の場合)
DOUBLE PRECISION (`dlamc2` の場合)
マシンに対する最大の正の数で、 $base^{emax(1-eps)}$ で与えられ、`base` は `beta` の浮動小数点値である。

?lamc3

?lamc1 ~ ?lamc5 から呼び出される。 `a` と `b` の加算を実行する前に `a` と `b` を格納させる。

構文

```
val = slamc3( a, b )
```

```
val = dlamc3( a, b )
```

説明

このルーチンは、`a` と `b` の加算を実行する前に `a` と `b` を格納させる。最適マイザーがこれらの値をレジスターに保持している場合に使用する。

入力パラメーター

`a, b` REAL (`slamc3` の場合)
DOUBLE PRECISION (`dlamc3` の場合)
`a` と `b` の値。

出力パラメーター

`val` REAL (`slamc3` の場合)
DOUBLE PRECISION (`dlamc3` の場合)
`a` と `b` を加算した結果。

?lamc4

?lamc2 のサービスルーチンである。

構文

```
call slamc4( emin, start, base )
```

```
call dlamc4( emin, start, base )
```

説明

このルーチンは [?lamc2](#) のサービスルーチンである。

入力パラメーター

<i>start</i>	REAL (slamc4 の場合) DOUBLE PRECISION (dlamc4 の場合) <i>emin</i> を決定する開始点。
<i>base</i>	INTEGER。マシンの基数。

出力パラメーター

<i>emin</i>	INTEGER。(緩やかに)アンダーフローを起こす前の最小指数で、 <i>a = start</i> と設定し、前の <i>a</i> が回復できなくなるまで基数による除算を行って計算される。
-------------	--

?lamc5

?lamc2 から呼び出される。
オーバーフローしないマシンの最大浮動小数点値
の計算を試みる。

構文

```
call slamc5( beta, p, emin, ieee, emax, rmax )
call dlamc5( beta, p, emin, ieee, emax, rmax )
```

説明

ルーチン ?lamc5 は、オーバーフローしない最大マシン浮動小数点値 *rmax* の計算を試みる。*emax + abs(emin)* の合計はおよそ 2 のべき乗であると仮定する。

例えば、Cyber 205 (*emin* = -28625, *emax* = 28718) のように、この仮定が満たされない場合、マシンでの実行は失敗する。*emin* にオーバーフローするであろう大きすぎる値 (すなわちゼロに近い) を与えても同様にマシンでの実行は失敗する。

入力パラメーター

<i>beta</i>	INTEGER。浮動小数点演算の基数。
<i>p</i>	INTEGER。浮動小数点演算の仮数における基数 <i>beta</i> の桁数。
<i>emin</i>	INTEGER。(緩やかに)アンダーフローを起こす前の最小指数。
<i>ieeee</i>	LOGICAL。演算システムが IEEE スタandard に準拠していると考えられるかどうかを示す論理フラグ。

出力パラメーター

<i>emax</i>	INTEGER。オーバーフローを起こす前の最大指数。
<i>rmax</i>	REAL (slamc5 の場合) DOUBLE PRECISION (dlamc5 の場合) 最大マシン浮動小数点値。

second/dsecnd

プロセスのユーザー時間を返す。

構文

```
val = second()
```

```
val = dsecnd()
```

説明

関数 `second/dsecnd` はプロセスのユーザー時間 (秒単位) を返す。これらの関数のバージョンはシステム関数の `etime` から時間を取得する。相違点は `dsecnd` が結果を倍精度で返すことである。

出力パラメーター

<code>val</code>	REAL (<code>second</code> の場合) DOUBLE PRECISION (<code>dsecnd</code> の場合) プロセスのユーザー時間。
------------------	--

xerbla

BLAS ルーチン、*LAPACK* ルーチン、*VML* ルーチンから呼び出されるエラー処理ルーチン。

構文

```
call xerbla( sname, info )
```

説明

ルーチン `xerbla` は *BLAS* ルーチン、*LAPACK* ルーチン、*VML* ルーチンのエラー処理ルーチンである。*BLAS* ルーチン、*LAPACK* ルーチンまたは *VML* ルーチンで不正な入力パラメーターの値があったときに呼び出される。

メッセージを出力し実行を停止する。

ルーチンを導入する場合に、システム固有の例外処理機能呼び出すのであれば、`stop` 文の変更を考慮するとよい。

入力パラメーター

<code>sname</code>	CHARACTER*6 <code>xerbla</code> を呼び出したルーチンの名前。
<code>info</code>	INTEGER。 呼び出しルーチンのパラメーター・リストにおいて、不正なパラメーターの位置。

ScaLAPACK ルーチン

6

この章では、分散型メモリー・アーキテクチャー用の ScaLAPACK ルーチンに関するインテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) の実装について説明する。ScaLAPACK ルーチンは、実数と複素数の密行列および帯行列をサポートし、連立 1 次方程式、線形の最小二乗問題、固有値と特異値問題を解くタスクに加えて、さまざまな関連する計算タスクを実行する。すべてのルーチンは、単精度と倍精度の両方で使用できる。



注: ScaLAPACK ルーチンは、インテル MKL のスーパーセットであるインテル[®] MKL クラスタ・エディションで提供されている。

この章の各セクションで、ScaLAPACK の [計算ルーチン](#) (計算タスクを個別に実行) と [ドライバルーチン](#) (1 回の呼び出しで標準的な問題の解を算出) について説明する。

一般に、ScaLAPACK は、メッセージ・パッシング・レイヤーとして MPI を使用するコンピューターのネットワーク上で実行され、事前に構築された 1 セットのコミュニケーション・サブプログラム (BLACS) と対象のアーキテクチャー用に最適化された BLAS のセットを使用する。ScaLAPACK のインテル MKL クラスタ・エディションのバージョンは、インテル[®] プロセッサ用に最適化されている。詳細なシステム要件と環境要件は、「[インテル MKL リリースノート](#)」および「[インテル MKL テクニカル・ユーマー・ノート](#)」を参照。

ScaLAPACK ルーチンの完全なリファレンスと関連情報は、[\[SLUG\]](#) を参照。

概要

ScaLAPACK 用計算環境のモデルは、プロセスの 1 次元配列 (帯行列または三角対角行列の計算の場合) または 2 次元のプロセスグリッド (密行列の計算の場合) として表される。ScaLAPACK を使用するには、すべてのグローバル行列またはベクトルは、ScaLAPACK ルーチンを呼び出す前に、この配列またはグリッド上に分割されなければならない。

ScaLAPACK は、密行列計算用のレイアウトとして、2 次元のブロック・サイクリック・データ分割を使用する。この分割により、利用可能なプロセッサ間で作業の平衡化が行われ、BLAS レベル 3 のルーチンを最適なローカル計算で使用することが可能になる。各グローバル配列と対応するプロセスおよびメモリー位置間のマッピングを確立す

るために必要なデータ分割に関する情報は、各グローバル配列と関連する *配列ディスクリプター* に格納される。
配列ディスクリプター構造の例を [表 6-1](#) に示す。

表 6-1 密行列用の配列ディスクリプターの内容

配列成分 #	名前	定義
1	<i>dtype</i>	ディスクリプターのタイプ (密行列の場合は 1)
2	<i>ctxt</i>	プロセスグリッドの BLACS コンテキスト・ハンドル
3	<i>m</i>	グローバル配列の行数
4	<i>n</i>	グローバル配列の列数
5	<i>mb</i>	行ブロック化係数
6	<i>nb</i>	列ブロック化係数
7	<i>rsrc</i>	グローバル配列の最初の行が分割されるプロセス列
8	<i>csrc</i>	グローバル配列の最初の列が分割されるプロセス行
9	<i>lld</i>	ローカル配列のリーディング・ディメンジョン

データ分割後にグリッドの特定のプロセスが受け取るグローバル密行列の行数と列数は、 $LOC_r()$ と $LOC_c()$ で示される。これらの数は、ScaLAPACK ルーチン `numroc` を使用して計算できる。

グローバルデータのブロック・サイクリック分割が完了したら、グローバル行列 *A* の部分行列で実行する操作を選択できる。操作は、グローバル部分配列 `sub(A)` を含み、以下の 6 つの値で定義される (密行列の場合)。

<i>m</i>	<code>sub(A)</code> の行数
<i>n</i>	<code>sub(A)</code> の列数
<i>a</i>	全体のグローバル配列 <i>A</i> を含むローカル配列へのポインター
<i>ia</i>	グローバル配列における <code>sub(A)</code> の行インデックス
<i>ja</i>	グローバル配列における <code>sub(A)</code> の列インデックス
<i>desca</i>	グローバル配列の配列ディスクリプター

ルーチン命名規則

この章の各ルーチンについて、ScaLAPACK 名を使用できる。ScaLAPACK ルーチンの命名規則は、LAPACK ルーチン (第 4 章の「[ルーチン命名規則](#)」を参照) に使用されているものと似ている。一般に、ScaLAPACK のルーチン名は、LAPACK 名の先頭に文字 *p* が追加された名前である。

ScaLAPACK 名 は、以下に示すように、`p?yyzzz` または `p?yyzz` という構造になっている。

先頭の文字 *p* は、ScaLAPACK ルーチン用の特殊なプリフィックスであり、個々のルーチンに追加される。

2 番目の記号 *?* は、データ型を示す。

<i>s</i>	実数、単精度	<i>c</i>	複素数、単精度
<i>d</i>	実数、倍精度	<i>z</i>	複素数、倍精度

3 番目と 4 番目の文字 *yy* は、行列のタイプを示す。

ge	一般行列
gb	一般帯行列
gg	一般行列のペア (汎用問題用)
dt	一般三重対角行列 (対角優位)
db	一般帯行列 (対角優位)
po	対称 / エルミート 正定値行列
pb	対称 / エルミート 正定値帯行列
pt	対称 / エルミート 正定値三重対角行列
sy	対称行列
st	対称三重対角行列 (実数)
he	エルミート行列
or	直交行列
tr	三角 (または準三角) 行列
tz	台形行列
un	ユニタリー行列

計算ルーチンでは、最後の 3 つの文字 **zzz** は、実行される処理を示し、LAPACK ルーチンと同じ意味を持つ。

ドライバールーチンでは、最後の 2 つの文字 **zz** または 3 つの文字 **zzz** は次の意味を持つ。

sv	1 次方程式を解くための簡易ドライバー
svx	1 次方程式を解くための高度ドライバー
ls	線形最小二乗問題を解くためのドライバー
ev	対称固有値問題を解くための簡易ドライバー
evx	対称固有値問題を解くための高度ドライバー
svd	特異値分解計算用のドライバー
gvx	汎用対称固有値問題を解くための高度ドライバー

簡易ドライバーは、汎用問題のみを解くドライバーである。高度ドライバーは、より用途が広く、その他の関連する演算 (例えば、1 次方程式を解いた後の、解の精度の改善と誤差範囲の計算など) を行うこともできる。

計算ルーチン

以下の各セクションでは、ScaLAPACK 計算ルーチンについて説明する。LAPACK 計算ルーチンは、次の目的の計算タスクを個別に実行する。

- [連立 1 次方程式を解く](#)
- [直交因子分解 および LLS 問題](#)
- [対称固有値問題](#)
- [非対称固有値問題](#)
- [特異値分解](#)
- [汎用対称固有値問題](#)

各[ドライバールーチン](#)も参照のこと。

1 次方程式

ScaLAPACK は、以下のタイプの行列を持つ連立方程式用のルーチンをサポートしている。

- 一般行列

- 一般帯行列
- 一般対角優位帯行列 (一般三重対角行列を含む)
- 対称/エルミート正定値行列
- 対称/エルミート正定値帯行列
- 対称/エルミート正定値三重対角行列

対角優位行列は、この行列の LU 因子分解にピボット演算が不要であることがあらかじめわかっている行列として定義される。

これらの各行列タイプについて、ライブラリーには、行列の因子分解、行列の平衡化、連立 1 次方程式の解の算出、行列の条件数の推定、1 次方程式の解の精度の改善と誤差範囲の計算、行列の逆転を計算するためのルーチンが用意されている。いくつかの行列タイプでは、一部の計算ルーチンのみが提供されている点に注意する (例えば、解の精度を改善するルーチンは帯行列や三角行列では提供されていない)。使用可能なルーチンの完全なリストは、表 6-2 を参照のこと。

特定の問題を解くために、2 つ以上の計算ルーチン呼び出しか、1 回の呼び出しで複数のタスクが組み合わさった対応する [ドライバルーチン](#) を呼び出せる。したがって、一般行列の連立 1 次方程式を解く場合、最初に `p?getrf` (LU 因子分解) を呼び出し、次に `p?getrs` (解の算出) を呼び出せる。さらに、`p?gerfs` を呼び出して、解の精度を改善し、誤差範囲を計算できる。代わりに、これらのタスクを 1 回の呼び出しで実行するドライバルーチン `p?gesvx` を使用することもできる。

表 6-2 に、実数行列の因子分解、平衡化、行列の逆転、条件数の推定、連立 1 次方程式の解の算出、解の精度の改善、誤差の推定を行うための ScaLAPACK 計算ルーチンを示す。

表 6-2 連立 1 次方程式用の計算ルーチン

行列のタイプ、 格納形式	行列の 因子分解	行列の 平衡化	方程式の 解の算出	条件数	誤差の 推定	逆行列の 計算
一般 (部分ピボット演算)	p?getrf	p?aeegu	p?getrs	p?aecon	p?gerfs	p?getri
一般帯 (部分ピボット演算)	p?gbtrf		p?gbtrs			
一般帯 (ピボット演算なし)	p?dbtrf		p?dbtrs			
一般三重対角 (ピボット演算なし)	p?dttrf		p?dttrs			
対称/エルミート正 定値	p?potrf	p?poequ	p?potrs	p?pocon	p?porfs	p?potri
対称/エルミート正 定値 (帯形式)	p?pbtrf		p?pbtrs			
対称/エルミート正 定値 (三重対角形式)	p?pttrf		p?pttrs			
三角			p?trtrs	p?trcon	p?trrfs	p?trtri

表中の ? は、s (単精度実数)、d (倍精度実数)、c (単精度複素数)、または z (倍精度複素数) を示す。

行列の因子分解用のルーチン

このセクションでは、行列の因子分解用の ScaLAPACK ルーチンについて説明する。以下の因子分解をサポートしている。

- 一般行列の LU 因子分解
- 対角優位行列の LU 因子分解
- 実対称 / 複素エルミート正定値行列のコレスキー因子分解

因子分解の計算には、フル格納と帯格納形式の行列を使用できる。

p?getrf

$m \times n$ の一般分散行列の LU 因子分解を行う。

構文

```
call psgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pdgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pcgetrf( m, n, a, ia, ja, desca, ipiv, info )
call pzgetrf( m, n, a, ia, ja, desca, ipiv, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ の LU 因子分解を行う。

$$A = PLU$$

ここで、 P は置換行列、 L は単位対角成分を含む下三角 ($m > n$ の場合は下台形)、 U は上三角 ($m < n$ の場合上台形) である。 L と U は $\text{sub}(A)$ に格納される。

このルーチンでは、行を交換し、部分的にピボット演算を行う。

入力パラメーター

m	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
a	(ローカル) REAL (psgetrf の場合) DOUBLE PRECISION (pdgetrf の場合) COMPLEX (pcgetrf の場合) DOUBLE COMPLEX (pzgetrf の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOC_c(ja+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
ia, ja	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+n-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 A の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen*)。分散行列 *A* の配列ディスクリプター。

出力パラメーター

a 因子分解 $A = PLU$ で得られた係数 *L* と *U* のローカル部分によって上書きされる。*L* の単位対角成分は格納されない。

ipiv (ローカル) INTEGER 配列。
ipiv の次元は $(LOC_r(m_a) + mb_a)$ 。
この配列には、ピボット演算情報が格納される。ローカル行 *i* は、グローバル行 *ipiv*(*i*) と交換される。この配列は、分散行列 *A* に関連付けられる。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。
info = *i* の場合、 u_{ii} は 0 である。因子分解は完了したが、係数 *U* は完全に特異である。連立 1 次方程式の解の算出に係数 *U* を使用すると、0 による除算が発生する。

p?gbtrf

$n \times n$ の一般帯分散行列の LU 因子分解を行う。

構文

```
call psgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pdgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pcgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
call pzgbtrf(n, bwl, bwu, a, ja, desca, ipiv, af, laf, work, lwork, info)
```

説明

このルーチンは、行交換を伴う部分ピボット演算を用いて、 $n \times n$ の一般実 / 複素帯分散行列 $A(1:n, ja:ja+n-1)$ の LU 因子分解を行う。

ここで行われる因子分解は、LAPACK ルーチン [?gbtrf](#) から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = PLUQ$$

ここで、*P* と *Q* は置換行列、*L* は下三角行列、*U* は上三角行列である。行列 *Q* は並列化のために列の順序を変更することを表し、*P* は数の安定のために部分的なピボット演算を使用して行の順序を変更することを表す。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。分散部分行列 $A(1:n, ja:ja+n-1)$ の行と列の数 ($n \geq 0$)。
<i>bw1</i>	(グローバル) INTEGER。A の帯内にある劣対角成分の数 ($0 \leq bw1 \leq n-1$)。
<i>bwu</i>	(グローバル) INTEGER。A の帯内にある優対角成分の数 ($0 \leq bwu \leq n-1$)。
<i>a</i>	(ローカル) REAL (psgbtrf の場合) DOUBLE PRECISION (pdgbtrf の場合) COMPLEX (pcgbtrf の場合) DOUBLE COMPLEX (pzgbtrf の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOC_c(ja+n-1)$) の配列へのポインター。 ここで、 $lld_a \geq 2*bw1 + 2*bwu + 1$ である。 因子分解する $n \times n$ 分散帯行列 $A(1:n, ja:ja+n-1)$ のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。 $desca(dtype_)$ = 501 の場合、 $dlen_ \geq 7$ 。 $desca(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq (NB+bwu)*(bw1+bwu)+6*(bw1+bwu)*(bw1+2*bwu)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> のサイズ ($lwork \geq 1$)。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。

出力パラメーター

<i>a</i>	終了時に、この配列には因子分解の詳細が格納される。行列で追加の置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。
<i>ipiv</i>	(ローカル) INTEGER 配列。 <i>ipiv</i> の次元は $desca(NB)$ 以上でなければならない。 ローカル因子分解用のピボットのインデックスが格納される。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。

<i>af</i>	(ローカル) REAL (psgbtrf の場合) DOUBLE PRECISION (pdgbtrf の場合) COMPLEX (pcgbtrf の場合) DOUBLE COMPLEX (pzgbtrf の場合) 配列、次元は (<i>laf</i>)。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?gbtrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に p?gbtrs を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が非特異でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

p?dbtrf

$n \times n$ の対角優位帯分散行列の LU 因子分解を行う。

構文

```
call psdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pddbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pcdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
call pzdbtrf( n, bwl, bwu, a, ja, desca, af, laf, work, lwork, info )
```

説明

このルーチンは、ピボット演算を使用しないで、 $n \times n$ の実 / 複素対角優位帯分散行列 $A(1:n, ja:ja+n-1)$ の LU 因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる点に注意する。並列用の行列では、追加の置換が実行される。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。分散部分行列 $A(1:n, ja:ja+n-1)$ の行と列の数 ($n \geq 0$)。
<i>bw1</i>	(グローバル) INTEGER。A の帯内にある劣対角成分の数 ($0 \leq bw1 \leq n-1$)。
<i>bwu</i>	(グローバル) INTEGER。A の帯内にある優対角成分の数 ($0 \leq bwu \leq n-1$)。
<i>a</i>	(ローカル) REAL (psdbtrf の場合) DOUBLE PRECISION (pddbtrf の場合) COMPLEX (pcdbtrf の場合) DOUBLE COMPLEX (pzdbtrf の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOC_c(ja+n-1)$) の配列へのポインター。 因子分解する $n \times n$ 分散帯行列 $A(1:n, ja:ja+n-1)$ のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。 $desca(dtype_)$ = 501 の場合、 $dlen_ \geq 7$ 。 $desca(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq NB * (bw1 + bwu) + 6 * (\max(bw1, bwu))^2$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq (\max(bw1, bwu))^2$ でなければならない。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。

出力パラメーター

<i>a</i>	終了時に、この配列には因子分解の詳細が格納される。行列で追加の置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。
<i>af</i>	(ローカル) REAL (psdbtrf の場合) DOUBLE PRECISION (pddbtrf の場合) COMPLEX (pcdbtrf の場合) DOUBLE COMPLEX (pzdbtrf の場合)

配列、次元は (*laf*)。
 補助非零要素空間。非零要素は、因子分解ルーチン `p?dbtrf` で作成され、*af* に格納される。
 因子分解ルーチンの後に `p?dbtrs` を使用して 1 次方程式の解を算出する場合、*af* は因子分解の後に変更されてはならない点に注意する。

work(1) 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。

info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。
info > 0 の場合：

info = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が対角優位でないため、因子分解を完了できなかったことを示す。
info = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

p?potrf

対称(エルミート)正定値分散行列のコレスキー因子分解を行う。

構文

```
call pspotrf( uplo, n, a, ia, ja, desca, info )
call pdpotrf( uplo, n, a, ia, ja, desca, info )
call pcpotrf( uplo, n, a, ia, ja, desca, info )
call pzpotrf( uplo, n, a, ia, ja, desca, info )
```

説明

このルーチンは、 $n \times n$ の実対称 / 複素エルミート正定値分散行列 $A(ia:ia+n-1, ja:ja+n-1)$ のコレスキー因子分解を行う。行列は、以下 $\text{sub}(A)$ として表す。

因子分解の形式は次のとおりである。

$$\text{sub}(A) = U^H U \quad (\text{uplo} = 'U' \text{ の場合}), \text{ または}$$

$$\text{sub}(A) = LL^H \quad (\text{uplo} = 'L' \text{ の場合})$$

ここで、 L は下三角行列、 U は上三角行列である。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER*1. 'U' または 'L' でなければならない。 sub(A) の上三角部分と下三角部分のどちらが格納されるかを指定する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> には行列 sub(A) の上三角部分が格納され、 sub(A) は $U^H U$ として因子分解される。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> には行列 sub(A) の下三角部分が格納され、 sub(A) は LL^H として因子分解される。
<i>n</i>	(グローバル) INTEGER。分散部分行列 sub(A) の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (pspotrf の場合) DOUBLE PRECISION (pdpotrf の場合) COMPLEX (pcpotrf の場合) DOUBLE COMPLEX (pzpotrf の場合) ローカルメモリーにある、次元 ($lld_a, LOC_c(ja+n-1)$) の配列への ポインター。 呼び出し時に、この配列は、因子分解する $n \times n$ の対称/エルミート 分散行列 sub(A) のローカル部分を含む。 <i>uplo</i> に応じて、配列 <i>a</i> は、行列 sub(A) の上三角部分または下三角部 分を含む (<i>uplo</i> を参照)。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 sub(A) の最初の行と最初 の列を示す、グローバル配列 A の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行 列 A の配列ディスクリプター。

出力パラメーター

<i>a</i>	<i>uplo</i> の指定に従って、 <i>a</i> の上三角部分または下三角部分が、コレス キー係数 U あるいは L によって上書きされる。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場 合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場 合、 <i>info</i> = - <i>i</i> 。 <i>info</i> = <i>k</i> > 0 の場合、次数 <i>k</i> の先頭の小行列式 $A(ia:ia+k-1, ja:ja+k-1)$ が正定値でないため、因子分解を完了でき なかったことを示す。

p?pbtrf

対称/エルミート正定値帯分散行列のコレスキー因子分解を行う。

構文

```
call pspbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pdpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pcpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
call pzpbtrf( uplo, n, bw, a, ja, desca, af, laf, work, lwork, info )
```

説明

このルーチンは、 $n \times n$ の実対称 / 複素エルミート正定値帯分散行列 $A(1:n, ja:ja+n-1)$ のコレスキー因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$A(1:n, ja:ja+n-1) = P U^H U P^T$ ($uplo='U'$ の場合)、または

$A(1:n, ja:ja+n-1) = P L L^H P^T$ ($uplo='L'$ の場合)

ここで、 P は置換行列、 U は上三角行列、 L は下三角行列である。

入力パラメーター

- uplo** (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
 $uplo='U'$ の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。
 $uplo='L'$ の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
- n** (グローバル) INTEGER。分散部分行列 $A(1:n, ja:ja+n-1)$ の次数 ($n \geq 0$)。
- bw** (グローバル) INTEGER。分散行列の優対角成分の数 ($uplo='U'$ の場合) または劣対角成分の数 ($uplo='L'$ の場合) ($bw \geq 0$)。
- a** (ローカル)
 REAL (pspbtrf の場合)
 DOUBLE PRECISION (pdpbtrf の場合)
 COMPLEX (pcpbtrf の場合)
 DOUBLE COMPLEX (pzpbtrf の場合)
 ローカルメモリーにある、次元 ($lld_a, LOC_c(ja+n-1)$) の配列へのポインター。
 呼び出し時に、この配列は、因子分解する対称/エルミート帯分散行列 $A(1:n, ja:ja+n-1)$ の上三角または下三角のローカル部分を含む。
- ja** (グローバル) INTEGER。 (A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca</i> (<i>dtype_</i>) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> (<i>dtype_</i>) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> ≥ (NB+2* <i>bw</i>)* <i>bw</i> でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> ≥ <i>bw</i> ² でなければならない。

出力パラメーター

<i>a</i>	終了時に、 <i>info</i> =0 の場合、 <i>uplo</i> の指定に従って、帯行列 <i>A</i> (1: <i>n</i> , <i>ja</i> : <i>ja</i> + <i>n</i> -1) のコレスキー因子分解で得られた三角係数 <i>U</i> または <i>L</i> が格納される。
<i>af</i>	(ローカル) REAL (<i>pspbtrf</i> の場合) DOUBLE PRECISION (<i>pdpbtrf</i> の場合) COMPLEX (<i>pcpbtrf</i> の場合) DOUBLE COMPLEX (<i>pzpbtrf</i> の場合) 配列、次元は (<i>laf</i>)。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に <i>p?pbtrs</i> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

p?pttrf

対称/エルミート正定値三重対角分散行列のコレスキー因子分解を行う。

構文

```
call pspttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pdpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pcpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
call pzpttrf( n, d, e, ja, desca, af, laf, work, lwork, info )
```

説明

このルーチンは、 $n \times n$ の実対称/複素エルミート正定値三重対角分散行列 $A(1:n, ja:ja+n-1)$ のコレスキー因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = PLDL^H P^T \text{ または}$$

$$A(1:n, ja:ja+n-1) = P U^H D U P^T$$

ここで、 P は置換行列、 U は上三角行列、 L は下三角行列である。

入力パラメーター

n	(グローバル) INTEGER。分散部分行列 $A(1:n, ja:ja+n-1)$ の次数 ($n \geq 0$)。
d, e	(ローカル) REAL (pspttrf の場合) DOUBLE PRECISION (pdpttrf の場合) COMPLEX (pcpttrf の場合) DOUBLE COMPLEX (pzpttrf の場合) それぞれ、ローカルメモリーにある、次元 ($desca(nb_)$) の配列へのポインター。 呼び出し時に、配列 d は、分散行列 A の主対角を格納するグローバルベクトルのローカル部分を含む。 呼び出し時に、配列 e は、分散行列 A の上対角を格納するグローバルベクトルのローカル部分を含む。
ja	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。 $desca(dtype_)$ = 501 の場合、 $dlen_ \geq 7$ 。 $desca(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。

<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq NB+2$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>d</i> および <i>e</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq 8 * NPCOL$ でなければならない。

出力パラメーター

<i>d, e</i>	終了時に、因子分解の詳細によって上書きされる。
<i>af</i>	(ローカル) REAL (<i>pspttrf</i> の場合) DOUBLE PRECISION (<i>pdpttrf</i> の場合) COMPLEX (<i>pcpttrf</i> の場合) DOUBLE COMPLEX (<i>pzpttrf</i> の場合) 配列、次元は (<i>laf</i>)。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?pttrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に <i>p?pttrs</i> を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

p?dttrf

対角優位三重対角分散行列の LU 因子分解を行う。

構文

```
call psdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pddttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pcdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
call pzdttrf( n, dl, d, du, ja, desca, af, laf, work, lwork, info )
```

説明

このルーチンは、ピボット演算を使用しないで、 $n \times n$ の実 / 複素対角優位三重対角分散行列 $A(1:n, ja:ja+n-1)$ の LU 因子分解を行う。

因子分解の結果は、LAPACK から返される因子分解とは異なる。並列用の行列では、追加の置換が実行される。

因子分解の形式は次のとおりである。

$$A(1:n, ja:ja+n-1) = P L U P^T$$

ここで、 P は置換行列、 U は上三角行列、 L は下三角行列である。

入力パラメーター

n	(グローバル) INTEGER。処理される行数と列数。すなわち、分散行列 $A(1:n, ja:ja+n-1)$ の次数 ($n \geq 0$)。
dl, d, du	(ローカル) REAL (pspttrf の場合) DOUBLE PRECISION (pdpttrf の場合) COMPLEX (pcpttrf の場合) DOUBLE COMPLEX (pzpttrf の場合) それぞれ、次元 ($desca(nb_)$) のローカル配列へのポインター。 呼び出し時に、配列 dl は、行列の劣対角成分を格納するグローバルベクトルのローカル部分を含む。全体的に、 $dl(1)$ は参照されず、 dl は d とアライメントされなければならない。 呼び出し時に、配列 d は、行列の対角成分を格納するグローバルベクトルのローカル部分を含む。 呼び出し時に、配列 du は、行列の優対角成分を格納するグローバルベクトルのローカル部分を含む。 $du(n)$ は参照されず、 du は d とアライメントされなければならない。
ja	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca(dtype_)</i> = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca(dtype_)</i> = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> ≥ 2*(NB+2) でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>d</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> ≥ 8*NPCOL でなければならない。

出力パラメーター

<i>dl, d, du</i>	終了時に、行列の係数を含む情報によって上書きされる。
<i>af</i>	(ローカル) REAL (<i>psdttrf</i> の場合) DOUBLE PRECISION (<i>pddttrf</i> の場合) COMPLEX (<i>pcdttrf</i> の場合) DOUBLE COMPLEX (<i>pzdttrf</i> の場合) 配列、次元は (<i>laf</i>)。 補助非零要素空間。非零要素は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。 因子分解ルーチンの後に p?dttrs を使用して 1 次方程式の解を算出する場合、 <i>af</i> は因子分解の後に変更されてはならない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が対角優位でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

連立 1 次方程式を解くためのルーチン

このセクションでは、連立 1 次方程式を解くための ScaLAPACK ルーチンについて説明する。通常は、これらのルーチンを呼び出す前に、連立方程式の行列を因子分解する必要がある (この章の「[行列の因子分解用のルーチン](#)」を参照)。ただし、解を求める連立方程式が三角行列を持つ場合は、因子分解の必要はない。

p?getrs

p?getrf によって行われた LU 因子分解を使用して、一般正方行列の分散連立 1 次方程式を解く。

構文

```
call psgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pdgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pcgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
call pzgetrs(trans, n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb,
             info)
```

説明

このルーチンは、[p?getrf](#) によって行われた LU 因子分解を使用して、 $n \times n$ の一般分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ の分散連立 1 次方程式を解く。

trans ルーチンで指定される連立方程式の形式は、次のいずれかである。

$\text{sub}(A) * X = \text{sub}(B)$ (転置なし)、

$\text{sub}(A)^T * X = \text{sub}(B)$ (転置あり)、

$\text{sub}(A)^H * X = \text{sub}(B)$ (共役転置)

ここで、 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ である。

このルーチンを呼び出す前に、p?getrf を呼び出して、 $\text{sub}(A)$ の LU 因子分解を行う必要がある。

入力パラメーター

trans (グローバル) CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。

方程式の形式を指定する。

trans = 'N' の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を X について解く。

trans = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を X について解く。

trans = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を X について解く。

<i>n</i>	(グローバル) INTEGER。1 次方程式の数。部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<i>a, b</i>	(グローバル) REAL (psgetrs の場合) DOUBLE PRECISION (pdgetrs の場合) COMPLEX (pcgetrs の場合) DOUBLE COMPLEX (pzgetrs の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(lld_a, LOC_c(ja+n-1))$ と $b(lld_b, LOC_c(jb+nrhs-1))$ へのポインター。 呼び出し時に、配列 <i>a</i> は因子分解 $\text{sub}(A) = PLU$ で得られた係数 <i>L</i> と <i>U</i> のローカル部分を含む。 <i>L</i> の単位対角成分は格納されない。 呼び出し時に、配列 <i>b</i> は右辺 $\text{sub}(B)$ を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₁)。分散行列 <i>A</i> の配列ディスクリプター。
<i>ipiv</i>	(ローカル) INTEGER 配列。 <i>ipiv</i> の次元は $(LOC_r(m_a) + mb_a)$ 。 この配列は、ピボット演算情報を含む。行列のローカル行 <i>i</i> は、グローバル行 <i>ipiv</i> (<i>i</i>) と交換される。 この配列は、分散行列 <i>A</i> に関連付けられる。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₁)。分散行列 <i>B</i> の配列ディスクリプター。

出力パラメーター

<i>b</i>	終了時に、解の分散行列 <i>X</i> によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?gbtrs

p?gbtrf によって行われた LU 因子分解を使用して、一般帯行列の分散連立 1 次方程式を解く。

構文

```
call psgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,
             af, laf, work, lwork, info)
call pdgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,
             af, laf, work, lwork, info)
call pcgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,
             af, laf, work, lwork, info)
call pzgbtrs(trans, n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb,
             af, laf, work, lwork, info)
```

説明

このルーチンは、p?gbtrf によって行われた LU 因子分解を使用して、一般帯行列 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ の分散連立 1 次方程式を解く。

trans ルーチンで指定される連立方程式の形式は、次のいずれかである。

$\text{sub}(A)*X = \text{sub}(B)$ (転置なし)、

$\text{sub}(A)^T*X = \text{sub}(B)$ (転置あり)、

$\text{sub}(A)^H*X = \text{sub}(B)$ (共役転置)

ここで、 $\text{sub}(B) = B(ib:ib+n-1, 1:nrhs)$ である。

このルーチンを呼び出す前に、[p?gbtrf](#) を呼び出して、 $\text{sub}(A)$ の LU 因子分解を行う必要がある。

入力パラメーター

trans	(グローバル) CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 trans = 'N' の場合、 $\text{sub}(A)*X = \text{sub}(B)$ を X について解く。 trans = 'T' の場合、 $\text{sub}(A)^T*X = \text{sub}(B)$ を X について解く。 trans = 'C' の場合、 $\text{sub}(A)^H*X = \text{sub}(B)$ を X について解く。
n	(グローバル) INTEGER。1 次方程式の数。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
bwl	(グローバル) INTEGER。A の帯内にある劣対角成分の数 ($0 \leq bwl \leq n-1$)。
bwu	(グローバル) INTEGER。A の帯内にある優対角成分の数 ($0 \leq bwu \leq n-1$)。

<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<i>a, b</i>	(グローバル) REAL (psgbtrs の場合) DOUBLE PRECISION (pdgbtrs の場合) COMPLEX (pcgbtrs の場合) DOUBLE COMPLEX (pzgbtrs の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(lld_a, LOC_c(ja+n-1))$ と $b(lld_b, LOC_c(nrhs))$ へのポインタ。 配列 <i>a</i> は、分散帯行列 <i>A</i> の LU 因子分解の詳細を含む。 呼び出し時に、配列 <i>b</i> は右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 $desca(dtype_)$ = 501 の場合、 $dlen_ \geq 7$ 。 $desca(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。
<i>ib</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 $desca(dtype_)$ = 501 の場合、 $dlen_ \geq 7$ 。 $desca(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq NB*(bw1+bwu)+6*(bw1+bwu)*(bw1+2*bwu)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル) <i>a</i> と同じタイプ。ワークスペース配列、次元は <i>lwork</i> 。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq nrhs*(NB+2*bw1+4*bwu)$ でなければならない。

出力パラメーター

<i>ipiv</i>	(ローカル) INTEGER 配列。 <i>ipiv</i> の次元は <i>desca</i> (NB) 以上でなければならない。 ローカル因子分解用のピボットのインデックスが格納される。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分によって上書きされる。
<i>af</i>	(ローカル)

REAL (psgbtrs の場合)
 DOUBLE PRECISION (pdgbtrs の場合)
 COMPLEX (pcgbtrs の場合)
 DOUBLE COMPLEX (pzgbtrs の場合)

配列、次元は (*laf*)。

補助非零要素空間。非零要素は、因子分解ルーチン *p?gbtrf* で作成され、*af* に格納される。

因子分解ルーチンの後に *p?gbtrs* を使用して 1 次方程式の解を算出する場合、*af* は因子分解の後に変更されてはならない点に注意する。

work(1) 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。

p?potrs

コレスキー因子分解された対称/エルミート分散
 正定値行列の連立 1 次方程式を解く。

構文

```
call pspotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pdpotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pcpotrs( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
call pzpots( uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info )
```

説明

ルーチン *p?potrs* は、次の分散連立 1 次方程式を *X* について解く。

$$\text{sub}(A) * X = \text{sub}(B)$$

ここで、*sub(A)* = *A*(*ia*:*ia*+*n*-1, *ja*:*ja*+*n*-1) は *n* × *n* の実対称/複素エルミート正定値分散行列、*sub(B)* は分散行列 *B*(*ib*:*ib*+*n*-1, *jb*:*jb*+*nrhs*-1) である。
 このルーチンは、次のコレスキー因子分解を使用する。

$$\text{sub}(A) = U^H U \text{ または } \text{sub}(A) = L L^H$$

因子分解は、[p?potrf](#) によって行われる。

入力パラメーター

uplo (グローバル) CHARACTER*1。 'U' または 'L' でなければならない。
uplo = 'U' の場合、*sub(A)* の上三角部分が格納される。
uplo = 'L' の場合、*sub(A)* の下三角部分が格納される。

<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<i>a, b</i>	(ローカル) REAL (pspotrs の場合) DOUBLE PRECISION (pdpotrs の場合) COMPLEX (pcpotrs の場合) DOUBLE COMPLEX (pzpotrs の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(lld_a, LOC_c(ja+n-1))$ と $b(lld_b, LOC_c(jb+nrhs-1))$ へのポインター。 配列 <i>a</i> は、p?potrf によって行われたコレスキー因子分解 $\text{sub}(A) = LL^H$ または $\text{sub}(A) = U^H U$ で得られた係数 <i>L</i> または <i>U</i> を含む。 呼び出し時に、配列 <i>b</i> は右辺 $\text{sub}(B)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 <i>B</i> の配列ディスクリプター。

出力パラメーター

<i>b</i>	解の行列 <i>X</i> のローカル部分によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?pbtrs

コレスキー因子分解された対称/エルミート正定
 値帯行列の連立1次方程式を解く。

構文

```
call pspbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
call pdpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
```

```
call pcpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
call pzpbtrs( uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
```

説明

ルーチン `p?pbtrs` は、次の分散連立 1 次方程式を X について解く。

$$\text{sub}(A)*X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ は $n \times n$ の実対称 / 複素エルミート正定値帯分散行列、 $\text{sub}(B)$ は分散行列 $B(ib:ib+n-1, 1:nrhs)$ である。
このルーチンは、次のコレスキー因子分解を使用する。

$$\text{sub}(A) = P U^H U P^T \quad \text{または} \quad \text{sub}(A) = P L L^H P^T$$

因子分解は、[p?pbtrf](#) によって行われる。

入力パラメーター

`uplo` (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
`uplo = 'U'` の場合、 $\text{sub}(A)$ の上三角部分が格納される。
`uplo = 'L'` の場合、 $\text{sub}(A)$ の下三角部分が格納される。

`n` (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

`bw` (グローバル) INTEGER。分散行列の優対角成分の数 (`uplo = 'U'` の場合) または劣対角成分の数 (`uplo = 'L'` の場合) ($bw \geq 0$)。

`nrhs` (グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。

`a, b` (ローカル)
REAL (`pspbtrs` の場合)
DOUBLE PRECISION (`pdpbtrs` の場合)
COMPLEX (`pcpbtrs` の場合)
DOUBLE COMPLEX (`pzpbtrs` の場合)
それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(1:d_a, LOC_c(ja+n-1))$ と $b(1:d_b, LOC_c(nrhs-1))$ へのポインター。
配列 a は、`p?pbtrf` によって返される、帯行列 A のコレスキー因子分解 $\text{sub}(A) = P U^H U P^T$ または $\text{sub}(A) = P L L^H P^T$ で得られた三角係数 U あるいは L を含む。
呼び出し時に、配列 b は、 $n \times nrhs$ の右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。

`ja` (グローバル) INTEGER。 (A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。

`desca` (グローバルおよびローカル) INTEGER 配列、次元は (`dlen_`)。分散行列 A の配列ディスクリプター。
`desca(dtype_)` = 501 の場合、`dlen_` ≥ 7 。
`desca(dtype_)` = 1 の場合、`dlen_` ≥ 9 。

<i>ib</i>	(グローバル) INTEGER。部分行列 $\text{sub}(B)$ の最初の行を示す、グローバル配列 B の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 B の配列ディスクリプター。 <i>descb</i> (<i>dtype</i> _l) = 502 の場合、 <i>dlen</i> _l ≥ 7。 <i>descb</i> (<i>dtype</i> _l) = 1 の場合、 <i>dlen</i> _l ≥ 9。
<i>af, work</i>	(ローカル) 配列、 <i>a</i> と同じタイプ。 配列 <i>af</i> は、次元 (<i>laf</i>) の配列。補助非零要素空間が格納される。非零要素は、因子分解ルーチン p?dbtrf で作成され、 <i>af</i> に格納される。 配列 <i>work</i> は、次元 <i>lwork</i> のワークスペース配列。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> ≥ <i>nrhs</i> * <i>bw</i> でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> ≥ <i>bw</i> ² でなければならない。

出力パラメーター

<i>b</i>	終了時に、 <i>info</i> =0 の場合、この配列には、 $n \times \text{nrhs}$ の解の分散行列 X のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?pttrs

p?pttrf によって行われた因子分解を使用して、
対称(エルミート)正定値三重対角分散行列の連立1次方程式を解く。

構文

```
call pspttrs( n, nrhs, d, e, ja, desca, b, ib, descb, af, laf, work,
             lwork, info )
call pdpttrs( n, nrhs, d, e, ja, desca, b, ib, descb, af, laf, work,
             lwork, info )
call pcpttrs( uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
```

```
call pzpttrs( uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
             work, lwork, info )
```

説明

ルーチン `p?pttrs` は、次の分散連立 1 次方程式を X について解く。

$$\text{sub}(A)*X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ は $n \times n$ の実対称 / 複素エルミート正定値三重対角分散行列、 $\text{sub}(B)$ は分散行列

$B(ib:ib+n-1, 1:nrhs)$ である。

このルーチンは、次の因子分解を使用する。

$$\text{sub}(A) = PLDL^H P^T \quad \text{または} \quad \text{sub}(A) = P U^H D U P^T$$

因子分解は、`p?pttrf` によって行われる。

入力パラメーター

<code>uplo</code>	(グローバル、複素数型の場合のみ使用される) CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 $\text{sub}(A)$ の上三角部分が格納される。 <code>uplo = 'L'</code> の場合、 $\text{sub}(A)$ の下三角部分が格納される。
<code>n</code>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<code>nrhs</code>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<code>d, e</code>	(ローカル) REAL (<code>pspttrs</code> の場合) DOUBLE PRECISION (<code>pdpttrs</code> の場合) COMPLEX (<code>pcpttrs</code> の場合) DOUBLE COMPLEX (<code>pzpttrs</code> の場合) それぞれ、ローカルメモリーにある、次元 (<code>desca(nb_)</code>) の配列へのポインター。 これらの配列は、 <code>p?pttrf</code> によって返される因子分解の詳細を含む。
<code>ja</code>	(グローバル) INTEGER。 (A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER 配列、次元は (<code>dlen_</code>)。分散行列 A の配列ディスクリプター。 <code>desca(dtype_) = 501</code> または <code>502</code> の場合、 <code>dlen_ \geq 7</code> 。 <code>desca(dtype_) = 1</code> の場合、 <code>dlen_ \geq 9</code> 。
<code>b</code>	(ローカル) d, e と同じタイプ。 ローカルメモリーにある、ローカル次元 $b(11d_b, LOC_c(nrhs))$ の配列へのポインター。 呼び出し時に、配列 b は、 $n \times nrhs$ の右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。
<code>ib</code>	(グローバル) INTEGER。 (B のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 B の行インデックス。

<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。 <i>descb(dtype_)</i> = 502 の場合、 <i>dlen_</i> ≥ 7。 <i>descb(dtype_)</i> = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>af, work</i>	(ローカル) REAL (<i>pspttrs</i> の場合) DOUBLE PRECISION (<i>pdpttrs</i> の場合) COMPLEX (<i>pcpttrs</i> の場合) DOUBLE COMPLEX (<i>pzpttrs</i> の場合) それぞれ、次元 (<i>laf</i>) と (<i>lwork</i>) の配列。 配列 <i>af</i> は、補助非零要素空間を含む。非零要素は、因子分解ルーチン <i>p?pttrf</i> で作成され、 <i>af</i> に格納される。 配列 <i>work</i> はワークスペース配列である。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 <i>laf</i> ≥ NB+2 でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 <i>lwork</i> ≥ (10+2*min(100, <i>nrhs</i>))*NPCOL+4* <i>nrhs</i> でなければならない。

出力パラメーター

<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?dttrs

p?dttrf によって行われた因子分解を使用して、
対角優位三重対角分散行列の連立1次方程式を解く。

構文

```
call psdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
call pdpttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
```

```
call pcdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
call pzdttrs( trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
```

説明

ルーチン `p?dttrs` は、以下のいずれかの連立方程式を X について解く。

$$\text{sub}(A) * X = \text{sub}(B),$$

$$(\text{sub}(A))^T * X = \text{sub}(B), \text{ または}$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ は対角優位三重対角分散行列、 $\text{sub}(B)$ は分散行列 $B(ib:ib+n-1, 1:nrhs)$ である。

このルーチンは、[p?dttrf](#) によって行われた LU 因子分解を使用する。

入力パラメーター

<code>trans</code>	(グローバル) CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <code>trans = 'N'</code> の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を X について解く。 <code>trans = 'T'</code> の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を X について解く。 <code>trans = 'C'</code> の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を X について解く。
<code>n</code>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<code>nrhs</code>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<code>dl, d, du</code>	(ローカル) REAL (<code>psdttrs</code> の場合) DOUBLE PRECISION (<code>pddttrs</code> の場合) COMPLEX (<code>pcdttrs</code> の場合) DOUBLE COMPLEX (<code>pzdttrs</code> の場合) それぞれ、次元 (<code>desca(nb_)</code>) のローカル配列へのポインター。 呼び出し時に、これらの配列は因子分解の詳細を含む。全体的に、 <code>dl(1)</code> および <code>du(n)</code> は参照されず、 <code>dl</code> および <code>du</code> は <code>d</code> とアライメントされなければならない。
<code>ja</code>	(グローバル) INTEGER。 (A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER 配列、次元は (<code>dlen_</code>)。分散行列 A の配列ディスクリプター。 <code>desca(dtype_) = 501</code> または <code>502</code> の場合、 <code>dlen_ ≥ 7</code> 。 <code>desca(dtype_) = 1</code> の場合、 <code>dlen_ ≥ 9</code> 。
<code>b</code>	(ローカル) d と同じタイプ。

- ローカルメモリーにある、ローカル次元 $b(11d_b, LOC_c(nrhs))$ の配列へのポインター。
呼び出し時に、配列 b は、 $n \times nrhs$ の右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。
- ib* (グローバル) INTEGER。 (B のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 B の行インデックス。
- descb* (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。
 $descb(dtype_)$ = 502 の場合、 $dlen_ \geq 7$ 。
 $descb(dtype_)$ = 1 の場合、 $dlen_ \geq 9$ 。
- af, work* (ローカル)
REAL ($psdttrs$ の場合)
DOUBLE PRECISION ($pddttrs$ の場合)
COMPLEX ($pcdttrs$ の場合)
DOUBLE COMPLEX ($pzdttrs$ の場合)
それぞれ、次元 (laf) と ($lwork$) の配列。
配列 af は、補助非零要素空間を含む。非零要素は、因子分解ルーチン $p?dttrf$ で作成され、 af に格納される。因子分解ルーチンの後に $p?dttrs$ を使用して 1 次方程式の解を算出する場合、 af は因子分解の後に変更されてはならない点に注意する。
配列 $work$ はワークスペース配列である。
- laf* (ローカル) INTEGER。配列 af の次元。
 $laf \geq NB*(bw1+bwu)+6*(bw1+bwu)*(bw1+2*bwu)$ でなければならない。
 laf が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが $af(1)$ に返される。
- lwork* (ローカルまたはグローバル) INTEGER。配列 $work$ のサイズ。
 $lwork \geq 10*NPCOL+4*nrhs$ でなければならない。

出力パラメーター

- b* 終了時に、この配列には解の分散行列 X のローカル部分が格納される。
- work(1)* 終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
- info* INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合：
 i 番目の引数が配列で、 j 番目の値が不正だった場合、
 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、
 $info = -i$ 。

p?dbtrs

p?dbtrf によって行われた因子分解を使用して、対角優位帯分散行列の連立1次方程式を解く。

構文

```
call psdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
call pddbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
call pcdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
call pzdbtrs( trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info )
```

説明

ルーチン p?dbtrs は、以下のいずれかの連立方程式を X について解く。

$$\text{sub}(A) * X = \text{sub}(B),$$

$$(\text{sub}(A))^T * X = \text{sub}(B), \text{ または}$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ は対角優位帯分散行列、 $\text{sub}(B)$ は分散行列 $B(ib:ib+n-1, 1:nrhs)$ である。

このルーチンは、[p?dbtrf](#) によって行われた LU 因子分解を使用する。

入力パラメーター

<i>trans</i>	(グローバル) CHARACTER*1。'N'、'T'、または'C'のいずれかでなければならない。 方程式の形式を指定する。 <i>trans</i> = 'N' の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を X について解く。 <i>trans</i> = 'T' の場合、 $\text{sub}(A)^T * X = \text{sub}(B)$ を X について解く。 <i>trans</i> = 'C' の場合、 $\text{sub}(A)^H * X = \text{sub}(B)$ を X について解く。
<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>bwl</i>	(グローバル) INTEGER。A の帯内にある劣対角成分の数 ($0 \leq bwl \leq n-1$)。
<i>bwu</i>	(グローバル) INTEGER。A の帯内にある優対角成分の数 ($0 \leq bwu \leq n-1$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。

<i>a, b</i>	(ローカル) REAL (psdbtrs の場合) DOUBLE PRECISION (pddbtrs の場合) COMPLEX (pcdbtrs の場合) DOUBLE COMPLEX (pzdbtrs の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(lld_a, LOC_c(ja+n-1))$ と $b(lld_b, LOC_c(nrhs))$ へのポインタ。 呼び出し時に、配列 <i>a</i> は、p?dbtrf によって行われた帯行列 <i>A</i> の <i>LU</i> 因子分解の詳細を含む。 呼び出し時に、配列 <i>b</i> は右辺の分散行列 <i>sub(B)</i> のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。 (<i>A</i> のすべて、または <i>A</i> の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca</i> (<i>dtype_</i>) = 501 の場合、 <i>dlen_</i> ≥ 7。 <i>desca</i> (<i>dtype_</i>) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>ib</i>	(グローバル) INTEGER。 (<i>B</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。 <i>descb</i> (<i>dtype_</i>) = 502 の場合、 <i>dlen_</i> ≥ 7。 <i>descb</i> (<i>dtype_</i>) = 1 の場合、 <i>dlen_</i> ≥ 9。
<i>af, work</i>	(ローカル) REAL (psdbtrs の場合) DOUBLE PRECISION (pddbtrs の場合) COMPLEX (pcdbtrs の場合) DOUBLE COMPLEX (pzdbtrs の場合) それぞれ、次元 (<i>laf</i>) と (<i>lwork</i>) の配列。 配列 <i>af</i> は、補助非零要素空間を含む。非零要素は、因子分解ルーチン p?dbtrf で作成され、 <i>af</i> に格納される。 配列 <i>work</i> はワークスペース配列である。
<i>laf</i>	(ローカル) INTEGER。配列 <i>af</i> の次元。 $laf \geq NB * (bw1 + bwu) + 6 * (\max(bw1, bwu))^2$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq (\max(bw1, bwu))^2$ でなければならない。

出力パラメーター

<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
----------	---

<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	INTEGER。 <code>info=0</code> の場合、実行は正常に終了したことを示す。 <code>info<0</code> の場合： <code>i</code> 番目の引数が配列で、 <code>j</code> 番目の値が不正だった場合、 <code>info = -(i*100+j)</code> 。 <code>i</code> 番目の引数がスカラーで値が不正だった場合、 <code>info = -i</code> 。

p?trtrs

三角分散行列の連立1次方程式を解く。

構文

```
call pstrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, info)
call pdtrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, info)
call pctrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, info)
call pztrtrs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, info)
```

説明

このルーチンは、以下のいずれかの連立方程式を X について解く。

$$\text{sub}(A) * X = \text{sub}(B)、$$

$$(\text{sub}(A))^T * X = \text{sub}(B)、\text{または}$$

$$(\text{sub}(A))^H * X = \text{sub}(B)$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ は次数 n の三角分散行列、 $\text{sub}(B)$ は分散行列 $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ である。

$\text{sub}(A)$ が非特異であることを確認するチェックが行われる。

入力パラメーター

<code>uplo</code>	(グローバル) CHARACTER*1。 'U' または 'L' でなければならない。 <code>sub(A)</code> が上三角と下三角のどちらであることを示す。 <code>uplo = 'U'</code> の場合、 <code>sub(A)</code> は上三角である。 <code>uplo = 'L'</code> の場合、 <code>sub(A)</code> は下三角である。
<code>trans</code>	(グローバル) CHARACTER*1。 'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <code>trans = 'N'</code> の場合、 $\text{sub}(A) * X = \text{sub}(B)$ を X について解く。

	$trans = 'T'$ の場合、 $sub(A)^T * X = sub(B)$ を X について解く。
	$trans = 'C'$ の場合、 $sub(A)^H * X = sub(B)$ を X について解く。
<i>diag</i>	(グローバル) CHARACTER*1. 'N' または 'U' でなければならない。 $diag = 'N'$ の場合、 $sub(A)$ は単位三角ではない。 $diag = 'U'$ の場合、 $sub(A)$ は単位三角である。
<i>n</i>	(グローバル) INTEGER. 分散部分行列 $sub(A)$ の次数 ($n \geq 0$).
<i>nrhs</i>	(グローバル) INTEGER. 右辺の数。すなわち、分散行列 $sub(B)$ の列数 ($nrhs \geq 0$).
<i>a, b</i>	(ローカル) REAL (pstrtrs の場合) DOUBLE PRECISION (pdtrtrs の場合) COMPLEX (pctrtrs の場合) DOUBLE COMPLEX (pztrtrs の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(lld_a, LOC_c(ja+n-1))$ と $b(lld_b, LOC_c(jb+nrhs-1))$ へのポインター。 配列 a は、三角分散行列 $sub(A)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $sub(A)$ の先頭の $n \times n$ 上三角部分は上三角行列を含む。 $sub(A)$ の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $sub(A)$ の先頭の $n \times n$ の下三角部分に、下三角行列を格納する。 $sub(A)$ の厳密な上三角部分は参照されない。 $diag = 'U'$ の場合、 $sub(A)$ の対角成分も参照されず、1 と仮定される。 呼び出し時に、配列 b は右辺の分散行列 $sub(B)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER. それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>ib, jb</i>	(グローバル) INTEGER. それぞれ、部分行列 $sub(B)$ の最初の行と最初の列を示す、グローバル配列 B の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。

出力パラメーター

<i>b</i>	終了時に、 $info=0$ の場合、 $sub(B)$ は解の行列 X によって上書きされる。
<i>info</i>	INTEGER. $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

$info > 0$ の場合 :

$info = i$ の場合、 $sub(A)$ の i 番目の対角成分はゼロで、部分行列が特異であり、解 X が計算されていないことを示す。

条件数を推定するためのルーチン

このセクションでは、行列の条件数を推定するための ScaLAPACK ルーチンについて説明する。条件数は、連立 1 次方程式の解の誤差を解析するために使用される。行列がほとんど特異である (ゼロに近い) 場合は、条件数が非常に大きくなる場合がある。このため、これらのルーチンでは、実際には条件数の逆数を計算する。

p?gecon

一般分散行列の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

構文

```
call psgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             iwork, liwork, info )
call pdgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             iwork, liwork, info )
call pcgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )
call pzgecon( norm, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )
```

説明

このルーチンは、[p?getrf](#) によって行われた LU 因子分解を使用して、一般分散実 / 複素行列 $sub(A) = A(ia:ia+n-1, ja:ja+n-1)$ の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

推定値が $\|(sub(A))^{-1}\|$ について得られる。次に、条件数の逆数が、

$$rcond = \frac{1}{\|sub(A)\| \times \|(sub(A))^{-1}\|} \text{ として計算される。}$$

入力パラメーター

norm (グローバル) CHARACTER*1. '1'、'0'、または 'I' のいずれかでなければならない。

1- ノルムの条件数と無限ノルムの条件数のどちらを使用するかを指定する。

norm = '1' または '0' の場合、1- ノルムが使用される。

norm = 'I' の場合、無限ノルムが使用される。

- n* (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
- a* (ローカル)
 REAL (psgecon の場合)
 DOUBLE PRECISION (pdgecon の場合)
 COMPLEX (pcgecon の場合)
 DOUBLE COMPLEX (pzgecon の場合)
 ローカルメモリーにある、次元 $a(\text{lld_a}, \text{LOC}_c(\text{ja}+n-1))$ の配列へのポインター。
 配列 *a* は、因子分解 $\text{sub}(A) = PLU$ で得られた係数 *L* と *U* のローカル部分を含む。*L* の単位対角成分は格納されない。
- ia, ja* (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。
- desca* (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *A* の配列ディスクリプター。
- anorm* (グローバル) REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
norm = '1' または '0' の場合、元の分散行列 $\text{sub}(A)$ の 1- ノルム。
norm = 'I' の場合、元の分散行列 $\text{sub}(A)$ の無限ノルム。
- work* (ローカル)
 REAL (psgecon の場合)
 DOUBLE PRECISION (pdgecon の場合)
 COMPLEX (pcgecon の場合)
 DOUBLE COMPLEX (pzgecon の場合)
 次元 (*lwork*) の配列 *work* はワークスペース配列。
- lwork* (ローカルまたはグローバル) INTEGER。配列 *work* の次元。
- 実数型の場合:

$$lwork \geq 2 * \text{LOC}_r(n + \text{mod}(ia-1, mb_a)) +$$

$$2 * \text{LOC}_c(n + \text{mod}(ja-1, nb_a)) +$$

$$\max(2, \max(nb_a * \max(1, \text{ceil}(\text{NPROW}-1, \text{NPCOL})),$$

$$\text{LOC}_c(n + \text{mod}(ja-1, nb_a)) + nb_a * \max(1, \text{ceil}(\text{NPCOL}-1, \text{NPROW})))$$
 でなければならない。
- 複素数型の場合:

$$lwork \geq 2 * \text{LOC}_r(n + \text{mod}(ia-1, mb_a)) +$$

$$\max(2, \max(nb_a * \text{ceil}(\text{NPROW}-1, \text{NPCOL}),$$

$$\text{LOC}_c(n + \text{mod}(ja-1, nb_a)) + nb_a * \text{ceil}(\text{NPCOL}-1, \text{NPROW})))$$
 でなければならない。
- LOC_r および LOC_c の値は、ScaLAPACK ツール関数 `numroc` を使用して計算できる。NPROW および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。
- iwork* (ローカル) INTEGER。
 ワークスペース配列、次元は (*liwork*)。実数型でのみ使用される。

<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n+\text{mod}(ia-1, mb_a))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (<i>pcgecon</i> の場合) DOUBLE PRECISION (<i>pzgecon</i> の場合) ワークスペース配列、次元は (<i>lrwork</i>)。複素数型でのみ使用される。
<i>lrwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>rwork</i> の次元。複素数型でのみ使用される。 $lrwork \geq 2*LOC_c(n+\text{mod}(ja-1, nb_a))$ でなければならない。

出力パラメーター

<i>rcond</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 分散行列 <i>sub(A)</i> の条件数の逆数。説明を参照。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork(1)</i>	終了時に、 <i>iwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork(1)</i>	終了時に、 <i>rwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?pocon

対称/エルミート正定値分散行列の条件数の逆数を (1- ノルムで) 推定する。

構文

```
call pspocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             iwork, liwork, info )
call pdpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             iwork, liwork, info )
call pcpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )
call pzpocon( uplo, n, a, ia, ja, desca, anorm, rcond, work, lwork,
             rwork, lrwork, info )
```

説明

このルーチンは、[p?potrf](#) によって行われたコレスキー因子分解 $\text{sub}(A) = U^H U$ または $\text{sub}(A) = LL^H$ を使用して、実対称 / 複素エルミート正定値分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ の条件数の逆数を (1- ノルムで) 推定する。

推定値が $\|(\text{sub}(A))^{-1}\|$ について得られる。次に、条件数の逆数が、

$$rcond = \frac{1}{\|\text{sub}(A)\| \times \|(\text{sub}(A))^{-1}\|}$$

入力パラメーター

- uplo* (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
 $\text{sub}(A)$ に格納されている係数が上三角と下三角のどちらであるかを指定する。
uplo = 'U' の場合、 $\text{sub}(A)$ には、コレスキー因子分解 $\text{sub}(A) = U^H U$ の上三角係数 U が格納される。
uplo = 'L' の場合、 $\text{sub}(A)$ には、コレスキー因子分解 $\text{sub}(A) = LL^H$ の下三角係数 L が格納される。
- n* (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
- a* (ローカル)
 REAL (pspocon の場合)
 DOUBLE PRECISION (pdpocon の場合)
 COMPLEX (pcpocon の場合)
 DOUBLE COMPLEX (pzpocon の場合)
 ローカルメモリーにある、次元 $a(\text{lld}_a, \text{LOC}_c(\text{ja}+n-1))$ の配列へのポインター。
 配列 a は、[p?potrf](#) によって行われたコレスキー因子分解 $\text{sub}(A) = U^H U$ または $\text{sub}(A) = LL^H$ で得られた係数 U または L を含む。
- ia, ja* (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
- desca* (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
- anorm* (グローバル) REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 対称 / エルミート分散行列 $\text{sub}(A)$ の 1- ノルム。
- work* (ローカル)
 REAL (pspocon の場合)
 DOUBLE PRECISION (pdpocon の場合)
 COMPLEX (pcpocon の場合)
 DOUBLE COMPLEX (pzpocon の場合)
 次元 ($lwork$) の配列 $work$ はワークスペース配列。
- lwork* (ローカルまたはグローバル) INTEGER。配列 $work$ の次元。

実数型の場合:

$$lwork \geq 2*LOC_r(n+\text{mod}(ia-1, mb_a)) + \\ 2*LOC_c(n+\text{mod}(ja-1, nb_a)) + \\ \max(2, \max(nb_a*\text{ceil}(NPROW-1, NPCOL), \\ LOC_c(n+\text{mod}(ja-1, nb_a)) + \\ nb_a*\text{ceil}(NPCOL-1, NPROW))) \text{ でなければならない。}$$

複素数型の場合:

$$lwork \geq 2*LOC_r(n+\text{mod}(ia-1, mb_a)) + \\ \max(2, \max(nb_a*\max(1, \text{ceil}(NPROW-1, NPCOL)), \\ LOC_c(n+\text{mod}(ja-1, nb_a)) + \\ nb_a*\max(1, \text{ceil}(NPCOL-1, NPROW))) \text{ でなければならない。}$$

<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 <i>liwork</i> \geq $LOC_r(n+\text{mod}(ia-1, mb_a))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (<i>pcpocon</i> の場合) DOUBLE PRECISION (<i>pzpocon</i> の場合) ワークスペース配列、次元は (<i>lrwork</i>)。複素数型でのみ使用される。
<i>lrwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>rwork</i> の次元。複素数型でのみ使用される。 <i>lrwork</i> \geq $2*LOC_c(n+\text{mod}(ja-1, nb_a))$ でなければならない。

出力パラメーター

<i>rcond</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 分散行列 <i>sub(A)</i> の条件数の逆数。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork(1)</i>	終了時に、 <i>iwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork(1)</i>	終了時に、 <i>rwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。

info < 0 の場合:

i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。

p?trcon

三角分散行列の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

構文

```
call pstrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,
             iwork, liwork, info )
call pdtrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,
             iwork, liwork, info )
call pctrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,
             rwork, lrwork, info )
call pztrcon( norm, uplo, diag, n, a, ia, ja, desca, rcond, work, lwork,
             rwork, lrwork, info )
```

説明

このルーチンは、三角分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ の条件数の逆数を、1- ノルムまたは無限ノルムのいずれかで推定する。

$\text{sub}(A)$ のノルムが計算され、推定値が $\|(\text{sub}(A))^{-1}\|$ に対して得られる。次に、条件数の逆数が

$$rcond = \frac{1}{\|\text{sub}(A)\| \times \|(\text{sub}(A))^{-1}\|} \text{ として計算される。}$$

入力パラメーター

norm (グローバル) CHARACTER*1。'1'、'0'、または 'I' のいずれかでなければならない。
1- ノルムの条件数と無限ノルムの条件数のどちらを使用するかを指定する。
norm = '1' または '0' の場合、1- ノルムが使用される。
norm = 'I' の場合、無限ノルムが使用される。

uplo (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合、 $\text{sub}(A)$ は上三角部分である。
uplo = 'L' の場合、 $\text{sub}(A)$ は下三角部分である。

diag (グローバル) CHARACTER*1。'N' または 'U' でなければならない。
diag = 'N' の場合、 $\text{sub}(A)$ は単位三角ではない。
diag = 'U' の場合、 $\text{sub}(A)$ は単位三角である。

n (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

<i>a</i>	(ローカル) REAL (pstrcon の場合) DOUBLE PRECISION (pdtrcon の場合) COMPLEX (pctrcon の場合) DOUBLE COMPLEX (pztrcon の場合) ローカルメモリーにある、次元 $a(lld_a, LOC_c(ja+n-1))$ の配列へのポインター。 配列 <i>a</i> は、三角分散行列 $sub(A)$ のローカル部分を含む。 <i>uplo</i> = 'U' の場合、この分散行列の先頭の $n \times n$ 上三角部分は上三角行列を含む。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、この分散行列の先頭の $n \times n$ 下三角部分は下三角行列を含む。厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合、 $sub(A)$ の対角成分も参照されず、1 と仮定される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (pstrcon の場合) DOUBLE PRECISION (pdtrcon の場合) COMPLEX (pctrcon の場合) DOUBLE COMPLEX (pztrcon の場合) 次元 (<i>lwork</i>) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb_a)) +$ $LOC_c(n + \text{mod}(ja-1, nb_a)) +$ $\max(2, \max(nb_a * \max(1, \text{ceil}(NPROW-1, NPCOL)),$ $LOC_c(n + \text{mod}(ja-1, nb_a)) +$ $nb_a * \max(1, \text{ceil}(NPCOL-1, NPROW))) \text{ でなければならない。}$ 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb_a)) +$ $\max(2, \max(nb_a * \text{ceil}(NPROW-1, NPCOL),$ $LOC_c(n + \text{mod}(ja-1, nb_a)) +$ $nb_a * \text{ceil}(NPCOL-1, NPROW))) \text{ でなければならない。}$
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (pcpocon の場合) DOUBLE PRECISION (pzpocon の場合) ワークスペース配列、次元は (<i>lrwork</i>)。複素数型でのみ使用される。

lwork (ローカルまたはグローバル) INTEGER。
配列 *rwork* の次元。複素数型でのみ使用される。
 $lwork \geq LOC_c(n + \text{mod}(ja-1, nb_a))$ でなければならない。

出力パラメーター

rcond (グローバル) REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
分散行列 $\text{sub}(A)$ の条件数の逆数。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

iwork(1) 終了時に、*iwork*(1) には、最適なパフォーマンスを得るために必要な *liwork* の最小値が格納される (実数型の場合)。

rwork(1) 終了時に、*rwork*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される (複素数型の場合)。

info (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
 $info = -(i*100+j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、
 $info = -i$ 。

解の精度の改善と誤差の推定

このセクションでは、算出された連立 1 次方程式の解の精度を改善し、解の誤差を推定するための ScaLAPACK ルーチンについて説明する。これらのルーチン呼び出す前に、連立方程式の行列を因子分解し、解を計算する必要がある ([「行列の因子分解用のルーチン」](#) および [「連立 1 次方程式を解くためのルーチン」](#) を参照)。

p?gerfs

連立 1 次方程式の算出された解を改善し、解の誤差範囲と後退誤差推定を提供する。

構文

```
call psgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
            iwork, liwork, info)

call pdgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
            iwork, liwork, info)

call pcgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
            rwork, lrwork, info)
```

```
call pzgerfs(trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            ipiv, b, ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork,
            rwork, lrwork, info)
```

説明

このルーチンは、以下のいずれかの連立 1 次方程式の算出された解を改善し、解の誤差範囲と後退誤差推定を提供する。

$$\begin{aligned} \text{sub}(A) * \text{sub}(X) &= \text{sub}(B), \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B), \text{ または} \\ \text{sub}(A)^H * \text{sub}(X) &= \text{sub}(B) \end{aligned}$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ 、 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+nrhs-1)$ 、および $\text{sub}(X) = X(\text{ix}:\text{ix}+n-1, \text{jx}:\text{jx}+nrhs-1)$ である。

入力パラメーター

- trans* (グローバル) CHARACTER*1. 'N'、'T'、または 'C' のいずれかでなければならない。
連立方程式の形式を指定する。
trans = 'N' の場合、連立方程式の形式は、 $\text{sub}(A) * \text{sub}(X) = \text{sub}(B)$ (転置なし) である。
trans = 'T' の場合、連立方程式の形式は、 $\text{sub}(A)^T * \text{sub}(X) = \text{sub}(B)$ (転置あり) である。
trans = 'C' の場合、連立方程式の形式は、 $\text{sub}(A)^H * \text{sub}(X) = \text{sub}(B)$ (共役転置) である。
- n* (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
- nrhs* (グローバル) INTEGER。右辺の数。すなわち、 $\text{sub}(B)$ と $\text{sub}(X)$ の列数 ($nrhs \geq 0$)。
- a, af, b, x* (ローカル)
REAL (psgerfs の場合)
DOUBLE PRECISION (pdgerfs の場合)
COMPLEX (pcgerfs の場合)
DOUBLE COMPLEX (pzgerfs の場合)
それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(\text{lld}_a, \text{LOC}_c(\text{ja}+n-1))$ 、 $af(\text{lld}_{af}, \text{LOC}_c(\text{jaf}+n-1))$ 、 $b(\text{lld}_b, \text{LOC}_c(\text{jb}+nrhs-1))$ 、および $x(\text{lld}_x, \text{LOC}_c(\text{jx}+nrhs-1))$ へのポインター。
配列 *a* は、分散行列 $\text{sub}(A)$ のローカル部分を含む。
配列 *af* は、[p?getrf](#) によって計算された行列 $\text{sub}(A) = PLU$ の分散要素のローカル部分を含む。
配列 *b* は、右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。
呼び出し時に、配列 *x* は解の分散行列 $\text{sub}(X)$ のローカル部分を含む。
- ia, ja* (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>iaf, jaf</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(AF)</i> の最初の行と最初の列を示す、グローバル配列 <i>AF</i> の行インデックスと列インデックス。
<i>descaf</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>AF</i> の配列ディスクリプター。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(B)</i> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。
<i>ix, jx</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(X)</i> の最初の行と最初の列を示す、グローバル配列 <i>X</i> の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>X</i> の配列ディスクリプター。
<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は $LOC_r(m_{af}) + mb_{af}$ 。 この配列は、 p?getrf によって計算されたピボット情報を含む。 <i>ipiv(i)=j</i> の場合、ローカル行 <i>i</i> は、グローバル行 <i>j</i> と交換される。 この配列は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル) REAL (<i>psgerfs</i> の場合) DOUBLE PRECISION (<i>pdgerfs</i> の場合) COMPLEX (<i>pcgerfs</i> の場合) DOUBLE COMPLEX (<i>pzgerfs</i> の場合) 次元 (<i>lwork</i>) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。
<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は (<i>liwork</i>)。実数型でのみ使用される。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。実数 ^ でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib-1, mb_b))$ でなければならない。
<i>rwork</i>	(ローカル) REAL (<i>pcgerfs</i> の場合) DOUBLE PRECISION (<i>pzgerfs</i> の場合) ワークスペース配列、次元は (<i>lrwork</i>)。複素数型でのみ使用される。
<i>lrwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>rwork</i> の次元。複素数型でのみ使用される。 $lrwork \geq LOC_r(n + \text{mod}(ib-1, mb_b))$ でなければならない。

出力パラメーター

<i>x</i>	終了時に、改善された解ベクトルが格納される。
<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元はそれぞれ $LOC_c(jb+nrhs-1)$ 。 配列 <i>ferr</i> には、 <i>sub(X)</i> の各解ベクトル用に推定された前進誤差範囲が格納される。 <i>XTRUE</i> が <i>sub(X)</i> に対応する真の解の場合、 <i>ferr</i> は、 $(sub(X) - XTRUE)$ の最大成分の大きさを <i>sub(X)</i> の最大成分の大きさを割った推定上限である。この推定値は <i>rcond</i> に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。 この配列は、分散行列 <i>X</i> に関連付けられる。 配列 <i>berr</i> は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、 <i>sub(X)</i> が正確な解となる <i>sub(A)</i> または <i>sub(B)</i> の任意のエントリーにおける最小相対変化)。この配列は、分散行列 <i>X</i> に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork(1)</i>	終了時に、 <i>iwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork(1)</i>	終了時に、 <i>rwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = $-(i*100+j)$ 。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = $-i$ 。

p?porfs

対称/エルミート正定値分散行列の連立1次方程式の算出された解を改善し、解の誤差範囲と後退誤差推定を提供する。

構文

```
call psporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
            ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork,
            liwork, info)

call pdporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
            ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork,
            liwork, info)
```

```

call pcporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork,
             lrwork, info)
call pzporfs(uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf, b,
             ib, jb, descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork,
             lrwork, info)

```

説明

ルーチン p?porfs は、連立 1 次方程式 $\text{sub}(A) * \text{sub}(X) = \text{sub}(B)$ の算出された解を改善する。

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ は実対称 / 複素エルミート正定値分散行列、
 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ 、
 $\text{sub}(X) = X(\text{ix}:\text{ix}+n-1, \text{jx}:\text{jx}+\text{nrhs}-1)$

はそれぞれ、右辺と解の部分行列である。

また、このルーチンは、解の誤差範囲と後退誤差推定を提供する。

入力パラメーター

uplo (グローバル) CHARACTER*1. 'U' または 'L' でなければならない。
 対称 / エルミート行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。
 uplo = 'U' の場合、 $\text{sub}(A)$ は上三角部分である。
 uplo = 'L' の場合、 $\text{sub}(A)$ は下三角部分である。

n (グローバル) INTEGER. 分散行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

nrhs (グローバル) INTEGER. 右辺の数。すなわち、 $\text{sub}(B)$ と $\text{sub}(X)$ の列数 ($\text{nrhs} \geq 0$)。

a, af, b, x (ローカル)
 REAL (psporfs の場合)
 DOUBLE PRECISION (pdporfs の場合)
 COMPLEX (pcporfs の場合)
 DOUBLE COMPLEX (pzporfs の場合)
 それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(\text{l1d}_a, \text{LOC}_c(\text{ja}+n-1))$ 、 $af(\text{l1d}_{af}, \text{LOC}_c(\text{ja}+n-1))$ 、 $b(\text{l1d}_b, \text{LOC}_c(\text{jb}+\text{nrhs}-1))$ 、および $x(\text{l1d}_x, \text{LOC}_c(\text{jx}+\text{nrhs}-1))$ へのポインター。
 配列 a は、 $n \times n$ の対称 / エルミート分散行列 $\text{sub}(A)$ のローカル部分を含む。
 uplo = 'U' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。厳密な下三角部分は参照されない。
 uplo = 'L' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
 配列 af は、p?potrf によって行われたコレスキー因子分解 $\text{sub}(A) = LL^H$ または $\text{sub}(A) = U^H U$ で得られた係数 L または U を含む。
 呼び出し時に、配列 b は右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。

呼び出し時に、配列 x は解のベクトル $\text{sub}(X)$ のローカル部分を含む。

ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
iaf, jaf	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(AF)$ の最初の行と最初の列を示す、グローバル配列 AF の行インデックスと列インデックス。
$descaf$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 AF の配列ディスクリプター。
ib, jb	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 B の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。
ix, jx	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 X の行インデックスと列インデックス。
$descx$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 X の配列ディスクリプター。
$work$	(ローカル) REAL (psporfs の場合) DOUBLE PRECISION (pdporfs の場合) COMPLEX (pcporfs の場合) DOUBLE COMPLEX (pzporfs の場合) 次元 ($lwork$) の配列 $work$ はワークスペース配列。
$lwork$	(ローカル) INTEGER。配列 $work$ の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia - 1, mb_a))$ でなければならない。 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia - 1, mb_a))$ でなければならない。
$iwork$	(ローカル) INTEGER。 ワークスペース配列、次元は ($liwork$)。実数型でのみ使用される。
$liwork$	(ローカルまたはグローバル) INTEGER。 配列 $iwork$ の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib - 1, mb_b))$ でなければならない。
$rwork$	(ローカル) REAL (pcporfs の場合) DOUBLE PRECISION (pzporfs の場合) ワークスペース配列、次元は ($lrwork$)。複素数型でのみ使用される。
$lrwork$	(ローカルまたはグローバル) INTEGER。 配列 $rwork$ の次元。複素数型でのみ使用される。 $lrwork \geq LOC_r(n + \text{mod}(ib - 1, mb_b))$ でなければならない。

出力パラメーター

x	終了時に、改善された解ベクトルが格納される。
-----	------------------------

<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元はそれぞれ $LOC_c(jb+nrhs-1)$ 。 配列 <i>ferr</i> には、 $\text{sub}(X)$ の各解ベクトル用に推定された前進誤差範囲が格納される。 X_{TRUE} が $\text{sub}(X)$ に対応する真の解の場合、 <i>ferr</i> は、 $(\text{sub}(X) - X_{TRUE})$ の最大成分の大きさを $\text{sub}(X)$ の最大成分の大きさを割った推定上限である。この推定値は <i>rcond</i> に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。 この配列は、分散行列 X に関連付けられる。 配列 <i>berr</i> は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、 $\text{sub}(X)$ が正確な解となる $\text{sub}(A)$ または $\text{sub}(B)$ の任意のエントリにおける最小相対変化)。この配列は、分散行列 X に関連付けられる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork</i> (1)	終了時に、 <i>iwork</i> (1) には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork</i> (1)	終了時に、 <i>rwork</i> (1) には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?trrfs

分散三角係数行列の連立1次方程式の解の誤差範囲と後退誤差推定を提供する。

構文

```
call pstrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork, liwork, info)
call pdtrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, iwork, liwork, info)
call pctrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork, lrwork, info)
call pztrrfs(uplo, trans, diag, n, nrhs, a, ia, ja, desca, b, ib, jb,
            descb, x, ix, jx, descx, ferr, berr, work, lwork, rwork, lrwork, info)
```

説明

ルーチン `p?trrfcs` は、以下のいずれかの連立 1 次方程式の解の誤差範囲と後退誤差推定を提供する。

$$\begin{aligned} \text{sub}(A) * \text{sub}(X) &= \text{sub}(B), \\ \text{sub}(A)^T * \text{sub}(X) &= \text{sub}(B), \text{ または} \\ \text{sub}(A)^H * \text{sub}(X) &= \text{sub}(B). \end{aligned}$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ は三角行列、 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ 、および $\text{sub}(X) = X(\text{ix}:\text{ix}+n-1, \text{jx}:\text{jx}+\text{nrhs}-1)$ である。

解の行列 X は、このルーチンに入る前に、`p?trtrs` または他の手段によって計算されていないなければならない。後退誤差が改善されないことがあるため、ルーチン `p?trrfcs` は精度の改善を繰り返して行わない。

入力パラメーター

<code>uplo</code>	(グローバル) CHARACTER*1。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 <code>sub(A)</code> は上三角である。 <code>uplo = 'L'</code> の場合、 <code>sub(A)</code> は下三角である。
<code>trans</code>	(グローバル) CHARACTER*1。'N'、'T'、または 'C' のいずれかでなければならない。 連立方程式の形式を指定する。 <code>trans = 'N'</code> の場合、連立方程式の形式は、 $\text{sub}(A) * \text{sub}(X) = \text{sub}(B)$ (転置なし) である。 <code>trans = 'T'</code> の場合、連立方程式の形式は、 $\text{sub}(A)^T * \text{sub}(X) = \text{sub}(B)$ (転置あり) である。 <code>trans = 'C'</code> の場合、連立方程式の形式は、 $\text{sub}(A)^H * \text{sub}(X) = \text{sub}(B)$ (共役転置) である。
<code>diag</code>	CHARACTER*1。'N' または 'U' でなければならない。 <code>diag = 'N'</code> の場合、 <code>sub(A)</code> は単位三角ではない。 <code>diag = 'U'</code> の場合、 <code>sub(A)</code> は単位三角である。
<code>n</code>	(グローバル) INTEGER。分散行列 <code>sub(A)</code> の次数 ($n \geq 0$)。
<code>nrhs</code>	(グローバル) INTEGER。右辺の数。すなわち、 <code>sub(B)</code> と <code>sub(X)</code> の列数 ($\text{nrhs} \geq 0$)。
<code>a, b, x</code>	(ローカル) REAL (<code>pstrrfcs</code> の場合) DOUBLE PRECISION (<code>pdtrrfcs</code> の場合) COMPLEX (<code>pctrdfcs</code> の場合) DOUBLE COMPLEX (<code>pztrrfcs</code> の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(\text{lld}_a, \text{LOC}_c(\text{ja}+n-1))$ 、 $b(\text{lld}_b, \text{LOC}_c(\text{jb}+\text{nrhs}-1))$ 、および $x(\text{lld}_x, \text{LOC}_c(\text{jx}+\text{nrhs}-1))$ へのポインター。

配列 a は、元の三角分散行列 $\text{sub}(A)$ のローカル部分を含む。
 $\text{uplo} = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。厳密な下三角部分は参照されない。
 $\text{uplo} = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
 $\text{diag} = 'U'$ の場合、 $\text{sub}(A)$ の対角成分も参照されず、1 と仮定される。
 呼び出し時に、配列 b は右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。

呼び出し時に、配列 x は解のベクトル $\text{sub}(X)$ のローカル部分を含む。

ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
ib, jb	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(B)$ の最初の行と最初の列を示す、グローバル配列 B の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。
ix, jx	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 X の行インデックスと列インデックス。
$descx$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 X の配列ディスクリプター。
$work$	(ローカル) REAL ($pstrrrfs$ の場合) DOUBLE PRECISION ($pdtrrrfs$ の場合) COMPLEX ($pctrrrfs$ の場合) DOUBLE COMPLEX ($pztrrrfs$ の場合) 次元 ($lwork$) の配列 $work$ はワークスペース配列。
$lwork$	(ローカル) INTEGER。配列 $work$ の次元。 実数型の場合: $lwork \geq 3 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。 複素数型の場合: $lwork \geq 2 * LOC_r(n + \text{mod}(ia-1, mb_a))$ でなければならない。
$iwork$	(ローカル) INTEGER。 ワークスペース配列、次元は ($liwork$)。実数型でのみ使用される。
$liwork$	(ローカルまたはグローバル) INTEGER。 配列 $iwork$ の次元。実数型でのみ使用される。 $liwork \geq LOC_r(n + \text{mod}(ib-1, mb_b))$ でなければならない。
$rwork$	(ローカル) REAL ($pctrrrfs$ の場合) DOUBLE PRECISION ($pztrrrfs$ の場合) ワークスペース配列、次元は ($lrwork$)。複素数型でのみ使用される。
$lrwork$	(ローカルまたはグローバル) INTEGER。 配列 $rwork$ の次元。複素数型でのみ使用される。 $lrwork \geq LOC_r(n + \text{mod}(ib-1, mb_b))$ でなければならない。

出力パラメーター

<i>ferr</i> , <i>berr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元はそれぞれ $LOC_c(jb+nrhs-1)$ 。 配列 <i>ferr</i> には、 <i>sub(X)</i> の各解ベクトル用に推定された前進誤差範囲が格納される。 XTRUE が <i>sub(X)</i> に対応する真の解の場合、 <i>ferr</i> は、(<i>sub(X)</i> - XTRUE) の最大成分の大きさを <i>sub(X)</i> の最大成分の大きさを割った推定上限である。この推定値は <i>rcond</i> に対する推定値と同程度の信頼性があり、ほとんどの場合、実際の誤差よりも少し多めに推定される。 この配列は、分散行列 <i>X</i> に関連付けられる。 配列 <i>berr</i> は、各解ベクトルの成分ごとの相対後退誤差を含む (すなわち、 <i>sub(X)</i> が正確な解となる <i>sub(A)</i> または <i>sub(B)</i> の任意のエントリーにおける最小相対変化)。この配列は、分散行列 <i>X</i> に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork(1)</i>	終了時に、 <i>iwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される (実数型の場合)。
<i>rwork(1)</i>	終了時に、 <i>rwork(1)</i> には、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値が格納される (複素数型の場合)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

行列の反転用のルーチン

このセクションでは、以前に得られた因子分解に基づいて行列の逆行列を計算する ScaLAPACK ルーチンについて説明する。連立方程式 $Ax = b$ を解く場合、最初に A^{-1} を計算し、次に行列・ベクトルの積 $x = A^{-1}b$ を計算しないように注意する。代わりに、ソルバールーチン呼び出す(「[連立1次方程式を解くためのルーチン](#)」を参照)。この方が効率が良く、高い精度の結果が得られる。

p?getri

LU 因子分解された分散行列の逆行列を計算する。

構文

```
call psgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pdgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pcgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
call pzgetri(n, a, ia, ja, desca, ipiv, work, lwork, iwork, liwork, info)
```

説明

このルーチンは、p?getrf によって行われた LU 因子分解を使用して、一般分散行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ の逆行列を計算する。この手法は、 U を反転した後で方程式を解いて、 $\text{Inv}A$ で示された $\text{sub}(A)$ の逆行列を計算する。

$$\text{Inv}A * L = U^{-1}$$

($\text{Inv}A$ の場合)。

入力パラメーター

n (グローバル) INTEGER。処理される行数と列数。すなわち、分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

a (ローカル)
 REAL (psgetri の場合)
 DOUBLE PRECISION (pdgetri の場合)
 COMPLEX (pcgetri の場合)
 DOUBLE COMPLEX (pzgetri の場合)
 ローカルメモリーにある、ローカル次元 $a(\text{lld}_a, \text{LOC}_c(ja+n-1))$ の配列へのポインタ。
 呼び出し時に、配列 a は、p?getrf によって行われた因子分解 $\text{sub}(A) = PLU$ で得られた係数 L と U のローカル部分を含む。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。

$desca$ (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

<i>work</i>	(ローカル) REAL (psgetri の場合) DOUBLE PRECISION (pdgetri の場合) COMPLEX (pcgetri の場合) DOUBLE COMPLEX (pzgetri の場合) 次元 (<i>lwork</i>) の配列 <i>work</i> はワークスペース配列。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 $lwork \geq LOC_r(n + \text{mod}(ia-1, mb_a)) * nb_a$ でなければならない。 配列 <i>work</i> は、 <i>sub(A)</i> の列ブロック全体を維持するために使用される。
<i>iwork</i>	(ローカル) INTEGER。 ピボットの物理的な転置に使用されるワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>iwork</i> の次元。 最小値 <i>liwork</i> は、以下のコードによって決定される。 If NPROW == NPCOL then <i>liwork</i> = LOCC(<i>n_a</i> + mod(<i>ja</i> -1, <i>nb_a</i>)) + <i>nb_a</i> Else <i>liwork</i> = LOCC(<i>n_a</i> + mod(<i>ja</i> -1, <i>nb_a</i>)) + max(ceil(ceil(LOCr(<i>m_a</i>)/ <i>mb_a</i>)/(lcm/NPROW)), <i>nb_a</i>) End if ここで、 <i>lcm</i> はプロセス列とプロセス行 (NPROW および NPCOL) の最小公倍数である。

出力パラメーター

<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は ($LOC_r(m_a) + mb_a$)。 この配列には、ピボット情報が格納される。 <i>ipiv</i> (<i>i</i>)= <i>j</i> の場合、ローカル行 <i>i</i> は、グローバル行 <i>j</i> と交換される。 この配列は、分散行列 <i>A</i> に関連付けられる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>iwork</i> (1)	終了時に、 <i>iwork</i> (1) には、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>i</i> の場合、 <i>U</i> (<i>i</i> , <i>i</i>) が完全に 0 である。因子分解は完了したが、係数 <i>U</i> は完全に特異で、連立方程式の解の算出に使用するとゼロ除算が発生する。

p?potri

対称/エルミート正定値分散行列の逆行列を計算する。

構文

```
call pspotri(uplo, n, a, ia, ja, desca, info)
call pdpotri(uplo, n, a, ia, ja, desca, info)
call pcpotri(uplo, n, a, ia, ja, desca, info)
call pzpotri(uplo, n, a, ia, ja, desca, info)
```

説明

このルーチンは、p?potrf によって行われたコレスキー因子分解 $\text{sub}(A) = U^H U$ または $\text{sub}(A) = LL^H$ を使用して、実対称/複素エルミート正定値分散行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ の逆行列を計算する。

入力パラメーター

uplo (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
対称/エルミート行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。

uplo = 'U' の場合、 $\text{sub}(A)$ の上三角部分が格納される。
uplo = 'L' の場合、 $\text{sub}(A)$ の下三角部分が格納される。

n (グローバル) INTEGER。処理される行数と列数。すなわち、分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

a (ローカル)
REAL (pspotri の場合)
DOUBLE PRECISION (pdpotri の場合)
COMPLEX (pcpotri の場合)
DOUBLE COMPLEX (pzpotri の場合)

ローカルメモリーにある、ローカル次元 $a(11d_a, LOC_c(ja+n-1))$ の配列へのポインター。

呼び出し時に、配列 *a* は、p?potrf によって行われたコレスキー因子分解 $\text{sub}(A) = U^H U$ または $\text{sub}(A) = LL^H$ で得られた係数 *U* または *L* を含む。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen*)。分散行列 *A* の配列ディスクリプター。

出力パラメーター

a 終了時に、 $\text{sub}(A)$ の (対称/エルミート) 逆行列の上三角または下三角のローカル部分によって上書きされる。

info (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。

info > 0 の場合：
info = *i* の場合、係数 *U* または *L* の (*i*,*i*) 成分がゼロで、逆行列は計算できない。

p?trtri

三角分散行列の逆行列を計算する。

構文

```
call pstrtri(uplo, diag, n, a, ia, ja, desca, info)
call pdtrtri(uplo, diag, n, a, ia, ja, desca, info)
call pctrtri(uplo, diag, n, a, ia, ja, desca, info)
call pztrtri(uplo, diag, n, a, ia, ja, desca, info)
```

説明

このルーチンは、実 / 複素上 / 下三角分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ の逆行列を計算する。

入力パラメーター

uplo (グローバル) CHARACTER*1。 'U' または 'L' でなければならない。
分散行列 $\text{sub}(A)$ が上三角と下三角のどちらであるかを指定する。
uplo = 'U' の場合、 $\text{sub}(A)$ は上三角である。
uplo = 'L' の場合、 $\text{sub}(A)$ は下三角である。

diag CHARACTER*1。 'N' または 'U' でなければならない。
分散行列 $\text{sub}(A)$ が単位三角かどうかを指定する。
diag = 'N' の場合、 $\text{sub}(A)$ は単位三角ではない。
diag = 'U' の場合、 $\text{sub}(A)$ は単位三角である。

n (グローバル) INTEGER。 処理される行数と列数。 すなわち、分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

a (ローカル)
REAL (pstrtri の場合)
DOUBLE PRECISION (pdtrtri の場合)
COMPLEX (pctrtri の場合)
DOUBLE COMPLEX (pztrtri の場合)
ローカルメモリーにある、ローカル次元 $a(\text{lld}_a, \text{LOC}_c(\text{ja}+n-1))$ の配列へのポインター。

配列 a は、三角分散行列 $\text{sub}(A)$ のローカル部分を含む。
 $\text{uplo} = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分は反転する上三角行列を含む。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。
 $\text{uplo} = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の下三角部分に、下三角行列を格納する。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
 $desca$ (グローバルおよびローカル) INTEGER 配列、次元は $(dlen_)$ 。分散行列 A の配列ディスクリプター。

出力パラメーター

a 終了時に、元の行列の (三角) 逆行列によって上書きされる。
 $info$ (グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合：
 i 番目の引数が配列で、 j 番目の値が不正だった場合、
 $info = -(i * 100 + j)$ 。 i 番目の引数がスカラーで値が不正だった場合、
 $info = -i$ 。
 $info > 0$ の場合：
 $info = k$ の場合、 $A(ia+k-1, ja+k-1)$ は完全にゼロである。三角関数 $\text{sub}(A)$ は特異で、逆行列は計算できない。

行列の平衡化

このセクションでは、行列の平衡化に必要なスケール係数の計算に使用される ScaLAPACK ルーチンについて説明する。ただし、これらのルーチンが行列を実際にスケールリングするわけではない。

p?geequ

一般矩形分散行列を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。

構文

```
call psgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pdgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pcgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
call pzgeequ(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, info)
```

説明

このルーチンは、 $m \times n$ の分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ を平衡化して条件数を小さくするための、行と列のスケール係数を計算する。出力配列 r に行のスケール係数を返し、配列 c に列のスケール係数を返す。これらの係数は、成分 $b_{ij} = r(i) * a_{ij} * c(j)$ の行列 B の各行と各列の最大成分の絶対値が 1 になるように選択される。

$r(i)$ と $c(j)$ は、 $SMLNUM$ = 最小の安全な数値と $BIGNUM$ = 最大の安全な数値の範囲内に制限される。これらのスケール係数を使用しても $\text{sub}(A)$ の条件数が小さくなることは保証されていないが、実際には有効に機能する。

補助関数 [p?lagge](#) は、p?geequ によって計算されたスケール係数を使用して、一般矩形行列をスケールリングする。

入力パラメーター

- m (グローバル) INTEGER。処理される行数。すなわち、分散部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。
- n (グローバル) INTEGER。処理される列数。すなわち、分散部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
- a (ローカル)
 REAL (psgeequ の場合)
 DOUBLE PRECISION (pdgeequ の場合)
 COMPLEX (pcgeequ の場合)
 DOUBLE COMPLEX (pzgeequ の場合)
- ローカルメモリーにある、ローカル次元 $a(\text{lld}_a, LOC_c(\text{ja}+n-1))$ の配列へのポインター。
- 配列 a は、平衡化係数を計算する $m \times n$ の分散行列のローカル部分を含む。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 A の配列ディスクリプター。

出力パラメーター

<i>r, c</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元はそれぞれ $LOC_r(m_a)$ および $LOC_c(n_a)$ 。 <i>info</i> = 0 または <i>info</i> > <i>ia</i> + <i>m</i> -1 の場合、配列 $r(ia:ia+m-1)$ には、 $\text{sub}(A)$ の行スケール係数が格納される。 r は分散行列 A とアライメントされ、すべてのプロセス列に複製される。 r は、分散行列 A に関連付けられる。 <i>info</i> = 0 の場合、配列 $c(ja:ja+n-1)$ には、 $\text{sub}(A)$ の列スケール係数が格納される。 c は分散行列 A とアライメントされ、すべてのプロセス行に複製される。 c は、分散行列 A に関連付けられる。
<i>rowcnd, colcnd</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>info</i> = 0 または <i>info</i> > <i>ia</i> + <i>m</i> -1 の場合、 <i>rowcnd</i> には最小の $r(i)$ を最大の $r(i)$ で割った値が格納される ($ia \leq i \leq ia+m-1$)。 $rowcnd \geq 0.1$ かつ <i>amax</i> が大きすぎる値でも小さすぎる値でもない場合、 $r(ia:ia+m-1)$ によるスケールリングは不要である。 <i>info</i> = 0 の場合、 <i>colcnd</i> には最小の $c(j)$ を最大の $c(j)$ で割った値が格納される ($ja \leq j \leq ja+n-1$)。 $colcnd \geq 0.1$ の場合、 $c(ja:ja+n-1)$ によるスケールリングは不要である。
<i>amax</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 最大行列成分の絶対値。 <i>amax</i> がオーバーフローまたはアンダーフローに非常に近い場合、行列をスケールリングする必要がある。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>i</i> で <i>i</i> ≤ <i>m</i> の場合、分散行列 $\text{sub}(A)$ の <i>i</i> 番目の行が完全にゼロである。 <i>i</i> > <i>m</i> の場合、分散行列 $\text{sub}(A)$ の (<i>i</i> - <i>m</i>) 番目の列が完全にゼロである。

p?poequ

対称(エルミート)正定値分散行列を平衡化して条件数を小さくするための行と列のスケール係数を計算する。

構文

```
call pspoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pdpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pcpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
call pzpoequ(n, a, ia, ja, desca, sr, sc, scond, amax, info)
```

説明

このルーチンは、実対称/複素エルミート正定値分散行列

$\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ を平衡化して (2-ノルムに関して) 条件数を小さくするための行と列のスケール係数を計算する。出力配列 *sr* に行スケール係数を返し、*sc* に列スケール係数を返す。

$$s(i) = \frac{1}{\sqrt{a_{i,i}}}$$

これらの係数は、スケールリング後に成分 $b_{ij}=s(i)*a_{ij}*s(j)$ の行列 *B* の対角成分が同じになるように選択される。

sr および *sc* をこのように選択すると、*B* の条件数は、すべての可能な対角スケールリングで、可能な最小の条件数に係数 *n* を掛けた値の範囲内に収まる。

補助関数 [p?lagsy](#) は、[p?geequ](#) によって計算されたスケール係数を使用して、一般矩形行列をスケールリングする。

入力パラメーター

- n* (グローバル) INTEGER。処理される行数と列数。すなわち、分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
- a* (ローカル)
 REAL (pspoequ の場合)
 DOUBLE PRECISION (pdpoequ の場合)
 COMPLEX (pcpoequ の場合)
 DOUBLE COMPLEX (pzpoequ の場合)
 ローカルメモリーにある、ローカル次元 $a(\text{lld}_a, \text{LOC}_c(\text{ja}+n-1))$ の配列へのポインター。
 配列 *a* は、スケール係数を計算する $n \times n$ の対称/エルミート正定値分散行列 $\text{sub}(A)$ を含む。 $\text{sub}(A)$ の対角成分だけが参照される。
- ia, ja* (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 *A* の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *A* の配列ディスクリプター。

出力パラメーター

sr, sc (ローカル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 配列、次元はそれぞれ $LOC_r(m_a)$ および $LOC_c(n_a)$ 。
info = 0 の場合、配列 *sr* (*ia:ia+n-1*) には、**sub(A)** の行スケール係数が格納される。*sr* は分散行列 *A* でアライメントされ、すべてのプロセス列に複製される。*sr* は、分散行列 *A* に関連付けられる。
info = 0 の場合、配列 *sc* (*ja:ja+n-1*) には、**sub(A)** の列スケール係数が格納される。*sc* は分散行列 *A* とアライメントされ、すべてのプロセス行に複製される。*sc* は、分散行列 *A* に関連付けられる。

scond (グローバル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
info = 0 の場合、*scond* には最小の *sr*(*i*) (または *sc*(*j*)) を最大の *sr*(*i*) (または *sc*(*j*)) で割った値が格納される ($ia \leq i \leq ia+n-1$ および $ja \leq j \leq ja+n-1$)。
scond ≥ 0.1 かつ *amax* が大きすぎる値でも小さすぎる値でもない場合、*sr* (または *sc*) によるスケールリングは不要である。

amax (グローバル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 最大行列成分の絶対値。*amax* がオーバーフローまたはアンダーフローに非常に近い場合、行列をスケールリングする必要がある。

info (グローバル) INTEGER。*info* = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。
info > 0 の場合：
info = *k* の場合、**sub(A)** の *k* 番目の対角エントリーが正の値ではない。

直交因子分解

このセクションでは、行列の QR (RQ) 因子分解と LQ (QL) 因子分解を行うための ScaLAPACK ルーチンについて説明する。一般化された QR 因子分解と RQ 因子分解と同様に RZ 因子分解用のルーチンも含まれている。因子分解の数学的な定義については、それぞれの LAPACK セクションを参照するか、[SLUG] を参照のこと。

表 5-1 に、行列の直交因子分解を実行するための ScaLAPACK ルーチンを示す。

表 6-3 直交因子分解用の計算ルーチン

行列のタイプ、因子分解	因子分解 (ピボット演算なし)	因子分解 (ピボット演算あり)	行列 Q の生成	行列 Q の適用
一般行列、 QR 因子分解	p?geqrf	p?geqpf	p?orgqr p?ungqr	p?ormqr p?unmqr
一般行列、 RQ 因子分解	p?gerqf		p?orgrq p?ungrq	p?ormrq p?unmrq
一般行列、 LQ 因子分解	p?gelqf		p?orglq p?unglq	p?ormlq p?unmlq
一般行列、 QL 因子分解	p?geqlf		p?orgql p?ungql	p?ormql p?unmql
台形行列、 RZ 因子分解	p?tzzrf			p?ormrz p?unmrz
行列のペア、 汎用 QR 因子分解	p?gqrf			
行列のペア、 汎用 RQ 因子分解	p?qrqf			

p?geqrf

$m \times n$ の一般行列の QR 因子分解を行う。

構文

```
call psgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgeqrf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ の QR 因子分解を行う。

$$A = QR$$

入力パラメーター

m (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。

<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq nb_a * (mp0+nq0+nb_a)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb_a)$ 、 $icoff = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mp0 = \text{numroc}(m+iroff, mb_a, MYROW, iarow, NPROW)$ 、 $nq0 = \text{numroc}(n+icoff, nb_a, MYCOL, iacol, NPCOL)$ 。 numroc および indxg2p は、ScaLAPACK ツール関数である。 $MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および $NPCOL$ は、サブルーチン blacs_gridinfo を呼び出して求められる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	$\text{sub}(A)$ の対角成分とその上の成分は、 $\min(m,n) \times n$ の上台形行列 R ($m \geq n$ の場合、 R は上三角行列) によって上書きされる。対角より下の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
<i>tau</i>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合)

DOUBLE COMPLEX (pzgeqrf の場合)
 配列、次元は $LOCc(ja+\min(m,n)-1)$ 。
 基本リフレクターのスカラー係数 τ が格納される。
 τ は、分散行列 A に関連付けられる。

work(1) 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ja) H(ja+1) \dots H(ja+k-1)$$

ここで、 $k = \min(m,n)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(j) = I - \tau * v * v'$$

ここで、 τ は実/複素スカラー、 v は実/複素ベクトルで、 $v(1:i-1) = 0$ および $v(i) = 1$ 。
 終了時に、 $v(i+1:m)$ は $A(ia+i:ia+m-1, ja+i-1)$ に、 τ は $\tau(ja+i-1)$ に格納される。

p?geqpf

$m \times n$ の一般行列の QR 因子分解をピボット演算付きで行う。

構文

```
call psgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pdgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pcgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
call pzgeqpf( m, n, a, ia, ja, desca, ipiv, tau, work, lwork, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の一般分散行列 $sub(A) = A(ia:ia+m-1, ja:ja+n-1)$ の列ピボット演算を用いた QR 因子分解を行う。

$$sub(A) P = Q R$$

入力パラメーター

m (グローバル) INTEGER。部分行列 $sub(A)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。部分行列 $sub(A)$ の列数 ($n \geq 0$)。

<i>a</i>	<p>(ローカル)</p> <p>REAL (psgeqpf の場合)</p> <p>DOUBLE PRECISION (pdgeqpf の場合)</p> <p>COMPLEX (pcgeqpf の場合)</p> <p>DOUBLE COMPLEX (pzgeqpf の場合)</p> <p>ローカルメモリーにある、ローカル次元 (lld_a, $LOCc(ja+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。</p>
<i>ia, ja</i>	<p>(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, JA:JA+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。</p>
<i>work</i>	<p>(ローカル)。</p> <p>REAL (psgeqpf の場合)</p> <p>DOUBLE PRECISION (pdgeqpf の場合)</p> <p>COMPLEX (pcgeqpf の場合)</p> <p>DOUBLE COMPLEX (pzgeqpf の場合)</p> <p>次元 <i>lwork</i> のワークスペース配列。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は</p> <p>実数型の場合: $lwork \geq \max(3, mp0+nq0) + LOCc(ja+n-1) + nq0$</p> <p>複素数型の場合: $lwork \geq \max(3, mp0+nq0)$</p> <p>ここで、</p> <p>$iroff = \text{mod}(ia-1, mb_a)$, $icoff = \text{mod}(ja-1, nb_a)$、</p> <p>$iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$、</p> <p>$iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$、</p> <p>$mp0 = \text{numroc}(m+iroff, mb_a, MYROW, iarow, NPROW)$、</p> <p>$nq0 = \text{numroc}(n+icoff, nb_a, MYCOL, iacol, NPCOL)$、</p> <p>$LOCc(ja+n-1) = \text{numroc}(ja+n-1, nb_a, MYCOL, csrc_a, NPCOL)$。</p> <p>$\text{numroc}$ および indxg2p は、ScaLAPACK ツール関数である。 $MYROW$、$MYCOL$、$NPROW$、および $NPCOL$ は、サブルーチン blacs_gridinfo を呼び出して求められる。</p> <p>$lwork = -1$ の場合、<i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、pxerbla はエラーメッセージを生成しない。</p>

出力パラメーター

<i>a</i>	sub(A) の対角成分とその上の成分は、 $\min(m,n) \times n$ の上台形行列 R ($m \geq n$ の場合、 R は上三角行列) によって上書きされる。対角より下の成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は $LOCc(ja+n-1)$ 。 $ipiv(i)=k$ 、sub(A)* P の i 番目のローカル行は、sub(A) の k 番目のグローバル行。 <i>ipiv</i> は、分散行列 A に関連付けられる。
<i>tau</i>	(ローカル) REAL (psgeqpf の場合) DOUBLE PRECISION (pdgeqpf の場合) COMPLEX (pcgeqpf の場合) DOUBLE COMPLEX (pzgeqpf の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$ 。 基本リフレクターのスカラー係数 <i>tau</i> が格納される。 <i>tau</i> は、分散行列 A に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。
 $Q = H(1)H(2)\dots H(n)$

それぞれの $H(i)$ は次のような形式を持つ。
 $H = I - \tau * v * v'$

ここで、*tau* は実/複素スカラー、 v は実/複素ベクトルで、 $v(1:i-1) = 0$ および $v(i) = 1$ 。
 終了時に、 $v(i+1:m)$ は $A(ia+i:ia+m-1, ja+i-1)$ に格納される。

行列 P は、*jpvt* で次のように表現される： $jpvt(j) = i$ の場合、 P の j 番目の列は i 番目の正当な単位ベクトルである。

p?orgqr

p?geqrf で求めた QR 因子分解の直交行列 Q を生成する。

構文

```
call psorgqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の実分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターによる積の最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

入カパラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($m \geq n \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
a	(ローカル) REAL (psorgqr の場合) DOUBLE PRECISION (pdorgqr の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOCc(JA+n-1)$) の配列へのポインター。 j 番目の列は、p?geqrf によって分散行列引数 $a(ia:*, ja:ja+k-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。
ia, ja	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
tau	(ローカル) REAL (psorgqr の場合) DOUBLE PRECISION (pdorgqr の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?geqrf によって返される、基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を含む。 tau は、分散行列 A に関連付けられる。
$work$	(ローカル)

REAL (psorgqr の場合)
DOUBLE PRECISION (pdorgqr の場合)
次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。
work の次元。 $lwork \geq nb_a * (nqa0 + mpa0 + nb_a)$ でなければならない。
ここで、

```

irowfa = mod(ia-1, mb_a), icoffa = mod(ja-1, nb_a),
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mpa0 = numroc(m+irowfa, mb_a, MYROW, iarow, NPROW),
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して求められる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a $m \times n$ の分散行列 *Q* のローカル部分が格納される。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、 $info = -(i*100+j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ungqr

p?geqrf で求めた QR 因子分解の複素ユニタリー行列 *Q* を生成する。

構文

```

call pcungqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungqr( m, n, k, a, ia, ja, desca, tau, work, lwork, info )

```

説明

このルーチンは、直交列を持つ $m \times n$ の複素分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積による最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

入力パラメーター

<i>m</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($m \geq n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
<i>a</i>	(ローカル) COMPLEX (pcungqr の場合) DOUBLE COMPLEX (pzungqr の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。 j 番目の列は、p?geqrf によって分散行列引数 $a(ia:*, ja:ja+k-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcungqr の場合) DOUBLE COMPLEX (pzungqr の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?geqrf によって返される、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を含む。 τ は、分散行列 A に関連付けられる。
<i>work</i>	(ローカル) COMPLEX (pcungqr の場合) DOUBLE COMPLEX (pzungqr の場合) 次元 $lwork$ のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq nb_a * (nqa0 + mpa0 + nb_a)$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mpa0 = \text{numroc}(m+iroffa, mb_a, MYROW, iarow, NPROW)$ 、

```
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)
```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a	$m \times n$ の分散行列 Q のローカル部分が格納される。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
info	(グローバル) INTEGER。 info = 0 の場合、実行は正常に終了したことを示す。 info < 0 の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ormqr

一般行列に p?geqrf によって求めた QR 因子分解の直交行列 Q を掛ける。

構文

```
call psormqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pdormqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$:	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$trans = 'T'$:	$Q^T \text{sub}(C)$	$\text{sub}(C) Q^T$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 Q または Q^T は左側から適用される。 ='R' の場合、 Q または Q^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='T' の場合、 Q^T が適用される (転置)。
<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> ='L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> ='R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+k-1)$) の配列へのポインター。j 番目の列は、p?geqrf によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 <i>side</i> ='L' の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ <i>side</i> ='R' の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?geqrf によって返される、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を含む。 τ は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。

<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psormqr</i> の場合) DOUBLE PRECISION (<i>pdormqr</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + mpc0) * nb_a) + nb_a * nb_a$ <i>side</i> = 'R' の場合、 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(n + icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0)) * nb_a) + nb_a * nb_a$ でなければならない。 ここで、 $lcmq = lcm / NPCOL$ で $lcm = ilcm(NPROW, NPCOL)$ 、 $iroffa = \text{mod}(ia - 1, mb_a)$ 、 $icoffa = \text{mod}(ja - 1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $npa0 = \text{numroc}(n + iroffa, mb_a, MYROW, iarow, NPROW)$ 、 $iroffc = \text{mod}(ic - 1, mb_c)$ 、 $icoffc = \text{mod}(jc - 1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW)$ 、 $iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL)$ 、 $mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW)$ 、 $nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$ <i>ilcm</i> 、 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>MYROW</i> 、 <i>MYCOL</i> 、 <i>NPROW</i> および <i>NPCOL</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエンタリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q^T \text{sub}(C)$ 、 $\text{sub}(C) * Q^T$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。

p?unmqr

複素行列に p?geqrf で求めた QR 因子分解のユニタリー行列 *Q* を掛ける。

構文

```
call pcunmqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pzunmqr( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N':	$Q \text{sub}(C)$	$\text{sub}(C) Q$
<i>trans</i> = 'T':	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、*Q* は、*k* 個の以下の基本リフレクターの積として定義される複素ユニタリー分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?geqrf](#) によって返される。*Q* の次数は、*side* = 'L' の場合は *m*、*side* = 'R' の場合は *n* である。

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、*Q* または Q^H は左側から適用される。
 = 'R' の場合、*Q* または Q^H は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、*Q* が適用される (転置なし)。
 = 'C' の場合、 Q^H が適用される (共役転置)。

m (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。

<i>k</i>	(グローバル) INTEGER。その積が行列 <i>Q</i> を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (<i>pcunmqr</i> の場合) DOUBLE COMPLEX (<i>pzunmqr</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+k-1)</i>) の配列へのポインター。 <i>j</i> 番目の列は、 <i>p?geqrf</i> によって分散行列引数 <i>a(ia:*,ja:ja+k-1)</i> の <i>k</i> 列に返される、基本リフレクター <i>H(j)</i> , $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。 <i>a(ia:*,ja:ja+k-1)</i> はルーチンにより変更されるが終了時に復元される。 <i>side</i> = 'L' の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ <i>side</i> = 'R' の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (<i>pcunmqr</i> の場合) DOUBLE COMPLEX (<i>pzunmqr</i> の場合) 配列、次元は <i>LOCc(ja+k-1)</i> 。 <i>p?geqrf</i> によって返される、基本リフレクター <i>H(j)</i> のスカラー係数 <i>tau(j)</i> を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (<i>pcunmqr</i> の場合) DOUBLE COMPLEX (<i>pzunmqr</i> の場合) ローカルメモリーにある、ローカル次元 (<i>lld_c</i> , <i>LOCc(jc+n-1)</i>) の配列へのポインター。 因子分解する分散行列 <i>sub(C)</i> のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (<i>pcunmqr</i> の場合) DOUBLE COMPLEX (<i>pzunmqr</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((nb_a*(nb_a-1))/2, (ngc0 + mpc0)*nb_a) + nb_a * nb_a$

side = 'R' の場合、

$$lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(n + icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0)) * nb_a) + nb_a * nb_a)$$

でなければならない。

ここで、

$$lcmq = lcm / NPCOL \text{ で } lcm = ilcm(NPROW, NPCOL),$$

$$iroffa = \text{mod}(ia - 1, mb_a),$$

$$icoffa = \text{mod}(ja - 1, nb_a),$$

$$iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW),$$

$$npa0 = \text{numroc}(n + iroffa, mb_a, MYROW, iarow, NPROW),$$

$$iroffc = \text{mod}(ic - 1, mb_c),$$

$$icoffc = \text{mod}(jc - 1, nb_c),$$

$$icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW),$$

$$iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL),$$

$$mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW),$$

$$nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$$

ilcm、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。
MYROW、*MYCOL*、*NPROW* および *NPCOL* は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- c* 積 $Q * \text{sub}(C)$ 、 $Q^H \text{sub}(C)$ 、 $\text{sub}(C) * Q^H$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
- work*(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。
- info* (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?gelqf

一般矩形行列の LQ 因子分解を行う。

構文

```
call psgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgelqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の実 / 複素分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ia}:\text{ia}+n-1) = L*Q$ の LQ 因子分解を行う。

入力パラメーター

<i>m</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
<i>a</i>	(ローカル) REAL (psgelqf の場合) DOUBLE PRECISION (pdgelqf の場合) COMPLEX (pcgelqf の場合) DOUBLE COMPLEX (pzgelqf の場合) ローカルメモリーにある、ローカル次元 (lld_a , $\text{LOCc}(\text{ja}+n-1)$) の配列へのポインター。因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A(\text{ia}:\text{ia}+m-1, \text{ia}:\text{ia}+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($\text{dlen}_$)。分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psgelqf の場合) DOUBLE PRECISION (pdgelqf の場合) COMPLEX (pcgelqf の場合) DOUBLE COMPLEX (pzgelqf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。work の次元は $\text{lwork} \geq \text{mb_a} * (\text{mp0} + \text{nq0} + \text{mb_a})$ でなければならない。ここで、 $\text{iroff} = \text{mod}(\text{ia}-1, \text{mb_a})$ 、 $\text{icoff} = \text{mod}(\text{ja}-1, \text{nb_a})$ 、

```

iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mp0 = numroc (m+iroff, mb_a, MYROW, iarow, NPROW),
nq0 = numroc (n+icoff, nb_a, MYCOL, iacol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、`pxerbla` はエラーメッセージを生成しない。

出力パラメーター

- a** `sub(A)` の対角成分とその下の成分は、 $m \times \min(m,n)$ の下台形行列 L ($m \leq n$ の場合、 L は下三角行列) によって上書きされる。対角より上の成分は、配列 `tau` とともに、基本リフレクターの積として直交/ユニタリーの行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
- tau** (ローカル)
 REAL (`psgelqf` の場合)
 DOUBLE PRECISION (`pdgelqf` の場合)
 COMPLEX (`pcgelqf` の場合)
 DOUBLE COMPLEX (`pzgelqf` の場合)
 配列、次元は $LOCr(ia+\min(m,n)-1)$ 。
 基本リフレクターのスカラー係数が格納される。`tau` は、分散行列 A に関連付けられる。
- work(1)** 終了時に、`work(1)` には、最適なパフォーマンスを得るために必要な `lwork` の最小値が格納される。
- info** (グローバル) INTEGER。
`info = 0` の場合、実行は正常に終了したことを示す。
`info < 0` の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ia+k-1) H(ia+k-2) \dots H(ia)$$

ここで、 $k = \min(m,n)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、`tau` は実/複素スカラー、`v` は実/複素ベクトルで、 $v(1:i-1) = 0$ および $v(i) = 1$ 。終了時に、 $v(i+1:n)$ は $A(ia+i-1:ia+i-1, ja+n-1)$ に、`tau` は `tau(ia+i-1)` に格納される。

p?orglq

p?gelqf で求めた LQ 因子分解の実直交行列 Q を生成する。

構文

```
call psorglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の実分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積による最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(k) \dots H(2) H(1)$$

[p?gelqf](#) によって返される。

入カパラメーター

<i>m</i>	(グローバル) INTEGER。部分行列 sub(Q) の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。部分行列 sub(Q) の列数 ($n \geq m \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
<i>a</i>	(ローカル) REAL (psorglq の場合) DOUBLE PRECISION (pdorglq の場合) ローカルメモリーにある、ローカル次元 (<i>lld_a</i> , LOC(<i>ja+n-1</i>)) の配列へのポインター。呼び出し時に、 <i>i</i> 番目の行は、p?gelqf によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の <i>k</i> 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psorglq の場合) DOUBLE PRECISION (pdorglq の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。work の次元は $lwork \geq mb_a * (mpa0 + nqa0 + mb_a)$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb_a)$ 、

```

icoffa = mod(ja-1, nb_a),
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mpa0 = numroc(m+iroffa, mb_a, MYROW, iarow, NPROW),
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)

```

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<code>a</code>	因子分解の対象となる $m \times n$ の分散行列 Q のローカル部分が格納される。
<code>tau</code>	(ローカル) REAL (<code>psorglq</code> の場合) DOUBLE PRECISION (<code>pdorglq</code> の場合) 配列、次元は $LOCr(ia+k-1)$ 。 基本リフレクター $H(i)$ のスカラー係数 <code>tau</code> が格納される。 <code>tau</code> は、分散行列 A に関連付けられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info < 0</code> の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?unglq

p?gelqf で求めた LQ 因子分解のユニタリー行列 Q を生成する。

構文

```

call pcunglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzunglq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )

```

説明

このルーチンは、直交列を持つ $m \times n$ の複素分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積による最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(k) \dots H(2)' H(1)'$$

[p?gelsqf](#) によって返される。

入力パラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($n \geq m \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
a	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。呼び出し時に、 i 番目の行は、 p?gelsqf によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の k 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
tau	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) 配列、次元は $LOCr(ia+k-1)$ 。 基本リフレクター $H(i)$ のスカラー係数 tau が格納される。 tau は、分散行列 A に関連付けられる。
$work$	(ローカル) COMPLEX (pcunglq の場合) DOUBLE COMPLEX (pzunglq の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq mb_a * (mpa0 + nqa0 + mb_a)$ でなければならない。ここで、 $irow = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、

$mpa0 = \text{numroc}(m + iroffa, mb_a, \text{MYROW}, iarow, \text{NPROW}),$

$nqa0 = \text{numroc}(n + icoffa, nb_a, \text{MYCOL}, iacol, \text{NPCOL})$

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a	因子分解の対象となる $m \times n$ の分散行列 Q のローカル部分が格納される。
$\text{work}(1)$	終了時に、 $\text{work}(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
info	(グローバル) INTEGER。 $\text{info} = 0$ の場合、実行は正常に終了したことを示す。 $\text{info} < 0$ の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $\text{info} = -(i * 100 + j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $\text{info} = -i$ 。

p?ormlq

一般行列に p?gelqf によって求めた LQ 因子分解のユニタリー行列 Q を掛ける。

構文

```
call psormlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             work, lwork, info )
```

```
call pdormlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{sub}(C)$	$\text{sub}(C) Q^T$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される実直交分散行列である。

$$Q = H(k) \dots H(2) H(1)$$

[p?gelqf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 Q または Q^T は左側から適用される。 ='R' の場合、 Q または Q^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='T' の場合、 Q^T が適用される (転置)。
<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormlq の場合) DOUBLE PRECISION (pdormlq の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) ($side = 'L'$ の場合) または (lld_a , $LOCc(ja+n-1)$) ($side = 'R'$ の場合) の配列へのポインター。 i 番目の行は、 p?gelqf によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の k 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormlq の場合) DOUBLE PRECISION (pdormlq の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?gelqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。 τ は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psormlq の場合) DOUBLE PRECISION (pdormlq の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。

<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₀)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psormlq</i> の場合) DOUBLE PRECISION (<i>pdormlq</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + \max(mqa0) + \text{numroc}(\text{numroc}(m + iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcm), nqc0)) * mb_a) + mb_a * mb_a$ <i>side</i> = 'R' の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + nqc0) * mb_a + mb_a * mb_a)$ でなければならない。 ここで、 $lcmp = lcm / NPROW$ で $lcm = ilcm(NPROW, NPCOL)$ 、 $iroffa = \text{mod}(ia - 1, mb_a)$ 、 $icoffa = \text{mod}(ja - 1, nb_a)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mqa0 = \text{numroc}(m + icoffa, nb_a, MYCOL, iacol, NPCOL)$ 、 $iroffc = \text{mod}(ic - 1, mb_c)$ 、 $icoffc = \text{mod}(jc - 1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW)$ 、 $iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL)$ 、 $mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW)$ 、 $nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$ <i>ilcm</i> 、 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>MYROW</i> 、 <i>MYCOL</i> 、 <i>NPROW</i> および <i>NPCOL</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエンタリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?unmlq

一般行列に p?gelqf によって求めた LQ 因子分解のユニタリー行列 *Q* を掛ける。

構文

```
call pcunmlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pzunmlq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N':	$Q \text{sub}(C)$	$\text{sub}(C) Q$
<i>trans</i> = 'T':	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される複素ユニタリー分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

p?gelqf によって返される。 Q の次数は、*side* = 'L' の場合は m 、*side* = 'R' の場合は n である。

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^H は左側から適用される。
 = 'R' の場合、 Q または Q^H は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'C' の場合、 Q^H が適用される (共役転置)。

m (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。

<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) (<i>side</i> = 'L' の場合) または (lld_a , $LOCc(ja+n-1)$) (<i>side</i> = 'R' の場合) の配列へのポインター。 ここで、 $lld_a \geq \max(1, LOCr(ia+k-1))$ である。 <i>i</i> 番目の行は、p?gelqf によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の <i>k</i> 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合) 配列、次元は $LOCc(ia+k-1)$ 。 p?gelqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。 <i>tau</i> は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmlq の場合) DOUBLE COMPLEX (pzunmlq の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。work の次元は <i>side</i> = 'L' の場合、

$$lwork \geq \max ((mb_a*(mb_a-1))/2, (mpc0 + \max mqa0) + \text{numroc}(\text{numroc}(m + iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0)) * mb_a + mb_a*mb_a$$

side = 'R' の場合、

$$lwork \geq \max ((mb_a*(mb_a-1))/2, (mpc0 + nqc0) * mb_a + mb_a*mb_a)$$

でなければならない。

ここで、

$$lcmp = lcm / NPROW \text{ で } lcm = ilcm(NPROW, NPCOL),$$

$$iroffa = \text{mod}(ia-1, mb_a),$$

$$icoffa = \text{mod}(ja-1, nb_a),$$

$$iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL),$$

$$mqa0 = \text{numroc}(m + icoffa, nb_a, MYCOL, iacol, NPCOL),$$

$$iroffc = \text{mod}(ic-1, mb_c),$$

$$icoffc = \text{mod}(jc-1, nb_c),$$

$$icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW),$$

$$iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL),$$

$$mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW),$$

$$nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$$

ilcm、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。
MYROW、*MYCOL*、*NPROW* および *NPCOL* は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- c* 積 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
- work*(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。
- info* (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?geqlf

一般行列の QL 因子分解を行う。

構文

```
call psgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgeqlf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の実/複素分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * L$ の QL 因子分解を行う。

入力パラメーター

m (グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($n \geq 0$)。

a (ローカル)
 REAL (psgeqlf の場合)
 DOUBLE PRECISION (pdgeqlf の場合)
 COMPLEX (pcgeqlf の場合)
 DOUBLE COMPLEX (pzgeqlf の場合)
 ローカルメモリーにある、ローカル次元 (*lld_a*, *LOCc(ja+n-1)*) の配列へのポインター。因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ia:ia+n-1)$ の最初の行と最初の列を示すグローバル配列 *a* の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *A* の配列ディスクリプター。

work (ローカル)
 REAL (psgeqlf の場合)
 DOUBLE PRECISION (pdgeqlf の場合)
 COMPLEX (pcgeqlf の場合)
 DOUBLE COMPLEX (pzgeqlf の場合)
 次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は $lwork \geq nb_a * (mp0 + nq0 + nb_a)$ でなければならない。ここで、
iroff = mod(*ia-1*, *mb_a*)、
icoff = mod(*ja-1*, *nb_a*)、
iarow = indxg2p(*ia*, *mb_a*, MYROW, *rsrc_a*, NPROW)、
iacol = indxg2p(*ja*, *nb_a*, MYCOL, *csrc_a*, NPCOL)、

$mp0 = \text{numroc}(m + iroff, mb_a, \text{MYROW}, iarow, \text{NPROW}),$

$nq0 = \text{numroc}(n + icoff, nb_a, \text{MYCOL}, iacol, \text{NPCOL})$

numroc および indxg2p は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、 $m \geq n$ の場合、分散部分行列 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の下三角部分に、 $n \times n$ の下三角行列 L が格納される。 $m \leq n$ の場合、 $(n-m)$ 番目の優対角成分とその下の成分に、 $m \times n$ 下台形行列 L が格納される。残りの成分は、配列 tau とともに、基本リフレクターの積として直交/ユニタリー行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
- tau** (ローカル)
REAL (psgeqlf の場合)
DOUBLE PRECISION (pdgeqlf の場合)
COMPLEX (pcgeqlf の場合)
DOUBLE COMPLEX (pzgeqlf の場合)
配列、次元は $(LOCc(ja+n-1))$ 。
基本リフレクターのスカラー係数が格納される。 tau は、分散行列 A に関連付けられる。
- work(1)** 終了時に、 $\text{work}(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
- info** (グローバル) INTEGER。
 $\text{info} = 0$ の場合、実行は正常に終了したことを示す。
 $\text{info} < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $\text{info} = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $\text{info} = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ja+k-1) \dots H(ja+1) H(ja)$$

ここで、 $k = \min(m, n)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \text{tau} * v * v'$$

ここで、 tau は実/複素スカラー、 v は実/複素ベクトルで、 $v(m-k+i+1:m) = 0$ および $v(m-k+i) = 1$ 。終了時に、 $v(m-k+i-1)$ は $A(ia+ia+m-k+i-2, ja+n-k+i-1)$ に、 tau は $\text{tau}(ja+n-k+i-1)$ に格納される。

p?orgql

p?geqlf で求めた QL 因子分解の実直交行列 Q を生成する。

構文

```
call psorgql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の実分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積による最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(k) \dots H(2) H(1)$$

[p?geqlf](#) によって返される。

入カパラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($m \geq n \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
a	(ローカル) REAL (psorgql の場合) DOUBLE PRECISION (pdorgql の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインタ。呼び出し時に、 j 番目の列は、p?geqlf によって分散行列引数 $A(ia:*ja+n-k:ja+n-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja+n-k \leq j \leq ja+n-1$ を定義するベクトルを含んでいなければならない。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
tau	(ローカル) REAL (psorgql の場合) DOUBLE PRECISION (pdorgql の場合) 配列、次元は ($LOCc(ja+n-1)$)。 基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を含む。 tau は、分散行列 A に関連付けられる。
$work$	(ローカル)

REAL (psorgql の場合)
 DOUBLE PRECISION (pdorgql の場合)
 次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は $lwork \geq nb_a * (nqa0 + mpa0 + nb_a)$ でなければならない。ここで、

$iroffa = \text{mod}(ia-1, mb_a)$ 、
 $icoffa = \text{mod}(ja-1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、
 $mpa0 = \text{numroc}(m+iroffa, mb_a, MYROW, iarow, NPROW)$ 、
 $nqa0 = \text{numroc}(n+icoffa, nb_a, MYCOL, iacol, NPCOL)$

indxg2p および *numroc* は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a 因子分解の対象となる $m \times n$ の分散行列 *Q* のローカル部分が格納される。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、 $info = -(i*100+j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ungql

p?geqlf で求めた QL 因子分解のユニタリ行列 *Q* を生成する。

構文

```
call pcungql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungql( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の複素分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積による最初の n 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ 。

$$Q = H(k) \dots H(2) H(1)$$

[p?geqlf](#) によって返される。

入力パラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($m \geq n \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($n \geq k \geq 0$)。
a	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) ローカルメモリーにある、ローカル次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。呼び出し時に、 j 番目の列は、 p?geqlf によって分散行列引数 $A(ia:*ja+n-k:ja+n-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja+n-k \leq j \leq ja+n-1$ を定義するベクトルを含んでいなければならない。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
tau	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) 配列、次元は $LOCr(ia+n-1)$ 。 基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を含む。 tau は、分散行列 A に関連付けられる。
$work$	(ローカル) COMPLEX (pcungql の場合) DOUBLE COMPLEX (pzungql の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq nb_a * (nqa0 + mpa0 + nb_a)$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、

```
mpa0 = numroc(m+iroffa, mb_a, MYROW, iarow, NPROW),
```

```
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)
```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a	因子分解の対象となる $m \times n$ の分散行列 Q のローカル部分が格納される。
work(1)	終了時に、work(1) には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
info	(グローバル) INTEGER。 info = 0 の場合、実行は正常に終了したことを示す。 info < 0 の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ormql

一般行列に p?geqlf によって求めた QL 因子分解のユニタリー行列 Q を掛ける。

構文

```
call psormql( side, trans, m, n, k, a, ia, ja, desca, tau, c,ic, jc,
             descc, work, lwork, info )
```

```
call pdormql( side, trans, m, n, k, a, ia, ja, desca, tau, c,ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$:	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T'$:	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される実直交分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

[p?geqlf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 Q または Q^T は左側から適用される。 ='R' の場合、 Q または Q^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='T' の場合、 Q^T が適用される (転置)。
<i>m</i>	(グローバル) INTEGER。分散行列 $sub(C)$ の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $sub(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+k-1)$) の配列へのポインター。j 番目の列は、p?geqlf によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の k 列に返される、基本リフレクター $H(j)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 $side = 'L'$ の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ $side = 'R'$ の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合) 配列、次元は $LOCc(ja+n-1)$ 。 p?geqlf によって返される、基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を含む。 tau は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psormql の場合) DOUBLE PRECISION (pdormql の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。

	因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psormql</i> の場合) DOUBLE PRECISION (<i>pdormql</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + mpc0) * nb_a + nb_a * nb_a)$ <i>side</i> = 'R' の場合、 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + \max npa0) + \text{numroc}(\text{numroc}(n + icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0)) * nb_a + nb_a * nb_a$

でなければならない。

ここで、

```

lcmp = lcm / NPCOL で lcm = ilcm (NPROW, NPCOL)、
iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW)、
npa0 = numroc(n + iroffa, mb_a, MYROW, iarow, NPROW)、
iroffc = mod(ic-1, mb_c)、
icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m + iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n + icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。
MYROW、*MYCOL*、*NPROW* および *NPCOL* は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?unmql

一般行列に p?geqlf によって求めた *QL* 因子分解のユニタリー行列 *Q* を掛ける。

構文

```
call pcunmql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pzunmql( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ を以下により上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N':	$Q \text{sub}(C)$	$\text{sub}(C) Q$
<i>trans</i> = 'C':	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、*Q* は、*k* 個の以下の基本リフレクターの積として定義される複素ユニタリー分散行列である。

$$Q = H(k)' \dots H(2)' H(1)'$$

p?geqlf によって返される。*Q* の次数は、*side* = 'L' の場合は *m*、*side* = 'R' の場合は *n* である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER = 'L' の場合、 <i>Q</i> または Q^H は左側から適用される。 = 'R' の場合、 <i>Q</i> または Q^H は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER = 'N' の場合、 <i>Q</i> が適用される (転置なし)。 = 'C' の場合、 Q^H が適用される (共役転置)。

<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+k-1)$) の配列へのポインター。 <i>j</i> 番目の列は、 <i>p?geqlf</i> によって分散行列引数 $a(ia:*,ja:ja+k-1)$ の <i>k</i> 列に返される、基本リフレクター $H(j)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:*,ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 <i>side</i> = 'L' の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ <i>side</i> = 'R' の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合) 配列、次元は $LOCc(ia+n-1)$ 。 <i>p?geqlf</i> によって返される、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を含む。 <i>tau</i> は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmq1 の場合) DOUBLE COMPLEX (pzunmq1 の場合) 次元 <i>lwork</i> のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は *side* = 'L' の場合、

$$lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + mpc0) * nb_a + nb_a * nb_a)$$

side = 'R' の場合、

$$lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + \max npa0) + \text{numroc}(\text{numroc}(n + icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0)) * nb_a + nb_a * nb_a$$

でなければならない。

ここで、

$$lcmp = lcm / NPCOL \text{ で } lcm = ilcm(NPROW, NPCOL),$$

$$iroffa = \text{mod}(ia - 1, mb_a),$$

$$icoffa = \text{mod}(ja - 1, nb_a),$$

$$iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW),$$

$$npa0 = \text{numroc}(n + iroffa, mb_a, MYROW, iarow, NPROW),$$

$$iroffc = \text{mod}(ic - 1, mb_c),$$

$$icoffc = \text{mod}(jc - 1, nb_c),$$

$$icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW),$$

$$iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL),$$

$$mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW),$$

$$nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$$

ilcm、*indxg2p* および *numroc* は、ScaLAPACK ツール関数である。
MYROW、*MYCOL*、*NPROW*、および *NPCOL* は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c 積 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?gerqf

一般矩形行列の RQ 因子分解を行う。

構文

```
call psgerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdgerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pcgerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pzgerqf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、次の式に従って、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の QR 因子分解を行う。

$$A = RQ$$

入力パラメーター

<i>m</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) ローカルメモリーにある、ローカル次元 (<i>lld_a</i> , $LOCc(ja+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psgeqrf の場合) DOUBLE PRECISION (pdgeqrf の場合) COMPLEX (pcgeqrf の場合) DOUBLE COMPLEX (pzgeqrf の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq mb_a * (mp0+nq0+mb_a)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb_a)$ 、 $icoff = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、

$iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、
 $mp0 = \text{numroc}(m + iroff, mb_a, MYROW, iarow, NPROW)$ 、
 $nq0 = \text{numroc}(n + icoff, nb_a, MYCOL, iacol, NPCOL)$
 numroc および indxg2p は ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、 $m \leq n$ の場合、分散部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の上三角部分に、 $m \times m$ の上三角行列 R が格納される。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の成分に、 $m \times n$ 上台形行列 R が格納される。残りの成分は、配列 `tau` とともに、基本リフレクターの積として直交 / ユニタリー行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
- tau** (ローカル)
 REAL (`psgeqrf` の場合)
 DOUBLE PRECISION (`pdgeqrf` の場合)
 COMPLEX (`pcgeqrf` の場合)
 DOUBLE COMPLEX (`pzgeqrf` の場合)
 配列、次元は $LOC_r(ia+m-1)$ 。
 基本リフレクターのスカラー係数 `tau` が格納される。
`tau` は、分散行列 A に関連付けられる。
- work(1)** 終了時に、`work(1)` には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
- info** (グローバル) INTEGER。
`info = 0` の場合、実行は正常に終了したことを示す。
`info < 0` の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。
 $Q = H(ia)H(ia+1)\dots H(ia+k-1)$

ここで、 $k = \min(m, n)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで、 $v(n-k+i+1:n) = 0$ および $v(n-k+i) = 1$ 。終了時に、 $v(1:n-k+i-1)/\text{conj}(v(1:n-k+i-1))$ は $A(ia+m-k+i-1, ja:ja+n-k+i-2)$ に、 τ は $\tau(ia+m-k+i-1)$ に格納される。

p?orgrq

p?gerqf で求めた RQ 因子分解の実直交行列 Q を生成する。

構文

```
call psorgrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pdorgrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の実分散行列 Q の全体または一部を生成する。これは、以下の次数 m の k 個の基本リフレクターの積の最後の m 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ 。

$$Q = H(1)H(2)...H(k)$$

[p?gerqf](#) によって返される。

入カパラメーター

<i>m</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($n \geq m \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
<i>a</i>	(ローカル) REAL (psorgrq の場合) DOUBLE PRECISION (pdorgrq の場合) ローカルメモリーにある、ローカル次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。 <i>i</i> 番目の列は、p?gerqf によって分散行列引数 $a(ia:*, ja:ja+k-1)$ の <i>k</i> 列に返される、基本リフレクター $H(i)$, $ja \leq j \leq ja+k-1$ を定義するベクトルを含んでいなければならない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $A(ia:ia+m-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psorgrq の場合) DOUBLE PRECISION (pdorgrq の場合) 配列、次元は <i>LOCc(ja+k-1)</i> 。 p?gerqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。 <i>tau</i> は、分散行列 A に関連付けられる。
<i>work</i>	(ローカル)

REAL (psorgqr の場合)
 DOUBLE PRECISION (pdorgqr の場合)
 次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は $lwork \geq mb_a * (mpa0 + nqa0 + mb_a)$ でなければならない。ここで、

$irow = \text{mod}(ia-1, mb_a)$ 、
 $icoffa = \text{mod}(ja-1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、
 $mpa0 = \text{numroc}(m+irow, mb_a, MYROW, iarow, NPROW)$ 、
 $nqa0 = \text{numroc}(n+icoffa, nb_a, MYCOL, iacol, NPCOL)$

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a $m \times n$ の分散行列 *Q* のローカル部分が格納される。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 *i* 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ungrq

p?gerqf で求めた *RQ* 因子分解のユニタリ行列 *Q* を生成する。

構文

```
call pcungrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
call pzungrq( m, n, k, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交列を持つ $m \times n$ の複素分散行列 Q の全体または一部を生成する。これは、以下の次数 n の k 個の基本リフレクターの積の最後の m 列として定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ 。

$$Q = H(1)' H(2)' \dots H(k)'$$

[p?gerqf](#) によって返される。

入力パラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(Q)$ の列数 ($n \geq m \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q となる基本リフレクターの個数 ($m \geq k \geq 0$)。
a	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。 i 番目の行は、 p?gerqf によって分散行列引数 $a(ia+m-k:ia+m-1, ja:*)$ の k 行に返される、基本リフレクター $H(i)$, $ia+m-k \leq i \leq ia+m-1$ を定義するベクトルを含んでいなければならない。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
tau	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) 配列、次元は $LOCr(ia+m-1)$ 。 p?gerqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $tau(i)$ を含む。 tau は、分散行列 A に関連付けられる。
$work$	(ローカル) COMPLEX (pcungrq の場合) DOUBLE COMPLEX (pzungrq の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq mb_a * (mpa0 + nqa0 + mb_a)$ でなければならない。ここで、 $irow = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mpa0 = \text{numroc}(m+irow, mb_a, MYROW, iarow, NPROW)$ 、

```
nqa0 = numroc(n+icoffa, nb_a, MYCOL, iacol, NPCOL)
```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a $m \times n$ の分散行列 Q のローカル部分が格納される。

$work(1)$ 終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。

$info$ (グローバル) INTEGER。
 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?ormrq

一般行列に p?gerqf によって求めた RQ 因子分解の直交行列 Q を掛ける。

構文

```
call psormrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pdormrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$:	$Q \text{sub}(C)$	$\text{sub}(C) Q$
$trans = 'T'$:	$Q^T \text{sub}(C)$	$\text{sub}(C) Q^T$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

[p?gerqf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 Q または Q^T は左側から適用される。 ='R' の場合、 Q または Q^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='T' の場合、 Q^T が適用される (転置)。
<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) (<i>side</i> = 'L' の場合) または (lld_a , $LOCc(ja+n-1)$) (<i>side</i> = 'R' の場合) の配列へのポインター。 i 番目の行は、p?gelqf によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の k 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormqr の場合) DOUBLE PRECISION (pdormqr の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?gerqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。 τ は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psormrq の場合) DOUBLE PRECISION (pdormrq の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。

<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₁)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psormrq</i> の場合) DOUBLE PRECISION (<i>pdormrq</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0)) * mb_a) + mb_a * mb_a)$ <i>side</i> = 'R' の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + nqc0) * mb_a) + mb_a * mb_a$ でなければならない。 ここで、 $lcmp = lcm / NPROW$ で $lcm = ilcm(NPROW, NPCOL)$ 、 $iroffa = \text{mod}(ia - 1, mb_a)$ 、 $icoffa = \text{mod}(ja - 1, nb_a)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mqa0 = \text{numroc}(n + icoffa, nb_a, MYCOL, iacol, NPCOL)$ 、 $iroffc = \text{mod}(ic - 1, mb_c)$ 、 $icoffc = \text{mod}(jc - 1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW)$ 、 $iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL)$ 、 $mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW)$ 、 $nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$ <i>ilcm</i> 、 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>MYROW</i> 、 <i>MYCOL</i> 、 <i>NPROW</i> および <i>NPCOL</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?unmrq

一般行列に p?gerqf によって求めた RQ 因子分解のユニタリー行列 Q を掛ける。

構文

```
call pcunmrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pzunmrq( side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ を以下により上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N':	$Q \text{sub}(C)$	$\text{sub}(C) Q$
<i>trans</i> = 'C':	$Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される複素ユニタリー分散行列である。

$$Q = H(1)' H(2)' \dots H(k)'$$

[p?gerqf](#) によって返される。 Q の次数は、*side* = 'L' の場合は m 、*side* = 'R' の場合は n である。

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^H は左側から適用される。
 = 'R' の場合、 Q または Q^H は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'C' の場合、 Q^H が適用される (共役転置)。

m (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。

<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) (<i>side</i> = 'L' の場合) または (lld_a , $LOCc(ja+n-1)$) (<i>side</i> = 'R' の場合) の配列へのポインター。 <i>i</i> 番目の行は、p?gerqf によって分散行列引数 $a(ia:ia+k-1, ja^*)$ の <i>k</i> 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja^*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合) 配列、次元は $LOCc(ja+k-1)$ 。 p?gerqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。 τ は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合) ローカルメモリーにある、ローカル次元 (lld_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmrq の場合) DOUBLE COMPLEX (pzunmrq の場合) 次元 $lwork$ のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 $work$ の次元は <i>side</i> = 'L' の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0)) * mb_a) + mb_a * mb_a$

$side = 'R'$ の場合、

$lwork \geq \max ((mb_a*(mb_a-1))/2, (mpc0 + nqc0)*mb_a) + mb_a * mb_a$
でなければならない。

ここで、

$lcmp = lcm / NPROW$ で $lcm = ilcm (NPROW, NPCOL)$ 、
 $iroffa = \text{mod}(ia-1, mb_a)$ 、
 $icoffa = \text{mod}(ja-1, nb_a)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、
 $mqa0 = \text{numroc}(m+icoffa, nb_a, MYCOL, iacol, NPCOL)$ 、
 $iroffc = \text{mod}(ic-1, mb_c)$ 、
 $icoffc = \text{mod}(jc-1, nb_c)$ 、
 $icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW)$ 、
 $iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL)$ 、
 $mpc0 = \text{numroc}(m+iroffc, mb_c, MYROW, icrow, NPROW)$ 、
 $nqc0 = \text{numroc}(n+icoffc, nb_c, MYCOL, iccol, NPCOL)$

$ilcm$ 、 $indxg2p$ および $numroc$ は、ScaLAPACK ツール関数である。
 $MYROW$ 、 $MYCOL$ 、 $NPROW$ および $NPCOL$ は、サブルーチン
 $blacs_gridinfo$ を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペース
のクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最
適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエン
トリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- c 積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによ
って上書きされる。
- $work(1)$ 終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な
 $lwork$ の最小値が格納される。
- $info$ (グローバル) INTEGER。
 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、
 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、
 $info = -i$ 。

p?tzrzf

上台形行列 A を上三角形式に縮退させる。

構文

```
call pstzrzf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pdtzrzf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pctzrzf( m, n, a, ia, ja, desca, tau, work, lwork, info )
call pztzrzf( m, n, a, ia, ja, desca, tau, work, lwork, info )
```

説明

このルーチンは、直交/ユニタリー変換を用いて、 $m \times n$ ($m \leq n$) の実/複素上台形行列 $\text{sub}(A) = (ia:ia+m-1, ja:ja+n-1)$ を上三角形式に縮退させる。上台形行列 A は、次のように因子分解される。

$$A = (R \ 0) * Z.$$

Z は $n \times n$ の直交/ユニタリー行列、 R は $m \times m$ の上三角行列である。

入力パラメーター

m	(グローバル) INTEGER。部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
a	(ローカル) REAL (pstzrzf の場合) DOUBLE PRECISION (pdtzrzf の場合) COMPLEX (pctzrzf の場合) DOUBLE COMPLEX (pztzrzf の場合)。 ローカルメモリーにある、次元 (lld_a , $LOC(ja+n-1)$) の配列へのポインター。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を含む。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
$work$	(ローカル) REAL (pstzrzf の場合) DOUBLE PRECISION (pdtzrzf の場合) COMPLEX (pctzrzf の場合) DOUBLE COMPLEX (pztzrzf の場合) 次元 $lwork$ のワークスペース配列。
$lwork$	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq mb_a * (mp0 + nq0 + mb_a)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb_a)$ 、

```

icoff = mod(ja-1, nb_a),
iarow = indxg2p(ia, mb_a, MYROW, rsrc_a, NPROW),
iacol = indxg2p(ja, nb_a, MYCOL, csrc_a, NPCOL),
mp0 = numroc(m+iroff, mb_a, MYROW, iarow, NPROW),
nq0 = numroc(n+icoff, nb_a, MYCOL, iacol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<code>a</code>	終了時に、 <code>sub(A)</code> の先頭の $m \times m$ 上三角部分には上三角行列 R が格納される。 <code>sub(A)</code> の最初の m 行の $m+1$ から n までの成分は、配列 <code>tau</code> とともに、基本リフレクター m の積として直交/ユニタリー行列 Z を表す。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>tau</code>	(ローカル) REAL (<code>pstzrzf</code> の場合) DOUBLE PRECISION (<code>pdtzrzf</code> の場合) COMPLEX (<code>pctzrzf</code> の場合) DOUBLE COMPLEX (<code>pztzrzf</code> の場合)。 配列、次元は $LOCr(ia+m-1)$ 。 基本リフレクターのスカラー係数が格納される。 <code>tau</code> は、分散行列 A に関連付けられる。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info < 0</code> の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

因子分解は Householder 法を用いて得る。`sub(A)` の $(m - k + 1)$ 番目の行にゼロを導入するために共役置換が使用される k 番目の変換行列 $Z(k)$ は、次の形式になる。

$$Z(k) = \begin{bmatrix} i & 0 \\ 0 & T(k) \end{bmatrix}$$

ここで、

$$T(k) = i - \tau * u(k) * u(k)',$$

$$u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$$

τ はスカラー、 $Z(k)$ は $(n-m)$ 成分で構成されるベクトルである。 τ および $Z(k)$ は、 $\text{sub}(A)$ の k 番目の行の成分をゼロにするために選択される。スカラー τ は、 $Z(k)$ の成分が $a(k, m+1), \dots, a(k, n)$ にあるように、 τ の k 番目の成分および $\text{sub}(A)$ の k 番目の行のベクトル $u(k)$ で返される。 R の成分は $\text{sub}(A)$ の上三角部分で返される。 Z は次の式で与えられる。

$$Z = Z(1) * Z(2) * \dots * Z(m)$$

p?ormrz

一般行列に p?tzrzf によって求めた上三角形式に縮退した直交行列を掛ける。

構文

```
call psormrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pdormrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'T':$	$Q^T \text{ sub}(C)$	$\text{sub}(C) Q^T$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される実直交分散行列である。

$$Q = H(1) H(2) \dots H(k)$$

p?tzrzf によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^T は左側から適用される。
 = 'R' の場合、 Q または Q^T は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'T' の場合、 Q^T が適用される (転置)。

<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>k</i>	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>l</i>	(グローバル)。 Householder リフレクターとして意味を持つ部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 、 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合) ローカルメモリーにある、次元 (ll_d_a , $LOCc(ja+m-1)$) (<i>side</i> = 'L' の場合) または (ll_d_a , $LOCc(ja+n-1)$) (<i>side</i> = 'R' の場合) の配列へのポインター。ここで、 $ll_d_a \geq \max(1, LOCr(ia+k-1))$ である。 <i>i</i> 番目の行は、p?tzrzf によって分散行列引数 $a(ia:ia+k-1, ja:*)$ の <i>k</i> 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合) 配列、次元は $LOCc(ia+k-1)$ 。 p?tzrzf によって返される、基本リフレクター $H(i)$ のスカラー係数 $\tau(i)$ を含む。tau は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) REAL (psormrz の場合) DOUBLE PRECISION (pdormrz の場合) ローカルメモリーにある、ローカル次元 (ll_d_c , $LOCc(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $\text{sub}(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル)

REAL (psormrz の場合)
 DOUBLE PRECISION (pdormrz の場合) 次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は *side* = 'L' の場合、

$$lwork \geq \max ((mb_a * (mb_a - 1)) / 2, (mpc0 + \max (mqa0 + \text{numroc}(\text{numroc}(n + iroffa, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0)) * mb_a) + mb_a * mb_a)$$

side = 'R' の場合、

$$lwork \geq \max ((mb_a * (mb_a - 1)) / 2, (mpc0 + nqc0) * mb_a) + mb_a * mb_a$$

でなければならない。

ここで、

$$lcmp = lcm / NPROW \text{ で } lcm = ilcm(NPROW, NPCOL),$$

$$iroffa = \text{mod}(ia - 1, mb_a), \text{ icoffa} = \text{mod}(ja - 1, nb_a),$$

$$iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL),$$

$$mqa0 = \text{numroc}(n + icoffa, nb_a, MYCOL, iacol, NPCOL),$$

$$iroffc = \text{mod}(ic - 1, mb_c),$$

$$icoffc = \text{mod}(jc - 1, nb_c),$$

$$icrow = \text{indxg2p}(ic, mb_c, MYROW, rsrc_c, NPROW),$$

$$iccol = \text{indxg2p}(jc, nb_c, MYCOL, csrc_c, NPCOL),$$

$$mpc0 = \text{numroc}(m + iroffc, mb_c, MYROW, icrow, NPROW),$$

$$nqc0 = \text{numroc}(n + icoffc, nb_c, MYCOL, iccol, NPCOL)$$

ilcm, *indxg2p* および *numroc* は、ScaLAPACK ツール関数である。
MYROW, *MYCOL*, *NPROW* および *NPCOL* は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c 積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによって上書きされる。

work(1) 終了時に、*work*(1) には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100 + *j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?unmrz

一般行列に p?tzrpf によって求めた上三角形式に縮退したユニタリー変換行列を掛ける。

構文

```
call pcunmrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

```
call pzunmrz( side, trans, m, n, k, l, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ を以下により上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N':$	$Q \text{ sub}(C)$	$\text{sub}(C) Q$
$trans = 'C':$	$Q^H \text{ sub}(C)$	$\text{sub}(C) Q^H$

ここで、 Q は、 k 個の以下の基本リフレクターの積として定義される複素ユニタリー分散行列である。

$$Q = H(1)' H(2)' \dots H(k)'$$

[pctzrpf](#)/[pztzrpf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

$side$	(グローバル) CHARACTER ='L' の場合、 Q または Q^H は左側から適用される。 ='R' の場合、 Q または Q^H は右側から適用される。
$trans$	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='C' の場合、 Q^H が適用される (共役転置)。
m	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
n	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
k	(グローバル) INTEGER。その積が行列 Q を定義する基本リフレクターの個数。次の制約がある。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 、 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
l	(グローバル)。 Householder リフレクターとして意味を持つ部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 $side = 'L'$ の場合、 $m \geq l \geq 0$ 、 $side = 'R'$ の場合、 $n \geq l \geq 0$ 。

<i>a</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合) ローカルメモリーにある、次元 ($lld_a, LOCC(ja+m-1)$) ($side = 'L'$ の場合) または ($lld_a, LOCC(ja+n-1)$) ($side = 'R'$ の場合) の配列へのポインター。ここで、 $lld_a \geq \max(1, LOCr(ja+k-1))$ である。 i 番目の行は、 p?gerqf によって分散行列引数 $a(ia:ia+k-1, ja^*)$ の k 行に返される、基本リフレクター $H(i)$, $ia \leq i \leq ia+k-1$ を定義するベクトルを含んでいなければならない。 $a(ia:ia+k-1, ja^*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合) 配列、次元は $LOCC(ia+k-1)$ 。 p?gerqf によって返される、基本リフレクター $H(i)$ のスカラー係数 $tau(i)$ を含む。 tau は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合) ローカルメモリーにある、ローカル次元 ($lld_c, LOCC(jc+n-1)$) の配列へのポインター。 因子分解する分散行列 $sub(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmrz の場合) DOUBLE COMPLEX (pzunmrz の場合) 次元 $lwork$ のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $side = 'L'$ の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(n + iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0)) * mb_a) + mb_a * mb_a)$ $side = 'R'$ の場合、 $lwork \geq \max((mb_a * (mb_a - 1)) / 2, (mpc0 + nqc0) * mb_a) + mb_a * mb_a$ でなければならない。

ここで、

```
lcmp = lcm / NPROW で lcm = ilcm (NPROW, NPCOL)、
iroffa = mod(ia-1, mb_a)、
icoffa = mod(ja-1, nb_a)、
iacol = indxg2p (ja, nb_a, MYCOL, csrc_a, NPCOL)、
mqa0 = numroc(m+icoffa, nb_a, MYCOL, iacol, NPCOL)、
iroffc = mod(ic-1, mb_c)、
icoffc = mod(jc-1, nb_c)、
icrow = indxg2p(ic, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p(jc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(m+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(n+icoffc, nb_c, MYCOL, iccol, NPCOL)
```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。
MYROW、MYCOL、NPROW および NPCOL は、サブルーチン
blacs_gridinfo を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペース
のクエリとみなされ、ルーチンはすべての *work* 配列の最小かつ最
適なサイズだけを計算する。各値は該当する *work* 配列の最初のエン
トリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	積 $Q * \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかによ って上書きされる。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?ggqrf

汎用 QR 因子分解を行う。

構文

```
call psggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
call pdggqrf(n, m, p, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
```


<i>a</i>	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) の配列へのポインター。因子分解する $m \times n$ の行列 $\text{sub}(A)$ のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>b</i>	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合) ローカルメモリーにある、次元 (lld_b , $LOCc(ja+p-1)$) の配列へのポインター。因子分解する $n \times p$ の行列 $\text{sub}(B)$ のローカル部分を含む。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 B の最初の行と最初の列を示す、グローバル配列 b の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psggqrf の場合) DOUBLE PRECISION (pdggqrf の場合) COMPLEX (pcggqrf の場合) DOUBLE COMPLEX (pzggqrf の場合) 次元 $lwork$ のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 $work$ の次元は $lwork \geq \max(nb_a * (npa0 + mqa0 + nb_a), \max((nb_a * (nb_a - 1)) / 2, (pqb0 + npb0) * nb_a) + nb_a * nb_a, mb_b * (npb0 + pqb0 + mb_b))$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $npa0 = \text{numroc}(n + iroffa, mb_a, MYROW, iarow, NPROW)$ 、 $mqa0 = \text{numroc}(m + icoffa, nb_a, MYCOL, iacol, NPCOL)$ $iroffb = \text{mod}(ib-1, mb_b)$ 、 $icoffb = \text{mod}(jb-1, nb_b)$ 、 $ibrow = \text{indxg2p}(ib, mb_b, MYROW, rsrc_b, NPROW)$ 、 $ibcol = \text{indxg2p}(jb, nb_b, MYCOL, csrc_b, NPCOL)$ 、

$npb0 = \text{numroc}(n + i\text{roffa}, mb_b, \text{MYROW}, ibrow, \text{NPROW}),$

$pqb0 = \text{numroc}(m + i\text{coffb}, nb_b, \text{MYCOL}, ibcol, \text{NPCOL})$

numroc および indxg2p は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最適なサイズだけを計算する。各値は該当する work 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、 $\text{sub}(A)$ の対角成分とその上の成分は、 $\min(n,m) \times m$ の上台形行列 R ($n \geq m$ の場合、 R は上三角行列) によって上書きされる。対角より下の成分は、配列 taua とともに、 $\min(n,m)$ 個の基本リフレクターの積として直交/ユニタリー行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
- taua, taub** (ローカル)
 REAL (psggqrf の場合)
 DOUBLE PRECISION (pdggqrf の場合)
 COMPLEX (pcggqrf の場合)
 DOUBLE COMPLEX (pzggqrf の場合)
 配列、次元は $\text{LOCc}(ja + \min(n,m) - 1)$ (taua の場合)、 $\text{LOCr}(ib + n - 1)$ (taub の場合)。配列 taua には、直交/ユニタリー行列 Q を表す基本リフレクターのスカラー係数が格納される。
 taua は、分散行列 A に関連付けられる。(次の「アプリケーション・ノート」を参照)。
 配列 taub には、直交/ユニタリー行列 Z を表す基本リフレクターのスカラー係数が格納される。
 taub は、分散行列 B に関連付けられる。(次の「アプリケーション・ノート」を参照)。
- work(1)** 終了時に、 $\text{work}(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
- info** (グローバル) INTEGER。
 $\text{info} = 0$ の場合、実行は正常に終了したことを示す。
 $\text{info} < 0$ の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $\text{info} = -(i * 100 + j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $\text{info} = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ja) H(ja+1) \dots H(ja+k-1)$$

ここで、 $k = \min(n,m)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = i - \text{taua} * v * v'$$

ここで、 τ_{aua} は実 / 複素スカラー、 v は実 / 複素ベクトルで、 $v(1:i-1)=0$ および $v(i)=1$ 。終了時に $v(i+1:n)$ は $A(ia+i:ia+n-1, ja+i-1)$ に τ_{aua} は $\tau_{aua}(ja+i-1)$ に格納される。 Q を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr/p?ungqr](#) を使用する。 Q を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr/p?unmqr](#) を使用する。

行列 Z は、次の基本リフレクターの積として表現される。

$$Z = H(ib) H(ib+1) \dots H(ib+k-1)$$

ここで、 $k = \min(n, p)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau_{aub} * v * v'$$

ここで、 τ_{aub} は実 / 複素スカラー、 v は実 / 複素ベクトルで、 $v(p-k+i+1:p)=0$ および $v(p-k+i)=1$ 。終了時に、 $v(1:p-k+i-1)$ は $B(ib+n-k+i-1, jb:jb+p-k+i-2)$ に、 τ_{aub} は $\tau_{aub}(ib+n-k+i-1)$ に格納される。 Z を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr/p?ungqr](#) を使用する。 Z を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr/p?unmqr](#) を使用する。

p?ggrqf

汎用 RQ 因子分解を行う。

構文

```
call psggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
call pdggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
call pcggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
call pzggrqf(m, p, n, a, ia, ja, desca, taua, b, ib, jb, descb, taub,
            work, lwork, info)
```

説明

このルーチンは、 $m \times n$ の行列 $\text{sub}(A)=(ia:ia+m-1, ja:ja+n-1)$ および $p \times n$ の行列 $\text{sub}(B)=(ib:ib+p-1, ja:ja+n-1)$ の汎用 RQ 因子分解を、

$$\text{sub}(A) = RQ, \quad \text{sub}(B) = ZTQ \quad \text{として行う。}$$

ここで、 Q は $n \times n$ の直交 / ユニタリー行列、 Z は $p \times p$ の直交 / ユニタリー行列、 R および T は、次のいずれかの形式である。

$$R = \begin{matrix} m & (& 0 & R_{12}) \\ & n-m & m \end{matrix}, \quad (m \leq n \text{ の場合}),$$

または

<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psggrqf</i> の場合) DOUBLE PRECISION (<i>pdggrqf</i> の場合) COMPLEX (<i>pcggrqf</i> の場合) DOUBLE COMPLEX (<i>pzggqrf</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq \max(mb_a * (mpa0 + nqa0 + mb_a), \max((mb_a * (mb_a - 1)) / 2, (ppb0 + nqb0) * mb_a) + mb_a * mb_a, nb_b * (ppb0 + nqb0 + nb_b))$ でなければならない。ここで、 $iroffa = \text{mod}(ia - 1, mb_a)$ 、 $icoffa = \text{mod}(ja - 1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、 $iacol = \text{indxg2p}(ja, nb_a, MYCOL, csrc_a, NPCOL)$ 、 $mpa0 = \text{numroc}(m + iroffa, mb_a, MYROW, iarow, NPROW)$ 、 $nqa0 = \text{numroc}(m + icoffa, nb_a, MYCOL, iacol, NPCOL)$ $iroffb = \text{mod}(ib - 1, mb_b)$ 、 $icoffb = \text{mod}(jb - 1, nb_b)$ 、 $ibrow = \text{indxg2p}(ib, mb_b, MYROW, rsrc_b, NPROW)$ 、 $ibcol = \text{indxg2p}(jb, nb_b, MYCOL, csrc_b, NPCOL)$ 、 $ppb0 = \text{numroc}(p + iroffb, mb_b, MYROW, ibrow, NPROW)$ 、 $nqb0 = \text{numroc}(n + icoffb, nb_b, MYCOL, ibcol, NPCOL)$

numroc および *indxg2p* は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW および NPCOL は、サブルーチン *blacs_gridinfo* を呼び出して決定できる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、 $m \leq n$ の場合、 $A(ia:ia+m-1, ja+n-m:ja+n-1)$ の上三角部分に、 $m \times m$ の上三角行列 <i>R</i> が格納される。 $m \geq n$ の場合、(<i>m</i> - <i>n</i>) 番目の劣対角成分とその上の成分に、 $m \times n$ の上台形行列 <i>R</i> が格納される。残りの成分は、配列 <i>taua</i> とともに、 $\min(n, m)$ 個の基本リフレクターの積として直交/ユニタリー行列 <i>Q</i> を表す。(次の「アプリケーション・ノート」を参照)。
----------	---

<i>taua, taub</i>	(ローカル) REAL (p _{sggqrf} の場合) DOUBLE PRECISION (pd _{ggqrf} の場合) COMPLEX (pc _{ggqrf} の場合) DOUBLE COMPLEX (pz _{ggqrf} の場合) 配列、次元は $LOCr(ia+m-1)$ (<i>taua</i> の場合)、 $LOCc(jb+\min(p,n)-1)$ (<i>taub</i> の場合)。 配列 <i>taua</i> には、直交/ユニタリー行列 Q を表す基本リフレクターのスカラー係数が格納される。 <i>taua</i> は、分散行列 A に関連付けられる。(次の「アプリケーション・ノート」を参照)。 配列 <i>taub</i> には、直交/ユニタリー行列 Z を表す基本リフレクターのスカラー係数が格納される。 <i>taub</i> は、分散行列 B に関連付けられる。(次の「アプリケーション・ノート」を参照)。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ia) H(ia+1) \dots H(ia+k-1)$$

ここで、 $k = \min(m,n)$ である。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau_{ia} * v * v'$$

ここで、 τ_{ia} は実/複素スカラー、 v は実/複素ベクトルで、 $v(n-k+i+1:n) = 0$ および $v(n-k+i) = 1$ 。終了時に、 $v(1:n-k+i-1)$ は $A(ia+m-k+i-1, ja:ja+n-k+i-2)$ に、 τ_{ia} は $\tau_{ia}(ia+m-k+i-1)$ に格納される。 Q を明示的に生成するには、ScaLAPACK サブルーチン [p?orqrg/p?unqrg](#) を使用する。 Q を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormrg/p?unmrg](#) を使用する。

行列 Z は、次の基本リフレクターの積として表現される。

$$Z = H(jb) H(jb+1) \dots H(jb+k-1)$$

ここで、 $k = \min(p,n)$ 。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau_{ia} * v * v'$$

ここで、 $taub$ は実 / 複素スカラー、 v は実 / 複素ベクトルで、 $v(1:i-1) = 0$ および $v(i) = 1$ 。終了時に、 $v(i+1:p)$ は $B(ib+i:ib+p-1, jb+i-1)$ に、 $taub$ は $taub(jb+i-1)$ に格納される。 Z を明示的に生成するには、ScaLAPACK サブルーチン [p?orgqr](#)/[p?ungqr](#) を使用する。 Z を使用して他の行列を更新するには、ScaLAPACK サブルーチン [p?ormqr](#)/[p?unmqr](#) を使用する。

対称固有値問題

ScaLAPACK を使用して対称固有値問題を解くには、通常、行列を実三重対角形式 T に縮退させてから三重対角行列 T の固定値と固定ベクトルを見つける必要がある。

ScaLAPACK には、三重対角対称固定値問題を解くルーチンだけでなく、直交 (またはユニタリー) 相似変換 $A = QTQ^H$ を用いて行列を三重対角形式に縮退させるルーチンが含まれている。これらのルーチンのリストを、[表 6-4](#) に示す。

必要な項目 (固有値のみ、または固有値と固有ベクトル) と使用するアルゴリズム (QR アルゴリズム、または二分法と逆反復法) に応じて、さまざまな対称固有値問題用のルーチンがある。

表 6-4 対称固有値問題を解くための計算ルーチン

機能	密対称 / エルミート 行列	直交 / ユニタリー 行列	対称三重対角行列
三重対角形式に縮退させる $A = QTQ^H$	p?sytrd / p?hetrd		
縮退後に行列を掛ける		p?ormtr / p?unmtr	
QR 法を用いて、三重対角行列 T の固有値と固有ベクトルをす べて求める			?stegr2 ^{*)}
二分法を用いて、三重対角行 列 T の固有値を選択的に求め る			p?stebz
逆反復法を用いて、三重対角 行列 T の固有ベクトルを選択 的に求める			p?stein

*) このルーチンは、補助 ScaLAPACK ルーチンの一部として説明される。

p?sytrd

直交相似変換を用いて、対称行列を実数対称三重対角形式に縮退させる。

構文

```
call pssytrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
call pdsytrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
```

説明

このルーチンは、直交相似変換を用いて、実対称行列 $\text{sub}(A)$ を対称三重対角形式 T に縮退させる。

$$Q' \text{sub}(A) * Q = T$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+\text{n}-1, \text{ja}:\text{ja}+\text{n}-1)$ 。

入力パラメーター

<code>uplo</code>	(グローバル) CHARACTER。 対称行列 <code>sub(A)</code> の上三角部分と下三角部分のどちらが格納されるかを指定する。 <code>uplo = 'U'</code> の場合は、上三角部分。 <code>uplo = 'L'</code> の場合は、下三角部分。
<code>n</code>	(グローバル) INTEGER。分散行列 <code>sub(A)</code> の次数 ($n \geq 0$)。
<code>a</code>	(ローカル) REAL (<code>pssytrd</code> の場合) DOUBLE PRECISION (<code>pdsytrd</code> の場合) ローカルメモリーにある、次元 (<code>lld_a</code> , <code>LOC(ja+n-1)</code>) の配列へのポインター。呼び出し時に、この配列は、対称分散行列 <code>sub(A)</code> のローカル部分を含む。 <code>uplo = 'U'</code> の場合、 <code>sub(A)</code> の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 <code>uplo = 'L'</code> の場合、 <code>sub(A)</code> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。厳密な上三角部分は参照されない。(次の「アプリケーション・ノート」を参照)。
<code>ia, ja</code>	(グローバル) INTEGER。それぞれ、部分行列 <code>A</code> の最初の行と最初の列を示す、グローバル配列 <code>a</code> の行インデックスと列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER 配列、次元は (<code>dlen_</code>)。分散行列 <code>A</code> の配列ディスクリプター。
<code>work</code>	(ローカル) REAL (<code>pssytrd</code> の場合) DOUBLE PRECISION (<code>pdsytrd</code> の場合) 次元 <code>lwork</code> のワークスペース配列。
<code>lwork</code>	(ローカルまたはグローバル) INTEGER。 <code>work</code> の次元は $lwork \geq \max(NB * (np + 1), 3 * NB)$ でなければならない。 ここで、 $NB = mb_a = nb_a$ 、 $np = \text{numroc}(n, NB, MYROW, iarow, NPROW)$ 、 $iarow = \text{indxg2p}(ia, NB, MYROW, rsrc_a, NPROW)$ <code>indxg2p</code> および <code>numroc</code> は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <code>blacs_gridinfo</code> を呼び出して決定できる。 <code>lwork = -1</code> の場合、 <code>lwork</code> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <code>work</code> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <code>work</code> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、*uplo* = 'U' の場合、**sub(A)** の対角成分と最初の優対角成分は、三重対角行列 *T* の対応する成分によって上書きされる。最初の優対角より上の成分は、配列 *tau* とともに、基本リフレクターの積として直交行列 *Q* を表す。*uplo* = 'L' の場合、**sub(A)** の対角成分と最初の劣対角成分は、三重対角行列 *T* の対応する成分によって上書きされる。最初の劣対角より下の成分は、配列 *tau* とともに、基本リフレクターの積として直交行列 *Q* を表す。(次の「アプリケーション・ノート」を参照)。
- d** (ローカル)
REAL (pssytrd の場合)
DOUBLE PRECISION (pdsytrd の場合)
配列、次元は *LOCc(ja+n-1)*。三重対角行列 *T* の対角成分。
 $d(i) = A(i, i)$
d は、分散行列 *A* に関連付けられる。
- e** (ローカル)
REAL (pssytrd の場合)
DOUBLE PRECISION (pdsytrd の場合)
配列、次元は *LOCc(ja+n-1)* (*uplo* = 'U' の場合)、または *LOCc(ja+n-2)* (それ以外の場合)。三重対角行列 *T* の非対角成分。
 $e(i) = A(i, i+1)$ (*uplo* = 'U' の場合)
 $e(i) = A(i+1, i)$ (*uplo* = 'L' の場合)
e は分散行列 *A* に関連付けられる。
- tau** (ローカル)
REAL (pssytrd の場合)
DOUBLE PRECISION (pdsytrd の場合)
配列、次元は *LOCc(ja+n-1)*。この配列には、基本リフレクターのスカラー係数 *tau* が格納される。
tau は、分散行列 *A* に関連付けられる。
- work(1)** 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。
- info** (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

アプリケーション・ノート

uplo = 'U' の場合、行列 *Q* は、以下の基本リフレクターの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの *H(i)* は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実スカラー、 v は実ベクトルで $v(i+1:n)=0$ および $v(i)=1$ である。終了時に、 $v(1:i-1)$ は $A(ia:ia+i-2,ja+i)$ に、 τ は $\tau(ja+i-1)$ に格納される。

$uplo='L'$ の場合、行列 Q は、以下の基本リフレクターの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実スカラー、 v は実ベクトルで $v(1:i)=0$ および $v(i+1)=1$ である。出力時、 $v(i+2:n)$ は $A(ia+i+1:ia+n-1,ja+i-1)$ に、 τ は $\tau(ja+i-1)$ に格納される。

終了時の $sub(A)$ の内容 ($n=5$ の場合) を次に示す。

$uplo='U'$ の場合、

$$\begin{bmatrix} d & e & v2 & v3 & v4 \\ & d & e & v3 & v4 \\ & & d & e & v4 \\ & & & d & e \\ & & & & d \end{bmatrix}$$

$uplo='L'$ の場合、

$$\begin{bmatrix} d & & & & \\ e & d & & & \\ v1 & e & d & & \\ v1 & v2 & e & d & \\ v1 & v2 & v3 & e & d \end{bmatrix}$$

ここで、 d と e は T の対角成分と非対角成分である。 v_i は $H(i)$ を定義するベクトルの成分を表す。

p?ormtr

一般行列に p?sytrd によって求めた上三角形式に縮退した直交変換行列を掛ける。

構文

```
call psormtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pdormtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$$\begin{array}{ll} \text{side} = 'L' & \text{side} = 'R' \\ \text{trans} = 'N': & Q \text{sub}(C) \quad \text{sub}(C) Q \\ \text{trans} = 'T': & Q^T \text{sub}(C) \quad \text{sub}(C) Q^T \end{array}$$

ここで、 Q は、次数 nq の実直交分散行列で、 $nq = m$ ($\text{side} = 'L'$ の場合)、 $nq = n$ ($\text{side} = 'R'$ の場合) である。 Q は、[p?sytrd](#) によって返される、 nq 個の基本リフレクターの積として定義される。

$\text{uplo} = 'U'$ の場合、 $Q = H(nq-1) \dots H(2) H(1)$ 。

$\text{uplo} = 'L'$ の場合、 $Q = H(1) H(2) \dots H(nq-1)$ 。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER ='L' の場合、 Q または Q^T は左側から適用される。 ='R' の場合、 Q または Q^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER ='N' の場合、 Q が適用される (転置なし)。 ='T' の場合、 Q^T が適用される (転置)。
<i>uplo</i>	(グローバル) CHARACTER。 ='U' の場合、 $A(ia:*, ja:*)$ の上三角は p?sytrd からの基本リフレクターを格納する。 ='L' の場合、 $A(ia:*, ja:*)$ の下三角は p?sytrd からの基本リフレクターを格納する。
<i>m</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+m-1)$) ($\text{side} = 'L'$ の場合) または ($lld_a, LOCc(ja+n-1)$) ($\text{side} = 'R'$ の場合) の配列へのポインター。 p?sytrd によって返される、基本リフレクターを定義する一連のベクトルを含む。 $\text{side} = 'L'$ の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ $\text{side} = 'R'$ の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

<i>tau</i>	(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合) 配列、 <i>ltau</i> の次元。ここで、 <i>side</i> = 'L' および <i>uplo</i> = 'U' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>m_a</i>)、 <i>side</i> = 'L' および <i>uplo</i> = 'L' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>ja+m-2</i>)、 <i>side</i> = 'R' および <i>uplo</i> = 'U' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>n_a</i>)、 <i>side</i> = 'R' および <i>uplo</i> = 'L' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>ja+n-2</i>)。 <i>tau</i> (<i>i</i>) は、 <i>p?sytrd</i> によって返される、基本リフレクター <i>H</i> (<i>i</i>) のスカラー係数を含んでいなければならない。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja+n-1</i>)) の配列へのポインター。分散行列 <i>sub</i> (<i>C</i>) のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psormtr の場合) DOUBLE PRECISION (pdormtr の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>uplo</i> = 'U' の場合、 <i>iaa</i> = <i>ia</i> 、 <i>jaa</i> = <i>ja+1</i> 、 <i>icc</i> = <i>ic</i> 、 <i>jcc</i> = <i>jc</i> 。 それ以外の場合、 <i>uplo</i> = 'L'、 <i>iaa</i> = <i>ia+1</i> 、 <i>jaa</i> = <i>ja</i> 。 <i>side</i> = 'L' の場合、 <i>icc</i> = <i>ic+1</i> 、 <i>jcc</i> = <i>jc</i> 。 それ以外の場合、 <i>icc</i> = <i>ic</i> 、 <i>jcc</i> = <i>jc+1</i> でなければならない。 <i>side</i> = 'L' の場合、 <i>mi</i> = <i>m-1</i> 、 <i>ni</i> = <i>n</i> 、 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + mpc0)*nb_a) + nb_a * nb_a$ <i>side</i> = 'R' の場合、 <i>mi</i> = <i>m</i> 、 <i>ni</i> = <i>n-1</i> 、 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + \max(np_a0 + \text{numroc}(\text{numroc}(ni+icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0))*nb_a) + nb_a * nb_a$ でなければならない。

```

ここで、 $lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ 、
 $iroffa = \text{mod}(iaa-1, mb\_a)$ 、
 $icoffa = \text{mod}(jaa-1, nb\_a)$ 、
 $iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ 、
 $npa0 = \text{numroc}(ni+iroffa, mb\_a, MYROW, iarow, NPROW)$ 、
 $iroffc = \text{mod}(icc-1, mb\_c)$ 、
 $icoffc = \text{mod}(jcc-1, nb\_c)$ 、
 $icrow = \text{indxg2p}(icc, mb\_c, MYROW, rsrc\_c, NPROW)$ 、
 $iccol = \text{indxg2p}(jcc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、
 $mpc0 = \text{numroc}(mi+iroffc, mb\_c, MYROW, icrow, NPROW)$ 、
 $nqc0 = \text{numroc}(ni+icoffc, nb\_c, MYCOL, iccol, NPCOL)$ 

```

$ilcm$ 、 indxg2p および numroc は、ScaLAPACK ツール関数である。
 $MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および $NPCOL$ は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。 $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c 積 $Q \text{ sub}(C)$ 、 $Q' \text{ sub}(C)$ 、 $\text{sub}(C) Q'$ 、または $\text{sub}(C) Q$ のいずれかによって上書きされる。

work(1) 終了時に、*work(1)* には、最適なパフォーマンスを得るために必要な *lwork* の最小値が格納される。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合：*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = $-(i*100+j)$ 。*i* 番目の引数がスカラーで値が不正だった場合、*info* = $-i$ 。

p?hetrd

ユニタリー相似変換を用いて、エルミート行列をエルミート三重対角形式に縮退させる。

構文

```

call pchetrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )
call pzhetrd( uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info )

```

説明

このルーチンは、ユニタリー相似変換を用いて、複素エルミート行列 $\text{sub}(A)$ をエルミート三重対角形式 T に縮退させる。

$$Q' \text{sub}(A) Q = T$$

ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ 。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER。 エルミート行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。 <i>uplo</i> = 'U' の場合は、上三角部分。 <i>uplo</i> = 'L' の場合は、下三角部分。
<i>n</i>	(グローバル) INTEGER。分散行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) COMPLEX (<i>pchetrd</i> の場合) DOUBLE COMPLEX (<i>pzhetrdr</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja</i> + <i>n</i> -1)) の配列へのポインター。呼び出し時に、この配列は、エルミート分散行列 $\text{sub}(A)$ のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。(次の「アプリケーション・ノート」を参照)。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (<i>pchetrd</i> の場合) DOUBLE COMPLEX (<i>pzhetrdr</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq \max(\text{NB} * (np + 1), 3 * \text{NB})$ でなければならない。ここで、 $\text{NB} = mb_a = nb_a$ 、 $np = \text{numroc}(n, \text{NB}, \text{MYROW}, iarow, \text{NPROW})$ 、 $iarow = \text{indxg2p}(ia, \text{NB}, \text{MYROW}, rsrc_a, \text{NPROW})$ <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、 $uplo = 'U'$ の場合、 $sub(A)$ の対角成分と最初の優対角成分は、三重対角行列 T の対応する成分によって上書きされる。最初の優対角より上の成分は、配列 tau とともに、基本リフレクターの積として直交行列 Q を表す。 $uplo = 'L'$ の場合、 $sub(A)$ の対角成分と最初の劣対角成分は、三重対角行列 T の対応する成分によって上書きされる。最初の劣対角より下の成分は、配列 tau とともに、基本リフレクターの積として直交行列 Q を表す。(次の「アプリケーション・ノート」を参照)。
- d** (ローカル)
REAL (pchetrd の場合)
DOUBLE PRECISION (pzhetrd の場合)
配列、次元は $LOCc(ja+n-1)$ 。三重対角行列 T の対角成分。
 $d(i) = A(i, i)$
 d は、分散行列 A に関連付けられる。
- e** (ローカル)
REAL (pchetrd の場合)
DOUBLE PRECISION (pzhetrd の場合)
配列、次元は $LOCc(ja+n-1)$ ($uplo = 'U'$ の場合)、または $LOCc(ja+n-2)$ (それ以外の場合)。三重対角行列 T の非対角成分。
 $e(i) = A(i, i+1)$ ($uplo = 'U'$ の場合)
 $e(i) = A(i+1, i)$ ($uplo = 'L'$ の場合) e は分散行列 A に関連付けられる。
- tau** (ローカル)
COMPLEX (pchetrd の場合)
DOUBLE COMPLEX (pzhetrd の場合)
配列、次元は $LOCc(ja+n-1)$ 。この配列は、基本リフレクターのスカラー係数 tau を含む。 tau は、分散行列 A に関連付けられる。
- work(1)** 終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
- info** (グローバル) INTEGER。
 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

$uplo = 'U'$ の場合、行列 Q は、以下の基本リフレクターの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = i - \tau * v * v'$$

ここで、 τ は複素スカラー、 v は複素ベクトルで $v(i+1:n) = 0$ および $v(i) = 1$ である。終了時に、 $v(1:i-1)$ は $A(ia:ia+i-2, ja+i)$ に、 τ は $\tau(ja+i-1)$ に格納される。

$uplo = 'L'$ の場合、行列 Q は、以下の基本リフレクターの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = i - \tau * v * v'$$

ここで、 τ は複素スカラー、 v は複素ベクトルで $v(1:i) = 0$ および $v(i+1) = 1$ である。終了時に、 $v(i+2:n)$ は $A(ia+i+1:ia+n-1, ja+i-1)$ に、 τ は $\tau(ja+i-1)$ に格納される。

終了時の $sub(A)$ の内容 ($n = 5$ の場合) を次に示す。

$uplo = 'U'$ の場合、

$$\begin{bmatrix} d & e & v2 & v3 & v4 \\ & d & e & v3 & v4 \\ & & d & e & v4 \\ & & & d & e \\ & & & & d \end{bmatrix}$$

$uplo = 'L'$ の場合、

$$\begin{bmatrix} d & & & & \\ e & d & & & \\ v1 & e & d & & \\ v1 & v2 & e & d & \\ v1 & v2 & v3 & e & d \end{bmatrix}$$

ここで、 d と e は T の対角成分と非対角成分である。 v_i は $H(i)$ を定義するベクトルの成分を表す。

p?unmtr

一般行列に p?hetrd によって求めた三角形式に縮退したユニタリー変換行列を掛ける。

構文

```
call pcunmtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
call pzunmtr( side, uplo, trans, m, n, a, ia, ja, desca, tau, c, ic, jc,
             descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': Q \text{sub}(C)$	$\text{sub}(C) Q$
$trans = 'C': Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 Q は、次数 nq の複素ユニタリー分散行列で、 $nq = m$ ($side = 'L'$ の場合)、 $nq = n$ ($side = 'R'$ の場合) である。 Q は、[p?hetrd](#) によって返される、 $nq-1$ 個の基本リフレクターの積として定義される。

$uplo = 'U'$ の場合、 $Q = H(nq-1) \dots H(2) H(1)$ 。

$uplo = 'L'$ の場合、 $Q = H(1) H(2) \dots H(nq-1)$ 。

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^H は左側から適用される。
 = 'R' の場合、 Q または Q^H は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'C' の場合、 Q^H が適用される (共役転置)。

uplo (グローバル) CHARACTER。
 = 'U' の場合、 $A(ia:*, ja:*)$ の上三角は [p?hetrd](#) からの基本リフレクターを格納する。
 = 'L' の場合、 $A(ia:*, ja:*)$ の下三角は [p?hetrd](#) からの基本リフレクターを格納する。

m (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。

a (ローカル)
 REAL ([pcunmtr](#) の場合)
 DOUBLE PRECISION ([pzunmtr](#) の場合)
 ローカルメモリーにある、次元 ($lld_a, LOCc(ja+m-1)$) ($side = 'L'$ の場合) または ($lld_a, LOCc(ja+n-1)$) ($side = 'R'$ の場合) の配列へのポインター。
[p?hetrd](#) によって返される、基本リフレクターを定義する一連のベクトルを含む。
 $side = 'L'$ の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$
 $side = 'R'$ の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$

ia, ja (グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

<i>tau</i>	(ローカル) COMPLEX (pcunmtr の場合) DOUBLE COMPLEX (pzunmtr の場合) 配列、 <i>ltau</i> の次元。ここで、 <i>side</i> = 'L' および <i>uplo</i> = 'U' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>m_a</i>)、 <i>side</i> = 'L' および <i>uplo</i> = 'L' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>ja+m-2</i>)、 <i>side</i> = 'R' および <i>uplo</i> = 'U' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>n_a</i>)、 <i>side</i> = 'R' および <i>uplo</i> = 'L' の場合、 <i>ltau</i> = <i>LOCc</i> (<i>ja+n-2</i>)。 <i>tau</i> (<i>i</i>) は、 <i>p?hetrd</i> によって返される、基本リフレクター <i>H</i> (<i>i</i>) のスカラー係数を含んでいなければならない。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmtr の場合) DOUBLE COMPLEX (pzunmtr の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja+n-1</i>)) の配列へのポインター。分散行列 <i>sub</i> (<i>C</i>) のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmtr の場合) DOUBLE COMPLEX (pzunmtr の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は <i>uplo</i> = 'U' の場合、 <i>iaa</i> = <i>ia</i> 、 <i>jaa</i> = <i>ja+1</i> 、 <i>icc</i> = <i>ic</i> 、 <i>jcc</i> = <i>jc</i> 。 それ以外の場合、 <i>uplo</i> = 'L'、 <i>iaa</i> = <i>ia+1</i> 、 <i>jaa</i> = <i>ja</i> 。 <i>side</i> = 'L' の場合、 <i>icc</i> = <i>ic+1</i> 、 <i>jcc</i> = <i>jc</i> 。 それ以外の場合、 <i>icc</i> = <i>ic</i> 、 <i>jcc</i> = <i>jc+1</i> でなければならない。 <i>side</i> = 'L' の場合、 <i>mi</i> = <i>m-1</i> 、 <i>ni</i> = <i>n</i> 。 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + mpc0)*nb_a) + nb_a * nb_a$ <i>side</i> = 'R' の場合、 <i>mi</i> = <i>m</i> 、 <i>mi</i> = <i>n-1</i> 、 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb_a, 0, 0, NPCOL), *nb_a, 0, 0, lcmq), mpc0))*nb_a) + nb_a * nb_a$ でなければならない。

```

ここで、 $lcmq = lcm / NPCOL$  で  $lcm = ilcm(NPROW, NPCOL)$ 、
 $iroffa = \text{mod}(iaa-1, mb\_a)$ 、
 $icoffa = \text{mod}(jaa-1, nb\_a)$ 、
 $iarow = \text{indxg2p}(iaa, mb\_a, MYROW, rsrc\_a, NPROW)$ 、
 $npa0 = \text{numroc}(ni+iroffa, mb\_a, MYROW, iarow, NPROW)$ 、
 $iroffc = \text{mod}(icc-1, mb\_c)$ 、
 $icoffc = \text{mod}(jcc-1, nb\_c)$ 、
 $icrow = \text{indxg2p}(icc, mb\_c, MYROW, rsrc\_c, NPROW)$ 、
 $iccol = \text{indxg2p}(jcc, nb\_c, MYCOL, csrc\_c, NPCOL)$ 、
 $mpc0 = \text{numroc}(mi+iroffc, mb\_c, MYROW, icrow, NPROW)$ 、
 $nqc0 = \text{numroc}(ni+icoffc, nb\_c, MYCOL, iccol, NPCOL)$ 

```

$ilcm$ 、 indxg2p および numroc は、ScaLAPACK ツール関数である。
 $MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および $NPCOL$ は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。 $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<code>c</code>	積 $Q \text{ sub}(C)$ 、 $Q' \text{ sub}(C)$ 、 $\text{sub}(C) Q'$ 、または $\text{sub}(C) Q$ のいずれかによって上書きされる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info < 0</code> の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -i$ 。

p?stebz

二分法を用いて、対称三重対角行列の固有値を計算する。

構文

```

call psstebz( ictxt, range, order, n, vl, vu, il, iu, abstol, d, e, m,
              nsplit, w, iblock, isplit, work, iwork, liwork, info)
call pdstebz( ictxt, range, order, n, vl, vu, il, iu, abstol, d, e, m,
              nsplit, w, iblock, isplit, work, iwork, liwork, info)

```

説明

このルーチンは、二分法を用いて、対称三重対角行列の固有値を計算する。計算される固定値は、すべての固定値、区間 $[v1\ vu]$ 内のすべての固定値、またはインデックス $i1 \sim iu$ の固定値である。work の静的分割は p?stebz のはじめに完了する。これにより、各プロセスは、ほぼ同数の固有値を見つける。

入力パラメーター

<i>ictxt</i>	(グローバル) INTEGER。 BLACS コンテキスト・ハンドル。
<i>range</i>	(グローバル) CHARACTER。'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> = 'A' の場合、固有値をすべて計算する。 <i>range</i> = 'V' の場合、次の区間内の固有値を計算する。 $[v1\ vu]$ <i>range</i> = 'I' の場合、インデックス $i1$ から iu の固有値を計算する。
<i>order</i>	(グローバル) CHARACTER。'B' または 'E' でなければならない。 <i>order</i> = 'B' の場合、一連の固有値は、分解された各ブロック内で最小のものから昇順に配置される。 <i>order</i> = 'E' の場合、固有値が行列全体で最小のものから昇順に配置される。
<i>n</i>	(グローバル) INTEGER。三重対角行列 T の次数 ($n \geq 0$)。
<i>v1, vu</i>	(グローバル) REAL (psstebz の場合) DOUBLE PRECISION (pdstebz の場合) <i>range</i> = 'V' の場合、区間 $[v1\ vu]$ の固有値の下限値と上限値を計算する。 <i>range</i> = 'A' または 'I' の場合、 $v1$ と vu は参照されない。
<i>i1, iu</i>	(グローバル) INTEGER。次の制約がある。 $1 \leq i1 \leq iu \leq n$ 。 <i>range</i> = 'I' の場合、(固有値が昇順であると仮定して) 最も小さな固有値のインデックスは $i1$ に、最も大きな固有値のインデックスは iu に返されなければならない。 $i1$ は 1 以上でなければならない。 iu は $i1$ 以上で n 以下でなければならない。 <i>range</i> = 'A' または 'V' の場合、 $i1$ と iu は参照されない。
<i>abstol</i>	(グローバル) REAL (psstebz の場合) DOUBLE PRECISION (pdstebz の場合) 各固有値に対する絶対許容値は、必ず指定しなければならない。固有値 (または集積値) は、幅 <i>abstol</i> の区間内に存在している場合に、収束しているものとみなされる。 $abstol \leq 0$ の場合、許容値は $ulp \ T\ $ (ulp はマシン精度で $\ T\ $ は T の 1- ノルム) になる。 固有値が最も正確に計算されるのは、 <i>abstol</i> がゼロではなく、アンダーフローのしきい値 $slamch('U')$ に設定されたときである。 固有ベクトルを後から逆反復法 (p?stein) で使用する場合、 <i>abstol</i> を $2 * p?lamch('S')$ に設定する必要がある。

- d* (グローバル)
 REAL (psstebz の場合)
 DOUBLE PRECISION (pdstebz の場合)
 配列、次元は (n) 。
 三重対角行列 T の n 個の対角成分を格納する。アンダーフローを防ぐために、行列を、その最大エントリーが $\text{overflow}^{(1/2)} * \text{underflow}^{(1/4)}$ よりも絶対値において超えないようにスケールしなければならない。最も精度を上げるには、行列を上記条件よりもはるかに小さくスケールしてはならない。
- e* (グローバル)
 REAL (psstebz の場合)
 DOUBLE PRECISION (pdstebz の場合)
 配列、次元は $(n-1)$ 。
 三重対角行列 T の $n-1$ 個の非対角成分を格納する。アンダーフローを防ぐために、行列を、その最大エントリーが $\text{overflow}^{(1/2)} * \text{underflow}^{(1/4)}$ よりも絶対値において超えないようにスケールしなければならない。最も精度を上げるには、行列を上記条件よりもはるかに小さくスケールしてはならない。
- work* (ローカル)
 REAL (psstebz の場合)
 DOUBLE PRECISION (pdstebz の場合)
 配列、次元は $\max(5n, 7)$ 。ワークスペース配列。
- lwork* (ローカル) INTEGER。
work 配列のサイズは $\max(5n, 7)$ 以上でなければならない。
lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。
- iwork* (ローカル) INTEGER。
 配列、次元は $\max(4n, 14)$ 。ワークスペース配列。
- liwork* (ローカル) INTEGER。
liwork 配列のサイズは $\max(4n, 14, \text{NPROCS})$ 以上でなければならない。
liwork = -1 の場合、*liwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- m* (グローバル) INTEGER。見つかった固有値の実際の個数。 $0 \leq m \leq n$
- nsplit* (グローバル) INTEGER。 T で見つかった対角ブロックの個数 ($1 \leq \text{nsplit} \leq n$)。

<i>w</i>	(グローバル) REAL (<i>psstebz</i> の場合) DOUBLE PRECISION (<i>pdstebz</i> の場合) 配列、次元は (<i>n</i>)。 終了時に、 <i>w</i> の最初の <i>m</i> 個の成分に、すべてのプロセスの固有値が格納される。
<i>iblock</i>	(グローバル) INTEGER。 配列、次元は (<i>n</i>)。 <i>e(j)</i> がゼロまたは小さい行 / 列 <i>j</i> では、行列 <i>T</i> はブロック対角行列に分割するとみなされる。終了時に、 <i>iblock(i)</i> は (1 からブロック数まで) どのブロックに固有値 <i>w(i)</i> が属するかを示す。



注: 二分法で一部またはすべての固有値が収束しない (理論上不可能な) イベントでは、*info* は 1 に設定される。収束しなかったイベントは、負のブロック番号で識別できる。

<i>isplit</i>	(グローバル) INTEGER。 配列、次元は (<i>n</i>)。 <i>T</i> を部分行列に分割した分割点が格納される。最初の部分行列は 1 から <i>isplit(1)</i> の行 / 列で構成され、2 番目の部分行列は <i>isplit(1)+1</i> ~ <i>isplit(2)</i> の行 / 列で構成され、以下同様である。 <i>nsplit</i> 番目は <i>isplit(nsplite-1)+1</i> ~ <i>isplit(nsplite)=n</i> の行 / 列で構成される (最初の <i>nsplite</i> 個の成分のみが使用される。しかし、 <i>nsplite</i> の値はわからないため、 <i>n</i> ワードを <i>isplit</i> 用に予約しておかなければならない)。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、一部またはすべての固有値が収束に失敗したか計算できなかったことを示す。 <i>info</i> = 1 の場合、二分法で一部の固有値が収束に失敗したことを示す。これらの固有値には、負のブロック番号が付けられる。固有値は、絶対的および相対的な許容値ほど正確ではない。 <i>info</i> = 2 の場合、出力された固有値の番号と要求した番号が一致していないことを示す。 <i>info</i> = 3 の場合、 <i>range='i'</i> で最初に使用された Gershgorin 区間が正しくないことを示す。固有値は計算されない。マシンの浮動小数点演算に問題がある可能性がある。 <i>fudge</i> パラメーターの値を大きくして再コンパイルし、再度実行する。

p?stein

逆反復法を用いて、三重対角行列の固有ベクトルを計算する。

構文

```
call psstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pdstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pcstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pzstein( n, d, e, m, w, iblock, isplit, orfac, z, iz, jz, descz,
             work, lwork, iwork, liwork, ifail, iclustr, gap, info)
```

説明

このルーチンは、逆反復法を用いて、対称三重対角行列 T の指定された固有値に対応する固有ベクトルを計算する。p?stein は、異なるプロセス上にあるベクトルを直交しない。直交化の範囲は、入力パラメーター $lwork$ によって制御される。直交する固有ベクトルは、同じプロセスによって計算される。p?stein は、プロセス間の $work$ の割り当てを決定してから、個々のプロセスで LAPACK ルーチンの修正版である [sstein2](#) (psstein および pcstein) または [sdtein2](#) (pdstein および pzstein) を呼び出す。ワークスペースの割り当てが十分でない場合、要求した直交化は行われない。



注: 得られた固有ベクトルが直交でない場合、 $lwork$ の値を増やしてコードを再度実行する。

$p = NPROW * NPCOL$ はプロセスの総数。

入力パラメーター

n (グローバル) INTEGER。行列 T の次数 ($n \geq 0$)。

m (グローバル) INTEGER。返すべき固有ベクトルの個数。

d, e, w (グローバル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 配列:
 $d(*)$ には、 T の対角成分を格納する。
 次元は (n) 。

$e(*)$ には、 T の非対角成分が格納される。
 次元は $(n-1)$ 。

$w(*)$ には、分割ブロックによってグループ化されたすべての固有値を格納する。固有値は、ブロック内で最も小さなものから最も大きなものまで提供される (`order = 'B'` の場合の `p?stebz` の出力配列 w がここで要求される)。配列は、すべてのプロセスで複製する必要がある。次元は (m)。

<code>iblock</code>	(グローバル) INTEGER。 配列、次元は (n)。 w の対応する固有値に関連する部分行列インデックス。一番上から最初の部分行列に属する固有値は 1、2 番目の部分行列に属する固有値は 2、以下同様 (<code>p?stebz</code> の出力配列 <code>iblock</code> がここで要求される)。
<code>isplit</code>	(グローバル) INTEGER。 配列、次元は (n)。 T を部分行列に分割した分割点。最初の部分行列は 1 から <code>isplit(1)</code> の行 / 列で構成され、2 番目の部分行列は <code>isplit(1)+1 ~ isplit(2)</code> の行 / 列で構成され、以下同様である。 <code>nsplit</code> 番目は <code>isplit(1)+1 ~ isplit(1)+nsplit</code> の行 / 列で構成される (<code>p?stebz</code> の出力配列 <code>isplit</code> がここで要求される)。
<code>orfac</code>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <code>orfac</code> は、直交化される固有ベクトルを指定する。互いの <code>orfac * T </code> 内にある固有値に対応する固有ベクトルは直交化される。しかし、ワークスペースが十分でない場合 (<code>lwork</code> を参照)、この許容値は、すべての固有ベクトルが 1 つのプロセスで格納できるようになるまで下げられる。 <code>orfac</code> がゼロの場合、直交化は行われない。 <code>orfac</code> が負の場合、デフォルト値の 10^3 が使用される。 <code>orfac</code> はすべての処理で同一でなければならない。
<code>iz, jz</code>	(グローバル) INTEGER。それぞれ、部分行列 Z の最初の行と最初の列を示す、グローバル配列 z の行インデックスと列インデックス。
<code>descz</code>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 Z の配列ディスクリプター。
<code>work</code>	(ローカル)。 REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) ワークスペース配列、次元は ($lwork$)。
<code>lwork</code>	(ローカル) INTEGER。 $lwork$ は、直交化の範囲を制御する。各プロセスで割り当てられる格納形式に対する固有ベクトルの数は次のとおりである。 $nvec = \text{floor}((lwork - \max(5*n, np00*mq00))/n)$ サイズ $nvec - \text{ceil}(m/p) + 1$ の固有値クラスターに対応する固有ベクトルは、直交することが保証される (直交性は、 <code>?stein2</code> で得られたものに似ている)。



注: $lwork$ は、 $\max(5*n, np00*mq00) + \text{ceil}(m/p)*n$ 以上でなければならない。また、すべてのプロセスで入力値が同じでなければならない。

重要なのは、異なるプロセスにおける *lwork* 入力の最小値である。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

<i>iwork</i>	(ローカル) INTEGER。 ワークスペース配列、次元は $(3n+p+1)$ 。
<i>liwork</i>	(ローカル) INTEGER。配列 <i>iwork</i> のサイズ。 $3*n+p+1$ 以上でなければならない。 <i>liwork</i> = -1 の場合、 <i>liwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>z</i>	(ローカル) REAL (<i>psstein</i> の場合) DOUBLE PRECISION (<i>pdstein</i> の場合) COMPLEX (<i>pcstein</i> の場合) DOUBLE COMPLEX (<i>pzstein</i> の場合) 配列、次元は $(descz(dlen_), n/NPCOL + NB)$ 。 <i>z</i> には、指定された固有値と関連する計算された固有ベクトルが格納される。収束に失敗したすべてのベクトルは、MAXIT 回反復した後、現在の反復に設定される (?stein2 を参照)。出力時に、 <i>z</i> は、ブロック・サイクリック形式で <i>p</i> 個のプロセスに分散される。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、ユーザーが要求する直交化を保証するワークスペース (<i>lwork</i>) の下限が格納される (<i>orfac</i> を参照)。必要な最小ワークスペースを過大評価する点がある点に注意する。
<i>iwork</i>	終了時に、 <i>iwork(1)</i> には、要求されたワークスペースの量が整数で格納される。 終了時に、 <i>iwork(2) ~ iwork(p+2)</i> には、各プロセスによって計算された固有ベクトルが格納される。プロセス <i>i</i> は、インデックス <i>iwork(i+2)+1 ~ iwork(i+3)</i> の固有ベクトルを計算する。
<i>ifail</i>	(グローバル) INTEGER。次元は (<i>m</i>)。 通常終了時は、 <i>ifail</i> のすべての成分はゼロになる。MAXIT 回反復した後、1 つ以上の固有ベクトルが収束に失敗した場合 (?stein を参照)、 <i>info</i> > 0 が返される。mod(<i>info,m+1</i>)>0 で、 <i>i</i> = 1 から mod(<i>info,m+1</i>) の場合、固有値 <i>w</i> (<i>ifail(i)</i>) に対応する固有ベクトルは収束に失敗した (<i>w</i> は出力時に固有値の配列を参照する)。
<i>iclustr</i>	(グローバル) INTEGER。配列、次元は $(2*p)$ 。 この出力配列には、ワークスペースが十分でなかったために直交できなかった、固有値のクラスターに対する固有ベクトルのインデックスが格納される (<i>lwork</i> , <i>orfac</i> , および <i>info</i> を参照)。インデックス <i>iclustr(2*I-1)</i> から <i>iclustr(2*I)</i> 、 <i>i</i> = 1 から <i>info/(m+1)</i> の固有値のクラスターに対応する固有ベクトルは、ワークスペースの不足により直

交できなかった。したがって、これらのクラスターに対応する固有ベクトルは直交しない。*iclustr* は、ゼロで終了する配列である。 $(iclustr(2*k).ne.0.and.iclustr(2*k+1).eq.0)$ (*k* がクラスター数の場合)。

gap

(グローバル)

REAL (単精度の場合)

DOUBLE PRECISION (倍精度の場合)

この出力配列には、固有ベクトルが直交できなかった固有値間のギャップが格納される。この配列の *info/m* 出力値は、配列 *iclustr* によって示された *info/(m+1)* クラスターに対応する。その結果、*i* 番目のクラスターに対応する固有ベクトルのドット積は、 $(O(n)*macheps)/gap(i)$ と同程度になる。

info

(グローバル) INTEGER。

info = 0 の場合、正常に終了したことを示す。

info < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = $-(i*100+j)$ 、

i 番目の引数がスカラーで値が不正ならば *info* = $-i$ 。

info < 0 の場合: *info* = $-i$ の場合、*i* 番目の引数の値が不正だったことを示す。

info > 0 の場合: $\text{mod}(\text{info}, m+1) = i$ の場合、*i* 固有ベクトルは MAXIT 回の反復で収束に失敗した。インデックスは、配列 *ifail* に格納される。

info/(m+1) = i の場合、固有値の *i* クラスターに対応する固有ベクトルが、ワークスペースが十分でなかったために直交できなかった。クラスターのインデックスは、配列 *iclustr* に格納される。

非対称固有値問題

このセクションでは、非対称固有値問題の解決、一般行列の Schur 因子分解の計算、およびそれらに関連する各種の計算タスクを実行するための ScaLAPACK ルーチンについて説明する。

ScaLAPACK を使って非対称固有値問題を解くには、通常、行列を上 Hessenberg 形式に縮退させた後、得られた Hessenberg 行列を使って固有値問題を解く必要がある。

表 6-5 に示されている ScaLAPACK ルーチンを利用すると、直交 (またはユニタリー) の相似変換 $A = QHQ^H$ を使って、該当行列を上 Hessenberg 形式に縮退させることができる。また、Hessenberg 行列を使って固有値問題を解き、縮退後に行列を乗算することもできる。

表 6-5 非対称固有値問題を解くための計算ルーチン

機能	一般行列	直交 / ユニタリー行列	Hessenberg 行列
Hessenberg 形式に縮退させる $A = QHQ^H$	p?gehrd		
縮退後に行列を掛ける		p?ormhr / p?unmhr	
固有値と Schur 因子分解を求める			p?lahqr

p?gehrd

一般行列を上 Hessenberg 形式に縮退させる。

構文

```
call psgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,
             info )
call pdgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,
             info )
call pcgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,
             info )
call pzgehrd( n, ilo, ihi, a, ia, ja, desca, tau, work, lwork,
             info )
```

説明

このルーチンは、直交またはユニタリーの相似変換を用いて、実 / 複素一般分散行列 $\text{sub}(A)$ を上 Hessenberg 形式 H に縮退させる。

$$Q' \text{sub}(A) Q = H$$

ここで、 $\text{sub}(A) = A(\text{ia}+n-1:\text{ia}+n-1, \text{ja}+n-1:\text{ja}+n-1)$ 。

入力パラメーター

n (グローバル) INTEGER。分散行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。

<i>ilo, ihi</i>	(グローバル) INTEGER。sub(A) はすでに、行 $ia:ia+ilo-2$ と $ia+ihi:ia+n-1$ および列 $ja:ja+ilo-2$ と $ja+ihi:ja+n-1$ で上三角になっていると仮定する。(次の「アプリケーション・ノート」を参照)。 $n > 0$ の場合、 $1 \leq ilo \leq ihi \leq n$ 。それ以外の場合、 $ilo = 1$ 、 $ihi = n$ を設定する。
<i>a</i>	(ローカル) REAL (psgehrd の場合) DOUBLE PRECISION (pdgehrd の場合) COMPLEX (pcgehrd の場合) DOUBLE COMPLEX (pzgehrd の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列へのポインター。呼び出し時に、この配列は、因子分解する $n \times n$ の一般分散行列 sub(A) のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psgehrd の場合) DOUBLE PRECISION (pdgehrd の場合) COMPLEX (pcgehrd の場合) DOUBLE COMPLEX (pzgehrd の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。work の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq NB * NB + NB * \max(ihip+1, ihlp+inlq)$ でなければならない。ここで、 $NB = mb_a = nb_a$ 、 $irow = \text{mod}(ia-1, NB)$ 、 $icoff = \text{mod}(ja-1, NB)$ 、 $irow = \text{mod}(ia+ilo-2, NB)$ 、 $iarow = \text{indxg2p}(ia, NB, MYROW, rsrc_a, NPROW)$ 、 $ihip = \text{numroc}(ihi+irow, NB, MYROW, iarow, NPROW)$ 、 $ilrow = \text{indxg2p}(ia+ilo-1, NB, MYROW, rsrc_a, NPROW)$ 、 $ihlp = \text{numroc}(ihi-ilo+irow+1, NB, MYROW, ilrow, NPROW)$ 、 $ilcol = \text{indxg2p}(ja+ilo-1, NB, MYCOL, csrc_a, NPCOL)$ 、 $inlq = \text{numroc}(n-ilo+irow+1, NB, MYCOL, ilcol, NPCOL)$ <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。 <i>lwork = -1</i> の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、 <i>sub(A)</i> の上三角と第 1 の劣対角は、上 Hessenberg 行列 <i>H</i> および第 1 の劣対角よりも下の成分によって上書きされ、配列 <i>tau</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表す。(次の「アプリケーション・ノート」を参照)。
<i>tau</i>	(ローカル)。 REAL (<i>psgehrd</i> の場合) DOUBLE PRECISION (<i>pdgehrd</i> の場合) COMPLEX (<i>pcgehrd</i> の場合) DOUBLE COMPLEX (<i>pzgehrd</i> の場合) 配列、次元は $\max(ja+n-2)$ 以上。 基本リフレクターのスカラー係数。(次の「アプリケーション・ノート」を参照)。 <i>tau</i> の成分 <i>ja:ja+ilo-2</i> と <i>ja+ihi:ja+n-2</i> はゼロに設定される。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

アプリケーション・ノート

行列 *Q* は、(*ihi-ilo*) 個の基本リフレクターの積として表現される。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの *H(i)* は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、*tau* は実/複素スカラー、*v* は実/複素ベクトルで $v(1:i) = 0$ 、 $v(i+1) = 1$ 、および $v(ihi+1:n) = 0$ 。終了時に、 $v(i+2:ihi)$ は $a(ia+ilo+i:ia+ihi-1, ja+ilo+i-2)$ に、*tau* は $\tau(ja+ilo+i-2)$ に格納される。 $a(ia:ia+n-1, ja:ja+n-1)$ の内容を次に示す ($n = 7$ 、 $ilo = 2$ および $ihi = 6$ の場合)。

呼び出し時

$$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$$

終了時

$$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & & h & h & h & h & h \\ v2 & h & h & h & h & h & \\ v2 & v3 & h & h & h & h & \\ v2 & v3 & v4 & h & h & h & \\ & & & & & & a \end{bmatrix}$$

a は元の行列 $\text{sub}(A)$ の成分、 H は上 Hessenberg 行列 H の変更された成分、 v_i は $H(ja+ilo+i-2)$ を定義するベクトルの成分を表す。

p?ormhr

一般行列に p?gehrd で求めた Hessenberg 形式に縮退した直交変換行列を掛ける。

構文

```
call psormhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info )
call pdormhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般実分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$$\text{side} = 'L' \qquad \text{side} = 'R'$$

$$\text{trans} = 'N': \quad Q \text{sub}(C) \qquad \text{sub}(C) Q$$

$$\text{trans} = 'T': \quad Q^T \text{sub}(C) \qquad \text{sub}(C) Q^T$$

ここで、 Q は、次数 nq の実直交分散行列で、 $nq = m$ ($\text{side} = 'L'$ の場合)、 $nq = n$ ($\text{side} = 'R'$ の場合) である。 Q は、p?gehrd によって返される、 $ihi-ilo$ 個の基本リフレクターの積として定義される。

$$Q = H(ilo)H(ilo+1)\dots H(ihi-1)$$

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^T は左側から適用される。
 = 'R' の場合、 Q または Q^T は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'T' の場合、 Q^T が適用される (転置)。

<i>m</i>	(グローバル) INTEGER。分散行列 <i>sub</i> (<i>C</i>) の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散行列 <i>sub</i> (<i>C</i>) の列数 ($n \geq 0$)。
<i>ilo, ihi</i>	(グローバル) INTEGER。 <i>ilo</i> と <i>ihi</i> は <i>p?gehrd</i> の以前の呼び出しで指定された値と同じ値でなければならない。 <i>Q</i> は、分散部分行列 $Q(ia+ilo:ia+ihi-1, ia+ilo:ja+ihi-1)$ を除いてユニタリ行列と等しい。 <i>side</i> = 'L' の場合、 $1 \leq ilo \leq ihi \leq \max(1, m)$ 。 <i>side</i> = 'R' の場合、 $1 \leq ilo \leq ihi \leq \max(1, n)$ 。 <i>ilo</i> と <i>ihi</i> は相対インデックス。
<i>a</i>	(ローカル) REAL (<i>psormhr</i> の場合) DOUBLE PRECISION (<i>pdormhr</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja+m-1</i>)) (<i>side</i> ='L' の場合) または (<i>lld_a</i> , <i>LOCc</i> (<i>ja+n-1</i>)) (<i>side</i> ='R' の場合) の配列へのポインター。 <i>p?gehrd</i> によって返される、基本リフレクターを定義する一連のベクトルを含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (<i>psormhr</i> の場合) DOUBLE PRECISION (<i>pdormhr</i> の場合) 配列、次元は <i>LOCc</i> (<i>ja+m-2</i>) (<i>side</i> ='L' の場合) <i>LOCc</i> (<i>ja+n-2</i>) (<i>side</i> ='R' の場合) <i>p?gehrd</i> によって返される、基本リフレクター <i>H</i> (<i>j</i>) のスカラー係数 <i>tau</i> (<i>j</i>) を含む。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL (<i>psormhr</i> の場合) DOUBLE PRECISION (<i>pdormhr</i> の場合) ローカルメモリーにある、次元 (<i>lld_c</i> , <i>LOCc</i> (<i>jc+n-1</i>)) の配列へのポインター。分散行列 <i>sub</i> (<i>C</i>) のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psormhr</i> の場合) DOUBLE PRECISION (<i>pdormhr</i> の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> は

$iaa = ia + ilo$; $jaa = ja + ilo - 1$ 以上でなければならない。
 $side = 'L'$ の場合、
 $mi = ihi - ilo$, $ni = n$, $icc = ic + ilo$, $jcc = jc$,
 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + mpc0) * nb_a) + nb_a * nb_a$
 $side = 'R'$ の場合、
 $mi = m$, $ni = ihi - ilo$, $icc = ic$, $jcc = jc + ilo$,
 $lwork \geq \max((nb_a * (nb_a - 1)) / 2,$
 $(nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni + icoffc, nb_a, 0, 0, NPCOL),$
 $nb_a, 0, 0, lcmq), mpc0)) * nb_a) + nb_a * nb_a$
 でなければならない。

ここで、 $lcmq = lcm / NPCOL$ で $lcm = ilcm(NPROW, NPCOL)$,

$iroffa = \text{mod}(iaa - 1, mb_a)$,
 $icoffa = \text{mod}(jaa - 1, nb_a)$,
 $iarow = \text{indxg2p}(iaa, mb_a, MYROW, rsrc_a, NPROW)$,
 $npa0 = \text{numroc}(ni + iroffa, mb_a, MYROW, iarow, NPROW)$,
 $iroffc = \text{mod}(icc - 1, mb_c)$,
 $icoffc = \text{mod}(jcc - 1, nb_c)$,
 $icrow = \text{indxg2p}(icc, mb_c, MYROW, rsrc_c, NPROW)$,
 $iccol = \text{indxg2p}(jcc, nb_c, MYCOL, csrc_c, NPCOL)$,
 $mpc0 = \text{numroc}(mi + iroffc, mb_c, MYROW, icrow, NPROW)$,
 $nqc0 = \text{numroc}(ni + icoffc, nb_c, MYCOL, iccol, NPCOL)$

$ilcm$ 、 indxg2p および numroc は、ScaLAPACK ツール関数である。
 $MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および $NPCOL$ は、サブルーチン
 blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペース
 のクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最
 適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエン
 トリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- c $\text{sub}(C)$ は、 $Q \text{sub}(C)$ 、 $Q' \text{sub}(C)$ 、 $\text{sub}(C)Q'$ 、または $\text{sub}(C)Q$ のいずれか
 によって上書きされる。
- $work(1)$ 終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な
 $lwork$ の最小値が格納される。
- $info$ (グローバル) INTEGER。
 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、
 $info = -(i * 100 + j)$ 。 i 番目の引数がスカラーで値が不正だった場合、
 $info = -i$ 。

p?unmhr

一般行列に p?gehrd で求めた *Hessenberg* 形式に縮退したユニタリー変換行列を掛ける。

構文

```
call pcunmhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info )
call pzunmhr( side, trans, m, n, ilo, ihi, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info )
```

説明

このルーチンは、 $m \times n$ の一般複素分散行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': Q \text{sub}(C)$	$\text{sub}(C) Q$
$trans = 'C': Q^H \text{sub}(C)$	$\text{sub}(C) Q^H$

ここで、 Q は、次数 nq の複素ユニタリー分散行列で、 $nq = m$ ($side = 'L'$ の場合)、 $nq = n$ ($side = 'R'$ の場合) である。 Q は、[p?gehrd](#) によって返される、 $ihi-ilo$ 個の基本リフレクターの積として定義される。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

入力パラメーター

side (グローバル) CHARACTER
 = 'L' の場合、 Q または Q^H は左側から適用される。
 = 'R' の場合、 Q または Q^H は右側から適用される。

trans (グローバル) CHARACTER
 = 'N' の場合、 Q が適用される (転置なし)。
 = 'C' の場合、 Q^H が適用される (共役転置)。

m (グローバル) INTEGER。分散部分行列 $\text{sub}(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散部分行列 $\text{sub}(C)$ の列数 ($n \geq 0$)。

ilo, ihi (グローバル) INTEGER。これらの値は、それぞれ p?gehrd に指定した ilo および ihi と同じでなければならない。 Q は、分散部分行列 $Q(ia+ilo:ia+ihi-1, ia+ilo:ja+ihi-1)$ を除いてユニタリー行列と等しい。
 $side = 'L'$ の場合、 $1 \leq ilo \leq ihi \leq \max(1, m)$ 。
 $side = 'R'$ の場合、 $1 \leq ilo \leq ihi \leq \max(1, n)$ 。
 ilo と ihi は相対インデックス。

a (ローカル)
 COMPLEX (pcunmhr の場合)
 DOUBLE COMPLEX (pzunmhr の場合)
 ローカルメモリーにある、次元 ($lld_a, LOCC(ja+m-1)$) ($side = 'L'$ の場

合) または ($lld_a, LOCc(ja+n-1)$) ($side = 'R'$ の場合) の配列へのポインター。
 $p?gehrd$ によって返される、基本リフレクターを定義する一連のベクトルを含む。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

tau (ローカル)
 COMPLEX ($pcunmhr$ の場合)
 DOUBLE COMPLEX ($pzunmhr$ の場合)
 配列、次元は $LOCc(ja+m-2)$ ($side = 'L'$ の場合)
 $LOCc(ja+n-2)$ ($side = 'R'$ の場合)
 $p?gehrd$ によって返される、基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を含む。 tau は、分散行列 A に関連付けられる。

c (ローカル)
 COMPLEX ($pcunmhr$ の場合)
 DOUBLE COMPLEX ($pzunmhr$ の場合)
 ローカルメモリーにある、次元 ($lld_c, LOCc(jc+n-1)$) の配列へのポインター。分散行列 $sub(C)$ のローカル部分を含む。

ic, jc (グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。

descc (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。

work (ローカル)
 COMPLEX ($pcunmhr$ の場合)
 DOUBLE COMPLEX ($pzunmhr$ の場合)
 次元 $lwork$ のワークスペース配列。

lwork (ローカルまたはグローバル)
 配列 $work$ の次元。 $lwork$ は $iaa = ia + ilo; jaa = ja + ilo - 1$ 以上でなければならない。
 $side = 'L'$ の場合、
 $mi = ihi - ilo, ni = n, icc = ic + ilo, jcc = jc,$
 $lwork \geq \max((nb_a * (nb_a - 1)) / 2, (nqc0 + mpc0) * nb_a) + nb_a * nb_a$
 $side = 'R'$ の場合、
 $mi = m, ni = ihi - ilo, icc = ic, jcc = jc + ilo,$
 $lwork \geq \max((nb_a * (nb_a - 1)) / 2,$
 $(nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni + icoffc, nb_a, 0, 0, NPCOL),$
 $nb_a, 0, 0, lcmq), mpc0)) * nb_a) + nb_a * nb_a$
 でなければならない。
 ここで、 $lcmq = lcm / NPCOL$ で $lcm = ilcm(NPROW, NPCOL)$,
 $iroffa = \text{mod}(iaa - 1, mb_a),$
 $icoffa = \text{mod}(jaa - 1, nb_a),$
 $iarow = \text{indxg2p}(iaa, mb_a, MYROW, rsrc_a, NPROW),$
 $npa0 = \text{numroc}(ni + iroffa, mb_a, MYROW, iarow, NPROW),$

```

irowfc = mod(icc-1, mb_c)、
icoffc = mod(jcc-1, nb_c)、
icrow = indxg2p (icc, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p (jcc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(mi+irowfc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

ilcm、indxg2p および numroc は、ScaLAPACK ツール関数である。
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン
blacs_gridinfo を呼び出して決定できる。

lwork = -1 の場合、lwork はグローバル入力であり、ワークスペース
のクエリーとみなされ、ルーチンはすべての work 配列の最小かつ最
適なサイズだけを計算する。各値は該当する work 配列の最初のエン
トリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	C は $Q^* \text{sub}(C)$ 、 $Q^* \text{sub}(C)$ 、 $\text{sub}(C)Q'$ 、または $\text{sub}(C)Q$ のいずれかによ って上書きされる。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。

p?lahqr

すでに Hessenberg 形式の行列の Schur 分解および
固有値を計算する。

構文

```

call pslahqr(wantt, wantz, n, ilo, ihi, a, desca, wr, wi, iloz, ihiz, z,
             descz, work, lwork, iwork, ilwork, info)
call pdlahqr(wantt, wantz, n, ilo, ihi, a, desca, wr, wi, iloz, ihiz, z,
             descz, work, lwork, iwork, ilwork, info)

```

説明

このルーチンは、既に Hessenberg 形式の行列の列 *ilo* から *ihi* までの Schur 分解および
固有値を見つけるために使用される補助ルーチンである。

入力パラメーター

<i>wantt</i>	(グローバル) LOGICAL。 <i>wantt</i> = .TRUE. の場合、すべての Schur 形式 T が必要である。 <i>wantt</i> = .FALSE. の場合、固有値のみが必要である。
--------------	--

<i>wantz</i>	(グローバル) LOGICAL。 <i>wantz</i> = .TRUE. の場合、Schur ベクトル <i>z</i> が必要である。 <i>wantz</i> = .FALSE. の場合、Schur ベクトルは必要ない。
<i>n</i>	(グローバル) INTEGER。Hessenberg 行列 <i>A</i> の次数 (<i>wantz</i> の場合は <i>z</i>) ($n \geq 0$)。
<i>ilo, ihi</i>	(グローバル) INTEGER。 <i>A</i> は、行と列が <i>ihi+1:n</i> の上準三角になっているとする。また、 $A(i_{lo}, i_{lo}-1) = 0$ ($i_{lo} = 1$ でない限り) であるとする。?lahqr は基本的に、行と列が <i>ilo</i> から <i>ihi</i> の Hessenberg 部分行列を対象とするが、 <i>wantt</i> が .TRUE.、 $1 \leq i_{lo} \leq \max(1, i_{hi}); i_{hi} \leq n$ の場合は <i>h</i> のすべてに変換が適用される。
<i>a</i>	(グローバル) REAL (pslahqr の場合) DOUBLE PRECISION (pdlahqr の場合) 配列、次元は (<i>desca</i> (<i>lld</i> _*),*)。呼び出し時は、上 Hessenberg <i>A</i> 行列。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _)。分散行列 <i>A</i> の配列ディスクリプター。
<i>iloz, ihiz</i>	(グローバル) INTEGER。 <i>wantz</i> が .TRUE. の場合、変換が適用される <i>z</i> の行を指定する。 $1 \leq i_{loz} \leq i_{lo}$ 、 $i_{hi} \leq i_{hiz} \leq n$ 。
<i>z</i>	(グローバル) REAL (pslahqr の場合) DOUBLE PRECISION (pdlahqr の場合) 配列。 <i>wantz</i> が .TRUE. の場合、呼び出し時に、 <i>z</i> は pdhseqr で蓄積された変換の現在の行列 <i>z</i> を含んでいなければならない。 <i>wantz</i> が .FALSE. の場合、 <i>z</i> は参照されない。
<i>descz</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _)。分散行列 <i>Z</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (pslahqr の場合) DOUBLE PRECISION (pdlahqr の場合) 次元 <i>lwork</i> のワークスペース配列。
<i>lwork</i>	(ローカル) INTEGER。 <i>work</i> の次元。 <i>lwork</i> は十分大きいと仮定される。すなわち、 $lwork \geq 3*n + \max(2*\max(descz(lld_), desca(lld_)) + 2*LOCq(n), 7*\text{ceil}(n/hbl)/lcm(NPROW, NPCOL))$ 。 <i>lwork</i> = -1 の場合、 <i>work</i> (1) は上記の数に設定され、コードは直ちに戻る。
<i>iwork</i>	(グローバルおよびローカル) INTEGER 配列、サイズは <i>ilwork</i> 。
<i>ilwork</i>	(ローカル) INTEGER。 <i>iblk</i> 整数配列の一部を保持する。

出力パラメーター

<i>a</i>	終了時に、 <i>wantt</i> が .TRUE. の場合、 <i>A</i> は行と列が <i>ilo:ihi</i> にある上準三角で、標準形式ではない 2×2 以上の対角ブロックを持つ。 <i>wantt</i> が .FALSE. の場合、 <i>A</i> の内容は指定されない。
----------	--

<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>wr,wi</i>	(グローバル複製出力) REAL (<i>pslahqr</i> の場合) DOUBLE PRECISION (<i>pdlahqr</i> の場合) 配列、次元はそれぞれ (<i>n</i>)。 計算された固有値 <i>ilo</i> から <i>ihi</i> の実数部分と虚数部分は、それぞれ <i>wr</i> と <i>wi</i> の対応する成分に格納される。2 個の固有値が複素共役ペアとして計算された場合、それらは <i>wr</i> と <i>wi</i> の連続する成分に格納される。すなわち <i>i</i> 番目と (<i>i+1</i>) 番目で <i>wi(i) > 0</i> かつ <i>wi(i+1) < 0</i> である。 <i>wantt</i> が <i>.TRUE.</i> の場合、固有値は <i>A</i> に返される Schur 形式の対角と同じ順序で格納される。 <i>A</i> は、次のリリースまで、より大きな対角ブロックで返されることがある。
<i>z</i>	終了時に、 <i>z</i> は更新される。変換は部分行列 <i>z(iloz:ihi, ilo:ihi)</i> のみに適用される。
<i>info</i>	(グローバル) INTEGER。 <i>info = 0</i> の場合、実行は正常に終了したことを示す。 < 0 の場合、パラメーター数 <i>-info</i> が正しくないか一貫していないことを示す。 > 0 の場合、 <i>p?lahqr</i> は合計 $30 \cdot (ihi - ilo + 1)$ 回の反復で <i>ilo</i> から <i>ihi</i> のすべての固有値を計算できなかったことを示す。 <i>info = i</i> の場合、 <i>wr</i> および <i>wi</i> の成分 <i>i+1:ihi</i> に、正常に計算された固有値が含まれていることを示す。

特異値分解

このセクションでは、 $m \times n$ の一般行列 A の特異値分解 (SVD) を計算する ScaLAPACK ルーチンについて説明する (LAPACK の「[特異値分解](#)」を参照)。

一般行列 A の SVD を見つけるために、この行列は、最初にユニタリー (直交) 変換を用いて二重対角行列 B に縮退され、次に二重対角行列の SVD が計算される。 B の SVD は LAPACK ルーチン [?bdsqr](#) を使用して計算されることに注意する。

[表 6-6](#) に、この分解を行う ScaLAPACK 計算ルーチンを示す。

表 6-6 特異値分解 (SVD) 用のルーチン

機能	一般行列	直交 / ユニタリー行列
A を二重対角行列に縮退させる	p?gebrd	
縮退後に行列を掛ける		p?ormbr / p?unmbr

p?gebrd

一般行列を二重対角形式に縮退させる。

構文

```
call psgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pdgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pcgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pzgebrd(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
```

説明

このルーチンは、直交 / ユニタリー変換を用いて、 $m \times n$ の一般実 / 複素分布行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ を上または下二重対角形式 B に縮退させる。

$$Q^* * \text{sub}(A) * P = B$$

$m \geq n$ の場合、 B は上二重対角である。 $m < n$ の場合、 B は下二重対角である。

入力パラメーター

m (グローバル) INTEGER。分散行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。

a (ローカル)
 REAL (psgebrd の場合)
 DOUBLE PRECISION (pdgebrd の場合)
 COMPLEX (pcgebrd の場合)
 DOUBLE COMPLEX (pzgebrd の場合)

ローカルメモリーにある、次元 ($lld_a, LOC(ja+n-1)$) の配列へのポインター。呼び出し時に、この配列は、分散行列 $\text{sub}(A)$ を含む。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>psgebrd</i> の場合) DOUBLE PRECISION (<i>pdgebrd</i> の場合) COMPLEX (<i>pcgebrd</i> の場合) DOUBLE COMPLEX (<i>pzgebrd</i> の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 <i>work</i> の次元は $lwork \geq nb * (mpa0 + nqa0 + 1) + nqa0$ でなければならない。ここで、 $NB = mb_a = nb_a$ 、 $irow = \text{indxg2p}(ia, nb, MYROW, rsrc_a, NPROW)$ 、 $icol = \text{indxg2p}(ja, NB, MYCOL, csrc_a, NPCOL)$ 、 $mpa0 = \text{numroc}(m + irow, NB, MYROW, irow, NPROW)$ 、 $nqa0 = \text{numroc}(n + icol, NB, MYCOL, icol, NPCOL)$ indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン <i>blacs_gridinfo</i> を呼び出して決定できる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に $m \geq n$ の場合、 <i>sub</i> (<i>A</i>) の対角成分と最初の優対角成分は、上二重対角行列 <i>B</i> によって上書きされる。対角成分より下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。最初の優対角成分より上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交行列 <i>P</i> を表現する。 $m < n$ の場合、対角成分と最初の劣対角成分は、下二重対角行列 <i>B</i> によって上書きされる。最初の劣対角より下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。対角より上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交行列 <i>P</i> を表す。(次の「アプリケーション・ノート」を参照)。
<i>d</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $LOCc(ja + \min(m, n) - 1)$ ($m \geq n$ の場合) または $LOCr(ia + \min(m, n) - 1)$ (それ以外の場合)。二重対角行列 <i>B</i> の分散対角成分。 $d(i) = a(i, i)$ 。 <i>d</i> は分散行列 <i>A</i> に関連付けられる。

<i>e</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $LOCr(ia+\min(m,n)-1)$ ($m \geq n$ の場合) または $LOCc(ja+\min(m,n)-2)$ (それ以外の場合)。二重対角分散行列 B の非対角成分。 $m \geq n$ の場合、 $e(i) = a(i, i+1)$ ($i = 1, 2, \dots, n-1$) $m < n$ の場合 $e(i) = a(i+1, i)$ ($i = 1, 2, \dots, m-1$) e は、分散行列 A に関連付けられる。
<i>tauq, taup</i>	(ローカル) REAL (psgebrd の場合) DOUBLE PRECISION (pdgebrd の場合) COMPLEX (pcgebrd の場合) DOUBLE COMPLEX (pzgebrd の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$ (<i>tauq</i> の場合) または $LOCr(ia+\min(m,n)-1)$ (<i>taup</i> の場合)。 それぞれ、直交/ユニタリ行列 Q と P を表す基本リフレクターのスカラ係数が格納される。 <i>tauq</i> と <i>taup</i> は、分散行列 A に関連付けられる。 (次の「アプリケーション・ノート」を参照)。
<i>work(1)</i>	終了時に、 <i>work(1)</i> には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

アプリケーション・ノート

行列 Q と P は基本リフレクターの積として表現される。

$m \geq n$ の場合、

$$Q = H(1) H(2) \dots H(n) \text{ および } P = G(1) G(2) \dots G(n-1).$$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$$H(i) = i - \text{tauq} * v * v' \text{ および } G(i) = i - \text{taup} * u * u'$$

ここで、*tauq* と *taup* は実/複素スカラー、 v と u は実/複素ベクトルである。

$v(1:i-1) = 0$ 、 $v(i) = 1$ および $v(i+1:m)$ は、終了時に、 $A(ia+i:ia+m-1, ja+i-1)$ に格納され、

$u(1:i) = 0$ 、 $u(i+1) = 1$ および $u(i+2:n)$ は、終了時に、 $A(ia+i-1, ja+i+1:ja+n-1)$ に格納され、

tauq は *tauq(ja+i-1)* に格納され、*taup* は *taup(ia+i-1)* に格納される。

$m < n$ の場合、

$$Q = H(1) H(2) \dots H(m-1) \text{ および } P = G(1) G(2) \dots G(m)$$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$$H(i) = i - \text{tauq} * v * v' \text{ および } G(i) = i - \text{taup} * u * u'$$

ここで、 tauq と taup は実 / 複素スカラー、 v と u は実 / 複素ベクトルである。

$v(1:i) = 0$ 、 $v(i+1) = 1$ および $v(i+2:m)$ は、終了時に、 $A(ia+i:ia+m-1, ja+i-1)$ に格納され、 $u(1:i-1) = 0$ 、 $u(i) = 1$ および $u(i+1:n)$ は、終了時に、 $A(ia+i-1, ja+i+1:ja+n-1)$ に格納される。

tauq は $\text{tauq}(ja+i-1)$ に格納され、 taup は $\text{taup}(ia+i-1)$ に格納される。

ルーチン終了後の $\text{sub}(A)$ の内容は、次の例に示される。

$m = 6$ および $n = 5$ ($m > n$)。

$$\begin{bmatrix} d & e & u1 & u1 & u1 \\ v1 & d & e & u2 & u2 \\ v1 & v2 & d & e & u3 \\ v1 & v2 & v3 & d & e \\ v1 & v2 & v3 & v4 & d \\ v1 & v2 & v3 & v4 & v5 \end{bmatrix}$$

$m = 5$ および $n = 6$ ($m < n$)。

$$\begin{bmatrix} d & u1 & u1 & u1 & u1 & u1 \\ e & d & u2 & u2 & u2 & u2 \\ v1 & e & d & u3 & u3 & u3 \\ v1 & v2 & e & d & u4 & u4 \\ v1 & v2 & v3 & e & d & u5 \end{bmatrix}$$

ここで、 d と e は B の対角成分と非対角成分である。 v_i は $H(i)$ を定義するベクトルの成分を表し、 u_i は $G(i)$ を定義するベクトルの成分を表す。

p?ormbr

一般行列に p?gebrd で求めた 二重対角形式に縮退した直交行列を掛ける。

構文

```
call psormbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
            jc, descc, work, lwork, info)
call pdormbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
            jc, descc, work, lwork, info)
```

説明

$vect = 'Q'$ の場合、このルーチンは、 $m \times n$ の一般実分散行列 $sub(C) = C(c:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': Q sub(C)$	$sub(C) Q$
$trans = 'T': Q^T sub(C)$	$sub(C) Q^T$

$vect = 'P'$ の場合、このルーチンは、 $sub(C)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': P sub(C)$	$sub(C) P$
$trans = 'T': P^T sub(C)$	$sub(C) P^T$

ここで、 Q と P^T は、実分散行列 $A(ia:*, ja:*)$ を二重対角形式 $A(ia:*, ja:*) = Q B P^T$ に縮退させるときに p?gebrd で求めた直交分散行列である。 Q と P^T は、それぞれ、基本リフレクター $H(i)$ と $G(i)$ の積として定義される。

$nq = m$ ($side = 'L'$ の場合) または $nq = n$ ($side = 'R'$ の場合)。したがって、 nq は適用される直交行列 Q または P^T の次数である。

$vect = 'Q'$ の場合、 $A(ia:*, ja:*)$ は $nq \times k$ 行列であると仮定する。

$nq \geq k$ の場合、 $Q = H(1) H(2) \dots H(k)$

$nq < k$ の場合、 $Q = H(1) H(2) \dots H(nq-1)$

$vect = 'P'$ の場合、 $A(ia:*, ja:*)$ は、 $k \times nq$ 行列であると仮定する。

$k < nq$ の場合、 $P = G(1) G(2) \dots G(k)$

$k \geq nq$ の場合、 $P = G(1) G(2) \dots G(nq-1)$

入力パラメーター

$vect$ (グローバル) CHARACTER。
 $vect = 'Q'$ の場合、 Q または Q^T が適用される。
 $vect = 'P'$ の場合、 P または P^T が適用される。

<i>side</i>	(グローバル) CHARACTER。 <i>side</i> ='L' の場合、 Q または Q^T 、 P または P^T は左側から適用される。 <i>side</i> ='R' の場合、 Q または Q^T 、 P または P^T は右側から適用される。
<i>trans</i>	(グローバル) CHARACTER。 <i>trans</i> ='N' の場合、 Q または P が適用される (転置なし)。 <i>trans</i> ='T' の場合、 Q^T または P^T が適用される。
<i>m</i>	(グローバル) INTEGER。分散行列 <i>sub</i> (<i>C</i>) の行数。
<i>n</i>	(グローバル) INTEGER。分散行列 <i>sub</i> (<i>C</i>) の列数。
<i>k</i>	(グローバル) INTEGER。 <i>vect</i> ='Q' の場合、p?gebrd により縮退された元の分散行列の列数。 <i>vect</i> ='P' の場合、p?gebrd により縮退された元の分散行列の行数。 次の制約がある。 $k \geq 0$ 。
<i>a</i>	(ローカル) REAL (psormbr の場合) DOUBLE PRECISION (pdormbr の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja</i> +min(<i>nq</i> , <i>k</i>)-1)) (<i>vect</i> ='Q' の場合) または (<i>lld_a</i> , <i>LOCc</i> (<i>ja</i> + <i>nq</i> -1)) (<i>vect</i> ='P' の場合) の配列へのポインター。 <i>nq</i> = <i>m</i> (<i>side</i> ='L' の場合) または <i>nq</i> = <i>n</i> (それ以外の場合)。 基本リフレクター $H(i)$ と $G(i)$ を定義するベクトル。p?gebrd から返されたとおりに、行列 Q と P を決定する積である。 <i>vect</i> ='Q' の場合、 $lld_a \geq \max(1, LOCr(ia+nq-1))$ 。 <i>vect</i> ='P' の場合、 $lld_a \geq \max(1, LOCr(ia+\min(nq,k)-1))$ 。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル) REAL (psormbr の場合) DOUBLE PRECISION (pdormbr の場合) 配列、次元は <i>LOCc</i> (<i>ja</i> +min(<i>nq</i> , <i>k</i>)-1) (<i>vect</i> ='Q' の場合) または <i>LOCr</i> (<i>ia</i> +min(<i>nq</i> , <i>k</i>)-1) (<i>vect</i> ='P' の場合)。 <i>tau</i> (<i>i</i>) は、配列引数 <i>tauq</i> または <i>taup</i> を使用して pdgebrd から返されたとおりに、 Q あるいは P を決定する、基本リフレクター $H(i)$ または $G(i)$ のスカラー係数を含んでいなければならない。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル) REAL (psormbr の場合) DOUBLE PRECISION (pdormbr の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>jc</i> + <i>n</i> -1)) の配列へのポインター。分散行列 <i>sub</i> (<i>C</i>) のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>C</i> の最初の行と最初の列を示す、グローバル配列 <i>c</i> の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。

work (ローカル)
 REAL (psormbr の場合)
 DOUBLE PRECISION (pdormbr の場合)
 次元 *lwork* のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。 *work* の次元は
side = 'L' の場合、
 $nq = m_0$
 ((*vect* = 'Q' かつ $nq \geq k$) または (*vect* が 'Q' と等しくなく、かつ $nq > k$))
 の場合、 $iaa=ia$ 、 $jaa=ja$ 、 $mi=m$ 、 $ni=n$ 、 $icc=ic$ 、 $jcc=jc$ 。
 それ以外の場合、
 $iaa=ia+1$ 、 $jaa=ja$ 、 $mi=m-1$ 、 $ni=n$ 、 $icc=ic+1$ 、 $jcc=jc$ でなければなら
 ない。
side = 'R' の場合、 $nq = n_0$
 ((*vect* = 'Q' かつ $nq \geq k$) または (*vect* が 'Q' と等しくなく、かつ $nq > k$))
 の場合、
 $iaa=ia$ 、 $jaa=ja$ 、 $mi=m$ 、 $ni=n$ 、 $icc=ic$ 、 $jcc=jc$ 。
 それ以外の場合、
 $iaa=ia$ 、 $jaa=ja+1$ 、 $mi=m$ 、 $ni=n-1$ 、 $icc=ic$ 、 $jcc=jc+1$ でなければなら
 ない。
vect = 'Q' の場合、
side = 'L' の場合、 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + mpc0)*nb_a + nb_a * nb_a)$
side = 'R' の場合、
 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0))*nb_a) + nb_a * nb_a)$ でなければなら
 ない。
vect が 'Q' と等しくない場合で、*side* = 'L' の場合、
 $lwork \geq \max((mb_a*(mb_a-1))/2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(mi+iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmq), nqc0))*mb_a) + mb_a * mb_a)$
side = 'R' の場合、
 $lwork \geq \max((mb_a*(mb_a-1))/2, (mpc0 + nqc0)*mb_a) + mb_a * mb_a$
 でなければなら
 ない。
 ここで、 $lcmp = lcm / NPROW$ 、 $lcmq = lcm / NPCOL$ 、
 $lcm = \text{ilcm}(NPROW, NPCOL)$ 、
 $iroffa = \text{mod}(iaa-1, mb_a)$ 、
 $icoffa = \text{mod}(jaa-1, nb_a)$ 、
 $iarow = \text{indxcg2p}(iaa, mb_a, MYROW, rsrc_a, NPROW)$ 、
 $iacol = \text{indxcg2p}(jaa, nb_a, MYCOL, csrc_a, NPCOL)$ 、

```

mqa0 = numroc(mi+icoffa, nb_a, MYCOL, iacol, NPCOL)、
npa0 = numroc(ni+iroffa, mb_a, MYROW, iarow, NPROW)、
iroffc = mod(icc-1, mb_c)、
icoffc = mod(jcc-1, nb_c)、
icrow = indxg2p (icc, mb_c, MYROW, rsrc_c, NPROW)、
iccol = indxg2p (jcc, nb_c, MYCOL, csrc_c, NPCOL)、
mpc0 = numroc(mi+iroffc, mb_c, MYROW, icrow, NPROW)、
nqc0 = numroc(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)

```

indxg2p および numroc は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン blacs_gridinfo を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c	終了時に、 $vect='Q'$ の場合、 $sub(C)$ は $Q*sub(C)$ 、 $Q'*sub(C)$ 、 $sub(C)*Q'$ 、または $sub(C)*Q$ のいずれかによって上書きされる。 $vect='P'$ の場合、 $sub(C)$ は $P*sub(C)$ 、 $P'*sub(C)$ 、 $sub(C)*P$ 、または $sub(C)*P'$ のいずれかによって上書きされる。
$work(1)$	終了時に、 $work(1)$ には、最適なパフォーマンスを得るために必要な $lwork$ の最小値が格納される。
$info$	(グローバル) INTEGER。 $info = 0$ の場合、実行は正常に終了したことを示す。 $info < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?unmbr

p?gebrd で求めた二重対角形式に縮退したユニタリー変換行列を一般行列に掛ける。

構文

```

call pcunmbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info)
call pzunmbr(vect, side, trans, m, n, k, a, ia, ja, desca, tau, c, ic,
             jc, descc, work, lwork, info)

```

説明

$vect = 'Q'$ の場合、このルーチンは、 $m \times n$ の一般複素分散行列 $sub(C) = C(ic:ic+m-1, jc:jc+n-1)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': Q \ sub(C)$	$sub(C) \ Q$
$trans = 'C': Q^H \ sub(C)$	$sub(C) \ Q^H$

$vect = 'P'$ の場合、このルーチンは、 $sub(C)$ を以下により上書きする。

$side = 'L'$	$side = 'R'$
$trans = 'N': P \ sub(C)$	$sub(C) \ P$
$trans = 'C': P^H \ sub(C)$	$sub(C) \ P^H$

ここで、 Q と P^H は、複素分散行列 $A(ia:*, ja:*)$ を二重対角形式 $A(ia:*, ja:*) = Q B P^H$ に縮退させるときに [p?gebrd](#) で求めたユニタリー分散行列である。 Q と P^H は、それぞれ、基本リフレクター $H(i)$ と $G(i)$ の積として定義される。

$nq = m$ ($side = 'L'$ の場合) または $nq = n$ ($side = 'R'$ の場合)。したがって、 nq は適用されるユニタリー行列 Q または P^H の次数である。

$vect = 'Q'$ の場合、 $A(ia:*, ja:*)$ は $nq \times k$ 行列であると仮定する。

$nq \geq k$ の場合、 $Q = H(1) H(2) \dots H(k)$

$nq < k$ の場合、 $Q = H(1) H(2) \dots H(nq-1)$

$vect = 'P'$ の場合、 $A(ia:*, ja:*)$ は、 $k \times nq$ 行列であると仮定する。

$k < nq$ の場合、 $P = G(1) G(2) \dots G(k)$

$k \geq nq$ の場合、 $P = G(1) G(2) \dots G(nq-1)$

入力パラメーター

$vect$ (グローバル) CHARACTER。
 $vect = 'Q'$ の場合、 Q または Q^H が適用される。
 $vect = 'P'$ の場合、 P または P^H が適用される。

$side$ (グローバル) CHARACTER。
 $side = 'L'$ の場合、 Q または Q^H 、 P または P^H は左側から適用される。
 $side = 'R'$ の場合、 Q または Q^H 、 P または P^H は右側から適用される。

$trans$ (グローバル) CHARACTER。
 $trans = 'N'$ の場合、 Q または P が適用される (転置なし)。
 $trans = 'C'$ の場合、 Q^H または P^H が適用される (共役転置)。

m (グローバル)
 INTEGER。分散行列 $sub(C)$ の行数 ($m \geq 0$)。

n (グローバル) INTEGER。分散行列 $sub(C)$ の列数 ($n \geq 0$)。

k (グローバル) INTEGER。
 $vect = 'Q'$ の場合、[p?gebrd](#) により縮退された元の分散行列の列数。
 $vect = 'P'$ の場合、[p?gebrd](#) により縮退された元の分散行列の行数。

次の制約がある。 $k \geq 0$ 。

<i>a</i>	(ローカル) COMPLEX (psormbr の場合) DOUBLE COMPLEX (pdormbr の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+\min(nq,k)-1)$) (<i>vect</i> ='Q' の場合) または ($lld_a, LOCc(ja+nq-1)$) (<i>vect</i> ='P' の場合) の配列へのポインター。 $nq = m$ (<i>side</i> ='L' の場合) または $nq = n$ (それ以外の場合)。 基本リフレクター $H(i)$ と $G(i)$ を定義するベクトル。 p?gebrd から返されたとおりに、行列 Q と P を決定する積である。 <i>vect</i> ='Q' の場合、 $lld_a \geq \max(1, LOCr(ia+nq-1))$ 。 <i>vect</i> ='P' の場合、 $lld_a \geq \max(1, LOCr(ia+\min(nq,k)-1))$ 。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル) COMPLEX (pcunmbr の場合) DOUBLE COMPLEX (pzunmbr の場合) 配列、次元は $LOCc(ja+\min(nq,k)-1)$ (<i>vect</i> ='Q' の場合) または $LOCr(ia+\min(nq,k)-1)$ (<i>vect</i> ='P' の場合)。 $tau(i)$ は、配列引数 $tauq$ または $taup$ を使用して p?gebrd から返されたとおりに、 Q または P を決定する、基本リフレクター $H(i)$ あるいは $G(i)$ のスカラー係数を含んでいなければならない。 tau は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル) COMPLEX (pcunmbr の場合) DOUBLE COMPLEX (pzunmbr の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(jc+n-1)$) の配列へのポインター。分散行列 $sub(C)$ のローカル部分を含む。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 C の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル) COMPLEX (pcunmbr の場合) DOUBLE COMPLEX (pzunmbr の場合) 次元 $lwork$ のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 $work$ の次元は <i>side</i> ='L' の場合、 $nq = m_0$ (<i>vect</i> ='Q' かつ $nq \geq k$) または (<i>vect</i> が 'Q' と等しくなく、かつ $nq > k$) の場合、 $iaa=ia, jaa=ja, mi=m, ni=n, icc=ic, jcc=jc$ 。 それ以外の場合、

$iaa=ia+1$ 、 $jaa=ja$ 、 $mi=m-1$ 、 $ni=n$ 、 $icc=ic+1$ 、 $jcc=jc$ でなければならぬ。

$side = 'R'$ の場合、 $nq = n$ 。

(($vect = 'Q'$ かつ $nq \geq k$) または ($vect$ が $'Q'$ と等しくなく、かつ $nq > k$)) の場合、

$iaa=ia$ 、 $jaa=ja$ 、 $mi=m$ 、 $ni=n$ 、 $icc=ic$ 、 $jcc=jc$ 。

それ以外の場合、

$iaa=ia$ 、 $jaa=ja+1$ 、 $mi=m$ 、 $ni=n-1$ 、 $icc=ic$ 、 $jcc=jc+1$ でなければならぬ。

$vect = 'Q'$ の場合、

$side = 'L'$ の場合、 $lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + mpc0)*nb_a + nb_a * nb_a)$

$side = 'R'$ の場合、

$lwork \geq \max((nb_a*(nb_a-1))/2, (nqc0 + \max(npa0 + \text{numroc}(\text{numroc}(ni+icoffc, nb_a, 0, 0, NPCOL), nb_a, 0, 0, lcmq), mpc0))*nb_a) + nb_a * nb_a * \text{numroc}(\text{numroc}(mi+iroffc, nb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0))*mb_a) + mb_a * mb_a$ でなければならぬ。

$vect$ が $'Q'$ と等しくない場合で、 $side = 'L'$ の場合、

$lwork \geq \max((mb_a*(mb_a-1))/2, (mpc0 + \max(mqa0 + \text{numroc}(\text{numroc}(mi+iroffc, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcmp), nqc0))*mb_a) + mb_a * mb_a$

$side = 'R'$ の場合、

$lwork \geq \max((mb_a*(mb_a-1))/2, (mpc0 + nqc0)*mb_a) + mb_a * mb_a$

でなければならぬ。

ここで、 $lcmp = lcm / NPROW$ 、 $lcmq = lcm / NPCOL$ 、 $lcm = ilcm(NPROW, NPCOL)$ 、

$iroffa = \text{mod}(iaa-1, mb_a)$ 、

$icoffa = \text{mod}(jaa-1, nb_a)$ 、

$iarow = \text{indxg2p}(iaa, mb_a, MYROW, rsrc_a, NPROW)$ 、

$iacol = \text{indxg2p}(jaa, nb_a, MYCOL, csrc_a, NPCOL)$ 、

$mqa0 = \text{numroc}(mi+icoffa, nb_a, MYCOL, iacol, NPCOL)$ 、

$npa0 = \text{numroc}(ni+iroffa, mb_a, MYROW, iarow, NPROW)$ 、

$iroffc = \text{mod}(icc-1, mb_c)$ 、

$icoffc = \text{mod}(jcc-1, nb_c)$ 、

$icrow = \text{indxg2p}(icc, mb_c, MYROW, rsrc_c, NPROW)$ 、

$iccol = \text{indxg2p}(jcc, nb_c, MYCOL, csrc_c, NPCOL)$ 、

$mpc0 = \text{numroc}(mi+iroffc, mb_c, MYROW, icrow, NPROW)$ 、

$nqc0 = \text{numroc}(ni+icoffc, nb_c, MYCOL, iccol, NPCOL)$

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

`lwork = -1` の場合、`lwork` はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<code>c</code>	終了時に、 <code>vect='Q'</code> の場合、 <code>sub(C)</code> は $Q*\text{sub}(C)$ 、 $Q'*\text{sub}(C)$ 、 $\text{sub}(C)*Q$ 、または $\text{sub}(C)*Q'$ のいずれかによって上書きされる。 <code>vect='P'</code> の場合、 <code>sub(C)</code> は $P*\text{sub}(C)$ 、 $P'*\text{sub}(C)$ 、 $\text{sub}(C)*P$ 、または $\text{sub}(C)*P'$ のいずれかによって上書きされる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info < 0</code> の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

汎用対称固有値問題

このセクションで説明する ScaLAPACK ルーチンを利用すると、一般対称固有値問題 (LAPACK の「汎用対称固有値問題」を参照) を標準の対称固有値問題 $Cy = \lambda y$ に縮退させることができる。縮退させた後の標準の対称固有値問題は、この章ですでに説明した一連の ScaLAPACK ルーチン (「対称固有値問題」を参照) を呼び出して解くことができる。

表 6-7 に、これらのルーチンの一覧を示す。

表 6-7 汎用固有値問題から標準固有値問題への縮退用の計算ルーチン

機能	実対称行列	複素エルミート行列
標準問題への縮退	p?sygst	p?hegst

p?sygst

実対称の汎用固有値問題を標準形式に縮退させる。

構文

```
call pssygst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
call pdsygst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
```

説明

このルーチンは、実対称の汎用固有値問題を標準形式に縮退させる。

以下において、 $\text{sub}(A)$ は $A(ia:ia+n-1, ja:ja+n-1)$ を表し、 $\text{sub}(B)$ は $B(ib:ib+n-1, jb:jb+n-1)$ を表す。

$ibtype = 1$ の場合、問題は

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

$\text{sub}(A)$ は、 $\text{inv}(U^T) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^T)$ によって上書きされる。

$ibtype = 2$ または 3 の場合、問題は

$$\text{sub}(A) \text{sub}(B)x = \lambda x \text{ または } \text{sub}(B) \text{sub}(A)x = \lambda x$$

$\text{sub}(A)$ は、 $U \text{sub}(A) U^T$ または $L^T \text{sub}(A) L$ によって上書きされる。

$\text{sub}(B)$ は、 $U^T U$ または LL^T として、[p?potrf](#) によって事前に因子分解されていなければならない。

入力パラメーター

$ibtype$ (グローバル) INTEGER。1、2、または3のいずれかでなければならない。
 $itype = 1$ の場合、 $\text{inv}(U^T) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^T)$ を計

算する。

$itype = 2$ または 3 の場合、 $U\text{sub}(A)U^T$ あるいは $L^T\text{sub}(A)L$ を計算する。

<i>uplo</i>	(グローバル) CHARACTER。'U' または 'L' でなければならない。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の上三角部分が格納される。 <i>sub(B)</i> は $U^T U$ として因子分解される。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の下三角部分が格納される。 <i>sub(B)</i> は LL^T として因子分解される。
<i>n</i>	(グローバル) INTEGER。行列 <i>sub(A)</i> と <i>sub(B)</i> の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (<i>pssygst</i> の場合) DOUBLE PRECISION (<i>pdsygst</i> の場合) ローカルメモリーにある、次元 ($lld_a, LOCC(ja+n-1)$) の配列へのポインター。呼び出し時に、配列は $n \times n$ の対称分散行列 <i>sub(A)</i> のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>b</i>	(ローカル) REAL (<i>pssygst</i> の場合) DOUBLE PRECISION (<i>pdsygst</i> の場合) ローカルメモリーにある、次元 ($lld_b, LOCC(jb+n-1)$) の配列へのポインター。呼び出し時に、配列は、 <i>p?potrf</i> によって返される <i>sub(B)</i> のコレスキー因子分解で得られた三角係数を含む。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。

出力パラメーター

<i>a</i>	終了時に、 <i>info</i> = 0 の場合、変換後の行列が <i>sub(A)</i> と同じ形式で格納される。
<i>scale</i>	(グローバル) REAL (<i>pssygst</i> の場合) DOUBLE PRECISION (<i>pdsygst</i> の場合) このルーチンで実行されたスケールリングを補正するための固有値。 <i>scale</i> は将来の拡張のために用意されており、現在は常に 1.0 として返される。

info (グローバル) INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i*100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。

p?hegst

エルミートの汎用固有値問題を標準形式に縮退させる。

構文

```
call pchegst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
call pzhegst( ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, scale,
             info )
```

説明

このルーチンは、複素エルミートの汎用固有値問題を標準形式に縮退させる。

以下において、*sub(A)* は $A(ia:ia+n-1, ja:ja+n-1)$ を表し、*sub(B)* は $B(ib:ib+n-1, jb:jb+n-1)$ を表す。

ibtype = 1 の場合、問題は

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

sub(A) は、 $\text{inv}(U^H) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^H)$ によって上書きされる。

ibtype = 2 または 3 の場合、問題は

$$\text{sub}(A)\text{sub}(B)x = \lambda x \text{ または } \text{sub}(B)\text{sub}(A)x = \lambda x$$

sub(A) は、 $U \text{sub}(A) U^H$ または $L^H \text{sub}(A) L$ によって上書きされる。

sub(B) は、 $U^H U$ または LL^H として、[p?potrf](#) によって事前に因子分解されていなければならない。

入力パラメーター

ibtype (グローバル) INTEGER。1、2、または3のいずれかでなければならない。
type = 1 の場合、 $\text{inv}(U^H) \text{sub}(A) \text{inv}(U)$ または $\text{inv}(L) \text{sub}(A) \text{inv}(L^H)$ を計算する。
itype = 2 または 3 の場合、 $U \text{sub}(A) U^H$ あるいは $L^H \text{sub}(A) L$ を計算する。

uplo (グローバル)
 CHARACTER。'U' または 'L' でなければならない。
uplo = 'U' の場合、*sub(A)* の上三角部分が格納される。*sub(B)* は $U^H U$

として因子分解される。
 $uplo = 'L'$ の場合、 $sub(A)$ の下三角部分が格納される。 $sub(B)$ は LL^H として因子分解される。

<i>n</i>	(グローバル) INTEGER。行列 $sub(A)$ と $sub(B)$ の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) COMPLEX (pchebst の場合) DOUBLE COMPLEX (pzhebst の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列へのポインター。呼び出し時に、配列は $n \times n$ のエルミート分散行列 $sub(A)$ のローカル部分を含む。 $uplo = 'U'$ の場合、 $sub(A)$ の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $sub(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>b</i>	(ローカル) COMPLEX (pchebst の場合) DOUBLE COMPLEX (pzhebst の場合) ローカルメモリーにある、次元 ($lld_b, LOCc(jb+n-1)$) の配列へのポインター。呼び出し時に、配列は、 p?potrf によって返される $sub(B)$ のコレスキー因子分解で得られた三角係数を含む。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 B の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。

出力パラメーター

<i>a</i>	終了時に、 $info = 0$ の場合、変換後の行列が $sub(A)$ と同じ形式で格納される。
<i>scale</i>	(グローバル) REAL (pchebst の場合) DOUBLE PRECISION (pzhebst の場合) このルーチンで実行されたスケーリングを補正するための固有値。 <i>scale</i> は将来の拡張のために用意されており、現在は常に 1.0 として返される。
<i>info</i>	(グローバル) INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、 <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 $info = -(i100+j)$ 。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

ドライバールーチン

表 6-8 に、連立 1 次方程式、最小二乗問題、標準固有値と特異値問題、および汎用対称固有値問題を解くための ScaLAPACK ドライバールーチンを示す。

表 6-8 ScaLAPACK ドライバールーチン

問題のタイプ	行列のタイプ 格納形式	ドライバー	
1 次方程式	一般 (部分ピボット演算)	p?gesv (簡易ドライバー) p?gesvx (高度ドライバー)	
	一般帯 (部分ピボット演算)	p?gbsv (簡易ドライバー)	
	一般帯 (ピボット演算なし)	p?dbsv (簡易ドライバー)	
	一般三重対角 (ピボット演算なし)	p?dtsv (簡易ドライバー)	
	対称 / エルミート正定値	p?posv (簡易ドライバー) p?posvx (高度ドライバー)	
	対称 / エルミート正定値 (帯形式)	p?pbsv (簡易ドライバー)	
	対称 / エルミート正定値 (三重対角形式)	p?ptsv (簡易ドライバー)	
	線形最小二乗問題	一般的な $m \times n$	p?qels
	対称固有値問題	対称 / エルミート	p?syev (簡易ドライバー) p?syevx / p?heevx (高度ドライバー)
特異値分解	一般的な $m \times n$	p?gesvd	
汎用対称固有値問題	対称 / エルミート、単一行列 正定値	p?sygvx / p?hegvx (高度ドライバー)	

p?gesv

正分散行列を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

構文

```
call psgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pdgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pcgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
call pzgesv(n, nrhs, a, ia, ja, desca, ipiv, b, ib, jb, descb, info)
```

説明

p?gesv は、実 / 複素連立 1 次方程式 $\text{sub}(A) * X = \text{sub}(B)$ の解を計算する。
 ここで、 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ は $n \times n$ の分散行列、 X と
 $\text{sub}(B) = B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+nrhs-1)$ は $n \times nrhs$ の分散行列である。

部分的なピボット演算と行の交換による LU 分解を使用して、 $\text{sub}(A)$ を $\text{sub}(A) = PLU$ と
 して因子分解する。ここで、 P は置換行列、 L は単位下三角行列、 U は上三角行列であ
 る。 L と U は $\text{sub}(A)$ に格納される。次に、 $\text{sub}(A)$ の因子分解された形式を使用して、連
 立方程式 $\text{sub}(A) * X = \text{sub}(B)$ を解く。

入力パラメーター

- n (グローバル) INTEGER。処理される行数と列数。すなわち、分散部分
 行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
- $nrhs$ (グローバル) INTEGER。右辺の数。
 分散部分行列 B と X の列数 ($nrhs \geq 0$)。
- a, b (ローカル)
 REAL (psgesv の場合)
 DOUBLE PRECISION (pdgesv の場合)
 COMPLEX (pcgesv の場合)
 DOUBLE COMPLEX (pzgesv の場合)
 それぞれ、ローカルメモリーにある、ローカル次元の配列
 $a(\text{lld}_a, LOC_c(\text{ja}+n-1))$ と $b(\text{lld}_b, LOC_c(\text{jb}+nrhs-1))$ へのポ
 インター。
 呼び出し時に、配列 a は因子分解の対象となる $n \times n$ の分散行列 $\text{sub}(A)$
 のローカル部分を含む。
 呼び出し時に、配列 b は右辺の分散行列 $\text{sub}(B)$ のローカル部分を含
 む。
- ia, ja (グローバル) INTEGER。それぞれ、 $\text{sub}(A)$ の最初の行と最初の列を示
 す、グローバル配列 A の行インデックスと列インデックス。
- $desca$ (グローバルおよびローカル) INTEGER 配列、次元は ($dlen$)。分散行
 列 A の配列ディスクリプター。
- ib, jb (グローバル) INTEGER。それぞれ、 $\text{sub}(B)$ の最初の行と最初の列を示
 す、グローバル配列 B の行インデックスと列インデックス。
- $descb$ (グローバルおよびローカル) INTEGER 配列、次元は ($dlen$)。分散行
 列 B の配列ディスクリプター。

出力パラメーター

- a $\text{sub}(A) = PLU$ の因子分解で得られた係数 L と U によって上書きされ
 る。 L の単位対角成分は格納されない。
- b 解の分散行列 X によって上書きされる。
- $ipiv$ (ローカル) INTEGER 配列。
 $ipiv$ の次元は $(LOC_r(m_a) + mb_a)$ 。
 この配列には、ピボット情報が格納される。行列の (ローカル) 行 i
 は、(グローバル) 行 $ipiv(i)$ と交換される。
 この配列は、分散行列 A に関連付けられる。

info (グローバル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。

info < 0 の場合：
i 番目の引数が配列で、*j* 番目の値が不正だった場合、
info = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、
info = -*i*。

info > 0 の場合：
info = *k* の場合、 $U(ia+k-1, ja+k-1)$ は完全にゼロである。因子分解は完了したが、係数 *U* が完全に特異であるため、解を計算できなかった。

p?gesvx

LU 因子分解を使用して、正方行列 *A* を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算し、解の誤差範囲を示す。

構文

```
call psgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
            descaf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
            ferr, berr, work, lwork, iwork, liwork, info)
call pdgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
            descaf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
            ferr, berr, work, lwork, iwork, liwork, info)
call pcgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
            descaf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
            ferr, berr, work, lwork, rwork, lrwork, info)
call pzgesvx(fact, trans, n, nrhs, a, ia, ja, desca, af, iaf, jaf,
            descaf, ipiv, equed, r, c, b, ib, jb, descb, x, ix, jx, descx, rcond,
            ferr, berr, work, lwork, rwork, lrwork, info)
```

説明

このルーチンは、LU 因子分解を使用して、実 / 複素連立 1 次方程式 $AX = B$ の解を計算する。ここで、*A* は $n \times n$ の部分行列 $A(ia:ia+n-1, ja:ja+n-1)$ 、*B* は $n \times nrhs$ の部分行列 $B(ib:ib+n-1, jb:jb+nrhs-1)$ 、および *X* は $n \times nrhs$ の部分行列 $X(ix:ix+n-1, jx:jx+nrhs-1)$ を表す。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

以下の説明で、*af* は部分配列 $af(iaf:iaf+n-1, jaf:jaf+n-1)$ を表す。

ルーチン ?gesvx は、以下の手順を実行する。

1. *fact* = 'E' の場合、連立方程式を平衡化するための実スケール係数 *R* と *C* を計算する。

$$trans = 'N': \text{diag}(R) * A * \text{diag}(C) * \text{diag}(C)^{-1} * X = \text{diag}(R) * B$$

$$trans = 'T': (\text{diag}(R) * A * \text{diag}(C))^T * \text{diag}(R)^{-1} * X = \text{diag}(C) * B$$

$$trans = 'c': (\text{diag}(R)*A*\text{diag}(C))^H * \text{diag}(R)^{-1} * X = \text{diag}(C)*B$$

連立方程式が平衡化されるかどうかは、行列 A のスケーリングによって決まる。平衡化が行われる場合、 A は $\text{diag}(R)*A*\text{diag}(C)$ によって上書きされ、 B は $\text{diag}(R)*B$ ($trans='N'$ の場合) または $\text{diag}(C)*B$ ($trans='T'$ または $'c'$ の場合) によって上書きされる。

2. $fact = 'N'$ または $'E'$ の場合、 LU 分解を使用して、($fact = 'E'$ の場合は平衡化の後に) 行列 A を $A = PLU$ として因子分解する。ここで、 P は置換行列、 L は単位下三角行列、 U は上三角行列である。

3. A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合、ステップ 4 から 6 はスキップされる。

4. A の因子分解された形式を使用して、連立方程式を X について解く。

5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。

6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(C)$ ($trans = 'N'$ の場合) または $\text{diag}(R)$ ($trans = 'T'$ または $'c'$ の場合) によって、行列 X を事前乗算する。

入力パラメーター

<i>fact</i>	(グローバル) CHARACTER*1. 'F'、'N'、または 'E' でなければならない。 ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。 $fact = 'F'$ の場合、呼び出し時に、 <i>af</i> と <i>ipiv</i> は、 A の因子分解された形式を含む。 <i>equed</i> が 'N' でない場合、行列 A は、 r と c で指定されたスケール係数によって平衡化されている。配列 <i>a</i> 、 <i>af</i> 、および <i>ipiv</i> は変更されない。 $fact = 'N'$ の場合、行列 A を <i>af</i> にコピーし、因子分解を実行する。 $fact = 'E'$ の場合、必要に応じて行列 A を平衡化してから、 <i>af</i> にコピーし、因子分解を実行する。
<i>trans</i>	(グローバル) CHARACTER*1. 'N'、'T'、または 'c' でなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ の場合、連立方程式の形式は $AX = B$ (転置なし) である。 $trans = 'T'$ の場合、連立方程式の形式は $A^T X = B$ (転置) である。 $trans = 'c'$ の場合、連立方程式の形式は $A^H X = B$ (共役転置) である。
<i>n</i>	(グローバル) INTEGER. 1 次方程式の数。部分行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER. 右辺の数。分散部分行列 B と X の列の数 ($nrhs \geq 0$)。
<i>a, af, b, work</i>	(ローカル) REAL (<i>psgesvx</i> の場合) DOUBLE PRECISION (<i>pdgesvx</i> の場合)

COMPLEX (pcgesvx の場合)

DOUBLE COMPLEX (pzgesvx の場合)

それぞれ、ローカルメモリーにある、ローカル次元の配列

$a(lld_a, LOC_c(ja+n-1))$ 、 $af(lld_af, LOC_c(ja+n-1))$ 、 $b(lld_b, LOC_c(jb+nrhs-1))$ 、および $work(lwork)$ へのポインター。

配列 a には、行列 A が格納される。 $fact = 'F'$ かつ $equed$ が 'N' でない場合、 A は、 r または c 、あるいはその両方のスケール係数によって平衡化されている。

配列 af は、 $fact = 'F'$ の場合は入力引数になる。この場合、配列は、呼び出し時に、行列 A の因子分解された形式、すなわち、[p?getrf](#) による因子分解 $A = PLU$ で得られた係数 L と U を含む。 $equed$ が 'N' でない場合、 af は、平衡化された行列 A の因子分解された形式になる。

呼び出し時に、配列 b は行列 B を含む。この行列の列は連立方程式の右辺である。

$work(*)$ は、ワークスペース配列。

$work$ の次元は、 $(lwork)$ 。

ia, ja	(グローバル) INTEGER。 それぞれ、部分行列 $A(ia:ia+n-1, ja:ja+n-1)$ の最初の行と最初の列を示すグローバル配列 A の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は $(dlen_)$ 。分散行列 A の配列ディスクリプター。
iaf, jaf	(グローバル) INTEGER。 それぞれ、部分配列 $af(iaf:iaf+n-1, jaf:jaf+n-1)$ の最初の行と最初の列を示すグローバル配列 af の行インデックスと列インデックス。
$descaf$	(グローバルおよびローカル) INTEGER 配列、次元は $(dlen_)$ 。分散行列 AF の配列ディスクリプター。
ib, jb	(グローバル) INTEGER。 それぞれ、部分行列 $B(ib:ib+n-1, jb:jb+nrhs-1)$ の最初の行と最初の列を示すグローバル配列 B の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER 配列、次元は $(dlen_)$ 。分散行列 B の配列ディスクリプター。
$ipiv$	(ローカル) INTEGER 配列。 $ipiv$ の次元は $(LOC_r(m_a) + mb_a)$ 。 配列 $ipiv$ は、 $fact = 'F'$ の場合は入力引数になる。 呼び出し時に、この配列は、 p?getrf による因子分解 $A = PLU$ で得られたピボットのインデックスを含む。行列の (ローカル) 行 i は、(グローバル) 行 $ipiv(i)$ と交換される。 この配列は、 $A(ia:ia+n-1, *)$ とアライメントされなければならない。
$equed$	(グローバル) CHARACTER*1。'N'、'R'、'C'、または 'B' でなければならない。 $equed$ は、 $fact = 'F'$ の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 $equed = 'N'$ の場合、平衡化は行われていない ($fact = 'N'$ の場合、常に真)。

equed = 'R' の場合、行の平衡化が行われた。すなわち、*A* が $\text{diag}(r)$ によって事前乗算された。
equed = 'C' の場合、列の平衡化が行われた。すなわち、*A* が $\text{diag}(c)$ によって事後乗算された。
equed = 'B' の場合、行と列の平衡化、すなわち、*A* は $\text{diag}(r)*A*\text{diag}(c)$ で置き換えられた。

r, c (ローカル) REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列、次元はそれぞれ $LOC_r(m_a)$ および $LOC_c(n_a)$ 。
配列 *r* には *A* の行のスケール係数が格納され、配列 *c* には *A* の列のスケール係数が格納される。これらの配列は、*fact* = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。
equed = 'R' または 'B' の場合、*A* は $\text{diag}(r)$ によって左辺で乗算される。*equed* = 'N' または 'C' の場合、*r* は使用されない。
fact = 'F' で、*equed* = 'R' または 'B' の場合、*r* の各成分は正の値でなければならない。
equed = 'C' または 'B' の場合、*A* は $\text{diag}(c)$ によって右辺で乗算される。*equed* = 'N' または 'R' の場合、*c* は使用されない。
fact = 'F' で、*equed* = 'C' または 'B' の場合、*c* の各成分は正の値でなければならない。
配列 *r* はすべてのプロセス列に複製され、分散行列 *A* とアライメントされる。
配列 *c* はすべてのプロセス行に複製され、分散行列 *A* とアライメントされる。

ix, jx (グローバル) INTEGER。
それぞれ、部分行列 $X(ix:ix+n-1, jx:jx+nrhs-1)$ の最初の行と最初の列を示すグローバル配列 *X* の行インデックスと列インデックス。

descx (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *X* の配列ディスクリプター。

lwork (ローカルまたはグローバル) INTEGER。配列 *work* の次元。
 $\max(p?gecon(lwork), p?gerfs(lwork)) + LOC_r(n_a)$ 以上でなければならない。

iwork (ローカル、*psgesvx*/*pdgesvx* のみ) INTEGER。ワークスペース配列。
iwork の次元は (*liwork*)。

liwork (ローカル、*psgesvx*/*pdgesvx* のみ) INTEGER。配列 *iwork* の次元。
 $LOC_r(n_a)$ 以上でなければならない。

rwork (ローカル) REAL (*pcgesvx* の場合)
DOUBLE PRECISION (*pzgesvx* の場合)
複素数型の場合にのみ使用されるワークスペース配列。
rwork の次元は (*lrwork*)。

lrwork (ローカルまたはグローバル、*pcgesvx*/*pzgesvx* のみ) INTEGER。配列 *rwork* の次元。
 $2*LOC_c(n_a)$ 以上でなければならない。

出力パラメーター

- x* (ローカル)
 REAL (psgesvx の場合)
 DOUBLE PRECISION (pdgesvx の場合)
 COMPLEX (pcgesvx の場合)
 DOUBLE COMPLEX (pzgesvx の場合)
 ローカルメモリーにある、ローカル次元
 $x(1ld_x, LOC_c(jx+nrhs-1))$ の配列へのポインター。
info = 0 の場合、配列 *x* には、元の連立方程式の解の行列 *X* が格納される。ただし、*equed* ≠ 'N' の場合、*A* と *B* は終了時に修正され、平衡化された連立方程式の解は次のようになる。
 $diag(c)^{-1} * X$ (*trans* = 'N' で、*equed* = 'C' または 'B' の場合)
 $diag(R)^{-1} * X$ (*trans* = 'T' または 'C' で、*equed* = 'R' または 'B' の場合)
- a* 配列 *a* は、*fact* = 'F' または 'N' の場合、あるいは *fact* = 'E' で *equed* = 'N' の場合、終了時に変更されない。
equed ≠ 'N' の場合、*A* は終了時に次のようにスケールされる。
equed = 'R': $A = diag(R) * A$
equed = 'C': $A = A * diag(c)$
equed = 'B': $A = diag(R) * A * diag(c)$
- af* *fact* = 'N' または 'E' の場合、*af* は出力引数になる。この引数は、終了時に、元の行列 *A* (*fact* = 'N' の場合) または平衡化された行列 *A* (*fact* = 'E' の場合) の因子分解 $A = PLU$ で得られた係数 *L* と *U* を返す。平衡化された行列の形式については、*a* の説明を参照のこと。
- b* $diag(R) * B$ によって上書きされる (*trans* = 'N' で、*equed* = 'R' または 'B' の場合)
 $diag(c) * B$ によって上書きされる (*trans* = 'T' で、*equed* = 'C' または 'B' の場合)
 変更されない (*equed* = 'N' の場合)
- r, c* これらの配列は、*fact* ≠ 'F' の場合は出力引数になる。
 「入力引数」セクションの *r, c* の説明を参照のこと。
- rcond* (グローバル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 平衡化後の行列 *A* の条件数の逆数の推定値 (平衡化が行われる場合)。推定値のアンダーフローが発生した場合、*rcond* = 0 に設定される。この場合、行列は有効な精度で特異になる。ただし、*rcond* が (有効な精度で) 1.0 より小さい場合、条件の良くない行列または特異な行列である。
- ferr, berr* (ローカル)
 REAL (単精度の場合)
 DOUBLE PRECISION (倍精度の場合)
 配列、次元はそれぞれ $LOC_c(n_b)$ 。各解ベクトルの成分ごとの前進誤差と相対後退誤差を格納する。
 配列 *ferr* と *berr* はすべてのプロセス行に複製され、行列 *B* および *X* とアライメントされる。

<i>ipiv</i>	<i>fact</i> = 'N' または 'E' の場合、 <i>ipiv</i> は出力引数になる。この引数には、元の行列 <i>A</i> (<i>fact</i> = 'N' の場合) または平衡化された行列 <i>A</i> (<i>fact</i> = 'E' の場合) の因子分解 $A = PLU$ で得られたピボットのインデックスが格納される。
<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>work</i> (1)	<i>info</i> = 0 の場合、終了時に <i>work</i> (1) は、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値を返す。
<i>iwork</i> (1)	<i>info</i> = 0 の場合、終了時に <i>iwork</i> (1) は、最適なパフォーマンスを得るために必要な <i>liwork</i> の最小値を返す。
<i>rwork</i> (1)	<i>info</i> = 0 の場合、終了時に <i>rwork</i> (1) は、最適なパフォーマンスを得るために必要な <i>lrwork</i> の最小値を返す。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> = <i>i</i> で、 <i>i</i> ≤ <i>n</i> の場合、 <i>U</i> (<i>i</i> , <i>i</i>) は完全にゼロである。因子分解は完了したが、係数 <i>U</i> が完全に特異であるため、解と誤差範囲を計算できなかった。 <i>info</i> = <i>i</i> で、 <i>i</i> = <i>n</i> + 1 の場合、 <i>U</i> は特異でないが、 <i>rcond</i> はマシンの精度より小さい。因子分解は完了したが、行列は有効な精度で完全に特異であるため、解と誤差範囲を計算できなかった。

p?gbsv

一般帯分散行列を係数行列とする、複数の右辺を持つ連立 1 次方程式の解を計算する。

構文

```
call psqbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
            lwork, info)
call pdqbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
            lwork, info)
call pcqbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
            lwork, info)
call pzqbsv(n, bwl, bwu, nrhs, a, ja, desca, ipiv, b, ib, descb, work,
            lwork, info)
```

説明

p?gbsv は、実 / 複素連立 1 次方程式

$$\text{sub}(A) * X = \text{sub}(B)$$

の解を計算する。ここで、 $\text{sub}(A) = A(1:n, ja:ja+n-1)$ は *bw1* 劣対角と *bwu* 優対角を持つ $n \times n$ の実 / 複素一般帯分散行列、 X および $\text{sub}(B) = B(ib:ib+n-1, 1:nrhs)$ は $n \times nrhs$ の分散行列である。

部分的なピボット演算と行の交換による *LU* 分解を使用して、 $\text{sub}(A)$ を $\text{sub}(A) = PLUQ$ として因子分解する。ここで、 P と Q は置換行列、 L と U はそれぞれ帯形式の下三角行列、上三角行列である。行列 Q は並列化のために列の順序を変更することを表し、 P は数の安定のために部分的なピボット演算を使用して行の順序を変更することを表す。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。処理される行数と列数。すなわち、分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>bw1</i>	(グローバル) INTEGER。A の帯内にある劣対角成分の数 ($0 \leq bw1 \leq n-1$)。
<i>bwu</i>	(グローバル) INTEGER。A の帯内にある優対角成分の数 ($0 \leq bwu \leq n-1$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ の列数 ($nrhs \geq 0$)。
<i>a, b</i>	(ローカル) REAL (psgbsv の場合) DOUBLE PRECISION (pdgbsv の場合) COMPLEX (pcgbsv の場合) DOUBLE COMPLEX (pzgbsv の場合) それぞれ、ローカルメモリーにある、ローカル次元の配列 $a(1ld_a, LOC_c(ja+n-1))$ と $b(1ld_b, LOC_c(nrhs))$ へのポインタ。 呼び出し時に、配列 <i>a</i> はグローバル配列 <i>A</i> のローカル部分を含む。 呼び出し時に、配列 <i>b</i> は右辺の分散行列 $\text{sub}(B)$ のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>A</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca(dtype_)</i> = 501 の場合、 <i>dlen_</i> ≥ 7 。 <i>desca(dtype_)</i> = 1 の場合、 <i>dlen_</i> ≥ 9 。
<i>ib</i>	(グローバル) INTEGER。(B のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>B</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。 <i>descb(dtype_)</i> = 502 の場合、 <i>dlen_</i> ≥ 7 。 <i>descb(dtype_)</i> = 1 の場合、 <i>dlen_</i> ≥ 9 。

<i>work</i>	(ローカル) REAL (psgbsv の場合) DOUBLE PRECISION (pdgbsv の場合) COMPLEX (pcgbsv の場合) DOUBLE COMPLEX (pzgbsv の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>work</i> のサイズ。 $lwork \geq (NB+bwu)*(bw1+bwu)+6*(bw1+bwu)*(bw1+2*bwu) + \max(nrhs*(NB+2*bw1+4*bwu), 1)$ でなければならない。

出力パラメーター

<i>a</i>	終了時に、因子分解の詳細が格納される。 因子分解の結果は、LAPACK から返される因子分解とは異なる点に注意する。並列用の行列では、追加の置換が実行される。
<i>b</i>	終了時に、この配列には解の分散行列 <i>X</i> のローカル部分が格納される。
<i>ipiv</i>	(ローカル) INTEGER 配列。 <i>ipiv</i> の次元は <i>desca</i> (NB) 以上でなければならない。 ローカル因子分解用のピボットのインデックスを含む。因子分解および解の算出を行う間、この配列の内容は変更すべきでない点に注意する。
<i>work</i> (1)	終了時に、 <i>work</i> (1) には、最適なパフォーマンスを得るために必要な <i>lwork</i> の最小値が格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合： <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合： <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が非特異でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が非特異でないため、因子分解を完了できなかったことを示す。

p?dbsv

一般帯形式の連立1次方程式を解く。

構文

```
call psdbsv(n, bw1, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
           info)
call pddbsv(n, bw1, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
           info)
```

```
call pcdbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
           info)
call pzdbsv(n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, work, lwork,
           info)
```

説明

このルーチンは、次の連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$ はバンド幅 bwl 、 bwu の $n \times n$ の実 / 複素帯形式の対角優位分散行列である。

行列の順序を変更して LU に因子分解するために、ピボット演算を用いないガウス消去が使用される。

入力パラメーター

n	(グローバル) INTEGER。 分散部分行列 A の次数 ($n \geq 0$)。
bwl	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bwl \leq n-1$ 。
bwu	(グローバル) INTEGER。 劣対角成分の数。 $0 \leq bwu \leq n-1$ 。
$nrhs$	(グローバル) INTEGER。 右辺の数。分散部分行列 B の列数 ($nrhs \geq 0$)。
a	(ローカル)。 REAL (psdbsv の場合) DOUBLE PRECISION (pddbsv の場合) COMPLEX (pcdbsv の場合) DOUBLE COMPLEX (pzdbsv の場合) ローカルメモリーにある、第 1 次元が $lld_a \geq (bwl+bwu+1)$ の配列へのポインター ($desca$ に格納される)。呼び出し時に、この配列は、分散行列のローカル部分を含む。
ja	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 a のインデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen$)。 $1d$ type ($dtype_a=501$ または 502) の場合、 $dlen \geq 7$ 。 $2d$ type ($dtype_a=1$) の場合、 $dlen \geq 9$ 。 分散行列 A の配列ディスクリプター。 A のマッピング情報をメモリーに格納する。
b	(ローカル) REAL (psdbsv の場合) DOUBLE PRECISION (pddbsv の場合) COMPLEX (pcdbsv の場合) DOUBLE COMPLEX (pzdbsv の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $lld_b \geq NB$ の配列へのポインター。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。

<i>ib</i>	(グローバル) INTEGER。 (<i>b</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1d type (<i>dtype_b</i> =502) の場合、 <i>dlen</i> ≥ 7。 2d type (<i>dtype_b</i> =1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>B</i> の配列ディスクリプター。 <i>B</i> のマッピング情報をメモリーに格納する。
<i>work</i>	(ローカル) REAL (<i>psdbsv</i> の場合) DOUBLE PRECISION (<i>pddbsv</i> の場合) COMPLEX (<i>pcdbsv</i> の場合) DOUBLE COMPLEX (<i>pzdbsv</i> の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq NB(bw1+bwu)+6$ $\max(bw1,bwu)*\max(bw1,bwu) + \max((\max(bw1,bwu)nrhs),$ $\max(bw1,bwu)\max(bw1,bwu))$

出力パラメーター

<i>a</i>	終了時に、この配列には因子分解の詳細が格納される。行列で置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) には、 <i>lwork</i> の最小値が格納される。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、 <i>info</i> = -(<i>i</i> *100+ <i>j</i>)。 <i>i</i> 番目の引数がスカラーで値が不正だった場合、 <i>info</i> = - <i>i</i> 。 <i>info</i> > 0 の場合: <i>info</i> = <i>k</i> ≤ NPROCS の場合、プロセッサ <i>info</i> に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 <i>info</i> = <i>k</i> > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ <i>info</i> -NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

p?dtsv

三重対角形式の連立1次方程式を解く。

構文

```
call psdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pddtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pcdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
call pzdtsv(n, nrhs, dl, d, du, ja, desca, b, ib, descb, work, lwork, info)
```

説明

このルーチンは、次の連立1次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$ は $n \times n$ の複素対角優位三重対角行列である。

行列の順序を変更して LU に因子分解するために、ピボット演算を用いないガウス消去が使用される。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。分散部分行列 <i>A</i> の次数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数。分散行列 <i>B</i> の列数 ($nrhs \geq 0$)。
<i>dl</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合) DOUBLE COMPLEX (pzdtsv の場合) 行列の下対角を格納するグローバルベクトルのローカル部分へのポインター。全体的に、 <i>dl</i> (1) は参照されず、 <i>dl</i> は <i>d</i> とアライメントされなければならない。サイズは $\geq desca(nb_)$ でなければならない。
<i>d</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合) DOUBLE COMPLEX (pzdtsv の場合) 行列の主対角を格納するグローバルベクトルのローカル部分へのポインター。
<i>du</i>	(ローカル)。 REAL (psdtsv の場合) DOUBLE PRECISION (pddtsv の場合) COMPLEX (pcdtsv の場合) DOUBLE COMPLEX (pzdtsv の場合) 行列の上対角を格納するグローバルベクトルのローカル部分へのポインター。全体的に、 <i>du</i> (<i>n</i>) は参照されず、 <i>du</i> は <i>d</i> とアライメントされなければならない。

<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1d type (<i>dtype_a</i> =501 または 502) の場合、 <i>dlen</i> ≥ 7。 2d type (<i>dtype_a</i> =1) の場合、 <i>dlen</i> ≥ 9。 分散行列 A の配列ディスクリプター。A のマッピング情報をメモリーに格納する。
<i>b</i>	(ローカル) REAL (<i>psdtsv</i> の場合) DOUBLE PRECISION (<i>pddtsv</i> の場合) COMPLEX (<i>pcdtsv</i> の場合) DOUBLE COMPLEX (<i>pzdtsv</i> の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $11d_b \geq NB$ の配列へのポインター。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ib</i>	(グローバル) INTEGER。(b のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1d type (<i>dtype_b</i> =502) の場合、 <i>dlen</i> ≥ 7。 2d type (<i>dtype_b</i> =1) の場合、 <i>dlen</i> ≥ 9。 分散行列 B の配列ディスクリプター。B のマッピング情報をメモリーに格納する。
<i>work</i>	(ローカル)。 REAL (<i>psdtsv</i> の場合) DOUBLE PRECISION (<i>pddtsv</i> の場合) COMPLEX (<i>pcdtsv</i> の場合) DOUBLE COMPLEX (<i>pzdtsv</i> の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq (12 * NPCOL + 3 * NB) + \max((10 + 2 * \min(100, nrhs)) * NPCOL + 4 * nrhs, 8 * NPCOL)$

出力パラメーター

<i>d1</i>	終了時に、行列の係数を含む情報が格納される。
<i>d</i>	終了時に、行列の係数を含む情報が格納される。 サイズは ≥ <i>desca</i> (<i>nb</i>) でなければならない。
<i>du</i>	終了時に、行列の係数を含む情報が格納される。 サイズは ≥ <i>desca</i> (<i>nb</i>) でなければならない。
<i>b</i>	終了時に、解の分散行列 X のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) には、 <i>lwork</i> の最小値が格納される。

info (ローカル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。
info > 0 の場合: *info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。
info = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

p?posv

対称正定値連立1次方程式を解く。

構文

```
call psposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pdposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pcposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
call pzposv(uplo, n, nrhs, a, ia, ja, desca, b, ib, jb, descb, info)
```

説明

このルーチンは、実/複素連立1次方程式を計算する。

$\text{sub}(A) * X = \text{sub}(B)$

ここで、 $\text{sub}(A)$ は $A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ を表し、 $n \times n$ の対称/エルミート分散正定値行列である。 $B(\text{ib}:\text{ib}+n-1, \text{jb}:\text{jb}+\text{nrhs}-1)$ を表す X と $\text{sub}(B)$ は、 $n \times \text{nrhs}$ の分散行列である。コレスキー因子分解を使用して $\text{sub}(A)$ を以下のように因子分解する。

$\text{sub}(A) = U^T * U$ (*uplo* = 'U' の場合)

$\text{sub}(A) = L * LT$ (*uplo* = 'L' の場合)。

ここで、 U は上三角行列、 L は下三角行列である。次に、 $\text{sub}(A)$ の因子分解された形式を使用して、連立方程式を解く。

入力パラメーター

uplo (グローバル) CHARACTER。 'U' または 'L' でなければならない。
 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。
n (グローバル) INTEGER。 分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
nrhs INTEGER。 右辺の数。 分散部分行列 $\text{sub}(B)$ の列数 ($\text{nrhs} \geq 0$)。
a (ローカル)
 REAL (psposv の場合)
 DOUBLE PRECISION (pdposv の場合)
 COMPLEX (pcposv の場合)

COMPLEX*16 (pzposv の場合)
ローカルメモリーにある、次元 ($lld_a, LOC(ja+n-1)$) の配列へのポインター。呼び出し時に、この配列は、因子分解する $n \times n$ の対称分散行列 $sub(A)$ のローカル部分を含む。
 $uplo = 'U'$ の場合、 $sub(A)$ の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。厳密な下三角部分は参照されない。
 $uplo = 'L'$ の場合、 $sub(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

b (ローカル)
REAL (psposv の場合)
DOUBLE PRECISION (pdposv の場合)
COMPLEX (pcposv の場合)
COMPLEX*16 (pzposv の場合)
ローカルメモリーにある、次元 ($lld_b, LOC(jb+nrhs-1)$) の配列へのポインター。呼び出し時は、分散行列 $sub(B)$ の右辺のローカル部分。

ib, jb (グローバル) INTEGER。それぞれ、部分行列 B の最初の行と最初の列を示す、グローバル配列 b の行インデックスと列インデックス。

descb (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。

出力パラメーター

a 終了時に、 $info = 0$ の場合、この配列には、コレスキー因子分解 $sub(A) = U^H U$ または LL^H で得られた係数 U あるいは L が格納される。

b 終了時に、 $info = 0$ の場合、 $sub(B)$ は解の行列 X によって上書きされる。

info (グローバル) INTEGER。
 $info = 0$ の場合、実行は正常に終了したことを示す。
 $info < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。
 $info > 0$ の場合： $info = k$ の場合、次数 k の先頭の小行列式 $A(ia:ia+k-1, ja:ja+k-1)$ が正定値でないため、因子分解を完了できず、解が計算できなかったことを示す。

p?posvx

対称/エルミート正定値連立1次方程式を解く。

構文

```
call psposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
            work, lwork, iwork, liwork, info)

call pdposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
            work, lwork, iwork, liwork, info)

call pcposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
            work, lwork, iwork, liwork, info)

call pzposvx(fact, uplo, n, nrhs, a, ia, ja, desca, af, iaf, jaf, descaf,
            equed, sr, sc, b, ib, jb, descb, x, ix, jx, descx, rcond, ferr, berr,
            work, lwork, iwork, liwork, info)
```

説明

このルーチンは、コレスキー因子分解 $A=U^T U$ または $A=LL^T$ を使用して実/複素連立1次方程式の解を計算する。

$$A(ia:ia+n-1, ja:ja+n-1) * X = B(ib:ib+n-1, jb:jb+nrhs-1)$$

ここで、 $A(ia:ia+n-1, ja:ja+n-1)$ は $n \times n$ の行列、 X と $B(ib:ib+n-1, jb:jb+nrhs-1)$ は $n \times nrhs$ の行列である。

このルーチンは、解の誤差範囲と条件数の推定値も示す。

次の入力/出力パラメーターの説明では、 y は $Y(iy:iy+m-1, jy:jy+k-1)$ が $m \times k$ の行列 (y は a , af , b および x) であることを示す。

ルーチン p?posvx は、以下の手順を実行する。

1. $fact = 'E'$ の場合、連立方程式を平衡化するための実スケール係数 s を計算する。

$$\text{diag}(sr) * A * \text{diag}(sc) * \text{inv}(\text{diag}(sc)) * X = \text{diag}(sr) * B$$

連立方程式が平衡化されるかどうかは、行列 A のスケーリングによって決まる。平衡化が行われる場合、 A は $\text{diag}(sr) * A * \text{diag}(sc)$ によって上書きされ、 B は $\text{diag}(sr) * B$ によって上書きされる。

2. $fact = 'N'$ または $'E'$ の場合、コレスキー分解を使用して、($fact = 'E'$ の場合は平衡化の後に) 行列 A を次のように因子分解する。

$$A = U^T U \text{ (uplo = 'U' の場合)}$$

$$A = LL^T \text{ (uplo = 'L' の場合)}$$

ここで、 U は上三角行列、 L は下三角行列である。

3. A の因子分解された形式を使用して、行列 A の条件数を推定する。条件数の逆数がマシンの精度より小さい場合、ステップ 4 から 6 はスキップされる。

4. A の因子分解された形式を使用して、連立方程式を X について解く。
5. 計算された解の行列の精度を繰り返し改善し、解の誤差範囲と後退誤差の推定値を計算する。
6. 平衡化が行われている場合、平衡化が行われる前の元の連立方程式を解くために、 $\text{diag}(sr)$ によって行列 X を事前乗算する。

入力パラメーター

<i>fact</i>	(グローバル) CHARACTER。'F'、'N'、または'E' でなければならない。 ルーチンの開始時に行列 A の因子分解された形式が与えられるかどうか、与えられない場合は、行列 A を因子分解する前に A を平衡化するかどうかを指定する。 <i>fact</i> = 'F' の場合、呼び出し時に、 <i>af</i> は A の因子分解された形式を含む。 <i>equed</i> = 'Y' の場合、行列 A は、 <i>s</i> で指定されたスケール係数によって平衡化されている。 <i>a</i> と <i>af</i> は変更されない。 <i>fact</i> = 'N' の場合、行列 A を <i>af</i> にコピーし、因子分解を実行する。 <i>fact</i> = 'E' の場合、必要に応じて行列 A を平衡化してから、 <i>af</i> にコピーし、因子分解を実行する。
<i>uplo</i>	(グローバル) CHARACTER。'U' または 'L' でなければならない。 A の上三角部分と下三角部分のどちらが格納されるかを指定する。
<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。 分散部分行列 B と X の列数 ($nrhs \geq 0$)。
<i>a</i>	(ローカル) REAL (<i>psposvx</i> の場合) DOUBLE PRECISION (<i>pdposvx</i> の場合) COMPLEX (<i>pcposvx</i> の場合) DOUBLE COMPLEX (<i>pzposvx</i> の場合) ローカルメモリーにある、ローカル次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja+n-1</i>)) の配列へのポインター。呼び出し時は、対称/エルミート行列 A 。ただし、 <i>fact</i> = 'F' で <i>equed</i> = 'Y' の場合、 A は平衡化された行列 $\text{diag}(sr)*A*\text{diag}(sc)$ を含む。 <i>uplo</i> = 'U' の場合、 A の先頭の $n \times n$ 上三角部分に行列 A の上三角部分を含む。 A の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 A の先頭の $n \times n$ 下三角部分は行列 A の下三角部分を含む。 A の厳密な上三角部分は参照されない。 A は、 <i>fact</i> = 'F' または 'N' の場合、または <i>fact</i> = 'E' で <i>equed</i> = 'N' の場合、終了時に変更されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。

<i>af</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合) ローカルメモリーにある、ローカル次元 (<i>lld_af</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。 <i>fact</i> = 'F' の場合、 <i>af</i> は入力引数になる。呼び出し時に、この配列は、 コレスキー因子分解 $A = U^T * U$ または $A = L * L^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を、 <i>A</i> と同じ格納形式で含む。 <i>equed</i> が 'N' でない場合、 <i>af</i> は、平衡化された行列 $\text{diag}(sr) * A * \text{diag}(sc)$ の因子分解された形式になる。
<i>iaf, jaf</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(AF)</i> の最初の行と最初の列を示す、グローバル配列 <i>af</i> の行インデックスと列インデックス。
<i>descaf</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>AF</i> の配列ディスクリプター。
<i>equed</i>	(グローバル) CHARACTER。'N' または 'Y' でなければならない。 <i>equed</i> は、 <i>fact</i> = 'F' の場合は入力引数になる。この引数は、実行された平衡化の形式を指定する。 <i>equed</i> = 'N' の場合、平衡化は行われていない (<i>fact</i> = 'N' の場合、常に真)。 <i>equed</i> = 'Y' の場合、平衡化が行われ、 <i>A</i> は $\text{diag}(sr) * A * \text{diag}(sc)$ で置き換えられた。
<i>sr</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合) 配列、次元は (<i>lld_a</i>)。 配列 <i>s</i> には、 <i>A</i> のスケール係数が格納される。この配列は、 <i>fact</i> = 'F' の場合にのみ入力引数になり、それ以外の場合は出力引数になる。 <i>equed</i> = 'N' の場合、 <i>s</i> は使用されない。 <i>fact</i> = 'F' で、 <i>equed</i> = 'Y' の場合、 <i>s</i> の各成分は正の値でなければならない。
<i>b</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合) ローカルメモリーにある、ローカル次元 (<i>lld_b</i> , <i>LOCc(jb+nrhs-1)</i>) の配列へのポインター。 呼び出し時は、 $n \times nrhs$ の行列 <i>B</i> の右辺。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。

<i>x</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合) ローカルメモリーにある、ローカル次元 (<i>lld_x</i> , <i>LOCc(jx+nrhs-1)</i>) の配列へのポインター。
<i>ix, jx</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>X</i> の最初の行と最初の列を示す、グローバル配列 <i>x</i> の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>X</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psposvx の場合) DOUBLE PRECISION (pdposvx の場合) COMPLEX (pcposvx の場合) DOUBLE COMPLEX (pzposvx の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork = \max(p?pocon(lwork), p?porfs(lwork)) + LOCr(n_a)$ 以上でなければならない。 $lwork = 3*desca(lld_)$ <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての <i>work</i> 配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。
<i>iwork</i>	(ローカル) INTEGER。ワークスペース配列、次元は (<i>liwork</i>)。
<i>liwork</i>	(ローカルまたはグローバル) INTEGER。配列 <i>iwork</i> の次元。 <i>liwork</i> はローカル入力であり、 $liwork = desca(lld_) liwork = LOCr(n_a)$ 以上でなければならない。 <i>liwork</i> = -1 の場合、 <i>liwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、 <i>fact</i> = 'E' で <i>equed</i> = 'Y' の場合、 <i>a</i> は $\text{diag}(sr)*a*\text{diag}(sc)$ によって上書きされる。
<i>af</i>	<i>fact</i> = 'N' の場合、 <i>af</i> は出力引数になる。この引数は、元の行列 <i>A</i> のコレスキー因子分解 $A = U^T*U$ または $A = L*L^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を返す。 <i>fact</i> = 'E' の場合、 <i>af</i> は出力引数になる。この引数は、平衡化された行列 <i>A</i> のコレスキー因子分解 $A = U^T*U$ または $A = L*L^T$ で得られた三角係数 <i>U</i> あるいは <i>L</i> を返す (平衡化された行列の形式は、 <i>A</i> の説明を参照)。

<i>equed</i>	<i>fact</i> ≠ 'F' の場合、 <i>equed</i> は出力引数になる。この引数は、実行された平衡化の形式を指定する (「入力引数」セクションの <i>equed</i> の説明を参照)。
<i>sr</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>sr</i> の説明を参照。
<i>sc</i>	この配列は、 <i>fact</i> ≠ 'F' の場合は出力引数になる。 「入力引数」セクションの <i>sc</i> の説明を参照。
<i>b</i>	終了時に、 <i>equed</i> = 'N' の場合、 <i>b</i> は変更されない。 <i>trans</i> = 'N' で <i>equed</i> = 'R' または 'B' の場合、 <i>b</i> は $\text{diag}(r)*b$; によって上書きされる。 <i>trans</i> = 'T' または 'C' で <i>equed</i> = 'C' または 'B' の場合、 <i>b</i> は $\text{diag}(c)*b$ に よって上書きされる。
<i>x</i>	(ローカル) REAL (<i>psposvx</i> の場合) DOUBLE PRECISION (<i>pdposvx</i> の場合) COMPLEX (<i>pcposvx</i> の場合) DOUBLE COMPLEX (<i>pzposvx</i> の場合) <i>info</i> = 0 の場合、元の連立方程式の $n \times nrhs$ の解の行列 <i>X</i> が格納される。ただし、 <i>equed</i> .ne. 'N' の場合、 <i>A</i> と <i>B</i> は終了時に修正され、平衡化された連立方程式の解は次のようになる。 $\text{inv}(\text{diag}(sc))*X$ (<i>trans</i> = 'N' かつ <i>equed</i> = 'C' または 'B' の場合) $\text{inv}(\text{diag}(sr))*X$ (<i>trans</i> = 'T' または 'C' かつ <i>equed</i> = 'R' または 'B' の場合)。
<i>rcond</i>	(グローバル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 平衡化後の行列 <i>A</i> の条件数の逆数の推定値 (平衡化が行われる場合)。 <i>rcond</i> がマシンの精度より小さい場合 (特に、 <i>rcond</i> = 0 の場合) は、行列は有効な精度で特異になる。この条件は、 <i>info</i> > 0 のリターンコードで示される。
<i>ferr</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(LOC, n_b)$ 以上。各解ベクトル用に推定された前進誤差範囲 <i>X(j)</i> (すなわち解の行列 <i>X</i> の <i>j</i> 番目の列)。 <i>xtrue</i> が真の解である場合、 <i>ferr(j)</i> は、 $(X(j) - xtrue)$ の最大エントリーの大きさを <i>X(j)</i> の最大エントリーの大きさで割って範囲を決める。誤差範囲の質は、コードで計算された $\text{norm}(\text{inv}(A))$ の推定の質に依存する。 $\text{norm}(\text{inv}(A))$ の推定が正確である場合、誤差範囲は保証される。
<i>berr</i>	(ローカル) REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列、次元は $\max(LOC, n_b)$ 以上。 各解ベクトルの成分ごとの相対後退誤差 <i>X(j)</i> (すなわち <i>X(j)</i> が正確な解となる <i>A</i> または <i>B</i> の任意のエントリーにおける最小相対変化)。
<i>iwork(1)</i>	(ローカル) 終了時に、 <i>iwork(1)</i> は、最小かつ最適な <i>liwork</i> 値を返す。

info (グローバル) INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
 < 0 の場合: *info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
 > 0 の場合: *info* = *i* で *i* ≤ *n* の場合、*info* = *i* は *a* の次数 *i* の先頭の
 小行列式が正定値ではないため因子分解を完了できず、解と誤差範囲
 を計算できなかったことを示す。
i = *n*+1 の場合: *rcond* はマシンの精度より小さいことを示す。因子分
 解は完了したが、行列は有効な精度で完全に特異であるため、解と誤
 差範囲を計算できなかった。

p?pbsv

対称/エルミート正定値帯形式の連立1次方程式
 を解く。

構文

```
call pspbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pdpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pcpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
call pzpbsv(uplo, n, bw, nrhs, a, ja, desca, b, ib, descb, work, lwork,
            info)
```

説明

このルーチンは、次の連立1次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$ は帯域が bw の $n \times n$ の実/複素対称正定値帯分散行列である。
 行列の順序を変更して LL' に因子分解するために、コレスキー分解が使用される。

入力パラメーター

uplo (グローバル) CHARACTER。'U' または 'L' でなければならない。
A の上三角部分と下三角部分のどちらが格納されるかを指定する。
uplo = 'U' の場合、*A* の上三角部分が格納される。
uplo = 'L' の場合、*A* の下三角部分が格納される。
n (グローバル) INTEGER。分散行列 *A* の次数 ($n \geq 0$)。
bw (グローバル) INTEGER。*L* または *U* の劣対角成分の数。 $0 \leq bw \leq n-1$ 。
nrhs (グローバル) INTEGER。右辺の数。
 行列 *B* の列数 ($nrhs \geq 0$)。
a (ローカル)。
 REAL (pspbsv の場合)
 DOUBLE PRECISION (pdpbsv の場合)
 COMPLEX (pcpbsv の場合)

	DOUBLE COMPLEX (pzpbsv の場合) ローカルメモリーにある、第 1 次元が $lld_a \geq (bw+1)$ の配列へのポインター (<i>desca</i> に格納される)。 呼び出し時に、この配列は、因子分解する対称分散行列 <i>sub(A)</i> のローカル部分を含む。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 <i>a</i> のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>b</i>	(ローカル) REAL (pspbsv の場合) DOUBLE PRECISION (pdpbsv の場合) COMPLEX (pcpbsv の場合) DOUBLE COMPLEX (pzpbsv の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $lld_b \geq NB$ の配列へのポインター。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ib</i>	(グローバル) INTEGER。(b のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1D type (<i>dtype_b</i> =502) の場合、 $dlen \geq 7$ 。 2D type (<i>dtype_b</i> =1) の場合、 $dlen \geq 9$ 。 分散行列 <i>B</i> の配列ディスクリプター。 <i>B</i> のマッピング情報をメモリーに格納する。
<i>work</i>	(ローカル)。 REAL (pspbsv の場合) DOUBLE PRECISION (pdpbsv の場合) COMPLEX (pcpbsv の場合) DOUBLE COMPLEX (pzpbsv の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work(1)</i> に返される。 $lwork \geq (NB+2*bw)*bw + \max((bw*nrhs), bw*bw)$

出力パラメーター

<i>a</i>	終了時に、この配列には因子分解の詳細が格納される。行列で置換が行われるため、返される係数は LAPACK から返される係数とは異なる点に注意する。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work(1)</i> には、 <i>lwork</i> の最小値が格納される。
<i>info</i>	(グローバル)。 INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合 : <i>i</i> 番目の引数が配列で、 <i>j</i> 番目の値が不正だった場合、

$info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

$info > 0$ の場合 : $info = k \leq NPROCS$ の場合、プロセッサ $info$ に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。

$info = k > NPROCS$ の場合、他のプロセッサとの対話を表すプロセッサ $info-NPROCS$ に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

p?ptsv

対称/エルミート正定値三重対角連立1次方程式を解く。

構文

```
call psptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pdptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pcptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
call pzptsv(n, nrhs, d, e, ja, desca, b, ib, descb, work, lwork, info)
```

説明

このルーチンは、次の連立1次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs)$$

ここで、 $A(1:n, ja:ja+n-1)$ は $n \times n$ の実対称正定値三重対角分散行列である。

行列の順序を変更して LL' に因子分解するために、コレスキー分解が使用される。

入力パラメーター

n (グローバル) INTEGER。行列 A の次数 ($n \geq 0$)。

$nrhs$ (グローバル) INTEGER。右辺の数。分散部分行列 B の列数 ($nrhs \geq 0$)。

d (ローカル)
 REAL (psptsv の場合)
 DOUBLE PRECISION (pdptsv の場合)
 COMPLEX (pcptsv の場合)
 DOUBLE COMPLEX (pzptsv の場合)
 行列の主対角を格納するグローバルベクトルのローカル部分へのポインター。

e (ローカル)
 REAL (psptsv の場合)
 DOUBLE PRECISION (pdptsv の場合)
 COMPLEX (pcptsv の場合)
 DOUBLE COMPLEX (pzptsv の場合)

	行列の上対角を格納するグローバルベクトルのローカル部分へのポインター。全体的に、 $du(n)$ は参照されず、 du は d とアライメントされなければならない。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1d type (<i>dtype_a</i> =501 または 502) の場合、 $dlen \geq 7$ 。 2d type (<i>dtype_a</i> =1) の場合、 $dlen \geq 9$ 。 分散行列 A の配列ディスクリプター。A のマッピング情報をメモリーに格納する。
<i>b</i>	(ローカル) REAL (<i>psptsv</i> の場合) DOUBLE PRECISION (<i>pdptsv</i> の場合) COMPLEX (<i>pcptsv</i> の場合) DOUBLE COMPLEX (<i>pzptsv</i> の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $lld_b \geq NB$ の配列へのポインター。呼び出し時に、この配列は、右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を含む。
<i>ib</i>	(グローバル) INTEGER。(b のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 b の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 1d type (<i>dtype_b</i> =502) の場合、 $dlen \geq 7$ 。 2d type (<i>dtype_b</i> =1) の場合、 $dlen \geq 9$ 。 分散行列 B の配列ディスクリプター。B のマッピング情報をメモリーに格納する。
<i>work</i>	(ローカル)。 REAL (<i>psptsv</i> の場合) DOUBLE PRECISION (<i>pdptsv</i> の場合) COMPLEX (<i>pcptsv</i> の場合) DOUBLE COMPLEX (<i>pzptsv</i> の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq (12*NPCOL+3*NB)+\max((10+2*\min(100,nrhs))*NPCOL+4*nrhs, 8*NPCOL)$

出力パラメーター

<i>d</i>	終了時に、行列の係数を含む情報が格納される。サイズは $\geq desca(nb_)$ でなければならない。
<i>e</i>	行列の係数を含む情報が格納される。サイズは $\geq desca(nb_)$ でなければならない。
<i>b</i>	終了時に、解の分散行列 X のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) には、 <i>lwork</i> の最小値が格納される。

info (ローカル) INTEGER。 *info* = 0 の場合、実行は正常に終了したことを示す。
info < 0 の場合: *i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info* = -(*i**100+*j*)。 *i* 番目の引数がスカラーで値が不正だった場合、*info* = -*i*。
info > 0 の場合: *info* = *k* ≤ NPROCS の場合、プロセッサ *info* に格納され、ローカルに因子分解された部分行列が正定値でないため、因子分解を完了できなかったことを示す。 *info* = *k* > NPROCS の場合、他のプロセッサとの対話を表すプロセッサ *info*-NPROCS に格納された部分行列が正定値ではないため、因子分解を完了できなかったことを示す。

p?gels

最大階数の行列で表される、優決定または劣決定の線形問題を解く。

構文

```
call psgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
            work, lwork, info )
call pdgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
            work, lwork, info )
call pcgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
            work, lwork, info )
call pzgels( trans, m, n, nrhs, a, ia, ja, desca, b, ib, jb, descb,
            work, lwork, info )
```

説明

このルーチンは、sub(A) の QR または LQ 因子分解を使用して、 $m \times n$ の行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ (またはその転置 / 共役転置行列) で表される、実 / 複素の優決定 / 劣決定の線形問題を解く。ただし、sub(A) は最大階数の行列とする。

次のオプションが提供されている。

1. *trans* = 'N' で、 $m \geq n$ の場合: 優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize } \| \text{sub}(B) - \text{sub}(A) X \|$$

2. *trans* = 'N' かつ $m < n$ の場合: 劣決定の問題 $\text{sub}(A) X = \text{sub}(B)$ の最小ノルム解を求める。

3. *trans* = 'T' かつ $m \geq n$ の場合: 劣決定の問題 $\text{sub}(A)^T X = \text{sub}(B)$ の最小ノルム解を求める。

4. *trans* = 'N' で、 $m < n$ の場合: 優決定の最小二乗解を求める。すなわち次の最小二乗問題を解く。

$$\text{minimize } \| \text{sub}(B) - \text{sub}(A)^T X \|$$

ここで、 $\text{sub}(B)$ は $B(ib:ib+m-1, jb:jb+nrhs-1)$ ($trans = 'N'$ の場合)、 $B(ib:ib+n-1, jb:jb+nrhs-1)$ (それ以外の場合) である。複数の右辺のベクトル b と解ベクトル x を、一度の呼び出しで処理できる。 $trans = 'N'$ の場合、解ベクトルは $n \times nrhs$ の右辺の行列 $\text{sub}(B)$ に格納され、それ以外の場合、 $m \times nrhs$ の右辺の行列 $\text{sub}(B)$ に格納される。

入力パラメーター

<i>trans</i>	(グローバル) CHARACTER。'N' または 'T' でなければならない。 <i>trans</i> = 'N' の場合、行列 $\text{sub}(A)$ で表される線形問題。 <i>trans</i> = 'T' の場合、転置行列 A^T で表される線形問題 (実数型の場合のみ)。
<i>m</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の行数 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の列数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。分散部分行列 $\text{sub}(B)$ と X の列数 ($nrhs \geq 0$)。
<i>a</i>	(ローカル) REAL (psgels の場合) DOUBLE PRECISION (pdgels の場合) COMPLEX (pcgels の場合) DOUBLE COMPLEX (pzdgels の場合) ローカルメモリーにある、次元 ($lld_a, LOCC(ja+n-1)$) の配列へのポインター。呼び出し時に、 $m \times n$ の行列 A を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>b</i>	(ローカル) REAL (psgels の場合) DOUBLE PRECISION (pdgels の場合) COMPLEX (pcgels の場合) DOUBLE COMPLEX (pzdgels の場合) ローカルメモリーにある、ローカル次元 ($lld_b, LOCC(jb+nrhs-1)$) の配列へのポインター。呼び出し時に、この配列は、右辺の分散行列 B のローカル部分を列方向に含む。 $\text{sub}(B)$ は $m \times nrhs$ ($trans='N'$ の場合) または $n \times nrhs$ (それ以外の場合)。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 B の最初の行と最初の列を示す、グローバル配列 b の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (psgels の場合) DOUBLE PRECISION (pdgels の場合)

COMPLEX (pcgels の場合)
DOUBLE COMPLEX (pzgels の場合)
次元 $lwork$ のワークスペース配列。

$lwork$ (ローカルまたはグローバル)
INTEGER。配列 $work$ の次元。
 $lwork$ はローカル入力であり、
 $lwork \geq ltau + \max(lwf, lws)$ でなければならない。ここで、
 $m \geq n$ の場合、
 $ltau = \text{numroc}(ja + \min(m, n) - 1, nb_a, MYCOL, csrc_a, NPCOL)$
 $lwf = nb_a * (mpa0 + nga0 + nb_a)$
 $lws = \max((nb_a * (nb_a - 1)) / 2, (nrhsqb0 + mpb0) * nb_a) + nb_a * nb_a$
それ以外の場合、
 $ltau = \text{numroc}(ia + \min(m, n) - 1, mb_a, MYROW, rsrc_a, NPROW)$
 $lwf = mb_a * (mpa0 + nga0 + mb_a)$
 $lws = \max((mb_a * (mb_a - 1)) / 2, (npb0 + \max(nga0 + \text{numroc}(\text{numroc}(n + iroffb, mb_a, 0, 0, NPROW), mb_a, 0, 0, lcm), nrhsqb0)) * mb_a) + mb_a * mb_a$
である。
ここで、 $lcmp = lcm / NPROW$ で $lcm = \text{ilcm}(NPROW, NPCOL)$ 、

$iroffa = \text{mod}(ia - 1, mb_a)$ 、
 $icoffa = \text{mod}(ja - 1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, MYROW, rsrc_a, NPROW)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, MYROW, rsrc_a, NPROW)$ 、
 $mpa0 = \text{numroc}(m + iroffa, mb_a, MYROW, iarow, NPROW)$ 、
 $nga0 = \text{numroc}(n + icoffa, nb_a, MYCOL, iacol, NPCOL)$ 、
 $iroffb = \text{mod}(ib - 1, mb_b)$ 、
 $icoffb = \text{mod}(jb - 1, nb_b)$ 、
 $ibrow = \text{indxg2p}(ib, mb_b, MYROW, rsrc_b, NPROW)$ 、
 $ibcol = \text{indxg2p}(jb, nb_b, MYCOL, csrc_b, NPCOL)$ 、
 $mpb0 = \text{numroc}(m + iroffb, mb_b, MYROW, icrow, NPROW)$ 、
 $nqb0 = \text{numroc}(n + icoffb, nb_b, MYCOL, ibcol, NPCOL)$ 、

ilcm 、 indxg2p および numroc は、ScaLAPACK ツール関数である。
MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a 終了時に、 $m \geq n$ の場合、 $\text{sub}(A)$ は QR 因子分解の詳細 ([p?qgeqrf](#) から返される値) によって上書きされる。 $m < n$ の場合、 $\text{sub}(A)$ は LQ 因子分解の詳細 ([p?qelqf](#) から返される値) によって上書きされる。
- b 終了時に、 $\text{sub}(B)$ は解ベクトルによって上書きされ、列方向に格納される。 $trans = 'N'$ かつ $m \geq n$ の場合、 $\text{sub}(B)$ の 1 から n 行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの

	列の $n+1$ から m の成分の二乗和で与えられる。 $trans = 'N'$ かつ $m < n$ の場合、 $sub(B)$ の 1 から n 行に、最小ノルム解のベクトルが格納される。 $trans = 'T'$ かつ $m \geq n$ の場合、 $sub(B)$ の 1 から m 行に、最小ノルム解のベクトルが格納される。 $trans = 'T'$ かつ $m < n$ の場合、 $sub(B)$ の 1 から m 行に、最小二乗解のベクトルが格納される。各列の解の二乗和の誤差は、それぞれの列の $m+1$ から n の成分の二乗和で与えられる。
<code>work(1)</code>	終了時に、 <code>work(1)</code> には、最適なパフォーマンスを得るために必要な <code>lwork</code> の最小値が格納される。
<code>info</code>	(グローバル) INTEGER。 <code>info = 0</code> の場合、実行は正常に終了したことを示す。 <code>info < 0</code> の場合: i 番目の引数が配列で、 j 番目の値が不正だった場合、 $info = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $info = -i$ 。

p?syev

対称対角行列の固有値と固有ベクトルを選択的に計算する。

構文

```
call pssyev( jobz, uplo, n, a, ia, ja, desca, w, z, iz, jz, descz, work,
            lwork, info )
call pdsyev( jobz, uplo, n, a, ia, ja, desca, w, z, iz, jz, descz, work,
            lwork, info )
```

説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、実対称行列 A のすべての固有値と (オプションで) 固有ベクトルを計算する。現在の形式では、ルーチンは、均一な方程式を仮定しており、異なるプロセスにおける固有値や固有ベクトルの整合性をチェックしない。このため、均一でない方程式は、エラーメッセージを出力せずに、正しくない値を返す可能性がある。

入力パラメーター

`np` = 与えられたプロセスに対するローカルの行数。

`nq` = 与えられたプロセスに対するローカルの列数。

<code>jobz</code>	(グローバル) CHARACTER。'N' または 'V' でなければならない。 固有ベクトルを計算する場合に指定する。 <code>jobz = 'N'</code> の場合、固有値のみを計算する。 <code>jobz = 'V'</code> の場合、固有値と固有ベクトルを計算する。
<code>uplo</code>	(グローバル) CHARACTER。'U' または 'L' でなければならない。 対称行列 A の上三角部分または下三角部分のどちらが格納されるかを指定する。

uplo = 'U' の場合、*a* には *A* の上三角部分が格納される。
uplo = 'L' の場合、*a* には *A* の下三角部分が格納される。

n (グローバル) INTEGER。行列 *A* ($n \geq 0$) の行数と列数。

a (ローカル)
 REAL (pssyev の場合)
 DOUBLE PRECISION (pdsyev の場合)
 グローバル次元 (n, n) とローカル次元 ($lld_a, LOCC(ja+n-1)$) のブ
 ロック・サイクリック配列。対称行列 *A* を格納する。*uplo* = 'U' の場
 合、行列 *A* の上三角部分だけが対称行列の成分を定義するために使用
 される。*uplo* = 'L' の場合、行列 *A* の下三角部分だけが対称行列の成分
 を定義するために使用される。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 *A* の最初の行と最初の列
 を示す、グローバル配列 *a* の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen*<sub>_)。分散行
 列 *A* の配列ディスクリプター。</sub>

iz, jz (グローバル) INTEGER。それぞれ、部分行列 *Z* の最初の行と最初の列
 を示す、グローバル配列 *z* の行インデックスと列インデックス。

descz (グローバルおよびローカル) INTEGER 配列、次元は (*dlen*<sub>_)。分散行
 列 *Z* の配列ディスクリプター。</sub>

work (ローカル)
 REAL (pssyev の場合)
 DOUBLE PRECISION (pdsyev の場合) 配列、次元は (*lwork*)。

lwork (ローカル)
 INTEGER。 *lwork* の定義に使用する変数の定義は以下を参照。
 固有ベクトルが要求されなかった場合 (*jobz* = 'N')、
 $lwork \geq 5 * n + sizesytrd + 1$
 ここで、*sizesytrd* は、[p?sytrd](#) のワークスペース要件で
 $\max(NB * (np + 1), 3 * NB)$ 。
 固有ベクトルが要求された場合 (*jobz* = 'V')、すべての固有ベクトルが
 計算されることを保証するために必要なワークスペースの量は次のと
 おりである。
 $qrmem = 2 * n - 2$
 $lwmin = 5 * n + n * ldc + \max(sizemqrleft, qrmem) + 1$

変数定義：
 $NB = desca(mb_)= desca(nb_)= * descz(mb_)= descz(nb_)$
 $nn = \max(n, NB, 2)$
 $desca(rsrc_)= desca(rsrc_)= descz(rsrc_)= * descz(csrc_)= 0$
 $np = \text{numroc}(nn, NB, 0, 0, NPROW)$
 $nq = \text{numroc}(\max(n, NB, 2), NB, 0, 0, NPCOL)$
 $nrc = \text{numroc}(n, NB, myprowc, 0, NPROCS)$
 $ldc = \max(1, nrc)$
sizemqrleft は、その *side* 引数が 'L' のときの *p?ormtr* のワークス
 ペース要件。
 定義済みの *myprowc* を使用して、新しいコンテキストは次のようにし
 て作成される。

```
call blacs_get(desca(ctxt_), 0, contextc) call
blacs_gridinit(contextc, 'R', NPROCS, 1) call
blacs_gridinfo(contextc, nprowc, npcclc, myprowc, mypcclc)
```

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、行列 A の下三角部分 ($uplo = 'L'$ の場合) または上三角部分 ($uplo = 'U'$ の場合) が、対角成分を含めて削除される。
- w** (グローバル)。
REAL (pssyev の場合)
DOUBLE PRECISION (pdsyev の場合)
配列、次元は (n) 。
正常終了時に、先頭の m 個の成分には、指定された固有値が昇順で格納される。
- z** (ローカル)。
REAL (pssyev の場合)
DOUBLE PRECISION (pdsyev の場合)
グローバル次元 (n, n) 、ローカル次元 $(lld_z, LOCC(jz+n-1))$ 。
 $jobz = 'V'$ の場合、正常終了時に、 z の先頭の m 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。 $jobz = 'N'$ の場合、 z は参照されない。
- work(1)** 終了時に、 $work(1)$ は計算の完了を保證するために必要なワークスペースを返す。入力パラメーターが正しくない場合、 $work(1)$ も正しくならない。
 $jobz = 'N'$ の場合、 $work(1)$ はワークスペースの最小 (最適) 値。
 $jobz = 'V'$ の場合、 $work(1)$ はすべての固有ベクトルを生成するために必要な最小ワークスペース。
- info** (グローバル)
INTEGER。
 $info = 0$ の場合、正常に終了したことを示す。
 $info < 0$ の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$ 、
 i 番目の引数がスカラーで値が不正ならば $info = -i$ 。
 $info > 0$ の場合、
 $info = 1 \sim n$ の場合、 i 番目の固有値は合計 $30n$ 回の反復で `?stegr2` に収束しなかった。
 $info = n+1$ の場合、`p?syev` は固有値がプロセスグリッドで同一でなかったことから不均一性を検出した。この場合、`p?syev` の結果は保証されない。

p?syevx

対称行列の固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

```
call pssyevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, iwork, liwork,
             ifail, iclustr, gap, info)
call pdsyevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
             abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, iwork, liwork,
             ifail, iclustr, gap, info)
```

説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、実対称行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

np = 与えられたプロセスに対するローカルの行数。

nq = 与えられたプロセスに対するローカルの列数。

jobz (グローバル) CHARACTER*1. 'N' または 'V' でなければならない。
固有ベクトルを計算する場合に指定する。
 $jobz = 'N'$ の場合、固有値のみを計算する。
 $jobz = 'V'$ の場合、固有値と固有ベクトルを計算する。

range (グローバル) CHARACTER*1. 'A'、'V'、または 'I' でなければならない。
 $range = 'A'$ の場合、固有値をすべて計算する。
 $range = 'V'$ の場合、半開区間
[vl, vu] の固有値をすべて計算する。
 $range = 'I'$ の場合、インデックスが $il \sim iu$ の固有値を計算する。

uplo (グローバル) CHARACTER*1. 'U' または 'L' でなければならない。
対称行列 A の上三角部分または下三角部分のどちらが格納されるかを指定する。
 $uplo = 'U'$ の場合、 a には A の上三角部分が格納される。
 $uplo = 'L'$ の場合、 a には A の下三角部分が格納される。

n (グローバル) INTEGER。行列 A ($n \geq 0$) の行数と列数。

a (ローカル)。
REAL (pssyevx の場合)
DOUBLE PRECISION (pdsyevx の場合)
グローバル次元 (n, n) とローカル次元 ($lld_a, LOCC(ja+n-1)$) のブロック・サイクリック配列。対称行列 A を格納する。 $uplo = 'U'$ の場

合、行列 A の上三角部分だけが対称行列の成分を定義するために使用される。 $uplo = 'L'$ の場合、行列 A の下三角部分だけが対称行列の成分を定義するために使用される。

- ia, ja* (グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
- desca* (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
- vl, vu* (グローバル)
REAL (pssyevx の場合)
DOUBLE PRECISION (pdsyevx の場合)
 $range = 'V'$ の場合、固有値を探す区間の下限値と上限値。 $vl \leq vu$ 。
 $range = 'A'$ または $'I'$ の場合は参照されない。
- il, iu* (グローバル)
INTEGER。 $range = 'I'$ の場合、求める固有値の最小インデックスと最大インデックス。
次の制約がある。
 $il \geq 1$
 $\min(il, n) \leq iu \leq n$
 $range = 'A'$ または $'V'$ の場合は参照されない。
- abstol* (グローバル)。
REAL (pssyevx の場合)
DOUBLE PRECISION (pdsyevx の場合)
 $jobz = 'V'$ の場合、 $abstol$ を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。
- 固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (eps はマシン精度)、近似固有値は、収束したものとして受け入れられる。
 $abstol + eps * \max(|a|, |b|)$
ここで、 eps はマシン精度である。 $abstol$ がゼロまたは負の場合、代わりに $eps * \text{norm}(T)$ を使用する。ここで、 $\text{norm}(T)$ は、 A を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。
- 固有値が最も正確に計算されるのは、 $abstol$ がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$ に設定されたときである。このルーチンで $((\text{mod}(info, 2).ne.0).or.*(\text{mod}(info/8, 2).ne.0))$ が返された場合、固有値または固有ベクトルのいくつかは収束に失敗したことを意味するので、 $abstol$ を $2 * p?lamch('S')$ に設定して、やり直してみる。
- orfac* (グローバル)。
REAL (pssyevx の場合)
DOUBLE PRECISION (pdsyevx の場合)
再直交化される固有ベクトルを指定する。互いの $tol = orfac * \text{norm}(A)$ 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 ($lwork$ を参照)、 tol は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。 $orfac$ がゼロの場合、再直交化は行われず。 $orfac$ が負の場合、デフォルト値の 10^3 が使用される。 $orfac$ はすべての処理で同一でなければならない。

<i>iz, jz</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散配列 <i>Z</i> のディスクリプター。 <i>descz(ctxt_)</i> は <i>desca(ctxt_)</i> と等しくなければならない。
<i>work</i>	(ローカル) REAL (<i>pssyevx</i> の場合) DOUBLE PRECISION (<i>pdsyevx</i> の場合) 配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。 <i>lwork</i> の定義に使用する変数の定義は以下を参照。 固有ベクトルが要求されなかった場合 (<i>jobz</i> = 'N')、 $lwork \geq 5 * n + \max(5 * nn, NB * (np0 + 1))$ 。 固有ベクトルが要求された場合 (<i>jobz</i> = 'V')、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。 $lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, \text{NPROW} * \text{NPCOL}) * nn$ 最小のワークスペースが与えられ、 <i>orfac</i> が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を <i>lwork</i> に追加する。 $(clustersize - 1) * n$ ここで、 <i>clustersize</i> は、クラスターが近い固有値のセットとして定義された、最大クラスターの固有値の数。 $\{w(k), \dots, w(k + clustersize - 1) \mid w(j + 1) \leq w(j) + orfac * 2 * \text{norm}(A)\}$ 変数定義: <i>neig</i> = 要求された固有ベクトルの数 $NB = \text{desca}(mb_) = \text{desca}(nb_) = \text{descz}(mb_) = \text{descz}(nb_)$ $nn = \max(n, NB, 2)$ $\text{desca}(rsrc_) = \text{desca}(nb_) = \text{descz}(rsrc_) = \text{descz}(csrc_) = 0$ $np0 = \text{numroc}(nn, NB, 0, 0, \text{NPROW})$ $mq0 = \text{numroc}(\max(neig, NB, 2), NB, 0, 0, \text{NPCOL})$ <i>iceil(x, y)</i> は $\text{ceiling}(x/y)$ を返す ScaLAPACK 関数。 <i>lwork</i> が小さすぎる場合: <i>lwork</i> が直交性を保証するには小さすぎる場合、 <i>p?syevx</i> は、固有値間の間隔が最も小さいクラスターで直交性を維持するように試みる。 <i>lwork</i> が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、 <i>info</i> = -23 が返される。 <i>range</i> = 'V' の場合、 <i>p?syevx</i> は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、 <i>range</i> = 'V' で <i>lwork</i> が十分大きく <i>p?syevx</i> で固有値が計算可能な場合、 <i>p?syevx</i> は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係：
適切なワークスペースが提供されれば、より優れたパフォーマンスを達成できる。逆に、状況によっては、以下に示すワークスペースよりも多くのワークスペースが提供されると、パフォーマンスが低下することがある。

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$$lwork \geq \max(lwork, 5 * n + nsytrd_lwopt)$$

ここで、

$lwork$ は、上で定義され、要求された固有ベクトルの数に依存する。

また、

$$nsytrd_lwopt = n + 2 * (anb + 1) * (4 * nps + 2) + (nps + 3) * nps$$

$$anb = p\text{?}jlaenv(\text{desca}(\text{ctxt}_), 3, 'p\text{?}sytrd', 'L', 0, 0, 0, 0)$$

$$sqnpc = \text{int}(\text{sqrt}(\text{dble}(\text{NPROW} * \text{NPCOL})))$$

$$nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2 * anb)$$

numroc は ScaLAPACK ツール関数である。

$p\text{?}jlaenv$ は ScaLAPACK 環境問い合わせ関数である MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して求められる。

n が大きな場合でも追加のワークスペースは必要ないが、 n が小さな場合にパフォーマンスが最も上昇するため、追加のワークスペース (プロセスあたり 1MB 未満) を提供するほうがよい。

$clustersize \geq n / \text{sqrt}(\text{NPROW} * \text{NPCOL})$ の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n - 1$) では、 $p\text{?}stein$ は 1 プロセッサ上の $\text{?}stein$ よりもパフォーマンスが高くなることはない。

$clustersize = n / \text{sqrt}(\text{NPROW} * \text{NPCOL})$ の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n / \text{sqrt}(\text{NPROW} * \text{NPCOL})$ の場合、実行時間はクラスターサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する `work` 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

`iwork` (ローカル) INTEGER。ワークスペース配列。

`liwork` (ローカル) INTEGER。`iwork` の次元。

$$liwork \geq 6 * nnp$$

ここで、 $nnp = \max(n, \text{NPROW} * \text{NPCOL} + 1, 4)$

$liwork = -1$ の場合、 $liwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての `work` 配列の最小かつ最適なサイズだけを計算する。各値は該当する `work` 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、行列 <i>A</i> の下三角部分 (<i>uplo</i> = 'L' の場合) または上三角部分 (<i>uplo</i> = 'U' の場合) が、対角成分を含めて上書きされる。
<i>m</i>	(グローバル) INTEGER。求められた固有値の個数 ($0 \leq m \leq n$)。
<i>nz</i>	(グローバル) INTEGER。計算された固有ベクトルの総数 ($0 \leq nz \leq m$)。 <i>z</i> の列数。 <i>jobz</i> .ne. 'V' の場合、 <i>nz</i> は参照されない。 <i>jobz</i> .eq. 'V' の場合、 <i>nz</i> = <i>m</i> (ユーザーが十分な空間を提供しないで、 <i>p?syevx</i> が計算を開始する前にこれを検出できない場合を除く)。要求されたすべての固有ベクトルを得るには、ユーザーは <i>z</i> (<i>m.le.descz(n_)</i>) の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。(<i>lwork</i> を参照)。 <i>p?syevx</i> は、 <i>range</i> .eq. 'V' の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。
<i>w</i>	(グローバル)。 REAL (<i>pssyevx</i> の場合) DOUBLE PRECISION (<i>pdsyevx</i> の場合) 配列、次元は (<i>n</i>)。 <i>w</i> の先頭から <i>m</i> 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	(ローカル)。 REAL (<i>pssyevx</i> の場合) DOUBLE PRECISION (<i>pdsyevx</i> の場合) 配列、グローバル次元 (<i>n, n</i>)、 ローカル次元 (<i>lld_z, LOCc(jz+n-1)</i>)。 <i>jobz</i> = 'V' の場合、正常終了時に、 <i>z</i> の先頭の <i>m</i> 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 <i>z</i> は参照されない。
<i>work</i> (1)	終了時に、最適なパフォーマンスを得るために適切なワークスペースを返す。
<i>iwork</i> (1)	終了時に、 <i>iwork</i> (1) には、要求された整数ワークスペースの量が格納される。
<i>ifail</i>	(グローバル) INTEGER。配列、次元は (<i>n</i>)。 <i>jobz</i> = 'V' の場合、正常終了時に、 <i>ifail</i> の先頭の <i>m</i> 個の成分はゼロである。終了時に、(<i>mod</i> (<i>info</i> ,2).ne.0) の場合、 <i>ifail</i> には収束に失敗した固有ベクトルのインデックスが格納される。 <i>jobz</i> = 'N' の場合、 <i>ifail</i> は参照されない。
<i>iclustr</i>	(グローバル) INTEGER。 配列、次元は ($2 * \text{NPROW} * \text{NPCOL}$)。 この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスターに対する固有ベクトルのインデックスが格納される (<i>lwork</i> , <i>orfac</i> , および <i>info</i> を参照)。インデックス <i>iclustr</i> ($2 * i - 1$) から <i>iclustr</i> ($2 * i$) の固有値のクラスターに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。

したがって、これらのクラスターに対応する固有ベクトルは直交しない。*iclustr()* は、ゼロで終了する配列である。

(*iclustr(2*k).ne.0.and.iclustr(2*k+1).eq.0*) (*k* がクラスター数の場合)。

jobz = 'N' の場合、*iclustr* は参照されない。

gap

(グローバル)

REAL (*pssyevx* の場合)

DOUBLE PRECISION (*pdsyevx* の場合)

配列、次元は (NPROW*NPCOL)。

この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 *iclustr* によって示されたクラスターに対応する。その結果、*i* 番目のクラスターに対応する固有ベクトルのドット積は、 $(C * n) / gap(i)$ と同程度になる。ここで、*C* は小さな定数である。

info

(グローバル) INTEGER。

info = 0 の場合、正常に終了したことを示す。

info < 0 の場合、

i 番目の引数が配列で *j* 番目の成分の値が不正ならば

info = -(*i**100+*j*)、*i* 番目の引数がスカラーで値が不正ならば

info = -*i*。

info > 0 の場合 : (mod(*info*,2).ne.0) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。abstol=2.0*p?lamch('U') でなければならない。

(mod(*info*/2,2).ne.0) の場合、1 つ以上の固有値のクラスターに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスターのインデックスは、配列 *iclustr* に格納される。

(mod(*info*/4,2).ne.0) の場合、空間の制限により、*p?syevx* で *vl* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。

(mod(*info*/8,2).ne.0) の場合、*p?stebz* は、固有値の計算に失敗したことを示す。abstol=2.0*p?lamch('U') でなければならない。

p?heevx

エルミート行列の固有値と (オプションで)
固有ベクトルを選択的に計算する。

構文

```
call pcheevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
            abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, rwork, lrwork,
            iwork, liwork, ifail, iclustr, gap, info)
```

```
call pzheevx(jobz, range, uplo, n, a, ia, ja, desca, vl, vu, il, iu,
            abstol, m, nz, w, orfac, z, iz, jz, descz, work, lwork, rwork, lrwork,
            iwork, liwork, ifail, iclustr, gap, info)
```

説明

このルーチンは、ScaLAPACK ルーチンの推奨される組み合わせの呼び出しによって、複素エルミート行列 A の固有値と (オプションで) 固有ベクトルを選択的に計算する。固有値と固有ベクトルを選択するには、希望する固有値の値の範囲かインデックスの範囲を指定する。

入力パラメーター

np = 与えられたプロセスに対するローカルの行数。

nq = 与えられたプロセスに対するローカルの列数。

$jobz$	(グローバル) CHARACTER*1. 'N' または 'V' でなければならない。固有ベクトルを計算する場合に指定する。 $jobz = 'N'$ の場合、固有値のみを計算する。 $jobz = 'V'$ の場合、固有値と固有ベクトルを計算する。
$range$	(グローバル) CHARACTER*1. 'A'、'V'、または 'I' でなければならない。 $range = 'A'$ の場合、固有値をすべて計算する。 $range = 'V'$ の場合、半開区間 $[v_l, v_u]$ の固有値をすべて計算する。 $range = 'I'$ の場合、インデックスが $i_l \sim i_u$ の固有値を計算する。
$uplo$	(グローバル) CHARACTER*1. 'U' または 'L' でなければならない。エルミート行列 A の上三角部分と下三角部分のどちらが格納されるかを指定する。 $uplo = 'U'$ の場合、 a には A の上三角部分が格納される。 $uplo = 'L'$ の場合、 a には A の下三角部分が格納される。
n	(グローバル) INTEGER。行列 A ($n \geq 0$) の行数と列数。
a	(ローカル)。 COMPLEX (pcheevx の場合) DOUBLE COMPLEX (pzheevx の場合) グローバル次元 (n, n) とローカル次元 ($lld_a, LOCC(ja+n-1)$) のブロック・サイクリック配列。エルミート行列 A を格納する。 $uplo = 'U'$ の場合、行列 A の上三角部分だけが対称行列の成分を定義するために使用される。 $uplo = 'L'$ の場合、行列 A の下三角部分だけがエルミート行列の成分を定義するために使用される。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。 $desca(ctxt_)$ が正しくない場合、p?heevx での正しいエラー報告は保証されない。
v_l, v_u	(グローバル) REAL (pcheevx の場合) DOUBLE PRECISION (pzheevx の場合) $range = 'V'$ の場合、固有値を探す区間の下限値と上限値。 $range = 'A'$ または 'I' の場合は参照されない。

<i>il, iu</i>	(グローバル) INTEGER。range='I' の場合、求める固有値の最小インデックスと最大インデックス。 次の制約がある。 $il \geq 1$ $\min(il, n) \leq iu \leq n$ range='A' または 'V' の場合は参照されない。
<i>abstol</i>	(グローバル)。 REAL (pcheevx の場合) DOUBLE PRECISION (pzheevx の場合) jobz='V' の場合、 <i>abstol</i> を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。 固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (<i>eps</i> はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol + eps * \max(a , b)$ ここで、 <i>eps</i> はマシン精度である。 <i>abstol</i> がゼロまたは負の場合、代わりに $eps * \text{norm}(T)$ を使用する。ここで、 $\text{norm}(T)$ は、 <i>A</i> を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。 固有値が最も正確に計算されるのは、 <i>abstol</i> がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$ に設定されたときである。このルーチンで $((\text{mod}(info, 2), \text{ne.0}).\text{or}(\text{mod}(info/8, 2), \text{ne.0}))$ が返された場合、固有値または固有ベクトルのいくつかは収束に失敗したことを意味するので、 <i>abstol</i> を $2 * p?lamch('S')$ に設定して、やり直してみる。
<i>orfac</i>	(グローバル)。 REAL (pcheevx の場合) DOUBLE PRECISION (pzheevx の場合) 再直交化される固有ベクトルを指定する。互いの $tol = orfac * \text{norm}(A)$ 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、 <i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。 <i>orfac</i> がゼロの場合、再直交化は行われぬ。 <i>orfac</i> が負の場合、デフォルト値の 10^3 が使用される。 <i>orfac</i> はすべての処理で同一でなければならない。
<i>iz, jz</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散配列 <i>Z</i> のディスクリプター。 <i>descz(ctxt_)</i> は <i>desca(ctxt_)</i> と等しくなければならない。
<i>work</i>	(ローカル) COMPLEX (pcheevx の場合) DOUBLE COMPLEX (pzheevx の場合) 配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカル)。 INTEGER。配列 <i>work</i> の次元。 固有値のみが要求された場合：

$$lwork \geq n + \max(NB * (np0 + 1), 3)$$

固有ベクトルが要求された場合 :

$$lwork \geq n + (np0 + mq0 + NB) * NB$$

$$mq0 = \text{numroc}(nn, NB, 0, 0, NPCOL)$$

$$lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, NPROW * NPCOL) * nn$$

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$$lwork \geq \max(lwork, nhetr_lwork)$$

ここで、 $lwork$ は、上で定義され、

$$nhetr_lwork = n + 2 * (anb + 1) * (4 * nps + 2) + (nps + 1) * nps$$

$$ictxt = \text{desca}(ctxt_)$$

$$anb = \text{pjlaenv}(ictxt, 3, 'pchettrd', 'L', 0, 0, 0, 0)$$

$$sqnpc = \text{sqrt}(\text{dble}(NPROW * NPCOL))$$

$$nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2 * anb)$$

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

$rwork$

(ローカル)

REAL (pcheevx の場合)

DOUBLE PRECISION (pzheevx の場合)

ワークスペース配列、次元は ($lrwork$)。

$lrwork$

(ローカル)

INTEGER。配列 $work$ の次元。

$lwork$ の定義に使用する変数の定義は以下を参照。

固有ベクトルが要求されなかった場合 ($jobz = 'N'$)、

$$lrwork \geq 5 * nn + 4 * n。$$

固有ベクトルが要求された場合 ($jobz = 'V'$)、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。

$$lrwork \geq 4 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig, NPROW * NPCOL) * nn$$

最小のワークスペースが与えられ、 $orfac$ が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を $lwork$ に追加する。

$$(clustersize - 1) * n$$

ここで、 $clustersize$ は、クラスターが近い固有値のセットとして定義された、最大クラスターの固有値の数。

$$\{w(k), \dots, w(k + clustersize - 1) \mid$$

$$w(j + 1) \leq w(j) + orfac * 2 * \text{norm}(A)\}$$

変数定義:

$neig$ = 要求された固有ベクトルの数

$NB = \text{desca}(mb_)= \text{desca}(nb_)= \text{descz}(mb_)= \text{descz}(nb_)$

$nn = \max(n, NB, 2)$

$desca(rsrc_)=desca(nb_)=descz(rsrc_)=descz(csrc_)=0$

$np0 = numroc(nn, NB, 0, 0, NPROW)$

$mq0 = numroc(\max(neig, NB, 2), NB, 0, 0, NPCOL)$ $iceil(x, y)$ は (x/y) を返す ScaLAPACK 関数。

$lwork$ が小さすぎる場合:

$lwork$ が直交性を保証するには小さすぎる場合、 $p?heevx$ は、固有値間の間隔が最も小さいクラスターで直交性を維持するように試みる。 $lwork$ が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、 $info=-23$ が返される。 $range='v'$ の場合、 $p?heevx$ は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、 $range='v'$ で $lwork$ が十分大きく $p?heevx$ で固有値が計算可能な場合、 $p?heevx$ は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

$clustersize \geq n/\sqrt{NPROW*NPCOL}$ の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界(すなわち、 $clustersize = n-1$)では、[p?stein](#) は 1 プロセッサ上の [?stein](#) よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW*NPCOL}$ の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW*NPCOL}$ の場合、実行時間はクラスターサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

$iwork$ (ローカル) INTEGER。ワークスペース配列。

$liwork$ (ローカル) INTEGER。 $iwork$ の次元。

$liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW*NPCOL + 1, 4)$

$liwork = -1$ の場合、 $liwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a 終了時に、行列 A の下三角部分 ($uplo = 'L'$ の場合) または上三角部分 ($uplo = 'U'$ の場合) が、対角成分を含めて上書きされる。

m (グローバル) INTEGER。検出された固有値の総数 ($0 \leq m \leq n$)。

nz (グローバル) INTEGER。計算された固有ベクトルの総数 ($0 \leq nz \leq m$)。 z の列数。

$jobz.ne. 'v'$ の場合、 nz は参照されない。

$jobz.eq. 'v'$ の場合、 $nz = m$ (ユーザーが十分な空間を提供しないで、

p?heevx が計算を開始する前にこれを検出できない場合を除く)。要求されたすべての固有ベクトルを得るには、ユーザーは z ($m.le.descz(n_)$) の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。(lwork を参照)。p?heevx は、range.eq. 'V' の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。

<i>w</i>	(グローバル)。 REAL (pcheevx の場合) DOUBLE PRECISION (pzheevx の場合) 配列、次元は (n)。 <i>w</i> の先頭から m 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	(ローカル)。 COMPLEX (pcheevx の場合) DOUBLE COMPLEX (pzheevx の場合) 配列、グローバル次元 (n, n)、 ローカル次元 ($lld_z, LOCC(jz+n-1)$)。 <i>jobz</i> ='V' の場合、正常終了時に、 <i>z</i> の先頭の m 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束できない場合、 <i>z</i> のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> ='N' の場合、 <i>z</i> は参照されない。
<i>work(1)</i>	終了時に、最適なパフォーマンスを得るために適切なワークスペースを返す。
<i>rwork</i>	(ローカル)。 REAL (pcheevx の場合) DOUBLE PRECISION (pzheevx の場合) 配列、次元は (<i>lwork</i>)。 終了時に、 <i>rwork(1)</i> には、効率的に実行するために必要なワークスペースの最適な量が格納される。 <i>jobz</i> ='N' の場合、 <i>rwork(1)</i> には、固有値を効率的に計算するために必要なワークスペースの最適な量が格納される。 <i>jobz</i> ='V' の場合、 <i>rwork(1)</i> には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要なワークスペースの最適な量が格納される。 <i>range</i> ='V' の場合、すべての固有ベクトルが必要であるとみなされる。
<i>iwork(1)</i>	(ローカル) 終了時に、 <i>iwork(1)</i> には、要求された整数ワークスペースの量が格納される。
<i>ifail</i>	(グローバル) INTEGER。 配列、次元は (n)。 <i>jobz</i> ='V' の場合、正常終了時に、 <i>ifail</i> の先頭の m 個の成分はゼロである。終了時に、($\text{mod}(\text{info},2).\text{ne}.0$) の場合、 <i>ifail</i> には収束に失敗した固有ベクトルのインデックスが格納される。 <i>jobz</i> ='N' の場合、 <i>ifail</i> は参照されない。

- iclustr* (グローバル) INTEGER。
 配列、次元は (2*NPROW*NPCOL)。
 この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスターに対する固有ベクトルのインデックスが格納される (*lwork*, *orfac*, および *info* を参照)。インデックス *iclustr*(2*i-1) から *iclustr*(2*i) の固有値のクラスターに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスターに対応する固有ベクトルは直交しない。*iclustr*() は、ゼロで終了する配列である。
 (*iclustr*(2*k).ne.0.and.*iclustr*(2*k+1).eq.0) (*k* がクラスター数の場合)。
jobz = 'N' の場合、*iclustr* は参照されない。
- gap* (グローバル)
 REAL (*pcheevx* の場合)
 DOUBLE PRECISION (*pzheevx* の場合)
 配列、次元は (NPROW*NPCOL)。
 この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 *iclustr* によって示されたクラスターに対応する。その結果、*i* 番目のクラスターに対応する固有ベクトルのドット積は、 $(C * n) / gap(i)$ と同程度になる。ここで、*C* は小さな定数である。
- info* (グローバル) INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、
i 番目の引数が配列で *j* 番目の成分の値が不正ならば
info = -(*i**100+*j*)、*i* 番目の引数がスカラーで値が不正ならば
info = -*i*。
info > 0 の場合、
(mod(*info*,2).ne.0) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。
abstol=2.0*p?lamch('U') でなければならない。
(mod(*info*/2,2).ne.0) の場合、1 つ以上の固有値のクラスターに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスターのインデックスは、配列 *iclustr* に格納される。
(mod(*info*/4,2).ne.0) の場合、空間の制限により、[p?syevx](#) で *v1* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。
(mod(*info*/8,2).ne.0) の場合、[p?stebz](#) は、固有値の計算に失敗したことを示す。*abstol*=2.0*p?lamch('U') でなければならない。

p?gesvd

一般行列の特異値分解と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。

構文

```
call psgesvd( jobu, jobvt, m, n, a, ia, ja, desca, s, u, iu, ju, descu,
              vt, ivt, jvt, descvt, work, lwork, info )
call pdgesvd( jobu, jobvt, m, n, a, ia, ja, desca, s, u, iu, ju, descu,
              vt, ivt, jvt, descvt, work, lwork, info )
```

説明

このルーチンは、行列 A ($m \times n$) の特異値分解 (SVD) と (オプションで) 左特異ベクトルまたは右特異ベクトルを計算する。SVD は、次のように記述される。

$$A = U \Sigma V^T$$

ここで、 Σ はその $\min(m, n)$ 対角成分を除いてゼロの $m \times n$ の行列、 U は $m \times m$ の直交行列、 V は $n \times n$ の直交行列である。 Σ の対角成分は A の特異値で、 U と V の列は、それぞれ、右と左の特異ベクトルに対応する。特異値は s に降順で返され、 U の先頭から $\min(m, n)$ の列と $vt = V^T$ の行のみが計算される。

入力パラメーター

$mp = A$ と U のローカル行数

$nq = A$ と VT のローカル列数

$size = \min(m, n)$

$sizeq = U$ のローカル列数

$sizep = VT$ のローカル行数

jobu (グローバル) CHARACTER*1。
行列 U のすべて、または一部を計算するオプションを指定する。

 $jobu = 'V'$ の場合、 U の先頭から $size$ 列 (左特異ベクトル) が配列 u に返される。
 $jobu = 'N'$ の場合、 U の列 (左特異ベクトル) は計算されない。

jobvt (グローバル) CHARACTER*1。
行列 V^T のすべて、または一部を計算するオプションを指定する。

 $jobvt = 'V'$ の場合、 V^T の先頭から $size$ 行 (右特異ベクトル) が配列 vt に返される。
 $jobvt = 'N'$ の場合、 V^T の行 (右特異ベクトル) は計算されない。

m (グローバル) INTEGER。行列 A の行数 ($m \geq 0$)。

n (グローバル) INTEGER。行列 A の列数 ($n \geq 0$)。

a (ローカル)。
DOUBLE PRECISION (psgesvd と pdgesvd の場合)
ブロック・サイクリック配列、グローバル次元 (m, n)、ローカル次元 (mp, nq)。

 $work(lwork)$ は、ワークスペース配列である。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 <i>A</i> の配列ディスクリプター。
<i>iu, ju</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>U</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>descu</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 <i>U</i> の配列ディスクリプター。
<i>ivt, jvt</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>VT</i> の最初の行と最初の列を示す、グローバル配列 <i>vt</i> の行インデックスと列インデックス。
<i>descvt</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 <i>VT</i> の配列ディスクリプター。
<i>work</i>	(ローカル) DOUBLE PRECISION (<i>psgesvd</i> と <i>pdgesvd</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカル) INTEGER。配列 <i>work</i> の次元。

$$lwork > 2 + 6 * sizeb + \max(watobd, wbdtosvd)$$

ここで、*sizeb* = max(*m*, *n*)、*watobd* および *wbdtosvd* は、行列 *A* を二重対角形式にし、二重対角行列から特異値分解 *USVT* に移動するため、それぞれ必要なワークスペースを参照する。

watobd には、次の内容が含まれている。

$$watobd = \max(\max(wpslange, wpsgebrd), \max(wpslared2d, wpslared1d))$$

ここで、*wpslange*、*wpslared1d*、*wpslared2d*、*wpsgebrd* は、それぞれ、サブプログラム *pslange*、*pslared1d*、*pslared2d*、*psgebrd* に必要なワークスペースである。標準の表記を使用すると、次のようになる。

$$mp = \text{numroc}(m, mb, \text{MYROW}, \text{desca}(\text{ctxt}_l), \text{NPROW})$$

$$nq = \text{numroc}(n, NB, \text{MYCOL}, \text{desca}(\text{lld}_l), \text{NPCOL})$$

上記サブプログラムに必要なワークスペースは次のとおりである。

$$wpslange = mp$$

$$wpslared1d = nq0$$

$$wpslared2d = mp0$$

$$wpsgebrd = NB * (mp + nq + 1) + nq$$

ここで、*nq0* と *mp0* は、それぞれ、*MYCOL* = 0 と *MYROW* = 0 で得られた値を参照する。一般的に、ワークスペースの上限は、プロセッサ (0,0) で必要なワークスペースによって与えられる。

$$watobd \leq NB * (mp0 + nq0 + 1) + nq0$$

均一なプロセスグリッドの場合、この上限は各プロセッサの最小ワークスペースの推定として使用できる。

wbdtosvd には、次の内容が含まれている。

```
wbdtosvd = size*(wantu*nru + wantvt*ncvt) + max(wsbdsqr,
max(wantu*wpsormbrqln, wantvt*wpsormbrprt))
```

ここで、

$wantu(wantvt) = 1$ (左(右)特異ベクトルが必要な場合) および
 $wantu(wantvt) = 0$ (それ以外の場合)

$wsbdsqr$ 、 $wpsormbrqln$ および $wpsormbrprt$ は、それぞれ、サブプログラム [sbdsqr](#)、[p?ormbr](#)(qln)、および [p?ormbr](#)(prt) に必要なワークスペース。ここで、 qln と prt は [p?ormbr](#) への呼び出しで指定する、引数 $vect$ 、 $side$ 、および $trans$ の値。 nru は、プロセスの 1 次元の「列」を分散したときの、行列 U のローカル行数に等しい。同様に、 $ncvt$ は、プロセスの 1 次元の「行」を分散したときの、行列 VT のローカル列数に等しい。LAPACK プロシージャ [sbdsqr](#) を呼び出すには、各プロセッサで、

```
wsbdsqr = max(1, 2*size + (2*size - 4)* max(wantu, wantvt))
```

が必要である。最終的には、次のとおり。

```
wpsormbrqln = max((NB*(NB-1))/2,
(sizeq+mp)*NB)+NB*NB
wpsormbrprt = max((mb*(mb-1))/2,
(sizep+nq)*mb)+mb*mb
```

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンは $work$ 配列の最小サイズだけを計算する。必要なワークスペースは $work$ の最初の成分として返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a	終了時に、 a の内容は削除される。
s	(グローバル)。 DOUBLE PRECISION (psgesvd と pdgesvd の場合)。 配列、次元は ($size$)。 $s(i) \geq s(i+1)$ となるようにソートされて A の特異値が格納される。
u	(ローカル)。 DOUBLE PRECISION (psgesvd と pdgesvd の場合) ローカル次元 (mp , $sizeq$)、グローバル次元 (m , $size$)。 $jobu = 'v'$ の場合、 u は U の先頭の $\min(m,n)$ 列を格納する。 $jobu = 'N'$ または $'o'$ の場合、 u は参照されない。
vt	(ローカル)。 DOUBLE PRECISION (psgesvd と pdgesvd の場合) ローカル次元 ($sizep$, nq)、グローバル次元 ($size$, n)。 $jobvt = 'v'$ の場合、 VT は V^T の先頭の $size$ 行を格納する。 $jobu = 'N'$ の場合、 VT は参照されない。
$work$	終了時に、 $info = 0$ の場合、 $work(1)$ は、 $lwork$ の必要最小サイズを返す。

<i>rwork</i>	(複素数型の場合) 終了時に、 <i>info</i> > 0 の場合、 <i>rwork</i> (1:min(<i>m</i> , <i>n</i>)-1) には、対角が (必ずしもソートされていない) <i>s</i> 内にある上二重対角行列 <i>B</i> の非収束優対角成分が格納される。 <i>B</i> は、 $A = u * B * vt$ を満たすため、 <i>A</i> と同じ特異値を持ち、 <i>u</i> と <i>vt</i> で関連付けられた特異ベクトルを持つ。
<i>info</i>	(グローバル) INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> < 0 の場合: <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。 <i>info</i> > 0 の場合: <i>info</i> = <i>i</i> の場合、 <i>p?bdsqr</i> は収束しなかった。 <i>info</i> = min(<i>m</i> , <i>n</i>) + 1 の場合、 p?gesvd は固有値がプロセスグリッドで同一でなかったことから不均一性を検出した。この場合、 <i>p?gesvd</i> の結果は保証されない。

p?sygvx

実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルを選択的に計算する。

構文

```
call pssygvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
  descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
  work, lwork, iwork, liwork, ifail, iclustr, gap, info)
call pdsygvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
  descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
  work, lwork, iwork, liwork, ifail, iclustr, gap, info)
```

説明

このルーチンは、次の形式で示される実数の汎用対称固有値問題について、固有値と (オプションで) 固有ベクトルをすべて計算する。

$$\text{sub}(A)x = \lambda \text{sub}(B)x, \text{sub}(A) \text{sub}(B)x = \lambda x, \text{または} \text{sub}(B) \text{sub}(A)x = \lambda x.$$

ここで、 $A(ia:ia+n-1, ja:ja+n-1)$ を表す $\text{sub}(A)$ は対称とみなされ、 $B(ib:ib+n-1, jb:jb+n-1)$ を表す $\text{sub}(B)$ もまた、正定値とみなされる。

入力パラメーター

ibtype (グローバル) INTEGER。1、2、または3のいずれかでなければならない。
 解決する問題のタイプを指定する。
ibtype = 1 の場合、問題のタイプは $\text{sub}(A)x = \lambda \text{sub}(B)x$ である。
ibtype = 2 の場合、問題のタイプは $\text{sub}(A)\text{sub}(B)x = \lambda x$ である。
ibtype = 3 の場合、問題のタイプは $\text{sub}(B) \text{sub}(A)x = \lambda x$ である。

<i>jobz</i>	(グローバル) CHARACTER*1. 'N' または 'V' でなければならない。 <i>jobz</i> ='N' の場合、固有値のみを計算する。 <i>jobz</i> ='V' の場合、固有値と固有ベクトルを計算する。
<i>range</i>	(グローバル)。 CHARACTER*1. 'A'、'V'、または 'I' のいずれかでなければならない。 <i>range</i> ='A' の場合、固有値をすべて計算する。 <i>range</i> ='V' の場合、ルーチンは区間 [<i>vl</i> , <i>vu</i>] の固有値を計算する。 <i>range</i> ='I' の場合、ルーチンはインデックスが <i>il</i> ~ <i>iu</i> の固有値を計算する。
<i>uplo</i>	(グローバル)。 CHARACTER*1. 'U' または 'L' でなければならない。 <i>uplo</i> ='U' の場合、配列 <i>a</i> と <i>b</i> は、sub(<i>A</i>) と sub(<i>B</i>) の上三角を格納する。 <i>uplo</i> ='L' の場合、配列 <i>a</i> と <i>b</i> は sub(<i>A</i>) と sub(<i>B</i>) の下三角を格納する。
<i>n</i>	(グローバル)。 INTEGER。行列 sub(<i>A</i>) と sub(<i>B</i>) の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc</i> (<i>ja</i> + <i>n</i> -1)) の配列へのポインター。呼び出し時に、この配列は、 $n \times n$ の対称分散行列 sub(<i>A</i>) のローカル部分を含む。If <i>uplo</i> ='U' の場合、sub(<i>A</i>) の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 <i>uplo</i> ='L' の場合、sub(<i>A</i>) の先頭の $n \times n$ の下三角部分に行列の上三角部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca</i> (<i>ctxt_</i>) が正しくない場合、p?sygvx での正しいエラー報告は保証されない。
<i>b</i>	(ローカル)。 REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合) ローカルメモリーにある、次元 (<i>lld_b</i> , <i>LOCc</i> (<i>jb</i> + <i>n</i> -1)) の配列へのポインター。呼び出し時に、この配列は、 $n \times n$ の対称分散行列 sub(<i>B</i>) のローカル部分を含む。 <i>uplo</i> ='U' の場合、sub(<i>B</i>) の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。 <i>uplo</i> ='L' の場合、sub(<i>A</i>) の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。 <i>descb</i> (<i>ctxt_</i>) は <i>desca</i> (<i>ctxt_</i>) と等しくなければならない。

<i>vl, vu</i>	(グローバル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合) <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 <i>range</i> = 'A' または 'I' の場合、 <i>vl</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	(グローバル) INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $il \geq 1, \min(il, n) \leq iu \leq n$ <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。
<i>abstol</i>	(グローバル) REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合) <i>jobz</i> = 'V' の場合、 <i>abstol</i> を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。 固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (<i>eps</i> はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol + eps * \max(a , b)$ ここで、 <i>eps</i> はマシン精度である。 <i>abstol</i> がゼロまたは負の場合、代わりに $eps * \text{norm}(T)$ を使用する。ここで、 $\text{norm}(T)$ は、 <i>A</i> を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。 固有値が最も正確に計算されるのは、 <i>abstol</i> がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$ に設定されたときである。このルーチンで $((\text{mod}(info, 2).ne.0).or.*(\text{mod}(info/8, 2).ne.0))$ が返された場合、固有値または固有ベクトルのいくつかは収束に失敗したことを意味するので、 <i>abstol</i> を $2 * p?lamch('S')$ に設定して、やり直してみる。
<i>orfac</i>	(グローバル)。 REAL (pssygvx の場合) DOUBLE PRECISION (pdsygvx の場合) 再直交化される固有ベクトルを指定する。互いの $tol = orfac * \text{norm}(A)$ 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、 <i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。 <i>orfac</i> がゼロの場合、再直交化は行われない。 <i>orfac</i> が負の場合、デフォルト値の 10^{-3} が使用される。 <i>orfac</i> はすべての処理で同一でなければならない。
<i>iz, jz</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散配列 <i>Z</i> のディスクリプター。 <i>descz</i> (<i>ctxt_</i>) は <i>desca</i> (<i>ctxt_</i>) と等しくなければならない。

work (ローカル)
 REAL (pssygvx の場合)
 DOUBLE PRECISION (pdsygvx の場合)
 ワークスペース配列、次元は (*lwork*)。

lwork (ローカル)
 INTEGER。
lwork の定義に使用する変数の定義は以下を参照。
 固有ベクトルが要求されなかった場合 (*jobz* = 'N')、
 $lwork \geq 5 * n + \max(5 * nn, NB * (np0 + 1))$ 。
 固有ベクトルが要求された場合 (*jobz* = 'V')、すべての固有ベクトルが
 計算されることを保証するために必要なワークスペースの量は次のと
 おりである。
 $lwork \geq 5 * n + \max(5 * nn, np0 * mq0 + 2 * NB * NB) + \text{iceil}(neig,$
 $NPROW * NPCOL) * nn$

最小のワークスペースが与えられ、*orfac* が小さすぎる場合、計算さ
 れるベクトル固有値は直交ではない。(パフォーマンスが低下しても)
 直交性を保証するには、次の値を *lwork* に追加する。
 $(clustersize - 1) * n$
 ここで、*clustersize* は、クラスターが近い固有値のセットとして定
 義された、最大クラスターの固有値の数。
 $\{w(k), \dots, w(k + clustersize - 1) \mid$
 $w(j + 1) \leq w(j) + orfac * 2 * \text{norm}(A)\}$

変数定義：
neig = 要求された固有ベクトルの数
 $NB = \text{desca}(mb_) = \text{desca}(nb_) = \text{descz}(mb_) = \text{descz}(nb_)$
 $nn = \max(n, NB, 2)$
 $\text{desca}(rsrc_) = \text{desca}(nb_) = \text{descz}(rsrc_) = \text{descz}(csrc_) = 0$
 $np0 = \text{numroc}(nn, NB, 0, 0, NPROW)$
 $mq0 = \text{numroc}(\max(neig, NB, 2), NB, 0, 0, NPCOL)$ *iceil*(*x*, *y*) は (*x*/*y*) を返
 す ScaLAPACK 関数。

lwork が小さすぎる場合：
lwork が直交性を保証するには小さすぎる場合、*p?syevx* は、固有値
 間隔が最も小さいクラスターで直交性を維持するように試みる。
lwork が小さすぎて要求された固有値をすべて計算できない場合、計
 算は行われず、*info* = -23 が返される。*range* = 'V' の場合、*p?sygvx* は
 固有値が計算されるまで要求されている固有ベクトルがわからない点
 に注意する。したがって、*range* = 'V' で *lwork* が十分大きく *p?sygvx*
 で固有値が計算可能な場合、*p?sygvx* は固有値とできるだけ多くの固
 有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係：
 適切なワークスペースが提供されれば、より優れたパフォーマンスを
 達成できる。逆に、状況によっては、以下に示すワークスペースより
 も多くのワークスペースが提供されると、パフォーマンスが低下する
 ことがある。

最適なパフォーマンスを得るには、より大きなワークスペースが必要
 である。すなわち、
 $lwork \geq \max(lwork, 5 * n + \text{nsytrd_lwopt}, \text{nsygst_lwopt})$

ここで、

$lwork$ は、上で定義され、要求された固有ベクトルの数に依存する。

また、

$$nsytrd_lwopt = n + 2*(anb+1)*(4*nps+2) + (nps + 3) * nps$$

```
nsygst_lwopt = 2*np0*NB + nq0*NB + NB*NB
anb = p?jlaenv(desca(ctxt_), 3, p?syttrd', 'L', 0, 0, 0, 0)
sqnpc = int(sqrt(dble(NPROW * NPCOL)))
nps = max(numroc(n, 1, 0, 0, sqnpc), 2*anb)
NB = desca(mb_)
np0 = numroc(n, NB, 0, 0, NPROW)
nq0 = numroc(n, NB, 0, 0, NPCOL)
```

$numroc$ は ScaLAPACK ツール関数である。

$p?jlaenv$ は ScaLAPACK 環境問い合わせ関数である

$MYROW$ 、 $MYCOL$ 、 $NPROW$ 、および $NPCOL$ は、サブルーチン $blacs_gridinfo$ を呼び出して求められる。

n が大きな場合でも追加のワークスペースは必要ないが、 n が小さな場合にパフォーマンスが最も上昇するため、追加のワークスペース (プロセスあたり 1MB 未満) を提供するほうがよい。

$clustersize \geq n/\sqrt{NPROW*NPCOL}$ の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n-1$) では、 $p?stein$ は 1 プロセッサ上の $?stein$ よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW*NPCOL}$ の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW*NPCOL}$ の場合、実行時間はクラスターサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

$iwork$ (ローカル) INTEGER。ワークスペース配列。

$liwork$ (ローカル) INTEGER。 $iwork$ の次元。

$$liwork \geq 6 * nnp$$

ここで、

$$nnp = \max(n, NPROW*NPCOL + 1, 4)$$

$liwork = -1$ の場合、 $liwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a** 終了時に、`jobz = 'V'` で `info = 0` の場合、`sub(A)` には、固有ベクトルの分散行列 Z が格納される。固有ベクトルは、次のように正規化される。
`ibtype = 1` または `2` の場合、
 $Z^T * \text{sub}(B) * Z = I$ 。
`ibtype = 3` の場合、 $Z^T * \text{inv}(\text{sub}(B)) * Z = I$ 。
`jobz = 'N'` の場合、終了時に、`sub(A)` の上三角 (`uplo = 'U'` の場合) または下三角 (`uplo = 'L'` の場合) が対角を含めて破棄される。
- b** 終了時に、`info ≤ n` の場合、行列を格納している `sub(B)` の一部が、コレスキー因子分解 $\text{sub}(B) = U^T U$ または $\text{sub}(B) = L L^T$ からの三角係数 U または L で上書きされる。
- m** (グローバル)
 INTEGER。求められた固有値の個数 ($0 \leq m \leq n$)。
- nz** (グローバル)
 INTEGER。
 計算された固有ベクトルの総数 ($0 \leq nz \leq m$)。 z の列数。
`jobz.ne. 'V'` の場合、`nz` は参照されない。
`jobz.eq. 'V'` の場合、`nz = m` (ユーザーが十分な空間を提供しないで、`p?sygvx` が計算を開始する前にこれを検出できない場合を除く)。要求されたすべての固有ベクトルを得るには、ユーザーは z (`m.le.descz(n_)`) の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。(`lwork` を参照)。 `p?sygvx` は、`range.eq. 'V'` の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。
- w** (グローバル)
 REAL (`pssygvx` の場合)
 DOUBLE PRECISION (`pdsygvx` の場合)
 配列、次元は (n) 。
 正常終了時に、先頭の m 個の成分には、指定された固有値が昇順で格納される。
- z** (ローカル)。
 REAL (`pssygvx` の場合)
 DOUBLE PRECISION (`pdsygvx` の場合)
 グローバル次元 (n, n) 、ローカル次元 $(lld_z, LOCC(jz+n-1))$ 。
`jobz = 'V'` の場合、正常終了時に、 z の先頭の m 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束できない場合、 z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは `ifail` に返される。
`jobz = 'N'` の場合、 z は参照されない。
- work** `jobz = 'N'` の場合、`work(1)` には、固有値を効率的に計算するために必要な、ワークスペースの最適な量が格納される。
`jobz = 'V'` の場合、`work(1)` には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要なワークスペースの最適な量が格納される。
`range = 'V'` の場合、すべての固有ベクトルが必要であるとみなされる。

- ifail* (グローバル)
 INTEGER。
 配列、次元は (*n*)。
info.ne.0 の場合、*ifail* は追加情報を提供する
 ($\text{mod}(\text{info}/16,2).\text{ne.0}$) の場合、*ifail(1)* は、正定値でない最小の小行列式の次数を示す。終了時に、($\text{mod}(\text{info},2).\text{ne.0}$) の場合、*ifail* には収束に失敗した固有ベクトルのインデックスが格納される。
 上記のエラー条件がどちらもセットされてなく、かつ *jobz = 'V'* の場合、*ifail* の先頭から *m* 個の成分はゼロに設定される。
- iclustr* (グローバル)
 INTEGER。
 配列、次元は ($2*\text{NPROW}*\text{NPCOL}$)。この配列には、ワークスペースが十分でなかったために再直交できなかった、固有値のクラスターに対する固有ベクトルのインデックスが格納される (*lwork*、*orfac*、および *info* を参照)。インデックス *iclustr(2*i-1)* から *iclustr(2*i)* の固有値のクラスターに対応する固有ベクトルは、ワークスペースの不足により再直交できなかった。したがって、これらのクラスターに対応する固有ベクトルは直交しない。*iclustr()* は、ゼロで終了する配列である。($\text{iclustr}(2*k).\text{ne.0}.\text{and}.\text{iclustr}(2*k+1).\text{eq.0}$) (*k* がクラスター数の場合)。 *iclustr* は参照されない (*jobz = 'N'* の場合)。
- gap* (グローバル)
 REAL (*pssygvx* の場合)
 DOUBLE PRECISION (*pdsygvx* の場合)
 配列、次元は ($\text{NPROW}*\text{NPCOL}$)。
 この配列には、固有ベクトルが再直交できなかった固有値間のギャップが格納される。この配列の出力値は、配列 *iclustr* によって示されたクラスターに対応する。その結果、*i* 番目のクラスターに対応する固有ベクトルのドット積は、 $(C * n) / \text{gap}(i)$ と同程度になる。ここで、*C* は小さな定数である。
- info* (グローバル)
 INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合：*i* 番目の引数が配列で、*j* 番目の値が不正だった場合、*info = -(i*100+j)*。*i* 番目の引数がスカラーで値が不正だった場合、*info = -i*。
info > 0 の場合、
 ($\text{mod}(\text{info},2).\text{ne.0}$) の場合、1 つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは *ifail* に格納される。
 ($\text{mod}(\text{info},2,2).\text{ne.0}$) の場合、1 つ以上の固有値のクラスターに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかった。クラスターのインデックスは、配列 *iclustr* に格納される。
 ($\text{mod}(\text{info}/4,2).\text{ne.0}$) の場合、空間の制限により、*p?sygvx* で *v1* と *vu* 間のすべての固有ベクトルを計算できなかった。計算された固有ベクトルの個数は、*nz* に返される。
 ($\text{mod}(\text{info}/8,2).\text{ne.0}$) の場合、[p?stebz](#) は、固有値の計算に失敗したことを示す。
 ($\text{mod}(\text{info}/16,2).\text{ne.0}$) の場合、*B* は正定値ではなかったことを示す。*ifail(1)* は、正定値でない最小の小行列式の次数を示す。

p?hegvx

複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルを選択的に計算する。

構文

```
call pchegvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info)
call pzhegvx(ibtype, jobz, range, uplo, n, a, ia, ja, desca, b, ib, jb,
             descb, vl, vu, il, iu, abstol, m, nz, w, orfac, z, iz, jz, descz,
             work, lwork, rwork, lrwork, iwork, liwork, ifail, iclustr, gap, info)
```

説明

このルーチンは、次の形式で示される複素数の汎用エルミート固有値問題について、固有値と(オプションで)固有ベクトルをすべて計算する。

$$\text{sub}(A)x = \lambda \text{sub}(B)x, \text{sub}(A)\text{sub}(B)x = \lambda x, \text{または} \text{sub}(B)\text{sub}(A)x = \lambda x.$$

ここで、 $A(ia:ia+n-1, ja:ja+n-1)$ を表す $\text{sub}(A)$ と $\text{sub}(B)$ はエルミートとみなされ、 $B(ib:ib+n-1, jb:jb+n-1)$ を表す $\text{sub}(B)$ もまた、正定値とみなされる。

入力パラメーター

ibtype (グローバル) INTEGER。1、2、または3のいずれかでなければならない。
 解決する問題のタイプを指定する。
ibtype = 1 の場合、問題のタイプは $\text{sub}(A)x = \lambda \text{sub}(B)x$ である。
ibtype = 2 の場合、問題のタイプは $\text{sub}(A)\text{sub}(B)x = \lambda x$ である。
ibtype = 3 の場合、問題のタイプは $\text{sub}(B)\text{sub}(A)x = \lambda x$ である。

jobz (グローバル) CHARACTER*1。'N' または 'V' でなければならない。
jobz = 'N' の場合、固有値のみを計算する。
jobz = 'V' の場合、固有値と固有ベクトルを計算する。

range (グローバル) CHARACTER*1。'A'、'V'、または 'I' のいずれかでなければならない。
range = 'A' の場合、固有値をすべて計算する。
range = 'V' の場合、ルーチンは区間 [*vl*, *vu*] の固有値を計算する。
range = 'I' の場合、ルーチンはインデックスが *il* ~ *iu* の固有値を計算する。

uplo (グローバル) CHARACTER*1。'U' または 'L' でなければならない。
uplo = 'U' の場合、配列 *a* と *b* は、 $\text{sub}(A)$ と $\text{sub}(B)$ の上三角を格納す

	る。 <i>uplo</i> = 'L' の場合、配列 <i>a</i> と <i>b</i> は <i>sub(A)</i> と <i>sub(B)</i> の下三角を格納する。
<i>n</i>	(グローバル)。 INTEGER。行列 <i>sub(A)</i> と <i>sub(B)</i> の次数 ($n \geq 0$)。
<i>a</i>	(ローカル) COMPLEX (<i>pchegvx</i> の場合) DOUBLE COMPLEX (<i>pzhegvx</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。呼び出し時に、この配列は、 $n \times n$ のエルミート分散行列 <i>sub(A)</i> のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。 <i>desca(ctxt_)</i> が正しくない場合、 <i>pzhegvx</i> での正しいエラー報告は保証されない。
<i>b</i>	(ローカル)。 COMPLEX (<i>pchegvx</i> の場合) DOUBLE COMPLEX (<i>pzhegvx</i> の場合) ローカルメモリーにある、次元 (<i>lld_b</i> , <i>LOCc(jb+n-1)</i>) の配列へのポインター。呼び出し時に、この配列は、 $n \times n$ のエルミート分散行列 <i>sub(B)</i> のローカル部分を含む。 <i>uplo</i> = 'U' の場合、 <i>sub(B)</i> の先頭の $n \times n$ 上三角部分は行列の上三角部分を含む。 <i>uplo</i> = 'L' の場合、 <i>sub(B)</i> の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>B</i> の最初の行と最初の列を示す、グローバル配列 <i>b</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。 <i>descb(ctxt_)</i> は <i>desca(ctxt_)</i> と等しくなければならない。
<i>v1, vu</i>	(グローバル) REAL (<i>pchegvx</i> の場合) DOUBLE PRECISION (<i>pzhegvx</i> の場合) <i>range</i> = 'V' の場合、固有値を探す区間の下限値と上限値。 <i>range</i> = 'A' または 'I' の場合、 <i>v1</i> と <i>vu</i> は参照されない。
<i>il, iu</i>	(グローバル) INTEGER。 <i>range</i> = 'I' の場合、返される最小固有値と最大固有値に対する昇順のインデックス。 次の制約がある。 $i1 \geq 1, \min(i1, n) \leq iu \leq n$ <i>range</i> = 'A' または 'V' の場合、 <i>il</i> と <i>iu</i> は参照されない。

<i>abstol</i>	(グローバル) REAL (pchegvx の場合) DOUBLE PRECISION (pzhegvx の場合) <i>jobz</i> ='V' の場合、 <i>abstol</i> を $p?lamch(context, 'U')$ に設定すると、ほとんどの直交固有ベクトルが生成される。 固有値に対する絶対エラー許容値。次に示す値以下の幅の区間 $[a, b]$ 内に存在していると判別された場合 (<i>eps</i> はマシン精度)、近似固有値は、収束したものとして受け入れられる。 $abstol + eps * \max(a , b)$ ここで、 <i>eps</i> はマシン精度である。 <i>abstol</i> がゼロまたは負の場合、代わりに $eps * \text{norm}(T)$ を使用する。ここで、 $\text{norm}(T)$ は、 <i>A</i> を三重対角形式に縮退して得られた三重対角行列の 1- ノルムである。 固有値が最も正確に計算されるのは、 <i>abstol</i> がゼロでなく、アンダーフローのしきい値の 2 倍、 $2 * p?lamch('S')$ に設定されたときである。このルーチンで $((\text{mod}(info, 2).ne.0).or.*(\text{mod}(info/8, 2).ne.0))$ が返された場合、固有値または固有ベクトルのいくつかは収束に失敗したことを意味するので、 <i>abstol</i> を $2 * p?lamch('S')$ に設定して、やり直してみる。
<i>orfac</i>	(グローバル)。 REAL (pchegvx の場合) DOUBLE PRECISION (pzhegvx の場合) 再直交化される固有ベクトルを指定する。互いの $tol = orfac * \text{norm}(A)$ 内にある固有値に対応する固有ベクトルは再直交される。しかし、ワークスペースが十分でない場合 (<i>lwork</i> を参照)、 <i>tol</i> は、すべての固有ベクトルが 1 つのプロセスで再直交化されるようになるまで下げられる。 <i>orfac</i> がゼロの場合、再直交化は行われず。 <i>orfac</i> が負の場合、デフォルト値の 10^{-3} が使用される。 <i>orfac</i> はすべての処理で同一でなければならない。
<i>iz, jz</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>Z</i> の最初の行と最初の列を示す、グローバル配列 <i>z</i> の行インデックスと列インデックス。
<i>descz</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散配列 <i>Z</i> のディスクリプター。 <i>descz</i> (<i>ctxt_</i>) は <i>desca</i> (<i>ctxt_</i>) と等しくなければならない。
<i>work</i>	(ローカル) COMPLEX (pchegvx の場合) DOUBLE COMPLEX (pzhegvx の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカル)。 INTEGER。配列 <i>work</i> の次元。 固有値のみが要求された場合： $lwork \geq n + \max(NB * (np0 + 1), 3)$ 固有ベクトルが要求された場合： $lwork \geq n + (np0 + mq0 + NB) * NB$ $mq0 = \text{numroc}(nn, NB, 0, 0, NPCOL)$

最適なパフォーマンスを得るには、より大きなワークスペースが必要である。すなわち、

$$lwork \geq \max(lwork, n, nhetrd_lwopt, nhegst_lwopt)$$

ここで、 $lwork$ は、上で定義され、

$$nhetrd_lwork = 2*(anb+1)*(4*nps+2) + (nps + 1) * nps$$

$$nhegst_lwopt = 2*np0*NB + nq0*NB + NB*NB$$

$$NB = desca(mb_)$$

$$np0 = \text{numroc}(n, NB, 0, 0, NPROW)$$

$$nq0 = \text{numroc}(n, NB, 0, 0, NPCOL)$$

$$ictxt = desca(ctxt_)$$

$$anb = \text{pjlaenv}(ictxt, 3, 'p?hettrd', 'L', 0, 0, 0, 0)$$

$$sqnpc = \text{sqrt}(\text{dble}(NPROW * NPCOL))$$

$$nps = \max(\text{numroc}(n, 1, 0, 0, sqnpc), 2*anb)$$

numroc は ScaLAPACK ツール関数である。

pjlaenv は ScaLAPACK 環境問い合わせ関数である。MYROW、MYCOL、NPROW、および NPCOL は、サブルーチン `blacs_gridinfo` を呼び出して決定できる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての $work$ 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

$rwork$

(ローカル)

REAL (`pchegvx` の場合)

DOUBLE PRECISION (`pzhegvx` の場合)

ワークスペース配列、次元は ($lrwork$)。

$lrwork$

(ローカル)

INTEGER。配列 $rwork$ の次元。

$lrwork$ の定義に使用する変数の定義は以下を参照。

固有ベクトルが要求されなかった場合 ($jobz = 'N'$)、

$$lrwork \geq 5 * nn + 4 * n.$$

固有ベクトルが要求された場合 ($jobz = 'V'$)、すべての固有ベクトルが計算されることを保証するために必要なワークスペースの量は次のとおりである。

$$lrwork \geq 4*n + \max(5*nn, np0 * mq0) +$$

$$\text{iceil}(\text{neig}, NPROW * NPCOL) * nn$$

最小のワークスペースが与えられ、 $orfac$ が小さすぎる場合、計算されるベクトル固有値は直交ではない。(パフォーマンスが低下しても) 直交性を保証するには、次の値を $lwork$ に追加する。

$$(clustersize-1)*n$$

ここで、 $clustersize$ は、クラスターが近い固有値のセットとして定義された、最大クラスターの固有値の数。

$$\{w(k), \dots, w(k+clustersize-1)\}$$

$$w(j+1) \leq w(j) + orfac * 2 * \text{norm}(A)$$

変数定義:

$neig$ = 要求された固有ベクトルの数

$$NB = desca(mb_) = desca(nb_) = descz(mb_) = descz(nb_)$$

$nn = \max(n, NB, 2)$
 $desca(rsrc_) = desca(nb_) = descz(rsrc_) = descz(csrc_) = 0$
 $np0 = \text{numroc}(nn, NB, 0, 0, NPROW)$
 $mq0 = \text{numroc}(\max(neig, NB, 2), NB, 0, 0, NPCOL)$ $\text{iceil}(x, y)$ は (x/y) を返す ScaLAPACK 関数。

lwork が小さすぎる場合:

lwork が直交性を保証するには小さすぎる場合、*p?hegvx* は、固有値間の間隔が最も小さいクラスターで直交性を維持するように試みる。*lwork* が小さすぎて要求された固有値をすべて計算できない場合、計算は行われず、*info*=-25 が返される。*range='v'* の場合、*p?hegvx* は固有値が計算されるまで要求されている固有ベクトルがわからない点に注意する。したがって、*range='v'* で *lwork* が十分大きく *p?hegvx* で固有値が計算可能な場合、*p?hegvx* は固有値とできるだけ多くの固有ベクトルを計算する。

ワークスペース、直交性とパフォーマンスの関係:

$clustersize \geq n/\sqrt{NPROW * NPCOL}$ の場合、すべての固有ベクトルを直交して計算するために十分なスペースを提供すると、パフォーマンスが大幅に低下する。限界 (すなわち、 $clustersize = n-1$) では、[p?stein](#) は 1 プロセッサ上の *?stein* よりもパフォーマンスが高くなることはない。

$clustersize = n/\sqrt{NPROW * NPCOL}$ の場合、すべての固有ベクトルの再直交化は、2 以上の係数では合計の実行時間が長くなる。

$clustersize > n/\sqrt{NPROW * NPCOL}$ の場合、実行時間はクラスターサイズの 2 乗で増加し、すべての他の係数は同等なままで十分なワークスペースが提供されていると仮定する。ワークスペースが少なくなると再直交化も少なくなるが、実行は速くなる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての *work* 配列で最適なパフォーマンスのために必要なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

iwork (ローカル) INTEGER。ワークスペース配列。

liwork (ローカル) INTEGER。 *iwork* の次元。

$liwork \geq 6 * nnp$

ここで、 $nnp = \max(n, NPROW * NPCOL + 1, 4)$

liwork = -1 の場合、*liwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

- a 終了時に、*jobz* = 'v' で *info* = 0 の場合、*sub(A)* には、固有ベクトルの分散行列 *Z* が格納される。固有ベクトルは、次のように正規化される。
ibtype = 1 または 2 の場合、 $Z^H * \text{sub}(B) * Z = i$ 。
ibtype = 3 の場合、 $Z^H * \text{inv}(\text{sub}(B)) * Z = i$ 。
jobz = 'N' の場合、終了時に、*sub(A)* の上三角 (*uplo* = 'U' の場合) または下三角 (*uplo* = 'L' の場合) が対角を含めて破棄される。

<i>b</i>	終了時に、 $info \leq n$ の場合、行列を格納している $\text{sub}(B)$ の一部が、コレスキー因子分解 $\text{sub}(B) = U^H U$ または $\text{sub}(B) = L L^H$ からの三角係数 U あるいは L で上書きされる。
<i>m</i>	(グローバル) INTEGER。求められた固有値の個数 ($0 \leq m \leq n$)。
<i>nz</i>	(グローバル) INTEGER。 計算された固有ベクトルの総数 ($0 \leq nz \leq m$)。 z の列数。 <i>jobz</i> .ne. 'V' の場合、 <i>nz</i> は参照されない。 <i>jobz</i> .eq. 'V' の場合、 $nz = m$ (ユーザーが十分な空間を提供しないで、 <i>p?hegvx</i> が計算を開始する前にこれを検出できない場合を除く)。要求されたすべての固有ベクトルを得るには、ユーザーは z (<i>m</i> .le. <i>descz</i> (<i>n</i>)) の固有ベクトルを保持するための十分な空間とそれらを計算するための十分なワークスペースの両方を提供しなければならない。(<i>lwork</i> を参照)。 <i>p?hegvx</i> は、 <i>range</i> .eq. 'V' の場合を除いて、計算を開始する前に空間が十分でないことを常に検出できる。
<i>w</i>	(グローバル) REAL (<i>pchegvx</i> の場合) DOUBLE PRECISION (<i>pzhegvx</i> の場合) 配列、次元は (<i>n</i>)。 正常終了時に、先頭の m 個の成分には、指定された固有値が昇順で格納される。
<i>z</i>	(ローカル)。 COMPLEX (<i>pchegvx</i> の場合) DOUBLE COMPLEX (<i>pzhegvx</i> の場合) グローバル次元 (n, n)、ローカル次元 (<i>lld_z</i> , <i>LOCc</i> ($jz+n-1$))。 <i>jobz</i> = 'V' の場合、正常終了時に、 z の先頭の m 列には、選択された固有値に対応する正規直交固有ベクトルが格納される。固有ベクトルが収束できない場合、 z のその列には、その固有ベクトルに対する最新の近似値が格納され、固有ベクトルのインデックスは <i>ifail</i> に返される。 <i>jobz</i> = 'N' の場合、 z は参照されない。
<i>work</i>	終了時に、 <i>work</i> (1) は、ワークスペースの最適値を返す。
<i>rwork</i>	終了時に、 <i>rwork</i> (1) には、効率的に実行するために必要なワークスペースの最適な量が格納される。 <i>jobz</i> ='N' の場合、 <i>rwork</i> (1) には、固有値を効率的に計算するために必要な、ワークスペースの最適な量が格納される。 <i>jobz</i> ='V' の場合、 <i>rwork</i> (1) には、直交性を保証しないで固有値と固有ベクトルを効率的に計算するために必要なワークスペースの最適な量が格納される。 <i>range</i> ='V' の場合、最適なワークスペースを計算するときに、すべての固有ベクトルが必要であるとみなされる。
<i>ifail</i>	(グローバル) INTEGER。 配列、次元は (<i>n</i>)。 <i>info</i> .ne. 0 の場合、 <i>ifail</i> は追加情報を提供する

($\text{mod}(\text{info}/16,2).\text{ne.0}$) の場合、 $\text{ifail}(1)$ は、正定値でない最小の小行列式の次数を示す。終了時に、($\text{mod}(\text{info},2).\text{ne.0}$) の場合、 $\text{ifail}(1)$ には収束に失敗した固有ベクトルのインデックスが格納される。

上記のエラー条件がどちらもセットされてなく、かつ $\text{jobz} = 'v'$ の場合、 ifail の先頭から m 個の成分はゼロに設定される。

iclustr

(グローバル)

INTEGER。

配列、次元は ($2*\text{NPROW}*\text{NPCOL}$)。この配列には、ワークスペースが十分でなかったために再直交できなかつた、固有値のクラスターに対する固有ベクトルのインデックスが格納される (lwork 、 orfac 、および info を参照)。インデックス $\text{iclustr}(2*i-1)$ から $\text{iclustr}(2*i)$ の固有値のクラスターに対応する固有ベクトルは、ワークスペースの不足により再直交できなかつた。したがって、これらのクラスターに対応する固有ベクトルは直交しない。 $\text{iclustr}()$ は、ゼロで終了する配列である。($\text{iclustr}(2*k).\text{ne.0}.\text{and}.\text{iclustr}(2*k+1).\text{eq.0}$) (k がクラスター数の場合)。 iclustr は参照されない ($\text{jobz} = 'N'$ の場合)。

gap

(グローバル)

REAL (pchegvx の場合)

DOUBLE PRECISION (pzhegvx の場合)

配列、次元は ($\text{NPROW}*\text{NPCOL}$)。

この配列には、固有ベクトルが再直交できなかつた固有値間のギャップが格納される。この配列の出力値は、配列 iclustr によって示されたクラスターに対応する。その結果、 i 番目のクラスターに対応する固有ベクトルのドット積は、 $(C * n) / \text{gap}(i)$ と同程度になる。ここで、 C は小さな定数である。

info

(グローバル)

INTEGER。

$\text{info} = 0$ の場合、正常に終了したことを示す。

$\text{info} < 0$ の場合： i 番目の引数が配列で、 j 番目の値が不正だった場合、 $\text{info} = -(i*100+j)$ 。 i 番目の引数がスカラーで値が不正だった場合、 $\text{info} = -i$ 。

$\text{info} > 0$ の場合、

($\text{mod}(\text{info},2).\text{ne.0}$) の場合、1つ以上の固有ベクトルは収束に失敗したことを示す。それらのインデックスは ifail に格納される。

($\text{mod}(\text{info}/2,2).\text{ne.0}$) の場合、1つ以上の固有値のクラスターに対する固有ベクトルが、ワークスペースが十分でなかったために再直交化できなかつた。クラスターのインデックスは、配列 iclustr に格納される。

($\text{mod}(\text{info}/4,2).\text{ne.0}$) の場合、空間の制限により、 p?sygvx で $v1$ と vu 間のすべての固有ベクトルを計算できなかつた。計算された固有ベクトルの個数は、 nz に返される。

($\text{mod}(\text{info}/8,2).\text{ne.0}$) の場合、 p?stebz は、固有値の計算に失敗したことを示す。

($\text{mod}(\text{info}/16,2).\text{ne.0}$) の場合、 B は正定値ではなかつたことを示す。 $\text{ifail}(1)$ は、正定値でない最小の小行列式の次数を示す。

ScaLAPACK 補助ルーチンとユーティリティー・ルーチン

7

この章では ScaLAPACK [補助ルーチン](#)および[ユーティリティー関数とルーチン](#)に関する Intel[®] マス・カーネル・ライブラリーの実装について説明する。ライブラリーは実数と複素数の両方のデータに対応したルーチンで構成される。



注: ScaLAPACK ルーチンは、Intel MKL のスーパーセットである Intel[®] MKL クラスタ・エディションで提供されている。

ScaLAPACK 補助ルーチンとユーティリティー・ルーチンで使用されているルーチン名の規則、数学表記、行列の格納形式は、これまでの章に記載されているものと同一である。一部のルーチンや関数では、`sc` や `dz` などの組み合わせられたキャラクター・コードを持つことができる。例えば、ルーチン `pscsum1` は、入力として複素数配列を使用し、出力として実数値を返す。

補助ルーチン

表 7-1 ScaLAPACK 補助ルーチン

ルーチン名	データ型	説明
p_zlacqv	<code>c, z</code>	複素ベクトルを共役する。
p_zmax1	<code>c, z</code>	実数部分が最大絶対値を持つ成分のインデックスを検出する (レベル 1 PBLAS の <code>p_zamax</code> に似ているが、実数部分の絶対値を使用している)。
p_zcombamax1	<code>c, z</code>	実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。
p_zsum1	<code>sc, dz</code>	レベル 1 PBLAS の <code>p_zasum</code> と同様に複素ベクトルの 1- ノルムを実行するが、真の絶対値を用いる。
p_zdbtrsv	<code>s, d, c, z</code>	一般三重対角行列の <i>LU</i> 因子分解を、ピボット演算を用いずに計算する。 <code>p_zdbtrs</code> によって呼び出される。
p_zdttrsv	<code>s, d, c, z</code>	一般帯行列の <i>LU</i> 因子分解を、行交換を伴う部分ピボット演算を用いて計算する。 <code>p_zdttrs</code> によって呼び出される。
p_zgebd2	<code>s, d, c, z</code>	直交 / ユニタリー変換を用いて、一般矩形行列を実数二重対角形式に縮退させる (非ブロック化アルゴリズム)。
p_zgehd2	<code>s, d, c, z</code>	直交 / ユニタリー相似変換を用いて、一般行列を上 Hessenberg 形式に縮退させる (非ブロック化アルゴリズム)。

表 7-1 ScaLAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
p?gelq2	s, d, c, z	一般矩形行列の LQ 因子分解を計算する (非ブロック化アルゴリズム)。
p?geql2	s, d, c, z	一般矩形行列の QL 因子分解を計算する (非ブロック化アルゴリズム)。
p?geqr2	s, d, c, z	一般矩形行列の QR 因子分解を計算する (非ブロック化アルゴリズム)。
p?qerq2	s, d, c, z	一般矩形行列の RQ 因子分解を計算する (非ブロック化アルゴリズム)。
p?getf2	s, d, c, z	一般行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する (ローカルブロック化アルゴリズム)。
p?labrd	s, d, c, z	直交 / ユニタリー変換を用いて、一般矩形行列 A の最初の nb 行と列を実数二重対角形式に縮退させ、A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。
p?lacon	s, d, c, z	行列 - ベクトル積の評価にリバース・コミュニケーションを使用して正方行列の 1- ノルムを推定する。
p?laconsb	s, d	2 つの連続する小さい対角成分を検出する。
p?lACP2	s, d, c, z	分散行列の一部または全部を他の分散行列にコピーする。
p?lACP3	s, d	グローバル並列配列からローカル複製配列にコピーする。逆の場合も同じ。
p?lACPY	s, d, c, z	1 つの 2 次元配列の一部または全部を他にコピーする。
p?laevswp	s, d, c, z	計算された固有ベクトルを ScaLAPACK 標準ブロック・サイクリック配列に移動する。
p?lahrd	s, d, c, z	直交 / ユニタリー相似変換を用いて、k 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の nb 列を縮退させ、A の縮退されていない部分に変換を適用するために必要となる補助行列を返す。
p?laiect	s, d, c, z	IEEE 演算を利用して固有値の計算を加速させる (C インターフェイス関数)。
p?lange	s, d, c, z	一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
p?lanhs	s, d, c, z	上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
p?lansy, p?lanhe	s, d, c, z /c, z	実対称行列または複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
p?lantr	s, d, c, z	三角行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。
p?lapiv	s, d, c, z	一般分散行列に置換行列を適用し、行または列のピボット演算を行う。
p?laqge	s, d, c, z	p?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。
p?laqsy	s, d, c, z	p?poequ で計算されたスケール係数を用いて対称 / エルミート行列をスケールリングする。
p?lared1d	s, d	入力配列 <i>bycol</i> は複数の行に分配され、すべてのプロセス列に同じ <i>bycol</i> のコピーが格納されることを前提に、配列を再分配する。

表 7-1 ScaLAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
p?lared2d	s, d	入力配列 <i>byrow</i> は複数の列に分配され、すべてのプロセス行に同じ <i>byrow</i> のコピーが格納されることを前提に、配列を再分配する。
p?larf	s, d, c, z	一般矩形行列に基本リフレクターを適用する。
p?larfb	s, d, c, z	ブロック・リフレクターまたはその転置 / 共役転置を一般矩形行列に適用する。
p?larfc	c, z	一般行列に基本リフレクターの共役転置を適用する。
p?larfg	s, d, c, z	基本リフレクター (Householder 行列) を生成する
p?larft	s, d, c, z	ブロック・リフレクター $H = I - VTV^H$ の三角ベクトル T を生成する。
p?larz	s, d, c, z	p?tzrzf が返したとおりの基本リフレクターを一般行列に適用する。
p?larzb	s, d, c, z	p?tzrzf が返したとおりのブロック・リフレクターまたはその転置 / 共役転置を、一般行列に適用する。
p?larzc	c, z	p?tzrzf が返したとおりの基本リフレクターの共役転置を、一般行列に適用 (乗算) する。
p?larzt	s, d, c, z	p?tzrzf が返したとおりのブロック・リフレクター $H = I - VTV^H$ の三角係数 T を生成する。
p?lascl	s, d, c, z	一般矩形行列に C_{to}/C_{from} として定義される実スカラーを掛ける。
p?laset	s, d, c, z	行列の非対角成分を α に、対角成分を β に初期化する。
p?lasmsub	s, d	安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。
p?lassq	s, d, c, z	スケーリング形式で表現された二乗和を更新する。
p?laswp	s, d, c, z	一般矩形行列に対して一連の行交換を実行する。
p?latra	s, d, c, z	一般正方分散行列の対角和を計算する。
p?latrd	s, d, c, z	直交 / ユニタリー相似変換を用いて、対称 / エルミート行列 A の最初の nb 行 / 列を実数三重対角形式に縮退する。
p?latrz	s, d, c, z	直交 / ユニタリー相似変換を用いて、実 / 複素上台形行列を上三角形式に縮退させる。
p?lauu2	s, d, c, z	積 UU^H または L^HL を計算する。ここで、 U と L は上三角または下三角行列 (ローカル非ブロック化アルゴリズム)。
p?lauum	s, d, c, z	積 UU^H または L^HL を計算する。ここで、 U と L は上三角または下三角行列。
p?lawil	s, d	Wilkinson 変換を実行する。
p?org2l/p?ung2l	s, d, c, z	p?geqlf で求めた QL 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
p?org2r/p?ung2r	s, d, c, z	p?geqrf で求めた QR 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
p?orgl2/p?ungl2	s, d, c, z	p?gelqf で求めた LQ 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。

表 7-1 ScaLAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
p?orgr2/p?ungr2	s, d, c, z	p?gerqf で求めた RQ 因子分解から、全部または一部の直交 / ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。
p?orm2l/p?unm2l	s, d, c, z	一般行列に p?geqlf で求めた QL 因子分解の直交 / ユニタリー行列を掛ける (非ブロック化アルゴリズム)。
p?orm2r/p?unm2r	s, d, c, z	一般行列に p?geqrf で求めた QR 因子分解の直交 / ユニタリー行列を掛ける (非ブロック化アルゴリズム)。
p?orml2/p?unml2	s, d, c, z	一般行列に p?gelqf で求めた LQ 因子分解の直交 / ユニタリー行列を掛ける (非ブロック化アルゴリズム)。
p?ormr2/p?unmr2	s, d, c, z	一般行列に p?gerqf で求めた RQ 因子分解の直交 / ユニタリー行列を掛ける (非ブロック化アルゴリズム)。
p?pbtrsv	s, d, c, z	前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pbtrf で計算された帯行列の係数である。
p?pttrsv	s, d, c, z	前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pttrf で計算された三重対角行列の係数である。
p?potf2	s, d, c, z	対称 / エルミート正定値行列のコレスキー因子分解を行う (ローカル非ブロック化アルゴリズム)。
p?rscl	s, d, cs, zd	ベクトルに実スカラーの逆数を掛ける。
p?syqs2/p?hegs2	s, d, c, z	p?potrf で得られた因子分解の結果を用いて、対称 / エルミート汎用固有値問題を標準形式に縮退させる (ローカル非ブロック化アルゴリズム)。
p?sytd2/p?hetd2	s, d, c, z	直交 / ユニタリー相似変換を用いて、対称 / エルミート行列を実数対称三重対角形式に縮退させる (ローカル非ブロック化アルゴリズム)。
p?trti2	s, d, c, z	三角行列の逆行列を計算する (ローカル非ブロック化アルゴリズム)。
?lamsh	s, d	送られるバルジの数を最大にするために、小さな (単一成分) 行列を介して複数のシフトを送る。
?laref	s, d	Householder リフレクターを行列の行または列に適用する。
?lasorte	s, d	固有ベアを実数および複素数データ型でソートする。
?lasrt2	s, d	数を昇順または降順でソートする。
?stein2	s, d	逆反復法を用いて、実対称の三重対角行列の指定された固有値に対応する固有ベクトルを計算する。
?dbtf2	s, d, c, z	一般帯行列の LU 因子分解を、ピボット演算を用いなくて計算する (ローカル非ブロック化アルゴリズム)。
?dbtrf	s, d, c, z	一般帯行列の LU 因子分解を、ピボット演算を用いなくて計算する (ローカルブロック化アルゴリズム)。
?dttrf	s, d, c, z	一般三重対角行列の LU 因子分解を、ピボット演算を用いなくて計算する (ローカルブロック化アルゴリズム)。
?dttrsv	s, d, c, z	?dttrf によって行われた LU 因子分解を使用して、一般三重対角行列を係数行列とする連立 1 次方程式を解く。
?pttrsv	s, d, c, z	?pttrf によって行われた LDL^H 因子分解を使用して、対称 (エルミート) 正定値三重対角行列を係数行列とする連立 1 次方程式を解く。

表 7-1 ScaLAPACK 補助ルーチン (続き)

ルーチン名	データ型	説明
?steqr2	s, d	暗黙的 <i>QL</i> 法または <i>QR</i> 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

p?lacgv

複素ベクトルを共役する。

構文

```
call pclacgv(n, x, ix, jx, descx, incx)
call pzlacgv(n, x, ix, jx, descx, incx)
```

説明

このルーチンは、長さ n 、 $\text{sub}(x)$ の複素ベクトルを共役する。ここで、 $\text{sub}(x)$ は $X(ix, jx:jx+n-1)$ ($incx = descx(m_)$ の場合)、または $X(ix:ix+n-1, jx)$ ($incx = 1$ の場合) である。

入力パラメーター

n (グローバル) INTEGER。乱数ベクトル $\text{sub}(x)$ の長さ。

x (ローカル)
 COMPLEX (pclacgv の場合)
 COMPLEX*16 (pzlacgv の場合)
 ローカルメモリーにある、次元 ($lld_x, *$) の配列へのポインタ。共役するベクトルを格納する。
 $x(i) = X(ix+(jx-1)*m_x + (i-1)*incx)$ 、 $1 \leq i \leq n$ 。

ix (グローバル) INTEGER。
 $\text{sub}(x)$ の最初の行を示す、グローバル配列 x の行インデックス。

jx (グローバル) INTEGER。
 $\text{sub}(x)$ の最初の列を示す、グローバル配列 x の列インデックス。

$descx$ (グローバルおよびローカル) INTEGER。
 配列、次元は ($dlen_$)。分散行列 X の配列ディスクリプター。

$incx$ (グローバル) INTEGER。
 X の成分のグローバルな増分。このバージョンでは、1 と m_x の 2 つの $incx$ の値のみサポートされる。
 $incx$ の値は、ゼロであってはならない。

出力パラメーター

x (ローカル) 終了時に、共役ベクトルが格納される。

p?max1

実数部分が最大絶対値を持つ成分のインデックスを検出する (レベル1 PBLAS の p?amax に似ているが、実数部分の絶対値を使用している)。

構文

```
call p?max1(n, amax, indx, x, ix, jx, descx, incx)
call pzmax1(n, amax, indx, x, ix, jx, descx, incx)
```

説明

このルーチンは、乱数ベクトル $\text{sub}(x)$ の最大絶対値を持つ成分のグローバル・インデックスを計算する。グローバル・インデックスは indx に戻され、値は amax に戻される。

ここで、 $\text{sub}(x)$ は次のとおりである。 $X(\text{ix}:\text{ix}+\text{n}-1, \text{jx})$ ($\text{incx} = 1$ の場合)

$X(\text{ix}, \text{jx}:\text{jx}+\text{n}-1)$ ($\text{incx} = \text{m}_x$ の場合)

入力パラメーター

n	(グローバル) INTEGER へのポインター。乱数ベクトル $\text{sub}(x)$ の成分の個数。 $n \geq 0$ 。
x	(ローカル) COMPLEX (p?max1 の場合) COMPLEX*16 (pzmax1 の場合) 次の以上のローカル部分を格納する配列。 ($(\text{jx}-1)*\text{m}_x + \text{ix} + (\text{n} - 1)*\text{abs}(\text{incx})$)。乱数ベクトル $\text{sub}(x)$ の成分を格納する。
ix	(グローバル) INTEGER。 $\text{sub}(x)$ の最初の行を示す、グローバル配列 X の行インデックス。
jx	(グローバル) INTEGER。 $\text{sub}(x)$ の最初の列を示す、グローバル配列 X の列インデックス。
descx	(グローバルおよびローカル) INTEGER。配列、次元は (dlen_x)。分散行列 X の配列ディスクリプター。
incx	(グローバル) INTEGER。 X の成分のグローバルな増分。このバージョンでは、1 と m_x の2つの incx の値のみサポートされる。 incx の値は、ゼロであってはならない。

出力パラメーター

amax	(グローバル出力) REAL へのポインター。 $\text{sub}(x)$ の範囲における、乱数ベクトル $\text{sub}(x)$ の最大成分の絶対値。
indx	(グローバル出力) INTEGER へのポインター。実数部分が最大絶対値を持つ、乱数ベクトル $\text{sub}(x)$ の成分のグローバル・インデックス。

?combamax1

実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。

構文

```
call ccombamax1(v1, v2)
call zcombamax1(v1, v2)
```

説明

このルーチンは、実数部分が最大絶対値を持つ成分とそのグローバル・インデックスを検出する。

入力パラメーター

v1	(ローカル) COMPLEX (ccombamax1 の場合) COMPLEX*16 (zcombamax1 の場合) 配列、次元は 2。 最初の絶対最大成分とそのグローバル・インデックス。 v1(1) = amax、 v1(2) = indx。
v2	(ローカル) COMPLEX (ccombamax1 の場合) COMPLEX*16 (zcombamax1 の場合) 配列、次元は 2。 2 番目の絶対最大成分とそのグローバル・インデックス。 v2(1) = amax、 v2(2) = indx。

出力パラメーター

v1	(ローカル) 最初の絶対最大成分とそのグローバル・インデックス。 v1(1) = amax、 v1(2) = indx。
----	--

p?sum1

レベル1 PBLAS の p?asum と同様に複素ベクトルの 1- ノルムを実行するが、真の絶対値を用いる。

構文

```
call pscsum1(n, asum, x, ix, jx, descx, incx)
call pdzsum1(n, asum, x, ix, jx, descx, incx)
```

説明

このルーチンは、複素乱数ベクトル $\text{sub}(x)$ の絶対値の合計を asum に格納する。

ここで、 $\text{sub}(x)$ は次のとおりである。 $X(ix:ix+n-1, jx:jx)$ ($incx = 1$ の場合)
 $X(ix:ix, jx:jx+n-1)$ ($incx = m_x$ の場合)

レベル 1 BLAS の $p?asum$ を基にしている。違いは、'真の'絶対値を使用していることである。

入力パラメーター

n	(グローバル) INTEGER へのポインター。乱数ベクトル $\text{sub}(x)$ の成分の個数。 $n \geq 0$ 。
x	(ローカル) COMPLEX (psscsum1 の場合) COMPLEX*16 (pdzsum1 の場合) 次の以上のローカル部分を格納する配列。 $((jx-1)*m_x + ix + (n-1)*abs(incx))$ 。乱数ベクトル $\text{sub}(x)$ の成分を格納する。
ix	(グローバル) INTEGER。 $\text{sub}(x)$ の最初の行を示す、グローバル配列 X の行インデックス。
jx	(グローバル) INTEGER。 $\text{sub}(x)$ の最初の列を示す、グローバル配列 X の列インデックス。
$descx$	(グローバルおよびローカル) INTEGER。配列、次元は 8。分散行列 X の配列ディスクリプター。
$incx$	(グローバル) INTEGER。 X の成分のグローバルな増分。このバージョンでは、1 と m_x の 2 つの $incx$ の値のみサポートされる。

出力パラメーター

asum	(ローカル) REAL へのポインター。 $\text{sub}(x)$ の範囲における、乱数ベクトル $\text{sub}(x)$ の最大成分の絶対値。
---------------	--

p?dbtrsv

一般三角行列の LU 因子分解を、ピボット演算を用いずに計算する。 $p?dbtrs$ によって呼び出される。

構文

```
call psdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pddbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pcdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
```

```
call pzdbtrsv(uplo, trans, n, bwl, bwu, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
```

説明

このルーチンは、次の三角帯行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs) \text{ または}$$

$$A(1:n, ja:ja+n-1)^T * X = B(ib:ib+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H * X = B(ib:ib+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$ は、ガウス消去コード PD@(dom_pre)BTRF によって生成される三角帯行列の係数であり、 $A(1:n, ja:ja+n-1)$ と af に格納される。

$A(1:n, ja:ja+n-1)$ には、 $uplo$ の値に従って、上三角行列または下三角行列が格納される。 $(A(1:n, ja:ja+n-1))$ と $(A(1:n, ja:ja+n-1)^T)$ のどちらの式を解くのかは、パラメータ $trans$ を介して、ユーザーによって決定される。

ルーチン [pzdbtrf](#) は、最初に呼び出さなければならない。

入力パラメーター

$uplo$	(グローバル) CHARACTER。 $uplo = 'U'$ の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 $uplo = 'L'$ の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
$trans$	(グローバル) CHARACTER。 $trans = 'N'$ の場合、 $A(1:n, ja:ja+n-1)$ を解く。 $trans = 'C'$ の場合、共役転置式 $A(1:n, ja:ja+n-1)$ を解く。
n	(グローバル) INTEGER。分散部分行列 A の次数 ($n \geq 0$)。
bwl	(グローバル) INTEGER。劣対角成分の数。 $0 \leq bwl \leq n-1$ 。
bwu	(グローバル) INTEGER。劣対角成分の数。 $0 \leq bwu \leq n-1$ 。
$nrhs$	(グローバル) INTEGER。右辺の数。分散部分行列 B の列数 ($nrhs \geq 0$)。
a	(ローカル)。 REAL (psdbtrsv の場合) DOUBLE PRECISION (pddbtrsv の場合) COMPLEX (pcdbtrsv の場合) COMPLEX*16 (pzdbtrsv の場合) ローカルメモリーにある、第 1 次元が $lld_a \geq (bwl+bwu+1)$ の配列へのポインター ($desca$ に格納される)。 $n \times n$ の非対称帯形式の分割コレスキー因子 L または $L^T A(1:n, ja:ja+n-1)$ のローカル部分を格納する。 このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。
ja	(グローバル) INTEGER。 (A のすべてまたは A の部分行列で) 処理される行列の先頭を指すグローバル配列 a のインデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 1d type (<i>dtype_a</i> = 501 または 502) の場合、 <i>dlen</i> ≥ 7。 2d type (<i>dtype_a</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>A</i> の配列ディスクリプター。 <i>A</i> のマッピング情報をメモリーに格納する。
<i>b</i>	(ローカル) REAL (<i>psdbtrsv</i> の場合) DOUBLE PRECISION (<i>pddbtrsv</i> の場合) COMPLEX (<i>pcdbtrsv</i> の場合) COMPLEX*16 (<i>pzdbtrsv</i> の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $11d_b \geq nb$ の配列へのポインター。 右辺 <i>B</i> (<i>ib:ib+n-1, 1:nrhs</i>) のローカル部分を格納する。
<i>ib</i>	(グローバル) INTEGER。 <i>b</i> のすべて、または <i>B</i> の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 1d type (<i>dtype_b</i> = 502) の場合、 <i>dlen</i> ≥ 7。 2d type (<i>dtype_b</i> = 1) の場合、 <i>dlen</i> ≥ 9。 分散行列 <i>B</i> の配列ディスクリプター。 <i>B</i> のマッピング情報をメモリーに格納する。
<i>laf</i>	(ローカル) INTEGER。ユーザー入力の補助非零要素空間 <i>af</i> のサイズ。 $laf \geq nb*(bw1+bwu)+6*\max(bw1, bwu)*\max(bw1, bwu)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル)。 REAL (<i>psdbtrsv</i> の場合) DOUBLE PRECISION (<i>pddbtrsv</i> の場合) COMPLEX (<i>pcdbtrsv</i> の場合) COMPLEX*16 (<i>pzdbtrsv</i> の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq \max(bw1, bwu)*nrhs$ 。

出カパラメーター

<i>a</i>	(ローカル)。 このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。

<i>af</i>	(ローカル) REAL (psdbtrsv の場合) DOUBLE PRECISION (pddbtrsv の場合) COMPLEX (pcdbtrsv の場合) COMPLEX*16 (pzdbtrsv の場合) 補助非零要素空間。非零要素は、因子分解ルーチン p?dbtrf で作成され、 <i>af</i> に格納される。因子分解ルーチンの後で、p?dbtrf を使って連立 1 次方程式を解く場合、因子分解後に <i>af</i> を変更してはならない。
<i>work</i>	終了時に、 <i>lwork</i> の最小値が <i>work(1)</i> に格納される。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = - (<i>i</i> *100+ <i>j</i>)、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> であることを示す。

p?dttrs

一般帯行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する。p?dttrs によって呼び出される。

構文

```
call psdttrs(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
call pddttrs(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
call pcdttrs(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
call pzdttrs(uplo, trans, n, nrhs, dl, d, du, ja, desca, b, ib, descb, af, laf,
work, lwork, info)
```

説明

このルーチンは、次の三重対角三角行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1) * X = B(ib:ib+n-1, 1:nrhs) \text{ または}$$

$$A(1:n, ja:ja+n-1)^T * X = B(ib:ib+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H * X = B(ib:ib+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$ は、ガウス消去コード PS@(dom_pre)TTRF によって生成される三重対角行列の係数であり、 $A(1:n, ja:ja+n-1)$ と *af* に格納される。

$A(1:n, ja:ja+n-1)$ には、*uplo* の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$ と $A(1:n, ja:ja+n-1)^T$ のどちらの式を解くのかは、パラメータ *trans* を介して、ユーザーによって決定される。

ルーチン [psdttrf](#) は、最初に呼び出さなければならない。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER。 <i>uplo</i> = 'U' の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 <i>uplo</i> = 'L' の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
<i>trans</i>	(グローバル) CHARACTER。 <i>trans</i> = 'N' の場合、 $A(1:n, ja:ja+n-1)$ を解く <i>trans</i> = 'C' の場合、共役転置式 $A(1:n, ja:ja+n-1)$ を解く。
<i>n</i>	(グローバル) INTEGER。分散部分行列 A の次数 ($n \geq 0$)。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。 分散部分行列 $B(ib:ib+n-1, 1:nrhs)$ の列数。 ($nrhs \geq 0$)。
<i>d1</i>	(ローカル)。 REAL (<i>psdttrsv</i> の場合) DOUBLE PRECISION (<i>pddttrsv</i> の場合) COMPLEX (<i>pcdttrsv</i> の場合) COMPLEX*16 (<i>pzdttrsv</i> の場合) 行列の下対角を格納するグローバルベクトルのローカル部分へのポインター。全体的に、 <i>d1</i> (1) は参照されず、 <i>d1</i> は d とアライメントされなければならない。 サイズは $\geq \text{desca}(nb_)$ でなければならない。
<i>d</i>	(ローカル)。 REAL (<i>psdttrsv</i> の場合) DOUBLE PRECISION (<i>pddttrsv</i> の場合) COMPLEX (<i>pcdttrsv</i> の場合) COMPLEX*16 (<i>pzdttrsv</i> の場合) 行列の主対角を格納するグローバルベクトルのローカル部分へのポインター。
<i>du</i>	(ローカル)。 REAL (<i>psdttrsv</i> の場合) DOUBLE PRECISION (<i>pddttrsv</i> の場合) COMPLEX (<i>pcdttrsv</i> の場合) COMPLEX*16 (<i>pzdttrsv</i> の場合) 行列の上対角を格納するグローバルベクトルのローカル部分へのポインター。全体的に、 <i>du</i> (n) は参照されず、 <i>du</i> は d とアライメントされなければならない。
<i>ja</i>	(グローバル) INTEGER。 (A のすべてまたは A の部分行列で) 処理される行列の先頭を指すグローバル配列 a のインデックス。
<i>desca</i>	(グローバルおよびローカル)。INTEGER 配列、次元は (<i>dlen</i>)。 1 <i>d</i> type (<i>dtype_a</i> = 501 または 502) の場合、 $dlen \geq 7$ 。 2 <i>d</i> type (<i>dtype_a</i> = 1) の場合、 $dlen \geq 9$ 。 分散行列 A の配列ディスクリプター。 A のマッピング情報をメモリーに格納する。

<i>b</i>	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdtttrsv の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $11d_b \geq nb$ の配列へのポインター。 右辺 $B(ib:ib+n-1, 1:nrhs)$ のローカル部分を格納する。
<i>ib</i>	(グローバル) INTEGER。 (<i>b</i> のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 <i>b</i> の行インデックス。
<i>descb</i>	(グローバルおよびローカル)。 INTEGER 配列、次元は (<i>dlen</i> _{_)} 。 1 <i>d</i> type (<i>dtype</i> _b = 502) の場合、 <i>dlen</i> ≥ 7 。 2 <i>d</i> type (<i>dtype</i> _b = 1) の場合、 <i>dlen</i> ≥ 9 。 分散行列 B の配列ディスクリプター。 B のマッピング情報をメモリーに格納する。
<i>laf</i>	(ローカル)。 INTEGER。 ユーザー入力の補助非零要素空間 <i>af</i> のサイズ。 $laf \geq 2*(nb+2)$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル)。 REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合) COMPLEX*16 (pzdtttrsv の場合) テンポラリー・ワークスペース。このスペースは、ルーチンの呼び出しの間に上書きされる。 <i>work</i> のサイズは <i>lwork</i> で与えられる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 ユーザー入力ワークスペース <i>work</i> のサイズ。 <i>lwork</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>work</i> (1) に返される。 $lwork \geq 10*npcol+4*nrhs$ 。

出力パラメーター

<i>d1</i>	(ローカル)。 終了時に、行列の係数を含む情報が格納される。
<i>d</i>	終了時に、行列の係数を含む情報が格納される。 サイズは $\geq desca(nb_)$ でなければならない。
<i>b</i>	終了時に、解の分散行列 X のローカル部分が格納される。
<i>af</i>	(ローカル) REAL (psdttrsv の場合) DOUBLE PRECISION (pddttrsv の場合) COMPLEX (pcdttrsv の場合)

COMPLEX*16 (pzdttrsv の場合)

補助非零要素空間。非零要素は、因子分解ルーチン p?dttrf で作成され、af に格納される。因子分解ルーチンの後で、p?dttrs を使って連立 1 次方程式を解く場合、因子分解後に af を変更してはならない。

work

終了時に、work(1) には、lwork の最小値が格納される。

info

(ローカル)。INTEGER。

info = 0 の場合、実行は正常に終了したことを示す。

info < 0 の場合、i 番目の引数が配列で j 番目の成分の値が不正ならば info = -(i*100+j)、i 番目の引数がスカラーで値が不正ならば info = -i。

p?gebd2

直交 / ユニタリー変換を用いて、一般矩形行列を実数二重対角形式に縮退させる (非ブロック化アルゴリズム)。

構文

```
call psgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pdgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pcgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
call pzgebd2(m, n, a, ia, ja, desca, d, e, tauq, taup, work, lwork, info)
```

説明

このルーチンは、直交 / ユニタリー変換を用いて、 $m \times n$ の一般実 / 複素分布行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ を上または下二重対角形式 B に縮退させる。

$$Q^* \text{sub}(A) P = B$$

$m \geq n$ の場合、 B は上二重対角である。 $m < n$ の場合、 B は下二重対角である。

入力パラメーター

m (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の行数。 $(m \geq 0)$ 。

n (グローバル) INTEGER。分散部分行列 $\text{sub}(A)$ の次数。 $(n \geq 0)$ 。

a (ローカル)。

REAL (psgebd2 の場合)

DOUBLE PRECISION (pdgebd2 の場合)

COMPLEX (pcgebd2 の場合)

COMPLEX*16 (pzgebd2 の場合)

ローカルメモリーにある、次元 ($\text{lld}_a, \text{LOCc}(\text{ja}+n-1)$) の配列へのポインター。一般分散行列 $\text{sub}(A)$ のローカル部分を格納する。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合) COMPLEX*16 (pzgebd2 の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq \max(mpa0, nqa0)$ でなければならない。ここで、 $nb = mb_a = nb_a$ 、 $iroffa = \text{mod}(ia-1, nb)$ $iarow = \text{indxg2p}(ia, nb, myrow, rsrc_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb, mycol, csrc_a, npcot)$ 、 $mpa0 = \text{numroc}(m+iroffa, nb, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb, mycol, iacol, npcot)$ 。 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i> 、 <i>mycol</i> 、 <i>nprow</i> 、および <i>npcol</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して求められる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエンタリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	(ローカル)。 終了時に、 $m \geq n$ の場合、 <i>sub(A)</i> の対角および第 1 の優対角成分は、上二重対角行列 <i>B</i> によって上書きされる。対角よりも下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。また、第 1 の優対角よりも上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交行列 <i>P</i> を表現する。 $m < n$ の場合、対角成分と最初の劣対角成分は、下二重対角行列 <i>B</i> によって上書きされる。最初の劣対角よりも下の成分は、配列 <i>tauq</i> とともに、基本リフレクターの積として直交/ユニタリーの行列 <i>Q</i> を表現する。対角よりも上の成分は、配列 <i>taup</i> とともに、基本リフレクターの積として直交行列 <i>P</i> を表す。次の「アプリケーション・ノート」を参照。
<i>d</i>	(ローカル)。 REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合)

	<p>COMPLEX*16 (pzgebd2 の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$ ($m \geq n$ の場合) または $LOCr(ia+\min(m,n)-1)$ (それ以外の場合)。二重対角行列 B の分散対角成分。 $d(i) = a(i,i)$。 d は、分散行列 A に関連付けられる。</p>
<i>e</i>	<p>(ローカル) REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合) COMPLEX*16 (pzgebd2 の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$ ($m \geq n$ の場合) または $LOCr(ia+\min(m,n)-2)$ (それ以外の場合)。二重対角行列 B の分散対角成分。 $m \geq n$ の場合、$e(i) = a(i, i+1)$ ($i = 1, 2, \dots, n-1$) $m < n$ の場合、$e(i) = a(i+1, i)$ ($i = 1, 2, \dots, m-1$) e は、分散行列 A に関連付けられる。</p>
<i>tauq</i>	<p>(ローカル)。 REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合) COMPLEX*16 (pzgebd2 の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$。直交 / ユニタリー行列 Q を表す基本リフレクターのスカラー係数。 <i>tauq</i> は、分散行列 A に関連付けられる。</p>
<i>taup</i>	<p>(ローカル)。 REAL (psgebd2 の場合) DOUBLE PRECISION (pdgebd2 の場合) COMPLEX (pcgebd2 の場合) COMPLEX*16 (pzgebd2 の場合) 配列、次元は $LOCr(ia+\min(m,n)-1)$。直交 / ユニタリー行列 P を表す基本リフレクターのスカラー係数。 <i>taup</i> は、分散行列 A に関連付けられる。</p>
<i>work</i>	<p>終了時に、<i>work</i>(1) は、最小かつ最適な <i>lwork</i> 値を返す。</p>
<i>info</i>	<p>(ローカル) INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info < 0$ の場合、i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$、i 番目の引数がスカラーで値が不正ならば $info = -i$。</p>

アプリケーション・ノート

行列 Q と P は基本リフレクターの積として表現される。

$m \geq n$ の場合、

$$Q = H(1) H(2) \dots H(n) \text{ and } P = G(1) G(2) \dots G(n-1)$$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$$H(i) = I - \tau_{uq} * v * v' \quad \text{および} \quad G(i) = I - \tau_{up} * u * u'$$

ここで、 τ_{uq} と τ_{up} は実 / 複素スカラー、 v と u は実 / 複素ベクトルである。
終了時に、 $v(1:i-1)=0$ 、 $v(i)=1$ 、 $v(i+i:m)$ は

$A(ia+i-ia+m-1, ja+i-1)$ に格納される。
 $u(1:i)=0$ 、 $u(i+1)=1$ 、および $u(i+2:n)$ は、終了時に
 $A(ia+i-1, ja+i+1:ja+n-1)$ に格納される。
 τ_{uq} は $TAUQ(ja+i-1)$ に格納され、 τ_{up} は $TAUP(ia+i-1)$ に格納される。

$m < n$ の場合

$v(1:i)=0$ 、 $v(i+1)=1$ 、および $v(i+2:m)$ は終了時に
 $A(ia+i+1:ia+m-1, ja+i-1)$ に格納される。
 $u(1:i-1)=0$ 、 $u(i)=1$ 、および $u(i+1:n)$ は終了時に
 $A(ia+i-1, ja+i:ja+n-1)$ に格納される。
 τ_{uq} は $TAUQ(ja+i-1)$ に格納され、 τ_{up} は $TAUP(ia+i-1)$ に格納される。

終了時の $\text{sub}(A)$ の内容を次に示す。

$m=6$ 、 $n=5$ ($m > n$) の場合：

$m=5$ 、 $n=6$ ($m < n$) の場合：

$$\begin{bmatrix} d & e & u1 & u1 & u1 \\ v1 & d & e & u2 & u2 \\ v1 & v2 & d & e & u3 \\ v1 & v2 & v3 & d & e \\ v1 & v2 & v3 & v4 & d \\ v1 & v2 & v3 & v4 & v5 \end{bmatrix}$$

$$\begin{bmatrix} d & u1 & u1 & u1 & u1 & u1 \\ e & d & u2 & u2 & u2 & u2 \\ v1 & e & d & u3 & u3 & u3 \\ v1 & v2 & e & d & u4 & u4 \\ v1 & v2 & v3 & e & d & u5 \end{bmatrix}$$

ここで、 d と e は B の対角成分と非対角成分である。 v_i は $H(i)$ を定義するベクトルの成分を表し、 u_i は $G(i)$ を定義するベクトルの成分を表す。

p?gehd2

直交 / ユニタリー相似変換を用いて、一般行列を上 Hessenberg 形式に縮退させる (非ブロック化アルゴリズム)。

構文

```
call psgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pdgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pcgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
call pzgehd2(n, ilo, ihi, a, ia, ja, desca, tau, work, lwork, info)
```

説明

このルーチンは、直交 / ユニタリーの相似変換 $Q' * \text{sub}(A) * Q = H$ によって、実 / 複素一般行列 $\text{sub}(A)$ を上 Hessenberg 形式 H に縮退させる。ここで、 $\text{sub}(A) = A(ia+n-1:ia+n-1, ja+n-1:ja+n-1)$ である。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。分散部分行列 <i>A</i> の次数。($n \geq 0$)。
<i>ilo, ihi</i>	(グローバル) INTEGER。sub(<i>A</i>) はすでに行 <i>ia:ia+ilo-2</i> と列 <i>ja:ja+jlo-2</i> 、行 <i>ia+ihi:ia+n-1</i> と <i>ja+jhi:ja+n-1</i> で上三角になっていると仮定する。詳細は、「アプリケーション・ノート」を参照。 $n > 0$ の場合は $1 \leq ilo \leq ihi \leq n$ 、それ以外の場合は $ilo = 1$ かつ $ihi = n$ 。
<i>a</i>	(ローカル) REAL (psgehd2 の場合) DOUBLE PRECISION (pdgehd2 の場合) COMPLEX (pcgehd2 の場合) COMPLEX*16 (pzgehd2 の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , $LOCc(ja+n-1)$) の配列へのポインター。呼び出し時に、この配列は、因子分解する $n \times n$ の一般分散行列 sub(<i>A</i>) のローカル部分を含む。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>A</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psgehd2 の場合) DOUBLE PRECISION (pdgehd2 の場合) COMPLEX (pcgehd2 の場合) COMPLEX*16 (pzgehd2 の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル)。INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq nb + \max(npa0, nb)$ でなければならない。ここで、 $nb = mb_a = nb_a$ 、 $iroffa = \text{mod}(ia-1, nb)$ $iarow = \text{indxg2p}(ia, nb, myrow, rsrc_a, nprow)$ 、 $npa0 = \text{numroc}(ihi+iroffa, nb, myrow, iarow, nprow)$ 。 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i> 、 <i>mycol</i> 、 <i>nprow</i> 、および <i>npcol</i> は、サブルーチン <i>blacs_gridinfo</i> で呼び出して求められる。 <i>lwork = -1</i> の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

- a** (ローカル) 終了時に、**sub(A)** の上三角と第 1 の劣対角は、上 Hessenberg 行列 **H** および第 1 の劣対角よりも下の成分で上書きされ、配列 **tau** とともに、基本リフレクターの積として直交/ユニタリーの行列 **Q** を表現する。次の「アプリケーション・ノート」を参照。
- tau** (ローカル)
 REAL (psgehd2 の場合)
 DOUBLE PRECISION (pdgehd2 の場合)
 COMPLEX (pcgehd2 の場合)
 COMPLEX*16 (pzgehd2 の場合)
 配列、次元は $LOCc(ja+n-2)$ 。基本リフレクターのスカラー係数 (次の「アプリケーション・ノート」を参照)。tau の成分 $ja:ja+ilo-2$ と $ja+ihi:ja+n-2$ はゼロに設定される。tau は、分散行列 **A** に関連付けられる。
- work** 終了時に、**work(1)** は、最小かつ最適な **lwork** 値を返す。
- info** (ローカル)。INTEGER。
info=0 の場合、正常に終了したことを示す。
info<0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば **info** = - (*i**100+*j*)、*i* 番目の引数がスカラーで値が不正ならば **info** = -*i*。

アプリケーション・ノート

行列 **Q** を (*ihi-ilo*) 個の基本リフレクターの積として表現する。

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1)$$

それぞれの **H(i)** は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、**tau** は実/複素スカラー、**v** は実/複素ベクトルで $v(1:i) = 0$ 、 $v(i+1) = 1$ および $v(ihi+1:n) = 0$ 。v(*i*+2:*ihi*) は、終了時に、 $A(ia+ilo+i:ia+ihi-1, ia+ilo+i-2)$ に、**tau** は $\tau(ja+ilo+i-2)$ に格納される。

終了時の $A(ia:ia+n-1, ja:ja+n-1)$ の内容を次に示す ($n=7$ 、 $ilo=2$ 、および $ihi=6$ の場合)。

入力

出力

$\begin{bmatrix} a & a & a & a & a & a & a \\ & a & a & a & a & a & a \\ & & a & a & a & a & a \\ & & & a & a & a & a \\ & & & & a & a & a \\ & & & & & a & a \\ & & & & & & a \end{bmatrix}$	$\begin{bmatrix} a & a & h & h & h & h & a \\ & a & h & h & h & h & a \\ & & h & h & h & h & h \\ & & & v2 & h & h & h & h \\ & & & & v2 & v3 & h & h & h \\ & & & & & v2 & v3 & v4 & h & h & h \\ & & & & & & & & & & a \end{bmatrix}$
---	---

ここで、 a は元の行列 $\text{sub}(A)$ の成分を表し、 h は上 Hessenberg 行列 H の更新された成分を表し、 v_i は $H(ja+ilo+i-2)$ を定義するベクトルの成分を表している。

p?gelq2

一般矩形行列の LQ 因子分解を計算する (非ブロック化アルゴリズム)。

構文

```
call psgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call pdgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call pcgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call pzgelq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

説明

このルーチンは、 $m \times n$ の実 / 複素行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = L * Q$ の LQ 因子分解を計算する。

入力パラメーター

m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
a	(ローカル)。 REAL (psgelq2 の場合) DOUBLE PRECISION (pdgelq2 の場合) COMPLEX (pcgelq2 の場合) COMPLEX*16 (pzgelq2 の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
$work$	(ローカル)。 REAL (psgelq2 の場合) DOUBLE PRECISION (pdgelq2 の場合) COMPLEX (pcgelq2 の場合) COMPLEX*16 (pzgelq2 の場合) 次元 ($lwork$) のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。
 配列 *work* の次元。
lwork はローカル入力であり、 $lwork \geq nq0 + \max(1, mp0)$ でなければならない。ここで、

$$\begin{aligned} iroff &= \text{mod}(ia-1, mb_a), \quad icoff = \text{mod}(ja-1, nb_a), \\ iarow &= \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow), \\ iacol &= \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcrow), \\ mp0 &= \text{numroc}(m+iroff, mb_a, myrow, iarow, nprow), \\ nq0 &= \text{numroc}(n+icoff, nb_a, mycol, iacol, npcrow), \end{aligned}$$

indxg2p および *numroc* は、ScaLAPACK ツール関数である。
myrow, *mycol*, *nprow*, および *npcrow* は、サブルーチン *blacs_gridinfo* を呼び出して求められる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a (ローカル)。
 終了時に、*sub(A)* の対角成分とその下の成分は、 $m \times \min(m, n)$ の下台形行列 L ($m \leq n$ の場合、 L は下三角行列) によって上書きされる。対角よりも上の成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリーの行列 Q を表現する (次の「アプリケーション・ノート」を参照)。

tau (ローカル)。
 REAL (*psgelq2* の場合)
 DOUBLE PRECISION (*pdgelq2* の場合)
 COMPLEX (*pcgelq2* の場合)
 COMPLEX*16 (*pzgelq2* の場合)
 配列、次元は $LOCr(ia+\min(m,n)-1)$ 。基本リフレクターのスカラ係数が格納される。*tau* は、分散行列 A に関連付けられる。

work 終了時に、*work(1)* は、最小かつ最適な *lwork* 値を返す。

info (ローカル)。INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 i 番目の引数がスカラーで値が不正ならば $info = -i$ 。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$\begin{aligned} Q &= H(ia+k-1) H(ia+k-2) \dots H(ia) \text{ (実数型の場合)} \\ Q &= H(ia+k-1)' H(ia+k-2)' \dots H(ia)' \text{ (複素数型の場合)} \end{aligned}$$

ここで、 $k = \min(m, n)$ 。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(1:i-1) = 0$ および $v(i) = 1$ 。
終了時に、 $v(i+1:n)$ (実数型の場合) と $\text{conjg}(v(i+1:n))$ (複素数型の場合) は
 $A(ia+i-1, ja+i:ja+n-1)$ に、 τ は $TAU(ia+i-1)$ に格納される。

p?sgeql2

一般矩形行列の QL 因子分解を計算する (非ブ
ロック化アルゴリズム)。

構文

```
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeql2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

説明

このルーチンは、 $m \times n$ の実 / 複素行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * L$ の QL 因子分解を計算する。

入力パラメーター

m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
a	(ローカル)。 REAL (psgeql2 の場合) DOUBLE PRECISION (pdgeql2 の場合) COMPLEX (pcgeql2 の場合) COMPLEX*16 (pzgeql2 の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列 へのポインター。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカ ル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最 初の列を示す、グローバル配列 a の行インデックスと列イン デックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 A の配列ディスクリプター。
$work$	(ローカル)。 REAL (psgeql2 の場合) DOUBLE PRECISION (pdgeql2 の場合)

COMPLEX (pcgeql2 の場合)
 COMPLEX*16 (pzgeql2 の場合)
 次元 (*lwork*) のワークスペース配列。

lwork (ローカルまたはグローバル) INTEGER。
 配列 *work* の次元。
lwork はローカル入力であり、 $lwork \geq mp0 + \max(1, nq0)$ でなければならない。ここで、
 $iroff = \text{mod}(ia-1, mb_a)$ 、 $icoff = \text{mod}(ja-1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcil)$ 、
 $mp0 = \text{numroc}(m+iroff, mb_a, myrow, iarow, nprow)$ 、
 $nq0 = \text{numroc}(n+icoff, nb_a, mycol, iacol, npcil)$ 、
 indxg2p および numroc は、ScaLAPACK ツール関数である。
 myrow、mycol、nprow、および npcil は、サブルーチン
 blacs_gridinfo を呼び出して求められる。

lwork = -1 の場合、*lwork* はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する *work* 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a (ローカル)。
 終了時に、 $m \geq n$ の場合、分散部分行列 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の下三角部分は、 $n \times n$ の下三角行列 L によって上書きされる。 $m \leq n$ の場合、 $(n-m)$ 番目の優対角成分とその下の各成分は、 $m \times n$ 下台形行列 L によって上書きされる。残りの成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリ行列 Q を表現する(次の「アプリケーション・ノート」を参照)。

tau (ローカル)。
 REAL (psgeql2 の場合)
 DOUBLE PRECISION (pdgeql2 の場合)
 COMPLEX (pcgeql2 の場合)
 COMPLEX*16 (pzgeql2 の場合)
 配列、次元は $LOCc(ja+n-1)$ 。基本リフレクターのスカラー係数が格納される。*tau* は、分散行列 A に関連付けられる。

work 終了時に、*work*(1) は、最小かつ最適な *lwork* 値を返す。

info (ローカル)。INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば $info = -(i*100+j)$ 、*i* 番目の引数がスカラーで値が不正でならば $info = -i$ であることを示す。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$Q = H(ja+k-1) \dots H(ja+1) H(ja)$ 。ここで、 $k = \min(m, n)$ 。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

τ は実数 / 複素スカラー、 v は実数 / 複素のベクトルで
 $v(m-k+i+1:m) = 0$ および $v(m-k+i) = 1$ 。終了時に、 $v(1:m-k+i-1)$ は
 $A(ia:ia+m-k+i-2, ja+n-k+i-1)$ に、 τ は $TAU(ja+n-k+i-1)$ に格納される。

p?geqr2

一般矩形行列の QR 因子分解を計算する (非ブ
 ロック化アルゴリズム)。

構文

```
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgeqr2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

説明

このルーチンは、 $m \times n$ の実 / 複素行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = Q * R$ の QR 因子分解を計算する。

入力パラメーター

m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
a	(ローカル)。 REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-1)$) の配列 へのポインター。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカ ル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最 初の列を示す、グローバル配列 a の行インデックスと列イン デックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 A の配列ディスクリプター。

<i>work</i>	(ローカル)。 REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル)。INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq mp0 + \max(1, nq0)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb_a)$, $icoff = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcrow)$ 、 $mp0 = \text{numroc}(m+iroff, mb_a, myrow, iarow, nprow)$ 、 $nq0 = \text{numroc}(n+icoff, nb_a, mycol, iacol, npcrow)$ 、 indxg2p および numroc は、ScaLAPACK ツール関数である。 $myrow$, $mycol$, $nprow$ 、および $npcol$ は、サブルーチン blacs_gridinfo を呼び出して求められる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。
出力パラメーター	
<i>a</i>	(ローカル)。 終了時に、 $\text{sub}(A)$ の対角成分とその上の成分は、 $\min(m, n) \times n$ の上台形行列 R ($m \geq n$ の場合、 R は上三角行列) によって上書きされる。対角よりも下の成分は、配列 <i>tau</i> とともに基本リフレクターの積として直交/ユニタリーの行列 Q を表現する(次の「アプリケーション・ノート」を参照)。
<i>tau</i>	(ローカル)。 REAL (psgeqr2 の場合) DOUBLE PRECISION (pdgeqr2 の場合) COMPLEX (pcgeqr2 の場合) COMPLEX*16 (pzgeqr2 の場合) 配列、次元は $LOCc(ja+\min(m,n)-1)$ 。基本リフレクターのスカラ係数が格納される。 <i>tau</i> は、分散行列 A に関連付けられる。
<i>work</i>	終了時に、 <i>work</i> (1) は、最小かつ最適な <i>lwork</i> 値を返す。
<i>info</i>	(ローカル)。INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 <i>i</i> 番目の引数がスカラーで値が不正でならば $info = -i$ であることを示す。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$$Q = H(ja) H(ja+1) \dots H(ja+k-1)。ここで、k = \min(m, n)。$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(j) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(1:i-1) = 0$ および $v(i) = 1$ 。終了時に、 $v(i+1:m)$ は $A(ia+i:ia+m-1, ja+i-1)$ に、 τ は $TAU(ja+i-1)$ に格納される。

p?gerq2

一般矩形行列の RQ 因子分解を計算する (非ブロック化アルゴリズム)。

構文

```
call psggerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
call psgerq2(m, n, a, ia, ja, desca, tau, work, lwork, info)
```

説明

このルーチンは、 $m \times n$ の実 / 複素分布行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1) = R * Q$ の RQ 因子分解を計算する。

入力パラメーター

m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
a	(ローカル)。 REAL (psgerq2 の場合) DOUBLE PRECISION (pdgerq2 の場合) COMPLEX (pcgerq2 の場合) COMPLEX*16 (pzgerq2 の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列へのポインター。因子分解する $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (<i>psgerq2</i> の場合) DOUBLE PRECISION (<i>pdgerq2</i> の場合) COMPLEX (<i>pcgerq2</i> の場合) COMPLEX*16 (<i>pzgerq2</i> の場合) 次元 (<i>lwork</i>) のワークスペース配列。
<i>lwork</i>	(ローカルまたはグローバル)。INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq nq0 + \max(1, mp0)$ でなければならない。ここで、 $iroff = \text{mod}(ia-1, mb_a), icoff = \text{mod}(ja-1, nb_a),$ $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow),$ $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcold),$ $mp0 = \text{numroc}(m+iroff, mb_a, myrow, iarow, nprow),$ $nq0 = \text{numroc}(n+icoff, nb_a, mycol, iacol, npcold),$ indxg2p および numroc は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcold$ は、サブルーチン <code>blacs_gridinfo</code> で呼び出して求められる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。
出力パラメーター	
<i>a</i>	(ローカル)。 終了時に、 $m \leq n$ の場合、 $A(ia+m-n:ia+m-1, ja:ja+n-1)$ の上三角部分は、 $m \times m$ の上三角行列 <i>R</i> によって上書きされる。 $m \geq n$ の場合、 $(m-n)$ 番目の劣対角成分とその上の成分は、 $m \times n$ 上台形行列 <i>R</i> によって上書きされる。残りの成分は、配列 <i>tau</i> とともに、基本リフレクターの積として直交/ユニタリー行列 <i>Q</i> を表現する (次の「アプリケーション・ノート」を参照)。
<i>tau</i>	(ローカル)。 REAL (<i>psgeqr2</i> の場合) DOUBLE PRECISION (<i>pdgeqr2</i> の場合) COMPLEX (<i>pcgeqr2</i> の場合) COMPLEX*16 (<i>pzgeqr2</i> の場合) 配列、次元は $LOCr(ia+m-1)$ 。基本リフレクターのスカラー係数が格納される。 <i>tau</i> は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	終了時に、 <i>work</i> (1) は、最小かつ最適な <i>lwork</i> 値を返す。

info (ローカル) INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i**100+*j*)、*i* 番目の引数がスカラーで値が不正ならば *info* = -*i*。

アプリケーション・ノート

行列 Q は基本リフレクターの積として表現される。

$Q = H(ia)H(ia+1) \dots H(ia+k-1)$ (実数型の場合)
 $Q = H(ia)'H(ia+1)' \dots H(ia+k-1)'$ (複素数型の場合)

ここで、 $k = \min(m, n)$ 。

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(n-k+i+1:n) = 0$ および $v(n-k+i) = 1$ 。終了時に、 $v(1:n-k+i-1)$ (実数型の場合) または $\text{conjg}(v(1:n-k+i-1))$ (複素数型の場合) は $A(ia+m-k+i-1, ja:ja+n-k+i-2)$ に、 τ は $TAU(ia+m-k+i-1)$ に格納される。

p?getf2

一般行列の LU 因子分解を、行交換を伴う部分ピボット演算を用いて計算する (ローカルブロック化アルゴリズム)。

構文

```
call psgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pdgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pcgetf2(m, n, a, ia, ja, desca, ipiv, info)
call pzgetf2(m, n, a, ia, ja, desca, ipiv, info)
```

説明

このルーチンは、行交換を伴う部分ピボット演算を用いて、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の LU 因子分解を計算する。

因子分解は形式 $\text{sub}(A) = P * L * U$ を持ち、 P は置換行列、 L は単位対角成分を持つ下三角 ($m > n$ の場合、下台形)、 U は上三角 ($m < n$ の場合、上台形) である。このルーチンは、アルゴリズムの right-looking 法により並列化されたレベル 2 BLAS バージョンである。

入力パラメーター

m (グローバル) INTEGER。
 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。
 ($m \geq 0$)。

<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。 ($nb_a - \text{mod}(ja-1, nb_a) \geq n \geq 0$)。
<i>a</i>	(ローカル)。 REAL (psgetf2 の場合) DOUBLE PRECISION (pdgetf2 の場合) COMPLEX (pcgetf2 の場合) COMPLEX*16 (pzgetf2 の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列へのポインター。 $m \times n$ の分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は ($LOCr(m_a) + mb_a$)。この配列には、ピボット情報が格納される。 <i>ipiv(i)</i> -> ローカル行 <i>i</i> と交換されたグローバル行。この配列は、分散行列 <i>A</i> に関連付けられる。
<i>info</i>	(ローカル) INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> であることを示す。 <i>info</i> > 0 の場合、 <i>info</i> = <i>k</i> ならば $u(ia+k-1, ja+k-1)$ は完全に 0 であることを示す。因子分解は完了したが、係数 <i>u</i> は完全に特異で、連立方程式の解の算出に <i>D</i> を使用すると 0 での除算が発生する。

p?labrd

直交/ユニタリー変換を用いて、一般矩形行列 *A* の最初の *nb* 行と列を実数二重対角形式に縮退させ、*A* の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

```
call pslabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
call pdlabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
```

```
call pslabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
```

```
call pzlabrd(m, n, nb, a, ia, ja, desca, d, e, tauq, taup, x, ix, jx, descx, y,
            iy, jy, descy, work)
```

説明

このルーチンは、直交/ユニタリー変換 $Q' * A * P$ を用いて $m \times n$ の一般実/複素分布行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の最初の nb 行と列を上または下二重対角形式に縮退させ、 $\text{sub}(A)$ の縮退されていない部分に変換を適用するために必要な行列 X と Y を返す。

$m \geq n$ の場合、 $\text{sub}(A)$ は上二重対角形式に縮退される。

$m < n$ の場合、 $\text{sub}(A)$ は下二重対角形式に縮退される。

このルーチンは、[p?gebrd](#) から呼び出される補助ルーチンである。

入力パラメーター

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
<i>nb</i>	(グローバル) INTEGER。縮退の対象となる $\text{sub}(A)$ の先頭の行数と列数。
<i>a</i>	(ローカル)。 REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合) ローカルメモリーにある、次元 ($lld_a, LOCc(ja+n-1)$) の配列へのポインター。一般分散行列 $\text{sub}(A)$ のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _{_)。分散行列 <i>A</i> の配列ディスクリプター。}
<i>ix, jx</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(X)$ の最初の行と最初の列を示す、グローバル配列 <i>x</i> の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _{_)。分散行列 <i>X</i> の配列ディスクリプター。}
<i>iy, jy</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(Y)$ の最初の行と最初の列を示す、グローバル配列 <i>y</i> の行インデックスと列インデックス。

descy (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *Y* の配列ディスクリプター。

work (ローカル)
 REAL (*pslabrd* の場合)
 DOUBLE PRECISION (*pdlabrd* の場合)
 COMPLEX (*pclabrd* の場合)
 COMPLEX*16 (*pzlabrd* の場合)
 ワークスペース配列、次元は (*lwork*)。
 $lwork \geq nb_a + nq$ 、
 $nq = \text{numroc}(n + \text{mod}(ia-1, nb_y), nb_y, mycol, iacol, npcol)$ $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcol)$
indxg2p および *numroc* は、ScaLAPACK ツール関数である。
myrow、*mycol*、*nprow*、および *npcol* は、サブルーチン *blacs_gridinfo* を呼び出して求められる。

出力パラメーター

a (ローカル)
 終了時に、行列の最初の *nb* 行と列は上書きされる。分散行列 *sub(A)* の残りの部分に変更されない。
 $m \geq n$ の場合、最初の *nb* 列の対角線とその下の成分は、配列 *tauq* とともに、基本リフレクターの積として直交/ユニタリー行列 *Q* を表現する。最初の *nb* 行の対角よりも上の成分は、配列 *taup* とともに、基本リフレクターの積として直交/ユニタリー行列 *P* を表現する。
 $m < n$ の場合、最初の *nb* 列の対角よりも下の成分は、配列 *tauq* とともに、基本リフレクターの積として直交/ユニタリー行列 *Q* を表現する。最初の *nb* 行の対角線とその上の成分は、配列 *taup* とともに、基本リフレクターの積として直交/ユニタリー行列 *P* を表現する。次の「アプリケーション・ノート」を参照。

d (ローカル)。
 REAL (*pslabrd* の場合)
 DOUBLE PRECISION (*pdlabrd* の場合)
 COMPLEX (*pclabrd* の場合)
 COMPLEX*16 (*pzlabrd* の場合)
 配列、次元は $LOCr(ia+\min(m,n)-1)$ ($m \geq n$ の場合) または $LOCc(ja+\min(m,n)-1)$ (それ以外の場合)。二重対角分散行列 *B* の非対角成分。
 $d(i) = A(ia+i-1, ja+i-1)$ 。
d は、分散行列 *A* に関連付けられる。

e (ローカル)。
 REAL (*pslabrd* の場合)
 DOUBLE PRECISION (*pdlabrd* の場合)
 COMPLEX (*pclabrd* の場合)
 COMPLEX*16 (*pzlabrd* の場合)
 配列、次元は $LOCr(ia+\min(m,n)-1)$ ($m \geq n$ の場合) または $LOCc(ja+\min(m,n)-2)$ (それ以外の場合)。二重対角分散行列 *B* の非対角成分。

$m \geq n$ の場合、 $E(i) = A(ia+i-1, ja+i)$ ($i = 1, 2, \dots, n-1$)。
 $m < n$ の場合、 $E(i) = A(ia+i, ja+i-1)$ ($i = 1, 2, \dots, m-1$)。
 E は、分散行列 A に関連付けられる。

$tauq, taup$	<p>(ローカル)。 REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合) 配列、次元は $LOCc(ja+\min(m, n)-1)$ ($tauq$ の場合) または $LOCr(ia+\min(m, n)-1)$ ($taup$ の場合)。直交/ユニタリ行列 Q ($tauq$ の場合)、P ($taup$ の場合) を表す基本リフレクターのスカ ラー係数が格納される。$tauq$ と $taup$ は、分散行列 A に関連付 けられる。次の「アプリケーション・ノート」を参照。</p>
x	<p>(ローカル) REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合) ローカルメモリーにある、次元 (lld_x, nb) の配列へのポイン ター。終了時に、$sub(A)$ の縮退されていない部分の更新に必要な $m \times nb$ の分散行列 $X(ix:ix+m-1, jx:jx+nb-1)$ のローカル部分が 格納される。</p>
y	<p>(ローカル)。 REAL (pslabrd の場合) DOUBLE PRECISION (pdlabrd の場合) COMPLEX (pclabrd の場合) COMPLEX*16 (pzlabrd の場合) ローカルメモリーにある、次元 (lld_y, nb) の配列へのポイン ター。終了時に、$sub(A)$ の縮退されていない部分の更新に必要な $n \times nb$ の分散行列 $Y(iy:iy+n-1, jy:jy+nb-1)$ のローカル部分が 格納される。</p>

アプリケーション・ノート

行列 Q と P は基本リフレクターの積として表現される。

$$Q = H(1)H(2) \dots H(nb) \quad \text{および} \quad P = G(1)G(2) \dots G(nb)$$

それぞれの $H(i)$ と $G(i)$ は次のような形式を持つ。

$$H(i) = I - tauq * v * v' \quad \text{および} \quad G(i) = I - taup * u * u'$$

ここで、 $tauq$ と $taup$ は実/複素スカラー、 v と u は実/複素ベクトルである。

$m \geq n$ の場合、終了時に、 $v(1:i-1) = 0$ 、 $v(i) = 1$ 、 $v(i:m)$ は

$A(ia+i-1:ia+m-1, ja+i-1)$ に格納される。終了時に、 $u(1:i) = 0$ 、 $u(i+1) = 1$ 、
 $u(i+1:n)$ は $A(ia+i-1, ja+i:ja+n-1)$ に格納される。 $tauq$ は $TAUQ(ja+i-1)$ に、 $taup$
は $TAUP(ia+i-1)$ に格納される。

$m < n$ の場合、終了時に、 $v(1:i) = 0$, $v(i+1) = 1$, $v(i+1:m)$ は

$A(ia+i+1:ia+m-1, ja+i-1)$ に格納される。終了時に、 $u(1:i-1) = 0$, $u(i) = 1$, $u(i:n)$ は $A(ia+i-1, ja+i:ja+n-1)$ に格納される。 $tauq$ は $TAUQ(ja+i-1)$ に、 $taup$ は $TAUP(ia+i-1)$ に格納される。ベクトル v と u の成分は $m \times nb$ の行列 V と $nb \times n$ の行列 U' を形成する。行列 $sub(A)$ の縮退されていない部分に次の形式のブロック更新を使って変換を適用するために、これらの行列と行列 X および Y が必要となる。
 $sub(A) := sub(A) - V*Y' - X*U'$ 。終了時の $sub(A)$ の内容を次に示す ($nb=2$ の場合)。

$m = 6, n = 5 (m > n)$ の場合：

$$\begin{bmatrix} 1 & 1 & u1 & u1 & u1 \\ v1 & 1 & 1 & u2 & u2 \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \end{bmatrix}$$

$m = 5, n = 6 (m < n)$ の場合：

$$\begin{bmatrix} 1 & u1 & u1 & u1 & u1 & u1 \\ 1 & 1 & u2 & u2 & u2 & u2 \\ v1 & 1 & a & a & a & a \\ v1 & v2 & a & a & a & a \\ v1 & v2 & a & a & a & a \end{bmatrix}$$

a は元の行列の変更されていない成分を意味し、 v_i は $H(i)$ を定義するベクトルの成分を意味し、 u_i は $G(i)$ を定義するベクトルの成分を意味する。

p?lacon

行列-ベクトル積の評価にリバース・コミュニケーションを使用して正方行列の 1- ノルムを推定する。

構文

```
call pslacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
call pdlacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
call pclacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
call pzlacon(n, v, iv, jv, descv, x, ix, jx, descx, isgn, est, kase)
```

説明

このルーチンは、実/ユニタリー分散行列 A の 1- ノルムを推定する。行列-ベクトル積の評価にはリバース・コミュニケーションが使用される。 x と v は、分散行列 A でアライメントされている。この情報は、 iv 、 ix 、 $descv$ 、および $descx$ 内に暗黙的に含まれる。

入力パラメーター

n (グローバル)。INTEGER。
 乱数ベクトル v と x の長さ ($n \geq 0$)。

v (ローカル)。
 REAL (pslacon の場合)
 DOUBLE PRECISION (pdlacon の場合)
 COMPLEX (pclacon の場合)

	COMPLEX*16 (pzlacon の場合) ローカルメモリーにある、次元 $LOCr(n+\text{mod}(iv-1, mb_v))$ の配列へのポインター。最終戻りでは、 $v = a*w$ 。 ここで、 $est = \text{norm}(v)/\text{norm}(w)$ (w は返されない)。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 v の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 V の配列ディスクリプター。
<i>x</i>	(ローカル)。 REAL (pslacon の場合) DOUBLE PRECISION (pdlacon の場合) COMPLEX (pclacon の場合) COMPLEX*16 (pzlacon の場合) ローカルメモリーにある、次元 $LOCr(n+\text{mod}(ix-1, mb_x))$ の配列へのポインター。
<i>ix, jx</i>	(グローバル) INTEGER。それぞれ、部分行列 X の最初の行と最初の列を示す、グローバル配列 x の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。分散行列 X の配列ディスクリプター。
<i>isgn</i>	(ローカル) INTEGER。 配列、次元は $LOCr(n+\text{mod}(ix-1, mb_x))$ 。 <i>isgn</i> は、 x および v でアライメントされている。
<i>kase</i>	(ローカル) INTEGER。 $p?lacon$ を初めて呼び出すときは <i>kase</i> はゼロでなければならない。

出力パラメーター

<i>x</i>	(ローカル) 中間戻りでは、 X は次の値で上書きされる。 $A * X$ (<i>kase</i> =1 の場合) $A' * X$ (<i>kase</i> =2 の場合) また、他のパラメーターは変更しない状態で $p?lacon$ を再度呼び出さなければならない。
<i>est</i>	(グローバル) REAL (単精度の場合)。 DOUBLE PRECISION (倍精度の場合)。
<i>kase</i>	(ローカル) INTEGER。 中間戻りでは、 <i>kase</i> は 1 か 2 となり、 X が $A * X$ または $A' * X$ によって上書きされるかどうかを示す。 $p?lacon$ からの最終戻りでは、 <i>kase</i> は再びゼロになる。

p?laconsb

2つの連続する小さい対角成分を検出する。

```
call pslaconsb(a, desca, i, l, m, h44, h33, h43h34, buf, lwork)
call pdlaconsb(a, desca, i, l, m, h44, h33, h43h34, buf, lwork)
```

説明

このルーチンは、*h44*、*h33*、および *h43h34* で与えられる倍精度シフト *QR* の反復法で開始する影響を分析し、このプロセスによって無視できる程度の劣対角成分が作成されるかどうか調べ、2つの連続する小さい対角成分を検出する。

入力パラメーター

a (グローバル)。
REAL (pslaconsb の場合)
DOUBLE PRECISION (pdlaconsb の場合)
配列、次元は (*desca* (*lld_*),*)。三重対角部分をスキャンされた Hessenberg 行列を格納する。終了時に変更されない。

descx (グローバルおよびローカル) INTEGER。
次元 (*dlen_*) の配列。分散行列 *A* の配列ディスクリプター。

i (グローバル) INTEGER。
A の縮退されていない部分行列の一番下のグローバル位置。終了時に変更されない。

l (グローバル) INTEGER。
A の縮退されていない部分行列の一番上のグローバル位置。終了時に変更されない。

h44, *h33*

h43h34 (グローバル)。
REAL (pslaconsb の場合)
DOUBLE PRECISION (pdlaconsb の場合)
これらの3種類の値は、倍精度シフト *QR* の反復法で使用される。

lwork (グローバル) INTEGER。
この値は、 $7 * \text{ceil}(\text{ceil}((i-1)/hbl) / \text{lcm}(nrow, npc1))$ 以上でなければならない。lcm は公倍数以上で、*nrow**xnpc1* は論理グリッドサイズである。

出力パラメーター

m (グローバル)。
終了時に、倍精度シフト *QR* の開始位置を格納する。次の条件を満たす。
 $1 \leq m \leq i-2$

<i>buf</i>	(ローカル)。 REAL (pslaconsb の場合) DOUBLE PRECISION (pdlaconsb の場合) サイズ <i>lwork</i> の配列。
<i>lwork</i>	(グローバル)。 終了時に、 <i>lwork</i> は、ワークバッファのサイズである。

p?lacp2

分散行列の一部または全部を他の分散行列にコピーする。

構文

```
call pslacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pdlacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pclacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pzlacp2(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
```

説明

このルーチンは、分散行列 *A* の一部または全部を他の分散行列 *B* にコピーする。コミュニケーションが実行されない場合、p?lacp2 はローカルコピー $\text{sub}(A) := \text{sub}(B)$ を実行する。ここで、 $\text{sub}(A)$ は $A(ia:ia+m-1, ja:ja+n-1)$ 、 $\text{sub}(B)$ は $B(ib:ib+m-1, jb:jb+n-1)$ である。

p?lacp2 は、行列オペランドの次元のみ再分配する。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER。 分散行列 $\text{sub}(A)$ のコピーする部分を指定する。 'U' の場合、上三角部分をコピーする。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。 'L' の場合、下三角部分をコピーする。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。 それ以外の場合、行列 $\text{sub}(A)$ の全部をコピーする。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
<i>a</i>	(ローカル)。 REAL (pslacp2 の場合) DOUBLE PRECISION (pdlacp2 の場合) COMPLEX (pclacp2 の場合) COMPLEX*16 (pzlacp2 の場合)

	ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。 $m \times n$ の分散行列 $sub(A)$ のローカル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
ib, jb	(グローバル) INTEGER。それぞれ、 $sub(B)$ の最初の行と最初の列を示す、グローバル配列 B の行インデックスと列インデックス。
$descb$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 B の配列ディスクリプター。

出力パラメーター

b	(ローカル)。 REAL (pslap2 の場合) DOUBLE PRECISION (pdlap2 の場合) COMPLEX (pclap2 の場合) COMPLEX*16 (pzlap2 の場合) ローカルメモリーにある、次元 (lld_b , $LOCc(jb+n-1)$) の配列へのポインター。終了時に、次のように設定された、分散行列 $sub(B)$ のローカル部分が格納される。 $uplo = 'U'$ の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(1 \leq i \leq j, 1 \leq j \leq n)$ $uplo = 'L'$ の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(j \leq i \leq m, 1 \leq j \leq n)$ それ以外の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1)$ $(1 \leq i \leq m, 1 \leq j \leq n)$
-----	--

p?lap3

グローバル並列配列からローカル複製配列にコピーする。逆の場合も同じ。

構文

```
call pslap3(m, i, a, desca, b, ldb, ii, jj, rev)
call pdlap3(m, i, a, desca, b, ldb, ii, jj, rev)
```

説明

このルーチンは、グローバル並列配列からローカル複製配列にコピーしたり、逆にローカル複製配列からグローバル並列配列にコピーする補助ルーチンである。コピーされた部分行列全体は1つ以上の成分に配置される点に注意する。配置先成分は厳密に指定することができる。すべての成分、あるいは成分の1行または1列を指定することもできる。

入カパラメーター

<i>m</i>	(グローバル) INTEGER。 <i>m</i> はコピーされた正方部分行列の次数。 $m \geq 0$ 。終了時に変更されない。
<i>i</i>	(グローバル) INTEGER。 $A(i, i)$ は、コピーを開始するグローバル位置。終了時に変更されない。
<i>a</i>	(グローバル) REAL (pslacc3 の場合) DOUBLE PRECISION (pdlacc3 の場合) 配列、次元は (<i>desca</i> (<i>lld</i>),*)。コピー先またはコピー元の並列行列。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>b</i>	(ローカル)。 REAL (pslacc3 の場合) DOUBLE PRECISION (pdlacc3 の場合) 配列、次元は (<i>ldb</i> , <i>m</i>)。 <i>rev</i> = 0 の場合、配列 $A(i:i+m-1, i:i+m-1)$ のグローバル成分。 <i>rev</i> = 1 の場合、終了時に変更されない。
<i>ldb</i>	(ローカル) INTEGER。 <i>B</i> のリーディング・ディメンジョン。
<i>ii</i>	(グローバル) INTEGER <i>rev</i> 0 と 1 を使用すると、データを再度送信したり、返したりすることができる。 <i>rev</i> = 0 の場合、 <i>ii</i> は複製 <i>B</i> を受け取る成分の行インデックスである。 $ii \geq 0, jj \geq 0$ の場合、成分 (<i>ii</i> , <i>jj</i>) がデータを受け取る。 $ii = -1, jj \geq 0$ の場合、列 <i>jj</i> のすべての行がデータを受け取る。 $ii \geq 0, jj = -1$ の場合、行 <i>ii</i> のすべての列がデータを受け取る。 $ii = -1, jj = -1$ の場合、すべての成分がデータを受け取る。 <i>rev</i> != 0 の場合、 <i>ii</i> は複製 <i>B</i> を送信する成分の行インデックスである。
<i>jj</i>	(グローバル) INTEGER。上述の <i>ii</i> と同じ。
<i>rev</i>	(グローバル) INTEGER。 <i>rev</i> = 0 を使用して、グローバル <i>A</i> をローカル複製 <i>B</i> (成分 (<i>ii</i> , <i>jj</i>)) に送る。 <i>rev</i> != 0 を使用して、成分 (<i>ii</i> , <i>jj</i>) からローカル複製 <i>B</i> をグローバル <i>A</i> 内の所有成分に送る (所有成分は、 <i>A</i> における成分の位置によって変更する)。

出力パラメーター

- a* (グローバル)。 *rev* = 1 の場合、終了時に、コピーされたデータを格納する。 *rev* = 0 の場合、終了時に変更されない。
- b* (ローカル)。 *rev* = 1 の場合、終了時に変更されない。

p?lacpy

1 つの 2 次元配列の一部または全部を他にコピーする。

構文

```
call pslacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pdlacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pclacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
call pzlacpy(uplo, m, n, a, ia, ja, desca, b, ib, jb, descb)
```

説明

このルーチンは、分散行列 *A* の一部または全部を他の分散行列 *B* にコピーする。コミュニケーションが実行されない場合、p?lacpy はローカルコピー $\text{sub}(A) := \text{sub}(B)$ を実行する。ここで、 $\text{sub}(A)$ は $A(ia:ia+m-1, ja:ja+n-1)$ 、 $\text{sub}(B)$ は $B(ib:ib+m-1, jb:jb+n-1)$ である。

入力パラメーター

- uplo* (グローバル)。 CHARACTER。
分散行列 $\text{sub}(A)$ のコピーする部分を指定する。
'U' の場合、上三角部分をコピーする。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。
'L' の場合、下三角部分をコピーする。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。
それ以外の場合、行列 $\text{sub}(A)$ の全部をコピーする。
- m* (グローバル) INTEGER。
演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。
($m \geq 0$)。
- n* (グローバル) INTEGER。
演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。
($n \geq 0$)。
- a* (ローカル)。
REAL (pslacpy の場合)
DOUBLE PRECISION (pdlacpy の場合)
COMPLEX (pclacpy の場合)
COMPLEX*16 (pzlacpy の場合)
ローカルメモリーにある、次元 (*lld_a*, $LOCc(ja+n-1)$) の配列へのポインター。分散行列 $\text{sub}(A)$ のローカル部分を格納する。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、 <i>sub(B)</i> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>b</i>	(ローカル) REAL (<i>pslacpy</i> の場合) DOUBLE PRECISION (<i>pdlacpy</i> の場合) COMPLEX (<i>pclacpy</i> の場合) COMPLEX*16 (<i>pzlacpy</i> の場合) ローカルメモリーにある、次元 (<i>lld_b</i> , <i>LOCc(jb+n-1)</i>) の配列へのポインター。終了時に、次のように設定された、分散行列 <i>sub(B)</i> のローカル部分が格納される。 $\text{uplo} = 'U' \text{ の場合、 } B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1) \quad (1 \leq i \leq j, 1 \leq j \leq n)$ $\text{uplo} = 'L' \text{ の場合、 } B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1) \quad (j \leq i \leq m, 1 \leq j \leq n)$ それ以外の場合、 $B(ib+i-1, jb+j-1) = A(ia+i-1, ja+j-1) \quad (1 \leq i \leq m, 1 \leq j \leq n)$
----------	---

p?laevswp

計算された固有ベクトルを ScaLAPACK 標準ブ
ロック・サイクリック配列に移動する。

構文

```
call pslaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,
             lrwork)
call pdlaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,
             lrwork)
call pclaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,
             lrwork)
call pzlaevswp(n, zin, ldzi, z, iz, jz, descz, nvs, key, rwork,
             lrwork)
```

説明

このルーチンは、(ソートされていない可能性のある)計算された固有ベクトルを ScaLAPACK 標準ブロック・サイクリック配列に移動する (ScaLAPACK はソートされているため、対応する固有ベクトルもソートされる)。

入力パラメーター

np = 与えられたプロセスに対するローカルの行数。

nq = 与えられたプロセスに対するローカルの列数。

n (グローバル) INTEGER。
行列 *A* の次数。 $n \geq 0$ 。

zin (ローカル)。
REAL (*pslaevswp* の場合)
DOUBLE PRECISION (*pdlaevswp* の場合)
COMPLEX (*pclaevswp* の場合)
COMPLEX*16 (*pzlaevswp* の場合)
配列、次元は (*ldzi*, *nvs(iam)*)。固有ベクトル。各固有ベクトルは、1つのプロセスに完全に属する。各プロセスは、隣接した *nvs(iam)* 固有ベクトルのセットを格納する。プロセスが格納する最初の固有ベクトルは、*nvs(i)* の $i=[0, iam-1)$ の合計。

ldzi (ローカル) INTEGER。配列 *zin* のリーディング・ディメンジョン。

iz, *jz* (グローバル) INTEGER。それぞれ、部分行列 *Z* の最初の行と最初の列を示す、グローバル配列 *Z* の行インデックスと列インデックス。

descz (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *Z* の配列ディスクリプター。

nvs (グローバル) INTEGER。
配列、次元は (*nprocs+1*)
nvs(i) = プロセス [0, *i-1*) の固有ベクトル数。
nvs(1) = プロセス [0, 1-1) = 0 の固有ベクトル数。
nvs(nprocs+1) = プロセス [0, *nprocs*) の固有ベクトル数 (= 固有ベクトルの総数)。

key (グローバル) INTEGER。
配列、次元は (*n*)。各固有ベクトルの (ソート後の) 実際のインデックスを指定する。

rwork (ローカル)。
REAL (*pslaevswp* の場合)
DOUBLE PRECISION (*pdlaevswp* の場合)
COMPLEX (*pclaevswp* の場合)
COMPLEX*16 (*pzlaevswp* の場合)
配列、次元は (*lrwork*)。

lrwork (ローカル) INTEGER。
work の次元。

出力パラメーター

z (ローカル)。
 REAL (pslaevswp の場合)
 DOUBLE PRECISION (pdlaevswp の場合)
 COMPLEX (pclaevswp の場合)
 COMPLEX*16 (pzlaevswp の場合)
 配列、グローバル次元は (n, n) 、ローカル次元は $(descz(dlen_), nq)$ 。固有ベクトル。固有ベクトルは、どちらの次元でもサイズ *nb* のブロック・サイクリック形式で分散される。

p?lahrd

直交/ユニタリー相似変換を用いて、*k* 番目の劣対角よりも下の成分がゼロになるように一般矩形行列の最初の *nb* 列を縮退させ、*A* の縮退されていない部分に変換を適用するために必要となる補助行列を返す。

構文

```
call pslahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pdlahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pclahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
call pzlahrd(n, k, nb, a, ia, ja, desca, tau, t, y, iy, jy, descy, work)
```

説明

このルーチンは、*k* 番目の劣対角よりも下の成分がゼロになるように、 $n \times (n-k+1)$ の一般実数分布行列 $A(ia:ia+n-1, ja:ja+n-k)$ の最初の *nb* 列を縮退させる。この縮退は直交/ユニタリー相似変換 $Q' * A * Q$ によって実行される。このルーチンは、ブロック・リフレクター $I - V * T * V'$ として Q を定義する行列 V と T 、および行列 $Y = A * V * T$ を返す。

このルーチンは、[p?gehrd](#) から呼び出される補助ルーチンである。次の入力/出力パラメーターの説明では、 $sub(A)$ は $A(ia:ia+n-1, ja:ja+n-1)$ を表す。

入力パラメーター

n (グローバル) INTEGER。分散部分行列 $sub(A)$ の次数。 $n \geq 0$ 。
k (グローバル) INTEGER。縮退のオフセット。最初の *nb* 列にある *k* 番目の劣対角よりも下の成分がゼロに縮退される。
nb (グローバル) INTEGER。縮退させる列数。
a (ローカル)。
 REAL (pslahrd の場合)
 DOUBLE PRECISION (pdlahrd の場合)
 COMPLEX (pclahrd の場合)
 COMPLEX*16 (pzlahrd の場合)

	ローカルメモリーにある、次元 (lld_a , $LOCc(ja+n-k)$) の配列へのポインター。 $n \times (n-k+1)$ の一般分散行列 $A(ia:ia+n-1, ja:ja+n-k)$ のローカル部分を格納する。
ia, ja	(グローバル) INTEGER。それぞれ、部分行列 $sub(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
$desca$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
iy, jy	(グローバル) INTEGER。それぞれ、部分行列 $sub(Y)$ の最初の行と最初の列を示す、グローバル配列 Y の行インデックスと列インデックス。
$descy$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 Y の配列ディスクリプター。
$work$	(ローカル)。 REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合) COMPLEX*16 (pzlahrd の場合) 配列、次元は (nb)。

出力パラメーター

a	(ローカル)。 終了時に、最初の nb 列にある k 番目の劣対角成分とその上の成分は、縮退された分散行列の対応する成分で上書きされる。 k 番目の劣対角よりも下の成分は、配列 tau とともに、基本リフレクターの積として行列 Q を表現する。 $A(ia:ia+n-1, ja:ja+n-k)$ のそのほかの列は変更されない。次の「アプリケーション・ノート」を参照。
tau	(ローカル)。 REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合) COMPLEX*16 (pzlahrd の場合) 配列、次元は $LOCc(ja+n-2)$ 。 基本リフレクターのスカラー係数 (次の「アプリケーション・ノート」を参照)。 tau は、分散行列 A に関連付けられる。
t	(ローカル)。 REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合) COMPLEX (pclahrd の場合) COMPLEX*16 (pzlahrd の場合) 配列、次元は (nb_a, nb_a) 上三角行列 T 。
y	(ローカル)。 REAL (pslahrd の場合) DOUBLE PRECISION (pdlahrd の場合)

COMPLEX (pclahrd の場合)

COMPLEX*16 (pzlahrd の場合)

ローカルメモリーにある、次元 (lld_y, nb_a) の配列へのポインター。終了時に、 $n \times nb$ の分散行列 Y のローカル部分が格納される。

$lld_y \geq LOCr(ia+n-1)$ 。

アプリケーション・ノート

行列 Q は基本リフレクター nb の積として表現される。

$$Q = H(1)H(2) \dots H(nb)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau v v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(1:i+k-1) = 0$ 、 $v(i+k) = 1$ 。終了時に、 $v(i+k+1:n)$ は $A(ia+i+k:ia+n-1, ja+i-1)$ に、 τ は $TAU(ja+i-1)$ に格納される。

ベクトル v の成分は、行列の縮退されていない部分に変換を適用するために、次の形式の更新を用いて、 T と Y とともに必要な $(n-k+1) \times nb$ の行列 V を形成する。

$A(ia:ia+n-1, ja:ja+n-k) := (I - V * T * V') * (A(ia:ia+n-1, ja:ja+n-k) - Y * V)$ 。終了時の $A(ia:ia+n-1, ja:ja+n-k)$ の内容を次に示す ($n=7$ 、 $k=3$ 、および $nb=2$ の場合)。

$$\begin{bmatrix} a & h & a & a & a \\ a & h & a & a & a \\ a & h & a & a & a \\ h & h & a & a & a \\ v1 & h & a & a & a \\ v1 & v2 & a & a & a \\ v1 & v2 & a & a & a \end{bmatrix}$$

a は元の行列 $A(ia:ia+n-1, ja:ja+n-k)$ の成分、 h は上 Hessenberg 行列 H の変更された成分、 v_i は $H(i)$ を定義するベクトルの成分を示す。

p?laiect

IEEE 演算を利用して固有値の計算を加速させる
(C インターフェイス関数)。

構文

```
void pslaiect(float *sigma, int *n, float *d, int *count);
void pdlaiectb(float *sigma, int *n, float *d, int *count);
void pdlaiectl(float *sigma, int *n, float *d, int *count);
```

説明

このルーチンは、 $(A - \sigma I)$ の負の固有値の個数を計算する。Sturm シーケンスループは、IEEE 演算を利用し、最内ループに条件を持たない。ビット 32 を実数ルーチン `pslaiect` の符号ビットとみなす。倍精度ルーチン `pdlaiectb` と `pdlaiectl` では、倍精度 WORD 型の格納順序が異なる。そのため、符号ビットの位置も異なる。`pdlaiectb` では、倍精度 WORD 型をビッグ・エンディアンで格納し、符号ビットはビット 32 である。`pdlaiectl` では、倍精度 WORD 型をリトル・エンディアンで格納し、符号ビットはビット 64 である。

すべての引数は参照による呼び出しであるため、このルーチンは Fortran コードから直接呼び出すことができる。

これは ScaLAPACK の内部サブルーチンであり、引数の妥当性は確認されない。

入力パラメーター

<i>sigma</i>	REAL (<code>pslaiect</code> の場合) DOUBLE PRECISION (<code>pdlaiectb/pdlaiectl</code> の場合) シフト。 <i>p?laiect</i> は、 <i>sigma</i> よりも小さいか等しい固有値を検出する。
<i>n</i>	INTEGER。 三重対角行列 <i>T</i> の次数。 $n \geq 1$ 。
<i>d</i>	REAL (<code>pslaiect</code> の場合) DOUBLE PRECISION (<code>pdlaiectb/pdlaiectl</code> の場合) 次元 $(2n - 1)$ の配列。三重対角行列 <i>T</i> の非対角成分の平方および非対角成分を格納する。高いキャッシュ・パフォーマンスを得るために、これらの成分はメモリーでインターリーブされるものみなす。 <i>T</i> の対角成分は $d(1), d(3), \dots, d(2n-1)$ 、非対角成分の平方は $d(2), d(4), \dots, d(2n-2)$ とする。オーバーフローを避けるためには、行列を、その最大の成分が $overflow^{(1/2)} * underflow^{(1/4)}$ よりも絶対値において超えないようにスケールしなければならない。また、最高の精度を得るためには、行列を上記条件よりもはるかに小さくスケールしてはならない。

出力パラメーター

<i>n</i>	INTEGER。 <i>sigma</i> よりも小さいか等しい <i>T</i> の固有値の総数。
----------	---

p?lange

一般矩形行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = pslange(norm, m, n, a, ia, ja, desca, work)
val = pdlange(norm, m, n, a, ia, ja, desca, work)
```

```
val = pclang(norm, m, n, a, ia, ja, desca, work)
val = pzlang(norm, m, n, a, ia, ja, desca, work)
```

説明

この関数は、分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

`p?lange` は次の値を返す。

```
(max(abs(A(i,j))), norm = 'M' または 'm', ia ≤ i ≤ ia+m-1,
(
    および ja ≤ j ≤ ja+n-1,
(
norm1(sub(A)), norm = '1', 'o' または 'o'
(
normI(sub(A)), norm = 'I' または 'i'
(
normF(sub(A)), norm = 'F', 'f', 'E' または 'e'
```

`norm1` は行列の 1- ノルム (最大列合計)、`normI` は行列の無限ノルム (最大行合計)、`normF` は行列の Frobenius ノルム (二乗和の平方根) を示す。`max(abs(A(i,j)))` は行列ノルムではない点に注意する。

入力パラメーター

<code>norm</code>	(グローバル) CHARACTER。 ルーチン <code>p?lange</code> が返す値を上述のように指定する。
<code>m</code>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 $m=0$ と指定した場合、 <code>p?lange</code> はゼロに設定される。 $m \geq 0$ 。
<code>n</code>	(グローバル)。 INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 $n=0$ と指定した場合、 <code>p?lange</code> はゼロに設定される。 $n \geq 0$ 。
<code>a</code>	(ローカル)。 REAL (<code>pslange</code> の場合) DOUBLE PRECISION (<code>pdlange</code> の場合) COMPLEX (<code>pclang</code> の場合) COMPLEX*16 (<code>pzlang</code> の場合) ローカルメモリーにある、次元 $(lld_a, LOCc(ja+n-1))$ の配列。分散行列 $\text{sub}(A)$ のローカル部分を格納する。
<code>ia, ja</code>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER 配列、次元は $(dlen_)$ 。分散行列 A の配列ディスクリプター。

`work` (ローカル)。
 REAL (`pslange` の場合)
 DOUBLE PRECISION (`pdlange` の場合)
 COMPLEX (`pclange` の場合)
 COMPLEX*16 (`pzlange` の場合)
 配列、次元は (`lwork`)。
 $norm = 'M'$ または $'m'$ の場合 (参照されない場合)、 $lwork \geq 0$ 。
 $norm = '1'$ 、 $'0'$ または $'o'$ の場合、 $nq0$ 。
 $norm = 'I'$ または $'i'$ の場合、 $mp0$ 。
 $norm = 'F'$ 、 $'f'$ 、 $'E'$ または $'e'$ の場合 (参照されない場合)、 0 。
 ここで、
 $irow = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcpl)$ 、
 $mp0 = \text{numroc}(m+irow, mb_a, myrow, iarow, nprow)$ 、
 $nq0 = \text{numroc}(n+icoffa, nb_a, mycol, iacol, npcpl)$ 、
 indxg2p および numroc は、ScaLAPACK ツール関数である。
 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcpl$ は、サブルーチン
`blacs_gridinfo` を呼び出して求められる。

出力パラメーター

`val` 関数の戻り値。

p?lanhs

上 Hessenberg 行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = pslanhs(norm, n, a, ia, ja, desca, work)
val = pdlanhs(norm, n, a, ia, ja, desca, work)
val = pclanhs(norm, n, a, ia, ja, desca, work)
val = pzlanshs(norm, n, a, ia, ja, desca, work)
```

説明

この関数は、分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

`p?lange` は次の値を返す。

```
(max(abs(A(i,j))), norm = 'M' または 'm', ia ≤ i ≤ ia+m-1,
(
    および ja ≤ j ≤ ja+n-1,
(
(norm1(sub(A)), norm = '1', '0' または 'o')
```

(

 $(\text{normI}(\text{sub}(A)), \text{norm} = 'I' \text{ または } 'i')$

(

 $(\text{normF}(\text{sub}(A)), \text{norm} = 'F', 'f', 'E' \text{ または } 'e')$

norm1 は行列の 1- ノルム (最大列合計)、 normI は行列の無限ノルム (最大行合計)、 normF は行列の Frobenius ノルム (二乗和の平方根) を示す。 $\max(\text{abs}(A(i,j)))$ は行列ノルムではない点に注意する。

入力パラメーター

<i>norm</i>	(グローバル) CHARACTER。 ルーチン <code>p?lange</code> が返す値を上述のように指定する。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <code>sub(A)</code> の列数。 $n=0$ と指定した場合、 <code>p?lanhs</code> はゼロに設定される。 $n \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (<code>pslanhs</code> の場合) DOUBLE PRECISION (<code>pdlanhs</code> の場合) COMPLEX (<code>pclanhs</code> の場合) COMPLEX*16 (<code>pzlanhs</code> の場合) ローカルメモリーにある、次元 (<code>lld_a, LOcc(ja+n-1)</code>) の配列。 分散行列 <code>sub(A)</code> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <code>sub(A)</code> の最初の行と最初の列を示す、グローバル配列 <code>A</code> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<code>dlen_</code>)。 分散行列 <code>A</code> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<code>pslanhs</code> の場合) DOUBLE PRECISION (<code>pdlanhs</code> の場合) COMPLEX (<code>pclanhs</code> の場合) COMPLEX*16 (<code>pzlanh</code> の場合) 配列、次元は (<code>lwork</code>)。 $\text{norm} = 'M'$ または $'m'$ の場合 (参照されない場合)、 $\text{lwork} \geq 0$ 。 $\text{norm} = '1'$ 、 $'o'$ または $'o'$ の場合、 $\text{nq}0$ 。 $\text{norm} = 'I'$ または $'i'$ の場合、 $\text{mp}0$ 。 $\text{norm} = 'F'$ 、 $'f'$ 、 $'E'$ または $'e'$ の場合 (参照されない場合)、 0 。 ここで、 $\text{iroffa} = \text{mod}(\text{ia}-1, \text{mb}_a)$ 、 $\text{icoffa} = \text{mod}(\text{ja}-1, \text{nb}_a)$ 、 $\text{iarow} = \text{indxg2p}(\text{ia}, \text{mb}_a, \text{myrow}, \text{rsrc}_a, \text{nprow})$ 、 $\text{iacol} = \text{indxg2p}(\text{ja}, \text{nb}_a, \text{mycol}, \text{csrc}_a, \text{npcol})$ 、 $\text{mp}0 = \text{numroc}(\text{m}+\text{iroffa}, \text{mb}_a, \text{myrow}, \text{iarow}, \text{nprow})$ 、 $\text{nq}0 = \text{numroc}(\text{n}+\text{icoffa}, \text{nb}_a, \text{mycol}, \text{iacol}, \text{npcol})$ 、 <code>indxg2p</code> および <code>numroc</code> は、ScaLAPACK ツール関数である。 <code>myrow</code> 、 <code>mycol</code> 、 <code>nprow</code> 、および <code>npcol</code> は、サブルーチン <code>blacs_gridinfo</code> を呼び出して求められる。

出力パラメーター

`val` 関数の戻り値。

p?lansy, p?lanhe

実対称行列または複素エルミート行列の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

構文

```
val = pslansy(norm, uplo, n, a, ia, ja, desca, work)
val = pdlansy(norm, uplo, n, a, ia, ja, desca, work)
val = pclansy(norm, uplo, n, a, ia, ja, desca, work)
val = pzlansy(norm, uplo, n, a, ia, ja, desca, work)
val = pclanhe(norm, uplo, n, a, ia, ja, desca, work)
val = pzlanhe(norm, uplo, n, a, ia, ja, desca, work)
```

説明

この関数は、分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の 1- ノルムの値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

p?lansy、p?lanhe は次の値を返す。

($\max(\text{abs}(A(i,j)))$), $\text{norm} = 'M'$ または $'m'$, $ia \leq i \leq ia+m-1$,
(および $ja \leq j \leq ja+n-1$,

(

($\text{norm}_1(\text{sub}(A))$), $\text{norm} = 'l'$, $'o'$ または $'o'$

(

($\text{norm}_I(\text{sub}(A))$), $\text{norm} = 'I'$ または $'i'$

(

($\text{norm}_F(\text{sub}(A))$), $\text{norm} = 'F'$, $'f'$, $'E'$ または $'e'$

norm_1 は行列の 1- ノルム (最大列合計)、 norm_I は行列の無限ノルム (最大行合計)、 norm_F は行列の Frobenius ノルム (二乗和の平方根) を示す。 $\max(\text{abs}(A(i,j)))$ は行列ノルムではない点に注意する。

入力パラメーター

`norm` (グローバル) CHARACTER。
ルーチン [p?lange](#) が返す値を上述のように指定する。

`uplo` (グローバル) CHARACTER。
対称行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらを参照させるかを指定する。

	'U' の場合、sub(A) の上三角部分を参照させる。 'L' の場合、sub(A) の下三角部分を参照させる。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 sub(A) の列数。n=0 と指定した場合、p?lansy はゼロに設定される。 n ≥ 0。
<i>a</i>	(ローカル)。 REAL (pslansy の場合) DOUBLE PRECISION (pdlansy の場合) COMPLEX (pclansy、pclanhe の場合) COMPLEX*16 (pzlansy、pzlanhe の場合) ローカルメモリーにある、次元 (lld_a, LOCC(ja+n-1)) の配列。 分散行列 sub(A) のローカル部分を格納する。 uplo='U' の場合、sub(A) の先頭の n × n の上三角部分に、ノルムを計算する上三角行列を格納する。この行列の厳密な下三角部分は参照されない。 uplo='L' の場合、sub(A) の先頭の n × n の下三角部分に、ノルムを計算する下三角行列を格納する。sub(A) の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 sub(A) の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (dlen_)。 分散行列 A の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (pslansy の場合) DOUBLE PRECISION (pdlansy の場合) COMPLEX (pclansy、pclanhe の場合) COMPLEX*16 (pzlansy、pzlanhe の場合) 配列、次元は (lwork)。 norm='M' または 'm' の場合 (参照されない場合)、lwork ≥ 0。 norm='I'、'O' または 'o'、'T' または 'i' の場合、2*nq0+np0+ldw。 ldw は次のように与えられる。 if(nprow.ne.npcol) then ldw = mb_a*ceil(ceil(np0/mb_a)/(lcm/nprow)) else ldw = 0 end if norm='F'、'f'、'E' または 'e' の場合 (参照されない場合)、0。 lcm は nprow と npcol の最小公倍数である。 lcm = ilcm(nprow, npcol)、ceil は ceiling 演算 (iceil)。 iroffa = mod(ia-1, mb_a)、icoffa = mod(ja-1, nb_a)、 iarow = indxg2p(ia, mb_a, myrow, rsrc_a, nprow)、 iacol = indxg2p(ja, nb_a, mycol, csrc_a, npcol)、 mp0 = numroc(m+iroffa, mb_a, myrow, iarow, nprow)、 nq0 = numroc(n+icoffa, nb_a, mycol, iacol, npcol)、

`indxg2p` および `numroc` は、ScaLAPACK ツール関数である。
`myrow`, `mycol`, `npro`, および `npcol` は、サブルーチン
`blacs_gridinfo` を呼び出して求められる。

出力パラメーター

`val` 関数の戻り値。

p?lantr

三角行列の 1- ノルムの値、Frobenius ノルムの値、
無限ノルムの値、あるいは最大絶対値の成分を返
す。

構文

```
val = pslantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pdlantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pclantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
val = pzlantr(norm, uplo, diag, m, n, a, ia, ja, desca, work)
```

説明

この関数は、台形または三角分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ の 1- ノルムの
値、Frobenius ノルムの値、無限ノルムの値、あるいは最大絶対値の成分を返す。

`p?lantr` は次の値を返す。

($\max(\text{abs}(A(i,j)))$), `norm = 'M'` または `'m'`, $ia \leq i \leq ia+m-1$,
(および $ja \leq j \leq ja+n-1$,

(

($\text{norm}_1(\text{sub}(A))$), `norm = '1'`, `'o'` または `'o'`

(

($\text{norm}_I(\text{sub}(A))$), `norm = 'I'` または `'i'`

(

($\text{norm}_F(\text{sub}(A))$), `norm = 'F'`, `'f'`, `'E'` または `'e'`

norm_1 は行列の 1- ノルム (最大列合計), norm_I は行列の無限ノルム (最大行合計),
 norm_F は行列の Frobenius ノルム (二乗和の平方根) を示す。 $\max(\text{abs}(A(i,j)))$ は行列ノ
ルムではない点に注意する。

入力パラメーター

`norm` (グローバル) CHARACTER。
ルーチン `p?lantr` が返す値を上述のように指定する。

<i>uplo</i>	(グローバル) CHARACTER。 対称行列 <i>sub(A)</i> の上三角部分と下三角部分のどちらを参照させるかを指定する。 'U' の場合、上台形。 'L' の場合、下台形。 <i>m = n</i> の場合、 <i>sub(A)</i> は台形ではなく三角であることに注意する。
<i>diag</i>	(グローバル) CHARACTER。分散行列 <i>sub(A)</i> が単位対角を持っているかどうかを指定する。 'N' の場合、単位対角ではない。 'U' の場合、単位対角。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。 <i>m = 0</i> と指定した場合、 <i>p?lantr</i> はゼロに設定される。 <i>m ≥ 0</i> 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。 <i>n = 0</i> と指定した場合、 <i>p?lantr</i> はゼロに設定される。 <i>n ≥ 0</i> 。
<i>a</i>	(ローカル)。 REAL (<i>p?lantr</i> の場合) DOUBLE PRECISION (<i>pdlantr</i> の場合) COMPLEX (<i>p?lantr</i> の場合) COMPLEX*16 (<i>pzlantr</i> の場合) ローカルメモリーにある、次元 (<i>lld_a, LOcc(ja+n-1)</i>) の配列。 分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>a</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>work</i>	(ローカル) REAL (<i>p?lantr</i> の場合) DOUBLE PRECISION (<i>pdlantr</i> の場合) COMPLEX (<i>p?lantr</i> の場合) COMPLEX*16 (<i>pzlantr</i> の場合) 配列、次元は (<i>lwork</i>)。 <i>norm = 'M'</i> または ' <i>m</i> ' の場合 (参照されない場合)、 <i>lwork ≥ 0</i> 。 <i>norm = 'l'</i> 、' <i>o</i> ' または ' <i>o</i> ' の場合、 <i>nq0</i> 。 <i>norm = 'I'</i> または ' <i>i</i> ' の場合、 <i>mp0</i> 。 <i>norm = 'F'</i> 、' <i>f</i> '、' <i>E</i> ' または ' <i>e</i> ' の場合 (参照されない場合)、 <i>0</i> 。 <i>lcm</i> は <i>npro</i> と <i>npcol</i> の最小公倍数である。 <i>lcm = ilcm(npro, npcol)</i> 、 <i>ceil</i> は ceiling 演算 (<i>iceil</i>)。 <i>iroffa = mod(ia-1, mb_a)</i> 、 <i>icoffa = mod(ja-1, nb_a)</i> 、 <i>iarow = indxg2p(ia, mb_a, myrow, rsrc_a, npro)</i> 、 <i>iacol = indxg2p(ja, nb_a, mycol, csrc_a, npcol)</i> 、

```
mp0 = numroc( m+iroffa, mb_a, myrow, iarow, nprow ),
nq0 = numroc( n+icoffa, nb_a, mycol, iacol, npc0l ),
indxg2p および numroc は、ScaLAPACK ツール関数である。
myrow, mycol, nprow, および npc0l は、サブルーチン
blacs_gridinfo を呼び出して求められる。
```

出力パラメーター

`val` 関数の戻り値。

p?lapiv

一般分散行列に置換行列を適用し、行または列のピボット演算を行う。

構文

```
call pslapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
call pdlapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
call pclapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
call pzlapiv(direc, rowcol, pivroc, m, n, a, ia, ja, desca, ipiv, ip, jp,
            descip, iwork)
```

説明

このルーチンは、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+m-1, \text{ja}:\text{ja}+n-1)$ に P (ipiv で示される置換行列) または $\text{inv}(P)$ を適用し、行または列のピボット演算を行う。ピボットのベクトルは、プロセス行 / 列に分配される。ピボットのベクトルは、分散行列 A とアライメントされなければならない。このルーチンは、必要に応じてピボットのベクトルを転置する。

例えば、行のピボット演算が $\text{sub}(A)$ の列に適用される場合、`rowcol = 'c'` と `pivroc = 'c'` を渡す。

入力パラメーター

`direc` (グローバル) CHARACTER*1。
置換順序を指定する。
'F' (順方向) の場合、行列の一番上から順方向にピボットを適用する。
 $P * \text{sub}(A)$ を計算する。
'B' (逆方向) の場合、行列の一番下から逆方向にピボットを適用する。
 $\text{inv}(P) * \text{sub}(A)$ を計算する。

<i>rowcol</i>	(グローバル) CHARACTER*1。 行、列どちらを置換するのかを指定する。 'R' の場合、行を置換する。 'C' の場合、列を置換する。
<i>pivroc</i>	(グローバル) CHARACTER*1。 <i>ipiv</i> をプロセス行 / 列に分配するかどうかを指定する。 'R' の場合、 <i>ipiv</i> をプロセス行に分配する。 'C' の場合、 <i>ipiv</i> をプロセス列に分配する
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。 $m = 0$ と指定した場合、 <i>p?lapiv</i> はゼロに設定される。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。 $n = 0$ と指定した場合、 <i>p?lapiv</i> はゼロに設定される。 $n \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (<i>pslapiv</i> の場合) DOUBLE PRECISION (<i>pdlapiv</i> の場合) COMPLEX (<i>pclapiv</i> の場合) COMPLEX*16 (<i>pzlapiv</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列。 分散行列 <i>sub(A)</i> のローカル部分を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は (<i>lipiv</i>)。 <i>lipiv</i> の値は次の通り。 <i>rowcol</i> = 'R' または 'r' の場合、 $\geq LOCr(ia+m-1) + mb_a$ (<i>pivroc</i> ='C' または 'c' の場合) $\geq LOCc(m + mod(jp-1, nb_p))$ (<i>pivroc</i> ='R' または 'r' の場合) <i>rowcol</i> = 'C' または 'c' の場合、 $\geq LOCr(n + mod(ip-1, mb_p))$ (<i>pivroc</i> ='C' または 'c' の場合) $\geq LOCc(ja+n-1) + nb_$ (<i>pivroc</i> ='R' または 'r' の場合) この配列には、ピボット情報が格納される。 <i>ipiv(i)</i> はローカル行 / 列 <i>i</i> と交換されたグローバル行 / 列。 <i>rowcol</i> = 'R'、'r' かつ <i>pivroc</i> = 'C'、'c' のとき、あるいは <i>rowcol</i> = 'C'、'c' かつ <i>pivroc</i> = 'R'、'r' のとき、サイズ <i>mb_a</i> (<i>nb_a</i>) の配列の一番最後の成分は、ワークスペースとして使用される。このような場合、分散行列 <i>A</i> に関連付けられる。
<i>ip, jp</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(P)</i> の最初の行と最初の列を示す、グローバル配列 <i>P</i> の行インデックスと列インデックス。

descip (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。乱数ベクトル *ipiv* の配列ディスクリプター。

iwork (ローカル)。INTEGER。配列、次元は (*ldw*)。 *ldw* は、転置に必要なワークスペースのサイズ、および転置された *ipiv* の格納と同じ大きさである。

lcm は *nprow* と *npcol* の最小公倍数である。

```

If( rowcol.eq.'r' .and. pivroc.eq.'r' ) then
  If( nprow.eq.npcol ) then
    ldw = LOCr( n_p + mod( jp-1, nb_p ) ) + nb_p
  else
    ldw = LOCr( n_p + mod( jp-1, nb_p ) ) +
          nb_p * ceil( ceil( LOCc( n_p ) / nb_p ) /
            ( lcm / npcol ) )
    end if
else if( rowcol.eq.'c' .and. pivroc.eq.'c' ) then
  if( nprow.eq.npcol ) then
    ldw = LOCc( m_p + mod( ip-1, mb_p ) ) + mb_p
  else
    ldw = LOCc( m_p + mod( ip-1, mb_p ) ) +
          mb_p * ceil( ceil( LOCr( m_p ) / mb_p ) /
            ( lcm / nprow ) )
    end if
else
  iwork is not referenced.
end if.

```

出力パラメーター

a (ローカル)
終了時に、置換された分散部分行列のローカル部分。

p?laqge

p?geequ で計算された行と列のスケール係数を使って一般矩形行列をスケールリングする。

構文

```

call pslaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pdlaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pclaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)
call pzlaqge(m, n, a, ia, ja, desca, r, c, rowcnd, colcnd, amax, equed)

```

説明

このルーチンは、 $m \times n$ の一般分散行列 $\text{sub}(A) = A(ia:ia+m-1, ja:ja+n-1)$ を、[p?geequ](#) で計算されたベクトル *r* と *c* に格納されている行と列のスケール係数を用いて平衡化する。

入力パラメーター

<i>m</i>	(グローバル)。INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(A)</i> の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル)。INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(A)</i> の列数。 ($n \geq 0$)。
<i>a</i>	(ローカル)。 REAL (<i>pslaqge</i> の場合) DOUBLE PRECISION (<i>pdlaqge</i> の場合) COMPLEX (<i>pclaqge</i> の場合) COMPLEX*16 (<i>pzlaqge</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。分散行列 <i>sub(A)</i> を格納する。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>r</i>	(ローカル)。 REAL (<i>pslaqge</i> の場合) DOUBLE PRECISION (<i>pdlaqge</i> の場合) COMPLEX (<i>pclaqge</i> の場合) COMPLEX*16 (<i>pzlaqge</i> の場合) 配列、次元は <i>LOCr(m_a)</i> 。 <i>sub(A)</i> の行スケール係数。 <i>r</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>r</i> は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル)。 REAL (<i>pslaqge</i> の場合) DOUBLE PRECISION (<i>pdlaqge</i> の場合) COMPLEX (<i>pclaqge</i> の場合) COMPLEX*16 (<i>pzlaqge</i> の場合) 配列、次元は <i>LOCc(n_a)</i> 。 <i>sub(A)</i> の行スケール係数。 <i>c</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>c</i> は、分散行列 <i>A</i> に関連付けられる。
<i>rowcnd</i>	(ローカル)。 REAL (<i>pslaqge</i> の場合) DOUBLE PRECISION (<i>pdlaqge</i> の場合) COMPLEX (<i>pclaqge</i> の場合) COMPLEX*16 (<i>pzlaqge</i> の場合) 最小の <i>r(i)</i> を最大の <i>r(i)</i> で割ったグローバル値 ($ia \leq i \leq ia+m-1$)。
<i>colcnd</i>	(ローカル)。 REAL (<i>pslaqge</i> の場合) DOUBLE PRECISION (<i>pdlaqge</i> の場合) COMPLEX (<i>pclaqge</i> の場合)

COMPLEX*16 (pzlaqge の場合)
 最小の $c(i)$ を最大の $r(i)$ で割ったグローバル値
 ($ia \leq i \leq ia+n-1$)。

amax (グローバル)。
 REAL (pslaqge の場合)
 DOUBLE PRECISION (pdlaqge の場合)
 COMPLEX (pclaqge の場合)
 COMPLEX*16 (pzlaqge の場合)
 最大分散部分行列成分の絶対値。

出力パラメーター

a (ローカル)。
 終了時に、平衡化された分散行列で上書きされる。平衡化された分散行列の形式は *equed* を参照のこと。

equed (グローバル) CHARACTER。
 実行された平衡化の形式を表す。
 'N' の場合、平衡化は行われていない。
 'R' の場合、行の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(r(ia:ia+m-1))$ で事前乗算された。
 'C' の場合、列の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(c(ja:ja+n-1))$ で事後乗算された。
 'B' の場合、行と列の平衡化、すなわち、 $\text{sub}(A)$ は $\text{diag}(r(ia:ia+m-1)) * \text{sub}(A) * \text{diag}(c(ja:ja+n-1))$ で置き換えられた。

p?laqsy

p?poequ で計算されたスケール係数を用いて対称/エルミート行列をスケールする。

構文

```
call pslaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pdlaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pclaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
call pzlaqsy(uplo, n, a, ia, ja, desca, sr, sc, scond, amax, equed)
```

説明

このルーチンは、対称分散行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ を、ベクトル *sr* と *sc* に格納されているスケール係数を用いて平衡化する。スケール係数は [p?poequ](#) で計算される。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER。 対称分散行列 <i>sub(A)</i> の上三角部分と下三角部分のどちらを参照させるかを指定する。 'U' の場合、上三角部分を参照させる。 'L' の場合、下三角部分を参照させる。
<i>n</i>	(グローバル) INTEGER。分散部分行列 <i>sub(A)</i> の次数。 $n \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (<i>pslaqsy</i> の場合) DOUBLE PRECISION (<i>pdlaqsy</i> の場合) COMPLEX (<i>pclaqsy</i> の場合) COMPLEX*16 (<i>pzlaqsy</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。分散行列 <i>sub(A)</i> のローカル部分を格納する。 分散対称行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 <i>sub(A)</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。 <i>sub(A)</i> の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>sr</i>	(ローカル) REAL (<i>pslaqsy</i> の場合) DOUBLE PRECISION (<i>pdlaqsy</i> の場合) COMPLEX (<i>pclaqsy</i> の場合) COMPLEX*16 (<i>pzlaqsy</i> の場合) 配列、次元は <i>LOCr(m_a)</i> 。 $A(ia:ia+m-1, ja:ja+n-1)$ のスケール係数。 <i>sr</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>sr</i> は、分散行列 <i>A</i> に関連付けられる。
<i>sc</i>	(ローカル)。 REAL (<i>pslaqsy</i> の場合) DOUBLE PRECISION (<i>pdlaqsy</i> の場合) COMPLEX (<i>pclaqsy</i> の場合) COMPLEX*16 (<i>pzlaqsy</i> の場合) 配列、次元は <i>LOCc(m_a)</i> 。 $A(ia:ia+m-1, ja:ja+n-1)$ のスケール係数。 <i>sr</i> は分散行列 <i>A</i> でアライメントされ、すべてのプロセス列に複製される。 <i>sr</i> は、分散行列 <i>A</i> に関連付けられる。
<i>scond</i>	(グローバル)。 REAL (<i>pslaqsy</i> の場合) DOUBLE PRECISION (<i>pdlaqsy</i> の場合) COMPLEX (<i>pclaqsy</i> の場合)

COMPLEX*16 (pzlaqsy の場合)
 最小の $sr(i)$ ($sc(j)$) を最大の $sr(i)$ ($sc(j)$) で割った値
 ($ia \leq i \leq ia+n-1$ および $ja \leq j \leq ja+n-1$)。
 (グローバル)。
 REAL (pslaqsy の場合)
 DOUBLE PRECISION (pdlaqsy の場合)
 COMPLEX (pclaqsy の場合)
 COMPLEX*16 (pzlaqsy の場合)
 最大分散部分行列成分の絶対値。

出力パラメーター

a $e_{\text{qued}} = 'Y'$ の場合、平衡化された行列、
 $\text{diag}(sr(ia:ia+n-1)) * \text{sub}(A) * \text{diag}(sc(ja:ja+n-1))$ が格納され
 る。
e_{qued} (グローバル) CHARACTER*1。
 平衡化が行われたかどうかを示す。
 'N' の場合、平衡化は行われていない。
 'Y' の場合、平衡化が行われ、 $\text{sub}(A)$ は
 $\text{diag}(sr(ia:ia+n-1)) * \text{sub}(A) * \text{diag}(sc(ja:ja+n-1))$ で置き換えら
 れている。

p?lared1d

入力配列 *bycol* は複数の行に分配され、すべての
 プロセス列に同じ *bycol* のコピーが格納されるこ
 とを前提に、配列を再分配する。

構文

```
call pslared1d(n, ia, ja, desc, bycol, byall, work, lwork)
call pdlared1d(n, ia, ja, desc, bycol, byall, work, lwork)
```

説明

このルーチンは、1次元配列を再分配する。入力配列 *bycol* は複数の行に分配され、す
 べてのプロセス列に同じ *bycol* のコピーが格納されると仮定する。出力配列 *byall* は
 すべてのプロセスで同じであり、配列全体を格納する。

入力パラメーター

np = *bycol*() のローカル行数。
n (グローバル)。INTEGER。
 再分配する成分の数。 $n \geq 0$ 。
ia, ja (グローバル) INTEGER。 *ia*、*ja* は 1 に等しくなければならない。
desc (グローバルおよびローカル) INTEGER 配列、次元は 8。 *bycol*
 の 2次元配列ディスクリプター。

<i>bycol</i>	(ローカル)。 REAL (pslared1d の場合) DOUBLE PRECISION (pdlared1d の場合) COMPLEX (pclared1d の場合) COMPLEX*16 (pzlared1d の場合) 分散ブロック・サイクリック配列、グローバル次元は (n)、ローカル次元は <i>np</i> 。 <i>bycol</i> はプロセス行に分配される。すべてのプロセス列には同じ値が格納されるものと仮定する。
<i>work</i>	(ローカル)。 REAL (pslared1d の場合) DOUBLE PRECISION (pdlared1d の場合) COMPLEX (pclared1d の場合) COMPLEX*16 (pzlared1d の場合) 次元は (<i>lwork</i>)。1つのプロセスから他のプロセスへ送られたバッファを格納する。
<i>lwork</i>	(ローカル) INTEGER。 配列 <i>work</i> のサイズ。 $lwork \geq \text{numroc}(n, \text{desc}(nb_), 0, 0, npcol)$ 。

出力パラメーター

<i>byall</i>	(グローバル)。 REAL (pslared1d の場合) DOUBLE PRECISION (pdlared1d の場合) COMPLEX (pclared1d の場合) COMPLEX*16 (pzlared1d の場合) グローバル次元は (n)、ローカル次元は (n)。 <i>byall</i> は、すべてのプロセスで完全に複製される。 <i>bycol</i> と同じ値を格納する。すべてのプロセスに、分配ではなく、複製される。
--------------	---

p?lared2d

入力配列 *byrow* は複数の列に分配され、すべてのプロセス行に同じ *byrow* のコピーが格納されることを前提に、配列を再分配する。

構文

```
call pslared2d(n, ia, ja, desc, byrow, byall, work, lwork)
call pdlared2d(n, ia, ja, desc, byrow, byall, work, lwork)
```

説明

このルーチンは、1次元配列を再分配する。入力配列 *byrow* は複数の列に分配され、すべてのプロセス行に同じ *byrow* のコピーが格納されると仮定する。出力配列 *byall* はすべてのプロセスで同じであり、配列全体を格納する。

入力パラメーター

$np = \text{byrow}()$ のローカル行数

n	(グローバル) INTEGER。 再分配する成分の数。 $n \geq 0$ 。
ia, ja	(グローバル) INTEGER。 ia, ja は 1 に等しくなければならない。
$desc$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 $byrow$ の 2 次元配列ディスクリプター。
$byrow$	(ローカル) REAL (pslared2d の場合) DOUBLE PRECISION (pdlared2d の場合) COMPLEX (pclared2d の場合) COMPLEX*16 (pzlared2d の場合) 分散ブロック・サイクリック配列、グローバル次元は (n)、ローカル次元は np 。 $bycol$ はプロセス列に分配される。すべてのプロセス行には同じ値が格納されるものと仮定する。
$work$	(ローカル)。 REAL (pslared2d の場合) DOUBLE PRECISION (pdlared2d の場合) COMPLEX (pclared2d の場合) COMPLEX*16 (pzlared2d の場合) 次元は ($lwork$)。1つのプロセスから他のプロセスへ送られたバッファを格納する。
$lwork$	(ローカル) INTEGER。 配列 $work$ のサイズ。 $lwork \geq \text{numroc}(n, \text{desc}(nb_), 0, 0, npcol)$ 。

出力パラメーター

$byall$	(グローバル)。 REAL (pslared2d の場合) DOUBLE PRECISION (pdlared2d の場合) COMPLEX (pclared2d の場合) COMPLEX*16 (pzlared2d の場合) グローバル次元は (n)、ローカル次元は (n)。 $byall$ は、すべてのプロセスで完全に複製される。 $bycol$ と同じ値を格納する。すべてのプロセスに、分配ではなく、複製される。
---------	---

p?larf

一般矩形行列に基本リフレクターを適用する。

構文

```
call pslarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pdlarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
```

```
call pclarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pzlarf(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
```

説明

このルーチンは、実 / 複素基本リフレクター Q (または Q^T) を、 $m \times n$ の実 / 複素分布行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ に左または右のいずれかから適用する。 Q は次の形式で表現される。

$$Q = I - \tau * v * v'$$

τ は実 / 複素スカラー、 v は実 / 複素ベクトルである。

$\tau = 0$ の場合、 Q は単位行列をとる。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、形式 $Q * \text{sub}(C)$ 。 'R' の場合、形式 $\text{sub}(C) * Q$ 、 $Q = Q^T$ 。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
<i>v</i>	(ローカル)。 REAL (pslarf の場合) DOUBLE PRECISION (pdlarf の場合) COMPLEX (pclarf の場合) COMPLEX*16 (pzlarf の場合) ローカルメモリーにある、Householder 変換 Q を表す分散行列 V のローカル部分を格納する、次元 ($11d_v, *$) の配列へのポインタ。 $v(\text{iv}:\text{iv}+m-1, \text{jv})$ ($\text{side} = 'L'$ および $\text{incv} = 1$ の場合) $v(\text{iv}, \text{jv}:\text{jv}+m-1)$ ($\text{side} = 'L'$ および $\text{incv} = m_v$ の場合) $v(\text{iv}:\text{iv}+n-1, \text{jv})$ ($\text{side} = 'R'$ および $\text{incv} = 1$ の場合) $v(\text{iv}, \text{jv}:\text{jv}+n-1)$ ($\text{side} = 'R'$ および $\text{incv} = m_v$ の場合) Q の表現におけるベクトル v 。 $\tau = 0$ の場合、 v は使用されない。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 V の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 V の配列ディスクリプター。
<i>incv</i>	(グローバル) INTEGER。 v の成分のグローバルな増分。このバージョンでは、1 と m_v の 2 つの incv の値のみサポートされる。 incv の値は、ゼロであってはならない。

<i>tau</i>	(ローカル)。 REAL (pslarf の場合) DOUBLE PRECISION (pdlarf の場合) COMPLEX (pclarf の場合) COMPLEX*16 (pzlarf の場合) 配列、次元は $LOCc(jv)$ ($incv = 1$ の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。 <i>tau</i> は、分散行列 v に関連付けられる。
<i>c</i>	(ローカル)。 REAL (pslarf の場合) DOUBLE PRECISION (pdlarf の場合) COMPLEX (pclarf の場合) COMPLEX*16 (pzlarf の場合) ローカルメモリーにある、 $sub(C)$ のローカル部分を格納する、次元 ($lld_c, LOCc(jc+n-1)$) の配列へのポインター。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 $sub(C)$ の最初の行と最初の列を示す、グローバル配列 c の行インデックスと列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (pslarf の場合) DOUBLE PRECISION (pdlarf の場合) COMPLEX (pclarf の場合) COMPLEX*16 (pzlarf の場合) 配列、次元は ($lwork$)。 If $incv = 1$, if $side = 'L'$, if $ivcol = iccol$, $lwork \geq nqc0$ else $lwork \geq mpc0 + \max(1, nqc0)$ end if else if $side = 'R'$, $lwork \geq nqc0 + \max(\max(1, mpc0), \text{numroc}(\text{numroc}(m + icoffc, nb_v, 0, 0, npcol), nb_v, 0, 0, lcmq))$ end if else if $incv = m_v$, if $side = 'L'$, $lwork \geq mpc0 + \max(\max(1, nqc0), \text{numroc}(\text{numroc}(m + iroffc, mb_v, 0, 0, nprw), mb_v, 0, 0, lcmv))$ else if $side = 'R'$, if $ivrow = icrow$, $lwork \geq mpc0$ else $lwork \geq nqc0 + \max(1, mpc0)$

```

        end if
    end if
end if,

lcm は nprow と npc01 の最小公倍数である。 lcm =
ilcm(nprow, npc01)、
lcmp = lcm / nprow、 lcmq = lcm / npc01、

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npc01 ),
mpc0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
npc0 = numroc( n+icoffc, nb_c, mycol, iccol, npc01 ),

ilcm、indxg2p、および numroc は、ScaLAPACK ツール関数で
ある。
myrow、mycol、nprow、および npc01 は、サブルーチン
blacs_gridinfo を呼び出して求められる。

```

出力パラメーター

c (ローカル)
 終了時に、 $\text{sub}(C)$ は $Q * \text{sub}(C)$ ($\text{side} = 'L'$ の場合)
 または $\text{sub}(C) * Q$ ($\text{side} = 'R'$ の場合) で上書きされる。

p?larfb

ブロック・リフレクターまたはその転置/共役転置を一般矩形行列に適用する。

構文

```

call pslarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t, c, ic,
            jc, descc, work)
call pdlarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t, c, ic,
            jc, descc, work)
call pclarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t, c, ic,
            jc, descc, work)
call pzlarfb(side, trans, direct, storev, m, n, k, v, iv, jv, descv, t, c, ic,
            jc, descc, work)

```

説明

このルーチンは、実/複素ブロック・リフレクター Q またはその転置 Q^T (または共役置換 Q^H) を、 $m \times n$ の実/複素分布行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ に左または右のいずれかから適用する。

入力パラメーター

side (グローバル) CHARACTER。

side = 'L' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型の場合) を左から適用する。

side = 'R' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型の場合) を右から適用する。

trans (グローバル) CHARACTER。
trans = 'N' の場合、 Q を適用する (転置なし)。
trans = 'T' の場合、 Q^T を適用する (転置、実数型の場合)。
trans = 'C' の場合、 Q^H を適用する (共役転置、複素数型の場合)。

direct (グローバル) CHARACTER。
 基本リフレクターの積からどのように Q が表現されているかを示す。
direct = 'F' の場合、 $Q = H(1) H(2) \dots H(k)$ (順方向)
direct = 'B' の場合、 $Q = H(k) \dots H(2) H(1)$ (逆方向)

storev (グローバル) CHARACTER。
 基本リフレクターを定義するベクトルがどのように格納されているかを示す。
storev = 'C' の場合、列方向
storev = 'R' の場合、行方向。

m (グローバル) INTEGER。
 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。
 ($m \geq 0$)。

n (グローバル) INTEGER。
 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。
 ($n \geq 0$)。

k (グローバル) INTEGER。
 行列 T の次数。

v (ローカル)
 REAL (pslarfb の場合)
 DOUBLE PRECISION (pdlarfb の場合)
 COMPLEX (pclarfb の場合)
 COMPLEX*16 (pzlarfb の場合)
 ローカルメモリーにある、次元
 ($lld_v, LOCc(jv+k-1)$) (*storev* = 'C' の場合)、
 ($lld_v, LOCc(jv+m-1)$) (*storev* = 'R' および *side* = 'L' の場合)、
 または
 ($lld_v, LOCc(jv+n-1)$) (*storev* = 'R' および *side* = 'R' の場合) の
 配列へのポインター。
 Householder 変換を表す乱数ベクトル V のローカル部分を格納する。
storev = 'C' および *side* = 'L' の場合、
 $lld_v \geq \max(1, LOCr(iv+m-1))$ 。
storev = 'C' および *side* = 'R' の場合、
 $lld_v \geq \max(1, LOCr(iv+n-1))$ 。
storev = 'R' の場合、 $lld_v \geq LOCr(jv+k-1)$ 。

<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(V)</i> の最初の行と最初の列を示す、グローバル配列 <i>V</i> の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>V</i> の配列ディスクリプター。
<i>c</i>	(ローカル)。 REAL (<i>pslarfb</i> の場合) DOUBLE PRECISION (<i>pdlarfb</i> の場合) COMPLEX (<i>pclarfb</i> の場合) COMPLEX*16 (<i>pzlarfb</i> の場合) ローカルメモリーにある、 <i>sub(C)</i> のローカル部分を格納する、次元 (<i>lld_c, LOCc(jc+n-1)</i>) の配列へのポインター。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(C)</i> の最初の行と最初の列を示す、グローバル配列 <i>C</i> の行インデックスと列インデックス。
<i>descC</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (<i>pslarfb</i> の場合) DOUBLE PRECISION (<i>pdlarfb</i> の場合) COMPLEX (<i>pclarfb</i> の場合) COMPLEX*16 (<i>pzlarfb</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。 <pre> If storev = 'C', if side = 'L', lwork ≥ (nqc0 + mpc0) * k else if side = 'R', lwork ≥ (nqc0 + max(npv0 + numroc(numroc(n + icoffc, nb_v, 0, 0, npc01), nb_v, 0, 0, lcmq), mpc0)) * k end if else if storev = 'R', if side = 'L', lwork ≥ (mpc0 + max(mqv0 + numroc(numroc(m+iroffc, mb_v, 0, 0, nprow), mb_v, 0, 0, lcmq), nqc0)) * k else if side = 'R', lwork ≥ (mpc0 + nqc0) * k end if end if, </pre> ここで、 <i>lcmq</i> = <i>lcm</i> / <i>npc01</i> 、 <i>lcm</i> = <i>icl</i> m(<i>nprow</i> , <i>npc01</i>)、

```

irowfv = mod( iv-1, mb_v ), icofv = mod( jv-1, nb_v ),
ivrow = indxg2p( iv, mb_v, myrow, rsrc_v, nprow ),
ivcol = indxg2p( jv, nb_v, mycol, csrc_v, npcol ),
MqV0 = numroc( m+icofv, nb_v, mycol, ivcol, npcol ),
NpV0 = numroc( n+irowfv, mb_v, myrow, ivrow, nprow ),

irowfc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcol ),
MpC0 = numroc( m+irowfc, mb_c, myrow, icrow, nprow ),
NpC0 = numroc( n+icoffc, mb_c, myrow, icrow, nprow ),
NqC0 = numroc( n+icoffc, nb_c, mycol, iccol, npcol ),

ilcm, indxg2p, および numroc は、ScaLAPACK ツール関数で
ある。
myrow, mycol, nprow, および npcol は、サブルーチン
blacs_gridinfo を呼び出して求められる。

```

出力パラメーター

t (ローカル)
REAL (pslarfb の場合)
DOUBLE PRECISION (pdlarfb の場合)
COMPLEX (pclarfb の場合)
COMPLEX*16 (pzlarfb の場合)
配列、次元は (*mb_v*, *mb_v*) (*storev* = 'R' の場合) または
(*nb_v*, *nb_v*) (*storev* = 'C' の場合)。三角行列 *t* は、ブロック・
リフレクターの表現である。

c (ローカル)
終了時に、sub(*C*) は、 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q$ 、または
 $\text{sub}(C) * Q'$ のいずれかで上書きされる。

p?larfc

一般行列に基本リフレクターの共役転置を適用する。

構文

```

call pclarfc(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)
call pzlarfc(side, m, n, v, iv, jv, descv, incv, tau, c, ic, jc, descc, work)

```

説明

このルーチンは、複素基本リフレクター Q^H を、 $m \times n$ の複素分布行列
 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ に左または右のいずれかから適用する。 Q は次の形
式で表現される。

$$Q = I - \tau * v * v'$$

tau は複素スカラー、*v* は複素ベクトルである。

$\tau = 0$ の場合、 Q は単位行列をとる。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER。 <i>side</i> = 'L' の場合、形式 $Q^H * \text{sub}(C)$ 。 <i>side</i> = 'R' の場合、形式 $\text{sub}(C) * Q^H$ 。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>v</i>	(ローカル)。 COMPLEX (pclarfc の場合) COMPLEX*16 (pzlarfc の場合) ローカルメモリーにある、Householder 変換 Q を表す分散行列 v のローカル部分を格納する、次元 ($11d_v, *$) の配列へのポインター。 $v(iv:iv+m-1, jv)$ (<i>side</i> = 'L' および $incv = 1$ の場合) $v(iv, jv:jv+m-1)$ (<i>side</i> = 'L' および $incv = m_v$ の場合) $v(iv:iv+n-1, jv)$ (<i>side</i> = 'R' および $incv = 1$ の場合) $v(iv, jv:jv+n-1)$ (<i>side</i> = 'R' および $incv = m_v$ の場合) Q の表現におけるベクトル v 。 $\tau = 0$ の場合、 v は使用されない。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 V の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 V の配列ディスクリプター。
<i>incv</i>	(グローバル) INTEGER。 v の成分のグローバルな増分。このバージョンでは、1 と m_v の2つの $incv$ の値のみサポートされる。 $incv$ の値は、ゼロであってはならない。
<i>tau</i>	(ローカル)。 COMPLEX (pclarfc の場合) COMPLEX*16 (pzlarfc の場合) 配列、次元は $LOCc(jv)$ ($incv = 1$ の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。 τ は、分散行列 V に関連付けられる。
<i>c</i>	(ローカル)。 COMPLEX (pclarfc の場合) COMPLEX*16 (pzlarfc の場合) ローカルメモリーにある、 $\text{sub}(C)$ のローカル部分を格納する、次元 ($11d_c, LOCc(jc+n-1)$) の配列へのポインター。

ic, jc (グローバル) INTEGER。それぞれ、部分行列 *sub(C)* の最初の行と最初の列を示す、グローバル配列 *C* の行インデックスと列インデックス。

desc (グローバルおよびローカル) INTEGER 配列、次元は (*dlen*)。分散行列 *C* の配列ディスクリプター。

work (ローカル)。

```

COMPLEX (pclarfc の場合)
COMPLEX*16 (pzlarfc の場合)
ワークスペース配列、次元は (lwork)。

If incv = 1,
  if side = 'L',
    if ivcol = iccol,
      lwork ≥ nqc0
    else
      lwork ≥ mpc0 + max( 1, nqc0 )
    end if
  else if side = 'R',
    lwork ≥ nqc0 + max( max( 1, mpc0 ), numroc( numroc(
      n+icoffc, nb_v, 0, 0, npcol ), nb_v, 0, 0, lcmq ) )
    end if
  else if incv = m_v,
    if side = 'L',
      lwork ≥ mpc0 + max( max( 1, nqc0 ), numroc(
numroc(
      m+iroffc, mb_v, 0, 0, nprow ), mb_v, 0, 0, lcmp
) )
    else if side = 'R',
      if ivrow = icrow,
        lwork ≥ mpc0
      else
        lwork ≥ nqc0 + max( 1, mpc0 )
      end if
    end if
  end if,

```

lcm は *nprow* と *npcol* の最小公倍数である。

lcm = *ilcm*(*nprow*, *npcol*), *lcmp* = *lcm* / *nprow*,
lcmq = *lcm* / *npcol*,

```

iroffc = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcol ),
mpc0 = numroc( m+iroffc, mb_c, myrow, icrow, nprow ),
nqc0 = numroc( n+icoffc, nb_c, mycol, iccol, npcol ),

```

ilcm、*indxg2p*、および *numroc* は、ScaLAPACK ツール関数である。

myrow、*mycol*、*nprow*、および *npcol* は、サブルーチン *blacs_gridinfo* を呼び出して求められる。

出力パラメーター

c (ローカル)。終了時に、 $\text{sub}(C)$ は $Q^H * \text{sub}(C)$ (*side* = 'L' の場合) または $\text{sub}(C) * Q^H$ (*side* = 'R' の場合) で上書きされる。

p?larfg

基本リフレクター (Householder 行列) を生成する。

構文

```
call pslarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pdlarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pclarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
call pzlarfg(n, alpha, iax, jax, x, ix, jx, descx, incx, tau)
```

説明

このルーチンは、次を満たすような次数 *n* の実 / 複素基本リフレクター *H* を生成する。

$$H * \text{sub}(X) = H * \begin{pmatrix} x(iax, jax) \\ \vdots \\ x \end{pmatrix} = \begin{pmatrix} \alpha \\ \vdots \\ 0 \end{pmatrix}, \quad H' * H = I,$$

ここで、*alpha* はスカラー (複素数型の場合、実スカラー)、 $\text{sub}(X)$ は (*n*-1) 成分で構成される実 / 複素乱数ベクトル $X(ix:ix+n-2, jx)$ (*incx* = 1 の場合) または $X(ix, jx:jx+n-2)$ (*incx* = *descx*(*m*)) の場合) である。*H* は次の形式で表現される。

$$H = I - \tau * \begin{pmatrix} 1 \\ \vdots \\ v \end{pmatrix} * \begin{pmatrix} 1 & v' \end{pmatrix}$$

tau は実数 / 複素スカラー、*v* は実数 / 複素の (*n*-1) 成分で構成されるベクトルである。*H* はエルミートではない点に注意する。

$\text{sub}(X)$ の成分がすべてゼロの場合 (かつ、複素数型の場合 $X(iax, jax)$ が実数ならば)、*tau* = 0 となり、*H* は単位行列をとる。

それ以外の場合、 $1 \leq \text{real}(\tau) \leq 2$ かつ $\text{abs}(\tau - 1) \leq 1$ 。

入力パラメーター

n (グローバル) INTEGER。
基本リフレクターのグローバル次数。 $n \geq 0$ 。

iax, jax (グローバル) INTEGER。
 $X(iax, jax)$ の *x* のグローバル行インデックスとグローバル列インデックス。

x (ローカル)
REAL (pslarfg の場合)
DOUBLE PRECISION (pdlarfg の場合)
COMPLEX (pclarfg の場合)
COMPLEX*16 (pzlarfg の場合)

ローカルメモリーにある、次元 ($lld_x, *$) の配列へのポインター。乱数ベクトル $sub(X)$ のローカル部分を格納する。このルーチンに入る前に、増分された配列 $sub(X)$ にベクトル x を格納しなければならない。

<i>ix, jx</i>	(グローバル) INTEGER。 それぞれ、 $sub(X)$ の最初の行と最初の列を示す、グローバル配列 X の行インデックスと列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。 次元 ($dlen_$) の配列。分散行列 X の配列ディスクリプター。
<i>incx</i>	(グローバル) INTEGER。 x の成分のグローバルな増分。このバージョンでは、1 と m_x の2つの <i>incx</i> の値のみサポートされる。 <i>incx</i> の値は、ゼロであってはならない。

出力パラメーター

<i>alpha</i>	(ローカル)。 REAL (pslafg の場合) DOUBLE PRECISION (pdlafg の場合) COMPLEX (pclafg の場合) COMPLEX*16 (pzlafg の場合) 終了時に、 <i>alpha</i> は、ベクトル $sub(X)$ のプロセス範囲において計算される。
<i>x</i>	(ローカル)。 終了時に、ベクトル v で上書きされる。
<i>tau</i>	(ローカル)。 REAL (pslarfg の場合) DOUBLE PRECISION (pdlarfg の場合) COMPLEX (pclarfg の場合) COMPLEX*16 (pzlarfg の場合) 配列、次元は $LOCc(jx)$ ($incx = 1$ の場合) または $LOCr(ix)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。 <i>tau</i> は、分散行列 X に関連付けられる。

p?larft

ブロック・リフレクター $H = I - VTV^H$ の三角ベクトル T を生成する。

構文

```
call pslarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pdlarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

```
call pclarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pzlarft(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

説明

このルーチンは、 k 個の基本リフレクターの積として定義されている次数 n の実 / 複素ブロック・リフレクター H の三角係数 T を生成する。

$direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ と T は上三角である。

$direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ と T は下三角である。

$storev = 'c'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは、分散行列 V の i 番目の列に次のように格納される。

$$H = I - V * T * V'$$

$storev = 'r'$ の場合、基本リフレクター $H(i)$ を定義するベクトルは、分散行列 V の i 番目の行に次のように格納される。

$$H = I - V' * T * V$$

入力パラメーター

<i>direct</i>	(グローバル) CHARACTER*1。 ブロック・リフレクターを生成するために基本リフレクターを乗算する次数を指定する。 $direct = 'F'$ の場合、 $H = H(1) H(2) \dots H(k)$ (順方向) $direct = 'B'$ の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	(グローバル) CHARACTER*1。 基本リフレクターを定義するベクトルがどのように格納されているかを指定する (次の「アプリケーション・ノート」を参照)。 $storev = 'c'$ の場合、列方向。 $storev = 'r'$ の場合、行方向。
<i>n</i>	(グローバル) INTEGER。 ブロック・リフレクター H の次数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 三角係数 T の次数 (基本リフレクターの個数に等しい)。 $1 \leq k \leq mb_v (= nb_v)$ 。
<i>v</i>	REAL (pslarft の場合) DOUBLE PRECISION (pdlarft の場合) COMPLEX (pclarft の場合) COMPLEX*16 (pzlarft の場合) ローカルメモリーにある、次元 ($LOCr(iv+n-1), LOCc(jv+k-1)$) ($storev = 'c'$ の場合) または ($LOCr(iv+k-1), LOCc(jv+n-1)$) ($storev = 'r'$ の場合) の配列へのポインター。 分散行列 V は Householder ベクトルを格納する。 (次の「アプリケーション・ノート」を参照)。

<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(V)</i> の最初の行と最初の列を示す、グローバル <i>z</i> 列 <i>v</i> の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_v</i>)。分散行列 <i>V</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>pslarft</i> の場合) DOUBLE PRECISION (<i>pdlarft</i> の場合) COMPLEX (<i>pclarft</i> の場合) COMPLEX*16 (<i>pzlarft</i> の場合) 配列、次元は <i>LOCr(iv+k-1)</i> (<i>incv = m_v</i> の場合) または <i>LOCc(jv+k-1)</i> (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。 <i>tau</i> は、分散行列 <i>V</i> に関連付けられる。
<i>work</i>	(ローカル)。 REAL (<i>pslarft</i> の場合) DOUBLE PRECISION (<i>pdlarft</i> の場合) COMPLEX (<i>pclarft</i> の場合) COMPLEX*16 (<i>pzlarft</i> の場合) ワークスペース配列、次元は $(k*(k-1)/2)$ 。

出力パラメーター

<i>v</i>	REAL (<i>pslarft</i> の場合) DOUBLE PRECISION (<i>pdlarft</i> の場合) COMPLEX (<i>pclarft</i> の場合) COMPLEX*16 (<i>pzlarft</i> の場合)
<i>t</i>	(ローカル)。 REAL (<i>pslarft</i> の場合) DOUBLE PRECISION (<i>pdlarft</i> の場合) COMPLEX (<i>pclarft</i> の場合) COMPLEX*16 (<i>pzlarft</i> の場合) 配列、次元は (<i>nb_v, nb_v</i>) (<i>storev = 'Col'</i> の場合) または (<i>mb_v, mb_v</i>) (それ以外の場合)。 <i>v</i> に関連するブロック・リフレクターの $k \times k$ の三角係数が格納される。 <i>direct = 'F'</i> の場合、 <i>t</i> は上三角。 <i>direct = 'B'</i> の場合、 <i>t</i> は下三角。

アプリケーション・ノート

行列 *V* の形状と *H(i)* を定義するベクトルの格納形式の例を、 $n=5$ 、 $k=3$ において次の図に示す。1 に等しい成分は格納されない。対応する配列成分は変更されるが、終了時に復元される。配列の残りの部分は使用されない。

$direct = 'F'$ および $storev = 'C'$:

$$V(iv:iv+n-1, jv:jv+k-1) = \begin{bmatrix} 1 & & & & \\ v1 & 1 & & & \\ v1 & v2 & 1 & & \\ v1 & v2 & v3 & & \\ v1 & v2 & v3 & & \end{bmatrix}$$

$direct = 'F'$ および $storev = 'R'$:

$$V(iv:iv+k-1, jv:jv+n-1) = \begin{bmatrix} 1 & v1 & v1 & v1 & v1 \\ & 1 & v2 & v2 & v2 \\ & & & 1 & v3 & v3 \end{bmatrix}$$

$direct = 'B'$ および $storev = 'C'$:

$$V(iv:iv+n-1, jv:jv+k-1) = \begin{bmatrix} v1 & v2 & v3 \\ v1 & v2 & v3 \\ 1 & v2 & v3 \\ & & 1 & v3 \\ & & & & 1 \end{bmatrix}$$

$direct = 'B'$ および $storev = 'R'$:

$$V(iv:iv+k-1, jv:jv+n-1) = \begin{bmatrix} v1 & v1 & 1 \\ v2 & v2 & v2 & 1 \\ v3 & v3 & v3 & v3 & 1 \end{bmatrix}$$

p?larz

`p?tzrzf` が返したとおりの基本リフレクターを一般行列に適用する。

構文

```
call pslarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
work)
call pdlarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
work)
call pclarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
work)
call pzlarz(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc, descc,
work)
```

説明

このルーチンは、実/複素基本リフレクター Q (または Q^T) を、 $m \times n$ の実/複素分布行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ に左または右のいずれかから適用する。 Q は次の形式で表現される。

$$Q = I - \tau * v * v'$$

τ は実/複素スカラー、 v は実/複素ベクトルである。

$\tau = 0$ の場合、 Q は単位行列をとる。

Q は `p?tzrzf` から返されたとおりの、 k 個の基本リフレクターの積である。

入力パラメーター

`side` (グローバル) CHARACTER。

side = 'L' の場合、形式 $Q * \text{sub}(C)$ 。
side = 'R' の場合、形式 $\text{sub}(C) * Q$, $Q = Q^T$ (実数型の場合)。

m (グローバル) INTEGER。
 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。
 ($m \geq 0$)。

n (グローバル) INTEGER。
 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。
 ($n \geq 0$)。

l (グローバル)。INTEGER。
 Householder リフレクターとして意味を持つ部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。*side* = 'L' の場合、 $m \geq l \geq 0$ 。
side = 'R' の場合、 $n \geq l \geq 0$ 。

v (ローカル)。
 REAL (pslarz の場合)
 DOUBLE PRECISION (pdlarz の場合)
 COMPLEX (pclarz の場合)
 COMPLEX*16 (pzlarz の場合)
 ローカルメモリーにある、Householder 変換 Q を表す分散行列 v のローカル部分を格納する、次元 ($lld_v, *$) の配列へのポインター。
 $v(iv:iv+l-1, jv)$ (*side* = 'L' および *incv* = 1 の場合)
 $v(iv, jv:jv+l-1)$ (*side* = 'L' および *incv* = m_v の場合)
 $v(iv:iv+l-1, jv)$ (*side* = 'R' および *incv* = 1 の場合)
 $v(iv, jv:jv+l-1)$ (*side* = 'R' および *incv* = m_v の場合)
 Q の表現におけるベクトル v 。tau = 0 の場合、 v は使用されない。

iv, jv (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 V の行インデックスと列インデックス。

descv (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_v$)。分散行列 V の配列ディスクリプター。

incv (グローバル) INTEGER。
 v の成分のグローバルな増分。このバージョンでは、1 と m_v の2つの *incv* の値のみサポートされる。
incv の値は、ゼロであってはならない。

tau (ローカル)。
 REAL (pslarz の場合)
 DOUBLE PRECISION (pdlarz の場合)
 COMPLEX (pclarz の場合)
 COMPLEX*16 (pzlarz の場合) 配列、次元は $LOCc(jv)$ (*incv* = 1 の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。
 τ は、分散行列 V に関連付けられる。

<i>c</i>	(ローカル)。 REAL (pslarz の場合) DOUBLE PRECISION (pdlarz の場合) COMPLEX (pclarz の場合) COMPLEX*16 (pzlarz の場合) ローカルメモリーにある、 sub(C) のローカル部分を格納する、 次元 (<i>lld_c</i> , <i>LOCc(jc+n-1)</i>) の配列へのポインター。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 sub(C) の最初の行 と最初の列を示す、グローバル配列 <i>C</i> の行インデックスと列イン デックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (pslarz の場合) DOUBLE PRECISION (pdlarz の場合) COMPLEX (pclarz の場合) COMPLEX*16 (pzlarz の場合) 配列、次元は (<i>lwork</i>)。 If <i>incv</i> = 1, if <i>side</i> = 'L', if <i>ivcol</i> = <i>iccol</i> , <i>lwork</i> ≥ <i>NqC0</i> else <i>lwork</i> ≥ <i>MpC0</i> + max(1, <i>NqC0</i>) end if else if <i>side</i> = 'R', <i>lwork</i> ≥ <i>NqC0</i> + max(max(1, <i>MpC0</i>), numroc(numroc(<i>n+icoffc</i> , <i>nb_v</i> , 0, 0, <i>npcol</i>), <i>nb_v</i> , 0, 0, <i>lcmq</i>)) end if else if <i>incv</i> = <i>m_v</i> , if <i>side</i> = 'L', <i>lwork</i> ≥ <i>MpC0</i> + max(max(1, <i>NqC0</i>), numroc(numroc(<i>m+iroffc</i> , <i>mb_v</i> , 0, 0, <i>npro</i>), <i>mb_v</i> , 0, 0, <i>lcmp</i>)) else if <i>side</i> = 'R', if <i>ivrow</i> = <i>icrow</i> , <i>lwork</i> ≥ <i>MpC0</i> else <i>lwork</i> ≥ <i>NqC0</i> + max(1, <i>MpC0</i>) end if end if end if, <i>lcm</i> は <i>npro</i> と <i>npcol</i> の最小公倍数である。 <i>lcm</i> = ilcm(<i>npro</i> , <i>npcol</i>), <i>lcmp</i> = <i>lcm</i> / <i>npro</i> , <i>lcmq</i> = <i>lcm</i> / <i>npcol</i> ,

```

irow = mod( ic-1, mb_c ), icoffc = mod( jc-1, nb_c ),
icrow = indxg2p( ic, mb_c, myrow, rsrc_c, nprow ),
iccol = indxg2p( jc, nb_c, mycol, csrc_c, npcrow ),
mpc0 = numroc( m+irow, mb_c, myrow, icrow, nprow ),
nqc0 = numroc( n+icoffc, nb_c, mycol, iccol, npcrow ),
ilcm, indxg2p, および numroc は、ScaLAPACK ツール関数で
ある。
myrow, mycol, nprow, および npcrow は、サブルーチン
blacs_gridinfo を呼び出して求められる。

```

出力パラメーター

c (ローカル)。終了時に、 $\text{sub}(C)$ は $Q * \text{sub}(C)$ ($\text{side} = 'L'$ の場合) または $\text{sub}(C) * Q$ ($\text{side} = 'R'$ の場合) で上書きされる。

p?larzb

`p?tzrzf` が返したとおりのブロック・リフレクターまたはその転置/共役転置を、一般行列に適用する。

構文

```

call pslarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,
            ic, jc, descc, work)
call pdlarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,
            ic, jc, descc, work)
call pclarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,
            ic, jc, descc, work)
call pzlarzb(side, trans, direct, storev, m, n, k, l, v, iv, jv, descv, t, c,
            ic, jc, descc, work)

```

説明

このルーチンは、実/複素ブロック・リフレクター Q またはその転置 Q^T (複素数型の場合、共役転置 Q^H) を、 $m \times n$ の実/複素分布行列 $\text{sub}(C) = C(\text{ic}:\text{ic}+m-1, \text{jc}:\text{jc}+n-1)$ に左または右のいずれかから適用する。

Q は `p?tzrzf` から返されたとおり、 k 個の基本リフレクターの積である。

現時点で、 $\text{storev} = 'R'$ と $\text{direct} = 'B'$ のみがサポートされている。

入力パラメーター

side (グローバル) CHARACTER。
 $\text{side} = 'L'$ の場合、 Q または Q^T (複素数型の場合 Q^H) を左から適用する。
 $\text{side} = 'R'$ の場合、 Q または Q^T (複素数型の場合 Q^H) を右から適用する。

<i>trans</i>	(グローバル) CHARACTER。 <i>trans</i> = 'N' の場合、 <i>Q</i> を適用する (転置なし)。 <i>trans</i> = 'T' の場合、転置。 Q^T (実数型) を適用する。 <i>trans</i> = 'C' の場合、共役転置。 Q^H (複素数型) を適用する。
<i>direct</i>	(グローバル) CHARACTER。 基本リフレクターの積からどのように <i>H</i> が表現されているかを示す。 <i>direct</i> = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない) <i>direct</i> = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)
<i>storev</i>	(グローバル) CHARACTER。 基本リフレクターを定義するベクトルがどのように格納されているかを示す。 <i>storev</i> = 'C' の場合、列方向 (現時点でこの機能はサポートされていない)。 <i>storev</i> = 'R' の場合、行方向。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>sub(C)</i> の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>sub(C)</i> の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 行列 <i>T</i> の次数。(積がブロック・リフレクターを定義する基本リフレクターの個数に等しい)。
<i>l</i>	(グローバル) INTEGER。 Householder リフレクターとして意味を持つ部分が格納されている分散部分行列 <i>sub(A)</i> の列数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$ 。
<i>v</i>	(ローカル)。 REAL (pslarzb の場合) DOUBLE PRECISION (pdlarzb の場合) COMPLEX (pclarzb の場合) COMPLEX*16 (pzlarzb の場合) ローカルメモリーにある、次元 (<i>lld_v</i> , <i>LOCc</i> (<i>lv+m-1</i>)) (<i>side</i> = 'L' の場合) または (<i>lld_v</i> , <i>LOCc</i> (<i>lv+m-1</i>)) (<i>side</i> = 'R' の場合) の配列へのポインター。 <i>p?tzrzf</i> が返したとおりの Householder 変換を表す乱数ベクトル <i>V</i> のローカル部分を格納する。 $lld_v \geq LOCr(iv+k-1)$ 。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(V)</i> の最初の行と最初の列を示す、グローバル配列 <i>V</i> の行インデックスと列インデックス。

ここで、 $lcmq = lcm / npc\text{ol}$ 、 $lcm = iclm(np\text{row}, npc\text{ol})$ 、 $iroffv = \text{mod}(iv-1, mb_v)$ 、 $icoffv = \text{mod}(jv-1, nb_v)$ 、 $ivrow = \text{indxg2p}(iv, mb_v, myrow, rsrc_v, np\text{row})$ 、 $ivcol = \text{indxg2p}(jv, nb_v, mycol, csrc_v, npc\text{ol})$ 、 $MqV0 = \text{numroc}(m+icoffv, nb_v, mycol, ivcol, npc\text{ol})$ 、 $NpV0 = \text{numroc}(n+iroffv, mb_v, myrow, ivrow, np\text{row})$ 、 $iroffc = \text{mod}(ic-1, mb_c)$ 、 $icoffc = \text{mod}(jc-1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, myrow, rsrc_c, np\text{row})$ 、 $iccol = \text{indxg2p}(jc, nb_c, mycol, csrc_c, npc\text{ol})$ 、 $MpC0 = \text{numroc}(m+iroffc, mb_c, myrow, icrow, np\text{row})$ 、 $NpC0 = \text{numroc}(n+icoffc, mb_c, myrow, icrow, np\text{row})$ 、 $NqC0 = \text{numroc}(n+icoffc, nb_c, mycol, iccol, npc\text{ol})$ 、 $ilcm$ 、 indxg2p 、および numroc は、ScaLAPACK ツール関数である。

$myrow$ 、 $mycol$ 、 $np\text{row}$ 、および $npc\text{ol}$ は、サブルーチン blacs_gridinfo を呼び出して求められる。

出力パラメーター

c (ローカル) 終了時に、 $\text{sub}(C)$ は、 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q$ 、または $\text{sub}(C) * Q'$ のいずれかで上書きされる。

p?larzc

$p?t\text{zrzf}$ が返したとおりの基本リフレクターの共役転置を、一般行列に適用(乗算)する。

構文

```
call pclarzc(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc,
             desc, work)
```

```
call pzlarzc(side, m, n, l, v, iv, jv, descv, incv, tau, c, ic, jc,
             desc, work)
```

説明

このルーチンは、複素基本リフレクター Q^H を、 $m \times n$ の複素分布行列 $\text{sub}(C) = C(ic:ic+m-1, jc:jc+n-1)$ に左または右のいずれかから適用する。 Q は次の形式で表現される。

$$Q = I - \tau * v * v'$$

τ は複素スカラー、 v は複素ベクトルである。

$\tau = 0$ の場合、 Q は単位行列をとる。

Q は [p?tzrzf](#) から返されたとおり、 k 個の基本リフレクターの積である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER。 <i>side</i> = 'L' の場合、形式 $Q^H * \text{sub}(C)$ 。 <i>side</i> = 'R' の場合、形式 $\text{sub}(C) * Q^H$ 。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>l</i>	(グローバル) INTEGER。 Householder リフレクターとして意味を持つ部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 <i>side</i> = 'L' の場合、 $m \geq l \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq l \geq 0$ 。
<i>v</i>	(ローカル)。 COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合) ローカルメモリーにある、Householder 変換 Q を表す分散行列 v のローカル部分を格納する、次元 ($lld_v, *$) の配列へのポインター。 $v(iv:iv+l-1, jv)$ (<i>side</i> = 'L' および <i>incv</i> = 1 の場合) $v(iv, jv:jv+l-1)$ (<i>side</i> = 'L' および <i>incv</i> = m_v の場合) $v(iv:iv+l-1, jv)$ (<i>side</i> = 'R' および <i>incv</i> = 1 の場合) $v(iv, jv:jv+l-1)$ (<i>side</i> = 'R' および <i>incv</i> = m_v の場合) Q の表現におけるベクトル v 。 $\tau = 0$ の場合、 v は使用されない。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(V)$ の最初の行と最初の列を示す、グローバル配列 V の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen$)。分散行列 V の配列ディスクリプター。
<i>incv</i>	(グローバル)。INTEGER。 v の成分のグローバルな増分。このバージョンでは、1 と m_v の2つの <i>incv</i> の値のみサポートされる。 <i>incv</i> の値は、ゼロであってはならない。
<i>tau</i>	(ローカル)。 COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合) 配列、次元は $LOCc(jv)$ (<i>incv</i> = 1 の場合) または $LOCr(iv)$ (それ以外の場合)。Householder ベクトルに関する Householder スカラーを格納する。 τ は、分散行列 V に関連付けられる。

<i>c</i>	(ローカル)。 COMPLEX (pclarzc の場合) COMPLEX*16 (pzlarzc の場合) ローカルメモリーにある、 <i>sub(C)</i> のローカル部分を格納する、 次元 (<i>lld_c</i> , <i>LOCc(jc+n-1)</i>) の配列へのポインター。
<i>ic, jc</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(C)</i> の最初の行 と最初の列を示す、グローバル配列 <i>C</i> の行インデックスと列イ ンデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 <pre>If incv = 1, if side = 'L', if ivcol = iccol, lwork ≥ NqC0 else lwork ≥ MpC0 + max(1, NqC0) end if else if side = 'R', lwork ≥ nqc0 + max(max(1, mpc0), numroc(numroc(n+icoffc, nb_v, 0, 0, npc0), nb_v, 0, 0, lcmq)) end if else if incv = m_v, if side = 'L', lwork ≥ mpc0 + max(max(1, nqc0), numroc(numroc(m+iroffc, mb_v, 0, 0, nprow), mb_v, 0, 0, lcmq)) else if side = 'R' if ivrow = icrow, lwork ≥ mpc0 else lwork ≥ nqc0 + max(1, mpc0) end if end if end if, end if,</pre> <i>lcm</i> は <i>nprow</i> と <i>npcol</i> の最小公倍数である。 <i>lcm</i> = <i>ilcm</i> (<i>nprow</i> , <i>npcol</i>)、 <i>lcmp</i> = <i>lcm</i> / <i>nprow</i> 、 <i>lcmq</i> = <i>lcm</i> / <i>npcol</i> 、 <i>iroffc</i> = mod(<i>ic</i> -1, <i>mb_c</i>)、 <i>icoffc</i> = mod(<i>jc</i> -1, <i>nb_c</i>)、 <i>icrow</i> = <i>indxg2p</i> (<i>ic</i> , <i>mb_c</i> , <i>myrow</i> , <i>rsrc_c</i> , <i>nprow</i>)、 <i>iccol</i> = <i>indxg2p</i> (<i>jc</i> , <i>nb_c</i> , <i>mycol</i> , <i>csrc_c</i> , <i>npcol</i>)、 <i>MpC0</i> = <i>numroc</i> (<i>m+iroffc</i> , <i>mb_c</i> , <i>myrow</i> , <i>icrow</i> , <i>nprow</i>)、 <i>NqC0</i> = <i>numroc</i> (<i>n+icoffc</i> , <i>nb_c</i> , <i>mycol</i> , <i>iccol</i> , <i>npcol</i>)、 <i>ilcm</i> 、 <i>indxg2p</i> 、および <i>numroc</i> は、ScaLAPACK ツール関数で ある。

myrow, *mycol*, *nprow*, および *npcol* は、サブルーチン *blacs_gridinfo* を呼び出して求められる。

出力パラメーター

c (ローカル) 終了時に、*sub(C)* は、 $Q^H * \text{sub}(C)$ (*side* = 'L' の場合) または $\text{sub}(C) * Q^H$ (*side* = 'R' の場合) で上書きされる。

p?larzt

p?tzrzt が返したとおりのブロック・リフレクター $H = I - VTV^H$ の三角係数 T を生成する。

構文

```
call pslarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pdlarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pclarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
call pzlarzt(direct, storev, n, k, v, iv, jv, descv, tau, t, work)
```

説明

このルーチンは、[p?tzrzt](#) によって返される k 個の基本リフレクターの積として定義されている次数 $> n$ の実 / 複素ブロック・リフレクター H の三角係数 T を生成する。

direct = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ と T は上三角である。

direct = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ と T は下三角である。

storev = 'C' の場合、基本リフレクター $H(i)$ を定義するベクトルは、配列 v の i 番目の列に次のように格納される。

$$H = i - v * t * v'$$

storev = 'C' の場合、基本リフレクター $H(i)$ を定義するベクトルは、配列 v の i 番目の行に次のように格納される。

$$H = i - v' * t * v$$

現時点で、*storev* = 'R' と *direct* = 'B' のみがサポートされている。

入力パラメーター

direct (グローバル) CHARACTER。
 ブロック・リフレクターを生成するために基本リフレクターを乗算する次数を指定する。
direct = 'F' の場合、 $H = H(1) H(2) \dots H(k)$ (順方向、現時点でこの機能はサポートされていない)
direct = 'B' の場合、 $H = H(k) \dots H(2) H(1)$ (逆方向)

<i>storev</i>	(グローバル) CHARACTER。 基本リフレクターを定義するベクトルがどのように格納されているかを指定する。 <i>storev</i> = 'c' の場合、列方向 (現時点でこの機能はサポートされていない)。 <i>storev</i> = 'r' の場合、行方向。
<i>n</i>	(グローバル)。INTEGER。 ブロック・リフレクター <i>H</i> の次数。 $n \geq 0$ 。
<i>k</i>	(グローバル)。INTEGER。 三角係数 <i>T</i> の次数 (基本リフレクターの個数に等しい)。 $1 \leq k \leq mb_v (= nb_v)$ 。
<i>v</i>	REAL (<i>pslarzt</i> の場合) DOUBLE PRECISION (<i>pdlarzt</i> の場合) COMPLEX (<i>pclarzt</i> の場合) COMPLEX*16 (<i>pzlarzt</i> の場合) ローカルメモリーにある、次元 (<i>LOCr</i> (<i>iv+k-1</i>), <i>LOCc</i> (<i>iv+n-1</i>)) の配列へのポインター。 分散行列 <i>V</i> は Householder ベクトルを格納する。 次の「アプリケーション・ノート」を参照。
<i>iv, jv</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(V)</i> の最初の行と最初の列を示す、グローバル配列 <i>V</i> の行インデックスと列インデックス。
<i>descv</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>V</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>pslarzt</i> の場合) DOUBLE PRECISION (<i>pdlarzt</i> の場合) COMPLEX (<i>pclarzt</i> の場合) COMPLEX*16 (<i>pzlarzt</i> の場合) 配列、次元は <i>LOCr</i> (<i>iv+k-1</i>) (<i>incv</i> = <i>m_v</i> の場合) または <i>LOCc</i> (<i>iv+k-1</i>) (それ以外の場合)。 Householder ベクトルに関する Householder スカラーを格納する。 <i>tau</i> は、分散行列 <i>V</i> に関連付けられる。
<i>work</i>	(ローカル)。 REAL (<i>pslarzt</i> の場合) DOUBLE PRECISION (<i>pdlarzt</i> の場合) COMPLEX (<i>pclarzt</i> の場合) COMPLEX*16 (<i>pzlarzt</i> の場合) ワークスペース配列、次元は $k*(k-1)/2$ 。

出力パラメーター

<i>v</i>	REAL (<i>pslarzt</i> の場合) DOUBLE PRECISION (<i>pdlarzt</i> の場合) COMPLEX (<i>pclarzt</i> の場合) COMPLEX*16 (<i>pzlarzt</i> の場合)
----------	---

$$\begin{array}{c}
 1 \\
 . 1 \\
 . . 1 \\
 . . . \\
 v = \dots \\
 \left[\begin{array}{ccc}
 v1 & v2 & v3 \\
 v1 & v2 & v3
 \end{array} \right]
 \end{array}$$

direct = 'B' および *storev* = 'R':

$$\begin{array}{c}
 v \\
 1 \dots \left[\begin{array}{ccccc}
 v1 & v1 & v1 & v1 & v1 \\
 v2 & v2 & v2 & v2 & v2 \\
 v3 & v3 & v3 & v3 & v3
 \end{array} \right] \\
 . 1 \dots \\
 . . 1 \dots
 \end{array}$$

p?lascl

一般矩形行列に c_{to}/c_{from} として定義される実スカラーを掛ける。

構文

```

call pslascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pdlascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pclascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)
call pzlascl(type, cfrom, cto, m, n, a, ia, ja, desca, info)

```

説明

このルーチンは、 $m \times n$ の実 / 複素行列 $\text{sub}(A)$ に実 / 複素スカラー $cto/cfrom$ を掛ける。ここで、 $\text{sub}(A)$ は $A(ia:ia+m-1, ja:ja+n-1)$ である。最終結果 $cto * A(i, j) / cfrom$ がオーバーフロー / アンダーフローしない限り、演算はオーバーフロー / アンダーフローなしで実行される。

$\text{sub}(A)$ の形式は *type* によって、フル、上三角、下三角、または上 Hessenberg から指定される。

入カパラメーター

type (グローバル) CHARACTER。

入力分散行列の格納形式の *type*。
type = 'G' の場合、sub(A) はフル行列。
type = 'L' の場合、sub(A) は下三角行列。
type = 'U' の場合、sub(A) は上三角行列。
type = 'H' の場合、sub(A) は上 Hessenberg 行列。

cfrom, cto

(グローバル)

REAL (pslascl/pclascl の場合)
 DOUBLE PRECISION (pdlascl/pzlascl の場合)

分散行列 sub(A) に *cto/cfrom* を掛ける。
 最終結果 $cto * A(i, j) / cfrom$ がオーバーフロー/アンダーフローなしで表現される場合、 $A(i, j)$ はオーバーフロー/アンダーフローなしで計算される。*cfrom* は非ゼロでなければならない。

m

(グローバル) INTEGER。

演算を行う行数、すなわち、分散部分行列 sub(A) の行数。 $m \geq 0$ 。

n

(グローバル) INTEGER。

演算を行う列数、すなわち、分散部分行列 sub(A) の列数。 $n \geq 0$ 。

a

(ローカル入力/ローカル出力)

REAL (pslascl の場合)
 DOUBLE PRECISION (pdlascl の場合)
 COMPLEX (pclascl の場合)
 COMPLEX*16 (pzlascl の場合)

ローカルメモリーにある、次元 ($lld_a, LOC(ja+n-1)$) の配列へのポインター。

分散行列 sub(A) のローカル部分を格納する。

ia, ja

(グローバル) INTEGER。

それぞれ、部分行列 sub(A) の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。

desca

(グローバルおよびローカル) INTEGER。

配列、次元は (*dlen_*)。分散行列 A の配列ディスクリプター。

出力パラメーター

a

(ローカル) 終了時に、*cto/cfrom* が掛けられる分散行列のローカル部分が格納される。

info

(ローカル) INTEGER。

info = 0 の場合、正常に終了したことを示す。

info < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = - (*i**100+*j*)、*i* 番目の引数がスカラーで値が不正でならば *info* = -*i* であることを示す。

p?laset

行列の非対角成分を α に、対角成分を β に初期化する。

構文

```
call pslaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pdlaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pclaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
call pzlaset(uplo, m, n, alpha, beta, a, ia, ja, desca)
```

説明

このルーチンは、 $m \times n$ の分散行列 $\text{sub}(A)$ で対角を β に、対角外を α に初期化する。ここで、 $\text{sub}(A)$ は $A(ia:ia+m-1, ja:ja+n-1)$ である。

入カパラメーター

<i>uplo</i>	(グローバル) CHARACTER。 設定対象の分散行列 $\text{sub}(A)$ の部分を指定する。 <i>uplo</i> = 'U' の場合、上三角部分が設定される。 $\text{sub}(A)$ の厳密な下三角部分は変更されない。 <i>uplo</i> = 'L' の場合、下三角部分が設定される。 $\text{sub}(A)$ の厳密な上三角部分は変更されない。 それ以外の場合、行列 $\text{sub}(A)$ のすべての部分が設定される。
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 ($m \geq 0$)。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 ($n \geq 0$)。
<i>alpha</i>	(グローバル)。 REAL (pslaset の場合) DOUBLE PRECISION (pdlaset の場合) COMPLEX (pclaset の場合) COMPLEX*16 (pzlaset の場合) 非対角成分に設定する定数。
<i>beta</i>	(グローバル)。 REAL (pslaset の場合) DOUBLE PRECISION (pdlaset の場合) COMPLEX (pclaset の場合) COMPLEX*16 (pzlaset の場合) 対角成分に設定する定数。

出力パラメーター

<i>a</i>	(ローカル)。 REAL (pslaset の場合) DOUBLE PRECISION (pdlaset の場合) COMPLEX (pclaset の場合) COMPLEX*16 (pzlaset の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。設定する分散行列 <i>sub(A)</i> のローカル部分が格納される。終了時に、 $m \times n$ の部分行列 <i>sub(A)</i> の先頭は、次のように設定される。 <i>uplo</i> = 'U' の場合、 $A(ia+i-1, ja+j-1) = \alpha$ ($1 \leq i \leq j-1, 1 \leq j \leq n$) <i>uplo</i> = 'L' の場合、 $A(ia+i-1, ja+j-1) = \alpha$ ($j+1 \leq i \leq m, 1 \leq j \leq n$) それ以外の場合、 $A(ia+i-1, ja+j-1) = \alpha$ ($1 \leq i \leq m, 1 \leq j \leq n, ia+i.ne.ja+j$) すべての <i>uplo</i> で、 $A(ia+i-1, ja+i-1) = \beta$ ($1 \leq i \leq \min(m,n)$)
<i>ia, ja</i>	(グローバル) INTEGER。 それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 次元 (<i>dlen_</i>) の配列。分散行列 <i>A</i> の配列ディスクリプター。

p?lasmsub

安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。

構文

```
call pslasmsub(a, desca, i, l, k, smlnum, buf, lwork)
call pdlasmsub(a, desca, i, l, k, smlnum, buf, lwork)
```

説明

このルーチンは、安全に 0 に設定できる小さい劣対角成分を行列の一番下から検出する。このルーチンは、グローバルな最大値を検出するため、すべてのプロセスで呼び出されなければならない。

入力パラメーター

a (グローバル)。

	REAL (pslasmsub の場合) DOUBLE PRECISION (pdlasmsub の場合) 配列、次元は (<i>desca</i> (<i>lld_</i>),*)。 三重対角部分をスキャンされた Hessenberg 行列を格納する。終了時に変更されない。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 次元 (<i>dlen_</i>) の配列。 分散行列 <i>A</i> の配列ディスクリプター。
<i>i</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番下のグローバル位置。終了時に変更されない。
<i>l</i>	(グローバル) INTEGER。 <i>A</i> の縮退されていない部分行列の一番上のグローバル位置。終了時に変更されない。
<i>smlnum</i>	(グローバル)。 REAL (pslasmsub の場合) DOUBLE PRECISION (pdlasmsub の場合) 与えられた行列に対する「小さい数」を格納する。終了時に変更されない。
<i>lwork</i>	(グローバル) INTEGER。 終了時に、 <i>lwork</i> は、 <i>work</i> バッファのサイズである。 この値は、 $2 * \text{ceil}(\text{ceil}((i-1)/\text{hbl}) / \text{lcm}(\text{nprow}, \text{npcol}))$ 以上でなければならない。 <i>lcm</i> は公倍数以上で、 <i>nprow</i> x <i>npcol</i> は論理グリッドサイズである。
出力パラメーター	
<i>k</i>	(グローバル) INTEGER。 終了時に、 <i>A</i> の縮退されていない部分行列の一番下が格納される。次の条件を満たす。 $1 \leq m \leq i-1$
<i>buf</i>	(ローカル)。 REAL (pslasmsub の場合) DOUBLE PRECISION (pdlasmsub の場合) サイズ <i>lwork</i> の配列。

p?lassq

スケーリング形式で表現された二乗和を更新する。

構文

```
call pslasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pdlasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pclasq(n, x, ix, jx, descx, incx, scale, sumsq)
call pzlasq(n, x, ix, jx, descx, incx, scale, sumsq)
```

説明

このルーチンは、次のように値 scl と $smsq$ を返す。

$$scl^2 * smsq = x(1)^2 + \dots + x(n)^2 + scale^2 * sumsq$$

ここで、

$x(i) = \text{sub}(x) = x(ix + (jx-1)*descx(m_) + (i-1)*incx)$ (plassq/pdlassq の場合)、
 $x(i) = \text{sub}(x) = \text{abs}(x(ix + (jx-1)*descx(m_) + (i-1)*incx))$ (pclassq/pzlassq の場合)。
 実数ルーチン plassq/pdlassq では、 $smsq$ の値は非負であるとみなされ、 scl は次の値を返す。

$$scl = \max(scale, \text{abs}(x(i)))$$

複素数ルーチン pclassq/pzlassq では、 $smsq$ の値は少なくとも 1 であるとみなされ、 ssq の値は、次のように満たされる。

$$1.0 \leq ssq \leq sumsq + 2n$$

$scale$ の値は非負であるとみなされ、 scl は次の値を返す。

$$scl = \max_i(scale, \text{abs}(\text{real}(x(i))), \text{abs}(\text{aimag}(x(i))))$$

plassq の全ルーチンで、 $scale$ と $smsq$ は、それぞれ $scale$ と $smsq$ によって与えられなければならない。また $scale$ と $smsq$ は scl と ssq にそれぞれ上書きされる。

plassq の全ルーチンは唯一、ベクトル $\text{sub}(x)$ を出力に渡す。

入力パラメーター

n	(グローバル) INTEGER。乱数ベクトル $\text{sub}(x)$ の長さ。
x	REAL (plassq の場合) DOUBLE PRECISION (pdlassq の場合) COMPLEX (pclassq の場合) COMPLEX*16 (pzlassq の場合) スケールされた二乗和を計算するベクトル。 $x(ix + (jx-1)*m_x + (i-1)*incx)$ ($1 \leq i \leq n$)
ix	(グローバル) INTEGER。 $\text{sub}(X)$ の最初の行を示す、グローバル配列 X の行インデックス。
jx	(グローバル) INTEGER。 $\text{sub}(X)$ の最初の列を示す、グローバル配列 X の列インデックス。
$descx$	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 X の配列ディスクリプター。
$incx$	(グローバル) INTEGER。 X の成分のグローバルな増分。このバージョンでは、1 と m_x の 2 つの $incx$ の値のみサポートされる。 $incx$ の値は、ゼロであってはならない。
$scale$	(ローカル)。 REAL (plassq/pclassq の場合) DOUBLE PRECISION (pdlassq/pzlassq の場合) 上の式で示した $scale$ の値を格納する。

sumsq (ローカル) REAL (*plassq*/*pclassq* の場合)
DOUBLE PRECISION (*pdlasq*/*pzlassq* の場合)
上の式で示した *sumsq* の値を格納する。

出力パラメーター

scale (ローカル)。終了時に、*scale* は、二乗和に対するスケール係数 *scl* で上書きされる。

sumsq (ローカル)。終了時に、*sumsq* は、*scl* が因子分解された二乗和 *smsq* の値で上書きされる。

p?laswp

一般矩形行列に対して一連の行交換を実行する。

構文

```
call pslaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pdlaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pclaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
call pzlaswp(direc, rowcol, n, a, ia, ja, desca, k1, k2, ipiv)
```

説明

このルーチンは、分散行列 $\text{sub}(A)=A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ に一連の行 / 列交換を実行する。交換は、 $\text{sub}(A)$ の行 / 列 k_1 から k_2 のそれぞれに対して開始される。ピボット演算情報はすでにプロセス行 / 列に送信済みであると仮定する。また、このルーチンは、同じ mb (または nb) ブロック内の k_1 - k_2 に対してのみ動作する点に注意する。フル行列のピボット演算を行う場合は、[p?lapiv](#) を使用する。

入力パラメーター

direc (グローバル) CHARACTER。
置換順序を指定する。
'F' (順方向)
'B' (逆方向)

rowcol (グローバル) CHARACTER。
行、列どちらを置換するのかを指定する。
'R' (行)
'C' (列)

n (グローバル) INTEGER。
rowcol='R' の場合、置換する分散行列 $A(*, \text{ja}:\text{ja}+n-1)$ の行数。
rowcol='C' の場合、置換する分散行列 $A(\text{ia}:\text{ia}+n-1, *)$ の列数

<i>a</i>	(ローカル)。 REAL (pslaswp の場合) DOUBLE PRECISION (pdlaswp の場合) COMPLEX (pclaswp の場合) COMPLEX*16 (pzlaswp の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , *) の配列へのポインター。 行 / 列交換を適用する分散行列のローカル部分を格納する。
<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 A の配列ディスクリプター。
<i>k1</i>	(グローバル) INTEGER。行 / 列交換を適用する <i>ipiv</i> の最初の成分。
<i>k2</i>	(グローバル) INTEGER。行 / 列交換を適用する <i>ipiv</i> の最後の成分。
<i>ipiv</i>	(ローカル) INTEGER。 配列、次元は <i>LOCr(m_a)+mb_a</i> (行ピボット演算の場合) または <i>LOCr(n_a)+nb_a</i> (列ピボット演算の場合)。行列 A に関連付けられており、 <i>ipiv(k)=l</i> は行 (または列) <i>k</i> と <i>l</i> の交換を意味する。

出力パラメーター

<i>a</i>	(ローカル)。 REAL (pslaswp の場合) DOUBLE PRECISION (pdlaswp の場合) COMPLEX (pclaswp の場合) COMPLEX*16 (pzlaswp の場合) 終了時に、置換された分散行列で上書きされる。
----------	--

p?latra

一般正方分散行列の対角和を計算する。

構文

```
val = pslatra(n, a, ia, ja, desca)
val = pdlatra(n, a, ia, ja, desca)
val = pclatra(n, a, ia, ja, desca)
val = pzlatra(n, a, ia, ja, desca)
```

説明

この関数は、 $n \times n$ の分散行列 $\text{sub}(A)$ の対角和を計算する。ここで $\text{sub}(A)$ は $A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ である。結果はグリッドのすべてのプロセスで残される。

入力パラメーター

<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (pslatra の場合) DOUBLE PRECISION (pdlatra の場合) COMPLEX (pclatra の場合) COMPLEX*16 (pzlatra の場合) ローカルメモリーにある、対角和を計算する分散行列のローカル部分格納する、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>val</i>	関数の戻り値。
------------	---------

p?latrd

直交/ユニタリー相似変換を用いて、対称/エルミート行列 *A* の最初の *nb* 行/列を実数三重対角形式に縮退する。

構文

```
call pslatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
            work)
call pdlatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
            work)
call pclatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
            work)
call pzlatrd(uplo, n, nb, a, ia, ja, desca, d, e, tau, w, iw, jw, descw,
            work)
```

説明

このルーチンは、直交/ユニタリー相似変換 $Q' * \text{sub}(A) * Q$ を用いて、実対称/複素エルミート行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+n-1, \text{ja}:\text{ja}+n-1)$ の nb 行を対称/複素三重対角形式に縮退し、 $\text{sub}(A)$ の縮退されていない部分に変換を適用するために必要となる V と W を返す。

$\text{uplo} = 'U'$ の場合、 p?latrd は、上三角が与えられる行列に対して最後の nb 行と nb 列の縮退を実行する。

$\text{uplo} = 'L'$ の場合、 p?latrd は、下三角が与えられる行列に対して最初の nb 行と nb 列の縮退を実行する。

このルーチンは、[p?sytrd](#)/[p?hetrd](#) から呼び出される補助ルーチンである。

入力パラメーター

uplo	(グローバル) CHARACTER。 対称/エルミート行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらかが格納されるかを指定する。 'U' の場合、上三角部分。 'L' の場合、下三角部分
n	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。
nb	(グローバル) INTEGER。 縮退する行数と列数。
a	REAL (pslatrd の場合) DOUBLE PRECISION (pdlatrd の場合) COMPLEX (pclatrd の場合) COMPLEX*16 (pzlatrd の場合) ローカルメモリーにある、次元 ($11d_a, LOCc(\text{ja}+n-1)$) の配列へのポインター。 対称/エルミート行列 $\text{sub}(A)$ のローカル部分を格納する。 $\text{uplo} = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。厳密な下三角部分は参照されない。 $\text{uplo} = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ 下三角部分は行列の下三角部分を含む。厳密な上三角部分は参照されない。
ia	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 A の行インデックス。
ja	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 A の列インデックス。
desca	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 A の配列ディスクリプター。
iw	(グローバル) INTEGER。 $\text{sub}(W)$ の最初の行を示す、グローバル配列 W の行インデックス。
jw	(グローバル) INTEGER。 $\text{sub}(W)$ の最初の列を示す、グローバル配列 W の列インデックス。
descw	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 W の配列ディスクリプター。

work (ローカル)。
 REAL (pslatrd の場合)
 DOUBLE PRECISION (pdlatrd の場合)
 COMPLEX (pclatrd の場合)
 COMPLEX*16 (pzlatrd の場合)
 次元 (*nb_a*) のワークスペース配列。

出力パラメーター

a (ローカル) 終了時に、*uplo* = 'U' の場合、最後の *nb* 列が三重対角形式に縮退され、*sub(A)* の対角成分は、三重対角形式の対角成分で上書きされる。対角よりも上の成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリ行列 *Q* を表現する。
uplo = 'L' の場合、最初の *nb* 列が三重対角形式に縮退され、*sub(A)* の対角成分は、三重対角形式の対角成分で上書きされる。対角よりも下の成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリ行列 *Q* を表現する。

d (ローカル)。
 REAL (pslatrd/pclatrd の場合)
 DOUBLE PRECISION (pdlatrd/pzlatrd の場合)
 配列、次元は *LOCc(ja+n-1)*。
 三重対角行列 *T* の対角成分。 $d(i) = a(i,i)$ *d* は、分散行列 *A* に関連付けられる。

e (ローカル)。
 REAL (pslatrd/pclatrd の場合)
 DOUBLE PRECISION (pdlatrd/pzlatrd の場合)
 配列、次元は *LOCc(ja+n-1)* (*uplo* = 'U' の場合) または *LOCc(ja+n-2)* (それ以外の場合)。
 三重対角行列 *T* の非対角成分。
 $e(i) = a(i, i+1)$ (*uplo* = 'U' の場合)
 $e(i) = a(i+1, i)$ (*uplo* = 'L' の場合)
e は分散行列 *A* に関連付けられる。

tau (ローカル)。
 REAL (pslatrd の場合)
 DOUBLE PRECISION (pdlatrd の場合)
 COMPLEX (pclatrd の場合)
 COMPLEX*16 (pzlatrd の場合)
 配列、次元は *LOCc(ja+n-1)*。
 この配列には、基本リフレクターのスカラー係数 *tau* が格納される。*tau* は、分散行列 *A* に関連付けられる。

w (ローカル)。
 REAL (pslatrd の場合)
 DOUBLE PRECISION (pdlatrd の場合)
 COMPLEX (pclatrd の場合)
 COMPLEX*16 (pzlatrd の場合)
 ローカルメモリーにある、次元 (*lld_w*, *nb_w*) の配列へのポイン

ター。
 $\text{sub}(A)$ の縮退されていない部分の更新に必要な $n \times nb_w$ の行列 W のローカル部分が格納される。

アプリケーション・ノート

$\text{uplo} = 'U'$ の場合、行列 Q は基本リフレクターの積として表現される。

$$Q = H(n) H(n-1) \dots H(n-nb+1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \text{tau} * v * v'$$

ここで、 tau は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(i:n) = 0$ および $v(i-1) = 1$ 。終了時に、 $v(1:i-1)$ は $A(ia:ia+i-1, ja+i)$ に、 tau は $\text{tau}(ja+i-1)$ に格納される。

$\text{uplo} = 'L'$ の場合、行列 Q は基本リフレクターの積として表現される。

$$Q = H(1) H(2) \dots H(nb)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \text{tau} * v * v'$$

ここで、 tau は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(1:i) = 0$ および $v(i+1) = 1$ 。終了時に、 $v(i+2:n)$ は $A(ia+i+1:ia+n-1, ja+i-1)$ に、 tau は $\text{tau}(ja+i-1)$ に格納される。

ベクトル v の成分は、行列の縮退されていない部分に変換を適用するために、行列 W とともに必要となる $n \times nb$ の行列 V を、形式の対称 / エルミート階数 $-2k$ 更新を使用し形成する。

$$\text{sub}(A) := \text{sub}(A) - v w' - w v'$$

終了時の a の内容を次に示す
 ($n = 5$ および $nb = 2$ の場合)。

$\text{uplo} = 'U'$ の場合

$\text{uplo} = 'L'$ の場合

$$\begin{bmatrix} a & a & a & v_4 & v_5 \\ & a & a & v_4 & v_5 \\ & & a & 1 & v_5 \\ & & & d & 1 \\ & & & & d \end{bmatrix} \qquad \begin{bmatrix} d \\ 1 & d \\ v_1 & 1 & a \\ v_1 & v_2 & a & a \\ v_1 & v_2 & a & a & a \end{bmatrix}$$

d は縮退された行列の対角成分、 a は変更されていない元の行列の成分、 v_i は $H(i)$ を定義するベクトルの成分を表す。

p?latrs

オーバーフローを防ぐために設定されたスケール係数を持つ三角連立方程式を解く。

構文

```
call pslatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
            scale, cnorm, work)
call pdlatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
            scale, cnorm, work)
call pclatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
            scale, cnorm, work)
call pzlatrs(uplo, trans, diag, normin, n, a, ia, ja, desca, x, ix, jx, descx,
            scale, cnorm, work)
```

説明

このルーチンは、三角連立方程式 $Ax = \sigma b$ 、 $A^T x = \sigma b$ 、または $A^H x = \sigma b$ を解く。ここで、 σ はオーバーフローを防ぐために設定されたスケール係数である。このルーチンの説明は、将来のリリースで拡張される。

入力パラメーター

<i>uplo</i>	CHARACTER*1。 行列 <i>A</i> が上三角か下三角かを指定する。 'U' の場合、上三角部分。 'L' の場合、下三角部分
<i>trans</i>	CHARACTER*1。 <i>A</i> に適用する演算を指定する。 'N' の場合、 $Ax = \sigma b$ を解く (置換なし) 'T' の場合、 $A^T x = \sigma b$ を解く (置換) 'C' の場合、 $A^H x = \sigma b$ を解く (共役置換)
<i>diag</i>	CHARACTER*1。 行列 <i>A</i> が単位三角かどうかを指定する。 'N' の場合、単位三角ではない。 'U' の場合、単位三角。
<i>normin</i>	CHARACTER*1。 <i>cnorm</i> を設定したかどうかを指定する。 'Y' の場合、 <i>cnorm</i> には列ノルムを格納した。 'N' の場合、 <i>cnorm</i> は設定していない。終了時に、ノルムが計算され、 <i>cnorm</i> に格納される。
<i>n</i>	INTEGER。 行列 <i>A</i> の次数。 $n \geq 0$
<i>a</i>	REAL (pslatrs/pclatrs の場合) DOUBLE PRECISION (pdlatrs/pzlatrs の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。三角行列 <i>A</i> を格納する。 <i>uplo</i> = 'U' の場合、配列 <i>a</i> の先頭の $n \times n$ の上三角部分に上三角行列を格納す

る。a の厳密な下三角部分は参照されない。uplo = 'L' の場合、配列 a の先頭の $n \times n$ の下三角部分に下三角行列を格納する。a の厳密な上三角部分は参照されない。diag = 'U' の場合も a の対角成分は参照されず 1 とみなされる。

<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 A の最初の行と最初の列を示す、グローバル配列 a の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> _l)。分散行列 A の配列ディスクリプター。
<i>x</i>	REAL (pslatrs/pclatrs の場合) DOUBLE PRECISION (pdlatrs/pzlatrs の場合) 配列、次元は (<i>n</i>)。三角法の右辺 <i>b</i> を格納する。
<i>ix</i>	(グローバル) INTEGER。sub(<i>x</i>) の最初の行を示す、グローバル配列 <i>x</i> の行インデックス。
<i>jx</i>	(グローバル) INTEGER。sub(<i>x</i>) の最初の列を示す、グローバル配列 <i>x</i> の列インデックス。
<i>descx</i>	(グローバルおよびローカル) INTEGER。 配列、次元は (<i>dlen</i> _l)。分散行列 X の配列ディスクリプター。
<i>cnorm</i>	REAL (pslatrs/pclatrs の場合) DOUBLE PRECISION (pdlatrs/pzlatrs の場合) 配列、次元は (<i>n</i>)。normin = 'Y' の場合、cnorm は入力引数となり、cnorm(<i>j</i>) には A の <i>j</i> 番目の列の非対角部分を格納する。trans = 'N' の場合、cnorm(<i>j</i>) は無限ノルムに等しいか大きくなければならない。trans = 'T' または 'C' の場合、cnorm(<i>j</i>) は 1-ノルムに等しいか大きくなければならない。
<i>work</i>	(ローカル)。 REAL (pslatrs の場合) DOUBLE PRECISION (pdlatrs の場合) COMPLEX (pclatrs の場合) COMPLEX*16 (pzlatrs の場合) テンポラリー・ワークスペース。

出力パラメーター

<i>x</i>	終了時に、 <i>x</i> は解のベクトル <i>x</i> によって上書きされる。
<i>scale</i>	REAL (pslatrs/pclatrs の場合) DOUBLE PRECISION (pdlatrs/pzlatrs の場合) 配列、次元は (<i>lda</i> , <i>n</i>)。上述のとおり三角法に対するスケール係数 <i>s</i> 。 <i>scale</i> = 0 は行列 A が特異であるか不適切にスケールされたことを示し、ベクトル <i>x</i> は $Ax = 0$ に対する完全または近似解となる。
<i>cnorm</i>	normin = 'N' の場合、cnorm は出力引数となり、cnorm(<i>j</i>) は A の <i>j</i> 番目の列の非対角部分の 1-ノルムを返す。

p?latrz

直交 / ユニタリー相似変換を用いて、実 / 複素上台形行列を上三角形式に縮退させる。

構文

```
call pslatz(m, n, l, a, ia, ja, desca, tau, work)
call pdlatrz(m, n, l, a, ia, ja, desca, tau, work)
call pclatz(m, n, l, a, ia, ja, desca, tau, work)
call pzlatrz(m, n, l, a, ia, ja, desca, tau, work)
```

説明

このルーチンは、直交 / ユニタリー変換を用いて、 $m \times n$ ($m \leq n$) の実 / 複素上台形行列 $\text{sub}(A) = [A(ia:ia+m-1, ja:ja+m-1) \ A(ia:ia+m-1, ja+n-l:ja+n-1)]$ を、上三角形式に縮退させる。

上台形行列 $\text{sub}(A)$ は、次のように因子分解される。

$$\text{sub}(A) = (R \ 0) * Z$$

Z は $n \times n$ の直交 / ユニタリー行列、 R は $m \times m$ の上三角行列である。

入力パラメーター

m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(A)$ の行数。 $m \geq 0$ 。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(A)$ の列数。 $n \geq 0$ 。
l	(グローバル) INTEGER。 Householder リフレクターとして意味を持った部分が格納されている分散部分行列 $\text{sub}(A)$ の列数。 $l > 0$ 。
a	(ローカル)。 REAL (pslatrz の場合) DOUBLE PRECISION (pdlatrz の場合) COMPLEX (pclatz の場合) COMPLEX*16 (pzlatrz の場合) ローカルメモリーにある、 次元 ($lld_a, LOC(ja+n-1)$) の配列へのポインター。 因子分解を実行する、 $m \times n$ の分散行列 $\text{sub}(A)$ のローカル部分を格納する。
ia	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 A の行インデックス。
ja	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 A の列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は (*dlen_*)。分散行列 *A* の配列ディスクリプター。

work (ローカル)。
 REAL (*pslatrz* の場合)
 DOUBLE PRECISION (*pdlatzr* の場合)
 COMPLEX (*pclatzr* の場合)
 COMPLEX*16 (*pzlatrz* の場合)

ワークスペース配列、次元は (*lwork*)。
 $lwork \geq nq0 + \max(1, mp0)$
 ここで、

$iroff = \text{mod}(ia-1, mb_a)$ 、 $icoff = \text{mod}(ja-1, nb_a)$ 、
 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、
 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npc0l)$ 、
 $mp0 = \text{numroc}(m+iroff, mb_a, myrow, iarow, nprow)$ 、
 $nq0 = \text{numroc}(n+icoff, nb_a, mycol, iacol, npc0l)$ 、
 numroc 、 indxg2p 、および numroc は ScaLAPACK ツール関数である。
 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npc0l$ は、サブルーチン `blacs_gridinfo` を呼び出して求められる。

出力パラメーター

a 終了時に、`sub(A)` の先頭の $m \times m$ の上三角部分には上三角行列 *R* が格納される。`sub(A)` の最初の m 行の $n-l+1$ から N までの成分は、配列 *tau* とともに、基本リフレクター m の積として直交/ユニタリー行列 *Z* を表現する。

tau (ローカル)。
 REAL (*pslatrz* の場合)
 DOUBLE PRECISION (*pdlatzr* の場合)
 COMPLEX (*pclatzr* の場合)
 COMPLEX*16 (*pzlatrz* の場合)

配列、次元は ($LOCr(ja+m-1)$)。基本リフレクターのスカラー係数が格納される。*tau* は、分散行列 *A* に関連付けられる。

アプリケーション・ノート

因子分解は Householder 法を用いて得る。`sub(A)` の $(m-k+1)$ 番目の行にゼロを導入するため (複素数ルーチンの場合は、共役置換) に使用される k 番目の変換行列 $Z(k)$ は次の形式で与えられる。

$$Z(k) = \begin{bmatrix} I & 0 \\ 0 & T(k) \end{bmatrix}$$

ここで、 $T(k) = I - \text{tau} * u(k) * u(k)'$ 、 $u(k) = \begin{bmatrix} 1 \\ 0 \\ z(k) \end{bmatrix}$

τ はスカラー、 $z(k)$ は $(n-m)$ 成分のベクトルである。 τ と $z(k)$ は、 $\text{sub}(A)$ の k 番目の行の成分をゼロにするために選択される。

スカラー τ は τ の k 番目の成分で返され、ベクトル $u(k)$ は $z(k)$ の成分が $a(k, m+1), \dots, a(k, n)$ に格納されるような $\text{sub}(A)$ の k 番目の行で返される。 R の成分は $\text{sub}(A)$ の上三角部分で返される。

Z は次の式で与えられる。

$$Z = Z(1)Z(2) \dots Z(m)$$

p?lauu2

積 UU^H または $L^H L$ を計算する。ここで、 U と L は上三角または下三角行列(ローカル非ブロック化アルゴリズム)。

構文

```
call pslauu2(uplo, n, a, ia, ja, desca)
call pdlauu2(uplo, n, a, ia, ja, desca)
call pclauu2(uplo, n, a, ia, ja, desca)
call pzlauu2(uplo, n, a, ia, ja, desca)
```

説明

このルーチンは、積 UU^H または $L^H L$ を計算する。三角係数 U または L は、分散行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ の上 O 角または下三角部分に格納される。

$uplo = 'U'$ または $'u'$ の場合、 $\text{sub}(A)$ の係数 U は結果の上三角で書き込まれる。
 $uplo = 'L'$ または $'l'$ の場合、 $\text{sub}(A)$ の係数 L は結果の下三角で書き込まれる。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS レベル 2 のルーチン](#) を呼び出す。このルーチンはコミュニケーションを実行しないため、演算を行う行列は 1 つのプロセスに対してローカルでなければならない。

入力パラメーター

$uplo$	(グローバル) CHARACTER*1。 行列 $\text{sub}(A)$ に格納されている三角係数が上三角か下三角かを指定する。 'U' の場合、上三角。 'L' の場合、下三角。
n	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、三角係数 U または L の次数。 $n \geq 0$ 。
a	(ローカル)。 REAL (pslauu2 の場合) DOUBLE PRECISION (pdlauu2 の場合) COMPLEX (pclauu2 の場合) COMPLEX*16 (pzlauu2 の場合)

	ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。 三角係数 <i>U</i> または <i>L</i> のローカル部分を格納する。
<i>ia</i>	(グローバル) INTEGER。 <i>sub(A)</i> の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 <i>sub(A)</i> の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>a</i>	(ローカル) 終了時に、 <i>uplo</i> = 'U' の場合、分散行列 <i>sub(A)</i> の上三角は積 <i>UU'</i> の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の下三角は積 <i>LL'</i> の下三角で上書きされる。
----------	---

p?lauum

積 UU^H または $L^H L$ を計算する。ここで、 U と L は上三角または下三角行列。

構文

```
call pslauum(uplo, n, a, ia, ja, desca)
call pdlauum(uplo, n, a, ia, ja, desca)
call pclauum(uplo, n, a, ia, ja, desca)
call pzlauum(uplo, n, a, ia, ja, desca)
```

説明

このルーチンは、積 UU' または LL' を計算する。三角係数 U または L は、行列 $\text{sub}(A) = A(\text{ia}:\text{ia}+\text{n}-1, \text{ja}:\text{ja}+\text{n}-1)$ の上三角または下三角部分に格納される。

uplo = 'U' または 'u' の場合、*sub(A)* の係数 U は結果の上三角で上書きされる。

uplo = 'L' または 'l' の場合、*sub(A)* の係数 L は結果の下三角で上書きされる。

このルーチンは、アルゴリズムのブロック化形式で、レベル 3 PBLAS を呼び出す。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER*1。 行列 <i>sub(A)</i> に格納されている三角係数が上三角か下三角かを指定する。 'U' の場合、上三角。 'L' の場合、下三角。
<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、三角係数 U または L の次数。 $n \geq 0$ 。

<i>a</i>	(ローカル)。 REAL (pslauum の場合) DOUBLE PRECISION (pdlauum の場合) COMPLEX (pclauum の場合) COMPLEX*16 (pzlauum の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。三角係数 <i>U</i> または <i>L</i> のローカル部分を格納する。
<i>ia</i>	(グローバル) INTEGER。 sub(<i>A</i>) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(<i>A</i>) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>a</i>	(ローカル) 終了時に、 <i>uplo</i> = 'U' の場合、分散行列 sub(<i>A</i>) の上三角は積 <i>UU'</i> の上三角で上書きされる。 <i>uplo</i> = 'L' の場合、sub(<i>A</i>) の下三角は積 <i>LL'</i> の下三角で上書きされる。
----------	--

p?lawil

Wilkinson 変換を実行する。

構文

```
call pslawil(ii, jj, m, a, desca, h44, h33, h43h34, v)
call pdlawil(ii, jj, m, a, desca, h44, h33, h43h34, v)
```

説明

このルーチンは、*h44*、*h33*、および *h43h34* で与えられる変換を、行 *m* から始まるように *v* に配置する。

入力パラメーター

<i>ii</i>	(グローバル) INTEGER。 <i>h(m+2, m+2)</i> の行所有成分。
<i>jj</i>	(グローバル) INTEGER。 <i>h(m+2, m+2)</i> の列所有成分。
<i>m</i>	(グローバル) INTEGER。 変換を開始する位置 (行 <i>m</i>)。終了時に変更されない。

<i>a</i>	(グローバル)。 REAL (pslawil の場合) DOUBLE PRECISION (pdlawil の場合) 配列、次元は (<i>desca</i> (<i>lld</i>),*)。Hessenberg 行列。終了時に変更されない。
<i>desca</i>	(グローバルおよびローカル) INTEGER。 次元 (<i>dlen</i>) の配列。分散行列 <i>A</i> の配列ディスクリプター。終了時に変更されない。
<i>h44</i> , <i>h33</i> , <i>h43h34</i>	(グローバル) REAL (pslawil の場合) DOUBLE PRECISION (pdlawil の場合) これらの3種類の値は、倍精度シフト <i>QR</i> の反復法で使用される。終了時に変更されない。

出力パラメーター

<i>v</i>	(グローバル)。 REAL (pslawil の場合) DOUBLE PRECISION (pdlawil の場合) 変換を格納する、サイズ3の配列。
----------	--

p?org2l/p?ung2l

p?geqlf で求めた *QL* 因子分解から、全部または一部の直交/ユニタリー行列 *Q* を生成する(非ブロック化アルゴリズム)。

構文

```
call psorg2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorg2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcung2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzung2l(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

説明

ルーチン p?org2l/p?ung2l は、直交列を持つ $m \times n$ の実/複素行列 *Q* を生成する。これは、次数 *m* の *k* 個の基本リフレクターの積の最後の *n* 列として次のように定義される。ここで、*Q* は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$Q = H(k) \dots H(2) H(1)$ (p?geqlf から返されたとおり)

入力パラメーター

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 <i>Q</i> の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 <i>Q</i> の列数。 $m \geq n \geq 0$ 。

<i>k</i>	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $n \geq k \geq 0$ 。
<i>a</i>	REAL (psorg21 の場合) DOUBLE PRECISION (pdorg21 の場合) COMPLEX (pcung21 の場合) COMPLEX*16 (pzung21 の場合) ローカルメモリーにある、 次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。 j 番目の列には、 p?qegqlf によって分散行列引数 $A(ia:*, ja+n-k: ja+n-1)$ の k 列に返されたとおりに、基本リフレクター $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$ は $ja+n-k \leq j \leq ja+n-k$ である。
<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (psorg21 の場合) DOUBLE PRECISION (pdorg21 の場合) COMPLEX (pcung21 の場合) COMPLEX*16 (pzung21 の場合) 配列、次元は $LOCc(ja+n-1)$ 。 p?qegqlf から返されたとおりに、基本リフレクター $H(j)$ のスカラー係数 $tau(j)$ を格納する。
<i>work</i>	(ローカル)。 REAL (psorg21 の場合) DOUBLE PRECISION (pdorg21 の場合) COMPLEX (pcung21 の場合) COMPLEX*16 (pzung21 の場合) ワークスペース配列、次元は ($lwork$)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 $work$ の次元。 $lwork$ はローカル入力であり、 $lwork \geq mpa0 + \max(1, nqa0)$ でなければならない。ここで、 $irowfa = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcyl)$ 、 $mpa0 = \text{numroc}(m+irowfa, mb_a, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb_a, mycol, iacol, npcyl)$ 。 indxg2p および numroc は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcyl$ は、サブルーチン blacs_gridinfo を呼び出して求められる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

a 終了時に、 $m \times n$ の分散行列 Q のローカル部分が格納される。

$work$ 終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。

$info$ (ローカル) INTEGER。
 $= 0$ の場合、正常に終了したことを示す。
 < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 i 番目の引数がスカラーで値が不正でならば $info = -i$ 。

p?org2r/p?ung2r

$p?geqrf$ で求めた QR 因子分解から、全部または一部の直交/ユニタリー行列 Q を生成する (非ブロック化アルゴリズム)。

構文

```
call psorg2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorg2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcung2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzung2r(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

説明

ルーチン $p?org2r/p?ung2r$ は、直交列を持つ $m \times n$ の実/複素行列 Q を生成する。これは、次数 m の k 個の基本リフレクターの積の最初の n 列として次のように定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(1)H(2) \dots H(k)$$

[p?geqrf](#) によって返される。

入力パラメーター

m (グローバル) INTEGER。
 演算を行う行数、すなわち、分散部分行列 Q の行数。 $m \geq 0$ 。

n (グローバル) INTEGER。
 演算を行う列数、すなわち、分散部分行列 Q の列数。 $m \geq n \geq 0$ 。

k (グローバル) INTEGER。
 その積が行列 Q を定義する基本リフレクターの個数。
 $n \geq k \geq 0$ 。

<i>a</i>	<p>REAL (psorg2r の場合) DOUBLE PRECISION (pdorg2r の場合) COMPLEX (pcung2r の場合) COMPLEX*16 (pzung2r の場合) ローカルメモリーにある、次元 (<i>lld_a</i>, <i>LOCc(ja+n-1)</i>) の配列へのポインター。 <i>j</i> 番目の列には、p?qgegrf によって分散行列引数 $A(ia:*, ja:ja+k-1)$ の <i>k</i> 列に返されたとおりに、基本リフレクター $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、$H(j)$ は $ja \leq j \leq ja+k-1$ である。</p>
<i>ia</i>	<p>(グローバル) INTEGER。 <i>sub(A)</i> の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。</p>
<i>ja</i>	<p>(グローバル) INTEGER。 <i>sub(A)</i> の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。</p>
<i>desca</i>	<p>(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。</p>
<i>tau</i>	<p>(ローカル)。 REAL (psorg2r の場合) DOUBLE PRECISION (pdorg2r の場合) COMPLEX (pcung2r の場合) COMPLEX*16 (pzung2r の場合) 配列、次元は <i>LOCc(ja+k-1)</i>。 p?qgegrf から返されたとおりに、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を格納する。この配列は、分散行列 <i>A</i> に関連付けられる。</p>
<i>work</i>	<p>(ローカル)。 REAL (psorg2r の場合) DOUBLE PRECISION (pdorg2r の場合) COMPLEX (pcung2r の場合) COMPLEX*16 (pzung2r の場合) ワークスペース配列、次元は (<i>lwork</i>)。</p>
<i>lwork</i>	<p>(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、$lwork \geq mpa0 + \max(1, nqa0)$ でなければならない。ここで、$irow = \text{mod}(ia-1, mb_a)$、$icoffa = \text{mod}(ja-1, nb_a)$、$iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$、$iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcot)$、$mpa0 = \text{numroc}(m+irow, mb_a, myrow, iarow, nprow)$、$nqa0 = \text{numroc}(n+icoffa, nb_a, mycol, iacol, npcot)$。 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i>、<i>mycol</i>、<i>nprow</i>、および <i>npcol</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して求められる。 <i>lwork</i> = -1 の場合、<i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエンタリーとして返され、pxerbla はエラーメッセージを生成しない。</p>

出力パラメーター

<i>a</i>	終了時に、 $m \times n$ の分散行列 Q のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) は、最小かつ最適な <i>lwork</i> 値を返す。
<i>info</i>	(ローカル) INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -(<i>i</i> *100+ <i>j</i>)、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> 。

p?orgl2/p?ungl2

p?gelqf で求めた LQ 因子分解から、全部または一部の直交/ユニタリー行列 Q を生成する(非ブロック化アルゴリズム)。

構文

```
call psorgl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorgl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcungl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzungl2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

説明

ルーチン p?orgl2/p?ungl2 は、直交列を持つ $m \times n$ の実/複素行列 Q を生成する。これは、次数 n の k 個の基本リフレクターの積の最初の m 行として次のように定義される。ここで、 Q は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$$Q = H(k) \dots H(2) H(1) \text{ (実数型の場合)},$$

$$Q = H(k)' \dots H(2)' H(1)' \text{ (複素数型の場合)}$$

[p?gelqf](#) によって返される。

入力パラメーター

<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 Q の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 Q の列数。 $n \geq m \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $m \geq k \geq 0$ 。
<i>a</i>	REAL (psorgl2 の場合) DOUBLE PRECISION (pdorgl2 の場合) COMPLEX (pcungl2 の場合) COMPLEX*16 (pzungl2 の場合) ローカルメモリーにある、

次元 (ld_a , $LOCc(ja+n-1)$) の配列へのポインター。
 i 番目の行には、[p?gelsqf](#) によって分散行列引数
 $A(ia:ia+k-1, ja:*)$ の k 行に返されたとおりに、基本リフレク
 ター $H(i)$ を定義するベクトルを格納しておかなければならない。
 ここで、 $H(i)$ は $ia \leq i \leq ia+k-1$ 。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>psorgl2</i> の場合) DOUBLE PRECISION (<i>pdorgl2</i> の場合) COMPLEX (<i>pcungl2</i> の場合) COMPLEX*16 (<i>pzungl2</i> の場合) 配列、次元は $LOCr(ja+k-1)$ 。 p?gelsqf から返されたとおりに、基本リフレクター $H(i)$ のスカ ラー係数 $\tau(i)$ を格納する。この配列は、分散行列 <i>A</i> に関連付 けられる。
<i>work</i>	(ローカル)。 REAL (<i>psorgl2</i> の場合) DOUBLE PRECISION (<i>pdorgl2</i> の場合) COMPLEX (<i>pcungl2</i> の場合) COMPLEX*16 (<i>pzungl2</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq nqa0 + \max(1, mpa0)$ でな なければならない。ここで、 $irow = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcrow)$ 、 $mpa0 = \text{numroc}(m+irow, mb_a, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb_a, mycol, iacol, npcrow)$ 。 indxg2p および numroc は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npcol$ は、サブルーチン blacs_gridinfo を呼び出して求められる。 $lwork = -1$ の場合、 <i>lwork</i> はグローバル入力であり、ワークス ペースのクエリーとみなされ、ルーチンはすべての配列の最小 かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の 最初のエントリーとして返され、 pxerbla はエラーメッセージ を生成しない。

出力パラメーター

<i>a</i>	終了時に、 $m \times n$ の分散行列 <i>Q</i> のローカル部分が格納される。
----------	--

work 終了時に、*work*(1) は、最小かつ最適な *lwork* 値を返す。

info (ローカル) INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば *info* = -(*i**100+*j*)、*i* 番目の引数がスカラーで値が不正でならば *info* = -*i*。

p?orgr2/p?ungr2

p?gerqf で求めた *RQ* 因子分解から、全部または一部の直交/ユニタリー行列 *Q* を生成する(非ブロック化アルゴリズム)。

構文

```
call psorgr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pdorgr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pcungr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
call pzungr2(m, n, k, a, ia, ja, desca, tau, work, lwork, info)
```

説明

ルーチン p?orgr2/p?ungr2 は、直交列を持つ $m \times n$ の実/複素行列 *Q* を生成する。これは、次数 *n* の *k* 個の基本リフレクターの積の最後の *m* 行として次のように定義される。ここで、*Q* は $A(ia:ia+m-1, ja:ja+n-1)$ である。

$Q = H(1) H(2) \dots H(k)$ (実数型の場合)、
 $Q = H(1)' H(2)' \dots H(k)'$ (複素数型の場合)

[p?gerqf](#) によって返される。

入力パラメーター

m (グローバル) INTEGER。
 演算を行う行数、すなわち、分散部分行列 *Q* の行数。 $m \geq 0$ 。

n (グローバル) INTEGER。
 演算を行う列数、すなわち、分散部分行列 *Q* の列数。 $n \geq m \geq 0$ 。

k (グローバル) INTEGER。
 その積が行列 *Q* を定義する基本リフレクターの個数。
 $m \geq k \geq 0$ 。

a REAL (psorgr2 の場合)
 DOUBLE PRECISION (pdorgr2 の場合)
 COMPLEX (pcungr2 の場合)
 COMPLEX*16 (pzungr2 の場合)
 ローカルメモリーにある、
 次元 (11*d*_{*a*}, LOCc(*ja*+*n*-1)) の配列へのポインター。
i 番目の行には、[p?gerqf](#) によって分散行列引数

$A(ia+m-k:ia+m-1, ja:*)$ の k 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(i)$ は $ia+m-k \leq i \leq ia+m-1$ 。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>psorgl2</i> の場合) DOUBLE PRECISION (<i>pdorgl2</i> の場合) COMPLEX (<i>pcungl2</i> の場合) COMPLEX*16 (<i>pzungl2</i> の場合) 配列、次元は $LOCr(ja+m-1)$ 。 p?gerqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数 $tau(i)$ を格納する。この配列は、分散行列 <i>A</i> に関連付けられる。
<i>work</i>	(ローカル)。 REAL (<i>psorgr2</i> の場合) DOUBLE PRECISION (<i>pdorgr2</i> の場合) COMPLEX (<i>pcungr2</i> の場合) COMPLEX*16 (<i>pzungr2</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、 $lwork \geq nqa0 + \max(1, mpa0)$ でなければならない。ここで、 $iroffa = \text{mod}(ia-1, mb_a)$ 、 $icoffa = \text{mod}(ja-1, nb_a)$ 、 $iarow = \text{indxg2p}(ia, mb_a, myrow, rsrc_a, nprow)$ 、 $iacol = \text{indxg2p}(ja, nb_a, mycol, csrc_a, npcot)$ 、 $mpa0 = \text{numroc}(m+iroffa, mb_a, myrow, iarow, nprow)$ 、 $nqa0 = \text{numroc}(n+icoffa, nb_a, mycol, iacol, npcot)$ 。 <i>indxg2p</i> および <i>numroc</i> は、ScaLAPACK ツール関数である。 <i>myrow</i> 、 <i>mycol</i> 、 <i>nprowv</i> 、および <i>npcol</i> は、サブルーチン <i>blacs_gridinfo</i> を呼び出して求められる。 <i>lwork</i> = -1 の場合、 <i>lwork</i> はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する <i>work</i> 配列の最初のエントリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>a</i>	終了時に、 $m \times n$ の分散行列 <i>Q</i> のローカル部分が格納される。
<i>work</i>	終了時に、 <i>work</i> (1) は、最小かつ最適な <i>lwork</i> 値を返す。

info (ローカル) INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば $info = -(i*100+j)$ 、*i* 番目の引数がスカラーで値が不正でならば $info = -i$ 。

p?orm2l/p?unm2l

一般行列に `p?geqlf` で求めた *QL* 因子分解の直交/ユニタリー行列を掛ける(非ブロック化アルゴリズム)。

構文

```
call psorm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pdorm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pcunm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pzunm2l(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
```

説明

ルーチン `p?orm2l/p?unm2l` は、 $m \times n$ の一般実/複素分布行列 $\text{sub}(C) = C(ic:ic+m-1, jc: jc+n-1)$ を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * \text{sub}(C)$	$\text{sub}(C) * Q$
$trans = 'T'$ (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C) * Q^T$
$trans = 'C'$ (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C) * Q^H$

Q は、*k* 個の基本リフレクターの積として定義される実数直交または複素ユニタリー行列である。

$$Q = H(k) \dots H(2) H(1)$$

`p?geqlf` によって返される。*Q* の次数は、 $side = 'L'$ の場合は *m*、 $side = 'R'$ の場合は *n* である。

入力パラメーター

side (グローバル) CHARACTER。
 'L' の場合、*Q*、 Q^T (実数型の場合) または Q^H (複素数型) を左から適用する。
 'R' の場合、*Q*、 Q^T (実数型の場合) または Q^H (複素数型の場合) を右から適用する。

<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 Q を適用する (転置なし) 'T' の場合、 Q^T を適用する (転置、実数型の場合) 'C' の場合、 Q^H を適用する (共役転置、複素数型の場合)
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (<i>psorm21</i> の場合) DOUBLE PRECISION (<i>pdorm21</i> の場合) COMPLEX (<i>pcunm21</i> の場合) COMPLEX*16 (<i>pzunm21</i> の場合) ローカルメモリーにある、次元 (<i>lld_a</i> , $LOCc(ja+k-1)$) の配列へのポインター。 <i>j</i> 番目の行には、 p?geqlf によって分散行列引数 $A(ia:*, ja: ja+k-1)$ の <i>k</i> 列に返されたとおりに、基本リフレクター $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$ は $ja \leq j \leq ja+k-1$ である。引数 $A(ia:*, ja: ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 <i>side</i> = 'L' の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ 。 <i>side</i> = 'R' の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$ 。
<i>ia</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>psorm21</i> の場合) DOUBLE PRECISION (<i>pdorm21</i> の場合) COMPLEX (<i>cunm21</i> の場合) COMPLEX*16 (<i>pzunm21</i> の場合) 配列、次元は $LOCc(ja+n-1)$ 。 p?geqlf から返されたとおりに、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を格納する。この配列は、分散行列 A に関連付けられる。
<i>c</i>	(ローカル)。 REAL (<i>psorm21</i> の場合) DOUBLE PRECISION (<i>pdorm21</i> の場合)

	COMPLEX (pcunm21 の場合) COMPLEX*16 (pzunm21 の場合) ローカルメモリーにある、次元 ($11d_c, LOCc(jc+n-1)$) の配列。 分散行列 $sub(C)$ のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 $sub(C)$ の最初の行を示す、グローバル配列 C の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 $sub(C)$ の最初の列を示す、グローバル配列 C の列インデックス。
<i>desc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psorm21 の場合) DOUBLE PRECISION (pdorm21 の場合) COMPLEX (pcunm21 の場合) COMPLEX*16 (pzunm21 の場合) ワークスペース配列、次元は ($lwork$)。 終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 $work$ の次元。 $lwork$ はローカル入力であり、次の値でなければならない。 $side = 'L'$ の場合、 $lwork \geq mpc0 + \max(1, nqc0)$ 。 $side = 'R'$ の場合、 $lwork \geq nqc0 + \max(\max(1, mpc0), numroc$ $(numroc(n+icoffc, nb_a, 0, 0, npc0), nb_a, 0, 0,$ $lcmq))$ 。 ここで、 $lcmq = lcm / npc0$ 、 $lcm = iclm(nprow, npc0)$ 、 $iroffc = \text{mod}(ic-1, mb_c)$ 、 $icoffc = \text{mod}(jc-1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, myrow, rsrc_c, nprow)$ 、 $iccol = \text{indxg2p}(jc, nb_c, mycol, csrc_c, npc0)$ 、 $Mqc0 = \text{numroc}(m+icoffc, nb_c, mycol, icrow, nprow)$ 、 $Npc0 = \text{numroc}(n+iroffc, mb_c, myrow, iccol, npc0)$ 、 $iclm$ 、 indxg2p 、および numroc は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npc0$ は、サブルーチン blacs_gridinfo を呼び出して求められる。 $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ A ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	終了時に、 $sub(C)$ は、 $Q*sub(C)$ 、 $Q'*sub(C)$ 、 $sub(C)*Q'$ 、または $sub(C)*Q$ のいずれかで上書きされる。
<i>work</i>	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。

`info` (ローカル) INTEGER。
 = 0 の場合、正常に終了したことを示す。
 < 0 の場合、*i* 番目の引数が配列で *j* 番目の成分の値が不正ならば `info = -(i*100+j)`、*i* 番目の引数がスカラーで値が不正でならば `info = -i`。



注: 分散部分行列 $A(ia:*, ja:*)$ と $C(ic:ic+m-1, jc:jc+n-1)$ は、次の式を満たすようにアライメントされていなければならない。
`side = 'L'` の場合、(`mb_a.eq.mb_c .AND. iroffa.eq.iroffc .AND. iarow.eq.icrow`)
`side = 'R'` の場合、(`mb_a.eq.nb_c .AND. iroffa.eq.iroffc`)

p?orm2r/p?unm2r

一般行列に `p?geqrf` で求めた *QR* 因子分解の直交/ユニタリ行列を掛ける(非ブロック化アルゴリズム)。

構文

```
call psorm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
  work, lwork, info)
call pdorm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
  work, lwork, info)
call pcunm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
  work, lwork, info)
call pzunm2r(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
  work, lwork, info)
```

説明

ルーチン `p?orm2r/p?unm2r` は、 $m \times n$ の一般実/複素分布行列 $\text{sub}(C)=C(ic:ic+m-1, jc:jc+n-1)$ を以下で上書きする。

	<code>side = 'L'</code>	<code>side = 'R'</code>
<code>trans = 'N'</code>	$Q * \text{sub}(C)$	$\text{sub}(C) * Q$
<code>trans = 'T'</code> (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C) * Q^T$
<code>trans = 'c'</code> (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C) * Q^H$

Q は、*k* 個の基本リフレクターの積として定義される実数直交または複素ユニタリ行列である。

$$Q = H(k) \dots H(2) H(1)$$

`p?geqrf` によって返される。*Q* の次数は、`side = 'L'` の場合は *m*、`side = 'R'` の場合は *n* である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型) を左から適用する。 'R' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型の場合) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 Q を適用する (転置なし) 'T' の場合、 Q^T を適用する (転置、実数型の場合) 'C' の場合、 Q^H を適用する (共役転置、複素数型の場合)
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 <i>side</i> = 'L' の場合、 $m \geq k \geq 0$ 。 <i>side</i> = 'R' の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合) ローカルメモリーにある、次元 (lld_a , $LOC(ja+k-1)$) の配列へのポインター。 <i>j</i> 番目の列には、 p?gegrf によって分散行列引数 $A(ia:*, ja:ja+k-1)$ の <i>k</i> 列に返されたとおりに、基本リフレクター $H(j)$ を定義するベクトルを格納しておかなければならない。ここで、 $H(j)$ は $ja \leq j \leq ja+k-1$ である。引数 $A(ia:*, ja:ja+k-1)$ はルーチンにより変更されるが終了時に復元される。 <i>side</i> = 'L' の場合、 $lld_a \geq \max(1, LOCr(ia+m-1))$ 。 <i>side</i> = 'R' の場合、 $lld_a \geq \max(1, LOCr(ia+n-1))$ 。
<i>ia</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 $\text{sub}(A)$ の最初の列を示す、グローバル配列 A の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (p sorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合)

配列、次元は $LOCc(ja+k-1)$ 。 [p?qegr](#) から返されたとおりに、基本リフレクター $H(j)$ のスカラー係数 $\tau(j)$ を格納する。この配列は、分散行列 A に関連付けられる。

<i>c</i>	(ローカル)。 REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合) ローカルメモリーにある、次元 ($lld_c, LOCc(jc+n-1)$) の配列へのポインター。 分散行列 $sub(C)$ のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 $sub(C)$ の最初の行を示す、グローバル配列 C の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 $sub(C)$ の最初の列を示す、グローバル配列 C の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。 分散行列 C の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (psorm2r の場合) DOUBLE PRECISION (pdorm2r の場合) COMPLEX (pcunm2r の場合) COMPLEX*16 (pzunm2r の場合) ワークスペース配列、次元は ($lwork$)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 $work$ の次元。 $lwork$ はローカル入力であり、次の値でなければならない。 $side='L'$ の場合、 $lwork \geq mpc0 + \max(1, nqc0)$ 。 $side='R'$ の場合、 $lwork \geq nqc0 + \max(\max(1, mpc0), numroc$ $(numroc(n+icoffc, nb_a, 0, 0, npc0), nb_a, 0, 0,$ $lcmq))$ 。 ここで、 $lcmq = lcm / npc0$ 、 $lcm = iclm(nprow, npc0)$ 、 $iroffc = \text{mod}(ic-1, mb_c)$ 、 $icoffc = \text{mod}(jc-1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, myrow, rsrc_c, nprow)$ 、 $iccol = \text{indxg2p}(jc, nb_c, mycol, csrc_c, npc0)$ 、 $Mqc0 = \text{numroc}(m+icoffc, nb_c, mycol, icrow, nprow)$ 、 $Npc0 = \text{numroc}(n+iroffc, mb_c, myrow, iccol, npc0)$ 、 $iclm$ 、 indxg2p 、および numroc は、ScaLAPACK ツール関数である。 $myrow$ 、 $mycol$ 、 $nprow$ 、および $npc0$ は、サブルーチン blacs_gridinfo を呼び出して求められる。 $lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、 pxerbla はエラーメッセージを生成しない。

出力パラメーター

<i>c</i>	終了時に、 $\text{sub}(C)$ は、 $Q*\text{sub}(C)$ 、 $Q'^*\text{sub}(C)$ 、 $\text{sub}(C)*Q'$ 、または $\text{sub}(C)*Q$ のいずれかで上書きされる。
<i>work</i>	終了時に、 <i>work</i> (1) は、最小かつ最適な <i>lwork</i> 値を返す。
<i>info</i>	(ローカル) INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば $\text{info} = -(i*100+j)$ 、 <i>i</i> 番目の引数がスカラーで値が不正でならば $\text{info} = -i$ 。



注: 分散部分行列 $A(ia:*, ja:*)$ と $C(ic:ic+m-1, jc:jc+n-1)$ は、次の式を満たすようにアライメントされていなければならない。
side = 'L' の場合、(*mb_a*.eq.*mb_c* .AND. *iroffa*.eq.*iroffc* .AND. *iarow*.eq.*icrow*)
side = 'R' の場合、(*mb_a*.eq.*nb_c* .AND. *iroffa*.eq.*iroffc*)

p?orml2/p?unml2

一般行列に p?gelqf で求めた LQ 因子分解の直交/ユニタリ行列を掛ける (非ブロック化アルゴリズム)。

構文

```
call psorml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pdorml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pcunml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
call pzunml2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
work, lwork, info)
```

説明

ルーチン p?orml2/p?unml2 は、 $m \times n$ の一般実/複素分布行列 $\text{sub}(C)=C(ic:ic+m-1, jc:jc+n-1)$ を以下で上書きする。

	<i>side</i> = 'L'	<i>side</i> = 'R'
<i>trans</i> = 'N'	$Q*\text{sub}(C)$	$\text{sub}(C)*Q$
<i>trans</i> = 'T' (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C)*Q^T$
<i>trans</i> = 'C' (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C)*Q^H$

Q は、 k 個の基本リフレクターの積として定義される実直交 / 複素ユニタリー分散行列である。

$$Q = H(k) \dots H(2) H(1) \text{ (実数型の場合)},$$

$$Q = H(k)' \dots H(2)' H(1)' \text{ (複素数型の場合)}$$

[p?qelqf](#) によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

<i>side</i>	(グローバル) CHARACTER。 'L' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型) を左から適用する。 'R' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型の場合) を右から適用する。
<i>trans</i>	(グローバル) CHARACTER。 'N' の場合、 Q を適用する (転置なし) 'T' の場合、 Q^T を適用する (転置、実数型の場合) 'C' の場合、 Q^H を適用する (共役転置、複素数型の場合)
<i>m</i>	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $sub(C)$ の行数。 $m \geq 0$ 。
<i>n</i>	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $sub(C)$ の列数。 $n \geq 0$ 。
<i>k</i>	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
<i>a</i>	(ローカル)。 REAL (psorml2 の場合) DOUBLE PRECISION (pdorml2 の場合) COMPLEX (pcunml2 の場合) COMPLEX*16 (pzunml2 の場合) ローカルメモリーにある、次元 (lld_a , $LOCc(ja+m-1)$) ($side='L'$ の場合) または (lld_a , $LOCc(ja+n-1)$) ($side='R'$ の場合) の配列へのポインター。 ここで、 $lld_a \geq \max(1, LOCr(ia+k-1))$ 。 i 番目の行には、 p?qelqf によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の k 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。 ここで、 $H(i)$ は $ia \leq i \leq ia+k-1$ 。引数 $A(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。
<i>ia</i>	(グローバル) INTEGER。 $sub(A)$ の最初の行を示す、グローバル配列 A の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 $sub(A)$ の最初の列を示す、グローバル配列 A の列インデックス。

<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>psorm12</i> の場合) DOUBLE PRECISION (<i>pdorm12</i> の場合) COMPLEX (<i>pcunml2</i> の場合) COMPLEX*16 (<i>pzunml2</i> の場合) 配列、次元は $LOCc(ia+k-1)$ 。 p?qelqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数 $tau(i)$ を格納する。この配列は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル)。 REAL (<i>psorm12</i> の場合) DOUBLE PRECISION (<i>pdorm12</i> の場合) COMPLEX (<i>pcunml2</i> の場合) COMPLEX*16 (<i>pzunml2</i> の場合) ローカルメモリーにある、次元 ($lld_c, LOCc(jc+n-1)$) の配列へのポインター。分散行列 <i>sub(C)</i> のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の行を示す、グローバル配列 <i>C</i> の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 <i>sub(C)</i> の最初の列を示す、グローバル配列 <i>C</i> の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (<i>psorm12</i> の場合) DOUBLE PRECISION (<i>pdorm12</i> の場合) COMPLEX (<i>pcunml2</i> の場合) COMPLEX*16 (<i>pzunml2</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、次の値でなければならない。 <i>side</i> = 'L' の場合、 $lwork \geq mqc0 + \max(\max(1, npc0), numroc(numroc(m+icoffc, mb_a, 0, 0, nrow), mb_a, 0, 0, lcmp))$ 。 <i>side</i> = 'R' の場合、 $lwork \geq npc0 + \max(1, mqc0)$ 。 ここで、 $lcmp = lcm / nrow$ 、 $lcm = iclm(nrow, npc0)$ 、 $iroffc = \text{mod}(ic-1, mb_c)$ 、 $icoffc = \text{mod}(jc-1, nb_c)$ 、 $icrow = \text{indxg2p}(ic, mb_c, myrow, rsrc_c, nrow)$ 、 $iccol = \text{indxg2p}(jc, nb_c, mycol, csrc_c, npc0)$ 、 $Mpc0 = \text{numroc}(m+icoffc, mb_c, mycol, icrow, nrow)$ 、 $Nqc0 = \text{numroc}(n+iroffc, nb_c, myrow, iccol, npc0)$ 。

ilcm、indxg2p、および numroc は、ScaLAPACK ツール関数である。

myrow、mycol、nprow、および npc1 は、サブルーチン blacs_gridinfo を呼び出して求められる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c	終了時に、 $sub(C)$ は、 $Q*sub(C)$ 、 $Q'*sub(C)$ 、 $sub(C)*Q'$ 、または $sub(C)*Q$ のいずれかで書き込まれる。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル) INTEGER。 $= 0$ の場合、正常に終了したことを示す。 < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 i 番目の引数がスカラーで値が不正でならば $info = -i$ 。



注: 分散部分行列 $A(ia:*, ja:*)$ と $C(ic:ic+m-1, jc:jc+n-1)$ は、次の式を満たすようにアライメントされていなければならない。
 $side = 'L'$ の場合、($nb_a.eq.mb_c$.AND. $icoffa.eq.iroffc$)
 $side = 'R'$ の場合、($nb_a.eq.nb_c$.AND. $icoffa.eq.icoffc$.AND. $iacol.eq.iccol$)

p?ormr2/p?unmr2

一般行列に p?gerqf で求めた RQ 因子分解の直交/ユニタリー行列を掛ける (非ブロック化アルゴリズム)。

構文

```
call psormr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pdormr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pcunmr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
call pzunmr2(side, trans, m, n, k, a, ia, ja, desca, tau, c, ic, jc, descc,
             work, lwork, info)
```

説明

ルーチン `p?ormr2/p?unmr2` は、 $m \times n$ の一般実 / 複素分布行列 $\text{sub}(C)=C(ic:ic+m-1,jc:jc+n-1)$ を以下で上書きする。

	$side = 'L'$	$side = 'R'$
$trans = 'N'$	$Q * \text{sub}(C)$	$\text{sub}(C) * Q$
$trans = 'T'$ (実数型の場合)	$Q^T * \text{sub}(C)$	$\text{sub}(C) * Q^T$
$trans = 'C'$ (複素数型の場合)	$Q^H * \text{sub}(C)$	$\text{sub}(C) * Q^H$

Q は、 k 個の基本リフレクターの積として定義される実直交 / 複素ユニタリー分散行列である。

$$Q = H(1) H(2) \dots H(k) \quad (\text{実数型の場合}),$$

$$Q = H(1) H(2) \dots H(k) \quad (\text{複素数型の場合})$$

`p?gerqf` によって返される。 Q の次数は、 $side = 'L'$ の場合は m 、 $side = 'R'$ の場合は n である。

入力パラメーター

$side$	(グローバル) CHARACTER。 'L' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型) を左から適用する。 'R' の場合、 Q 、 Q^T (実数型の場合) または Q^H (複素数型の場合) を右から適用する。
$trans$	(グローバル) CHARACTER。 'N' の場合、 Q を適用する (転置なし) 'T' の場合、 Q^T を適用する (転置、実数型の場合) 'C' の場合、 Q^H を適用する (共役転置、複素数型の場合)
m	(グローバル) INTEGER。 演算を行う行数、すなわち、分散部分行列 $\text{sub}(C)$ の行数。 $m \geq 0$ 。
n	(グローバル) INTEGER。 演算を行う列数、すなわち、分散部分行列 $\text{sub}(C)$ の列数。 $n \geq 0$ 。
k	(グローバル) INTEGER。 その積が行列 Q を定義する基本リフレクターの個数。 $side = 'L'$ の場合、 $m \geq k \geq 0$ 。 $side = 'R'$ の場合、 $n \geq k \geq 0$ 。
a	(ローカル)。 REAL (psormr2 の場合) DOUBLE PRECISION (pdormr2 の場合) COMPLEX (pcunmr2 の場合) COMPLEX*16 (pzunmr2 の場合) ローカルメモリーにある、次元 $(11d_a, LOCC(ja+m-1))$ ($side='L'$ の場合) または $(11d_a, LOCC(ja+n-1))$ ($side='R'$ の場合) の配列へのポインター。 ここで、 $11d_a \geq \max(1, LOCr(ia+k-1))$ 。 i 番目の行には、

[p?gerqf](#) によって分散行列引数 $A(ia:ia+k-1, ja:*)$ の k 行に返されたとおりに、基本リフレクター $H(i)$ を定義するベクトルを格納しておかなければならない。

ここで、 $H(i)$ は $ia \leq i \leq ia+k-1$ 。

引数 $A(ia:ia+k-1, ja:*)$ はルーチンにより変更されるが終了時に復元される。

<i>ia</i>	(グローバル) INTEGER。 sub(A) の最初の行を示す、グローバル配列 <i>A</i> の行インデックス。
<i>ja</i>	(グローバル) INTEGER。 sub(A) の最初の列を示す、グローバル配列 <i>A</i> の列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₁)。 分散行列 <i>A</i> の配列ディスクリプター。
<i>tau</i>	(ローカル)。 REAL (<i>psormr2</i> の場合) DOUBLE PRECISION (<i>pdormr2</i> の場合) COMPLEX (<i>pcunmr2</i> の場合) COMPLEX*16 (<i>pzunmr2</i> の場合) 配列、次元は $LOCc(ia+k-1)$ 。 p?gerqf から返されたとおりに、基本リフレクター $H(i)$ のスカラー係数 $tau(i)$ を格納する。この配列は、分散行列 <i>A</i> に関連付けられる。
<i>c</i>	(ローカル)。 REAL (<i>psormr2</i> の場合) DOUBLE PRECISION (<i>pdormr2</i> の場合) COMPLEX (<i>pcunmr2</i> の場合) COMPLEX*16 (<i>pzunmr2</i> の場合) ローカルメモリーにある、次元 (<i>lld_c</i> , $LOCc(jc+n-1)$) の配列へのポインター。分散行列 sub(<i>C</i>) のローカル部分を格納する。
<i>ic</i>	(グローバル) INTEGER。 sub(<i>C</i>) の最初の行を示す、グローバル配列 <i>C</i> の行インデックス。
<i>jc</i>	(グローバル) INTEGER。 sub(<i>C</i>) の最初の列を示す、グローバル配列 <i>C</i> の列インデックス。
<i>descc</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i> ₁)。 分散行列 <i>C</i> の配列ディスクリプター。
<i>work</i>	(ローカル)。 REAL (<i>psormr2</i> の場合) DOUBLE PRECISION (<i>pdormr2</i> の場合) COMPLEX (<i>pcunmr2</i> の場合) COMPLEX*16 (<i>pzunmr2</i> の場合) ワークスペース配列、次元は (<i>lwork</i>)。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。 配列 <i>work</i> の次元。 <i>lwork</i> はローカル入力であり、次の値でなければならない。 <i>side</i> = 'L' の場合、 $lwork \geq mpc0 + \max(\max(1, nqc0), numroc$

($\text{numroc}(m+i\text{roffc}, mb_a, 0, 0, nprow), mb_a, 0, 0, lcmp)$)。

$side = 'R'$ の場合、 $lwork \geq nqc0 + \max(1, mpc0)$)。

ここで、 $lcmp = lcm / nprow$ 、 $lcm = iclm(nprow, npc01)$ 、

$i\text{roffc} = \text{mod}(ic-1, mb_c)$ 、 $i\text{coffc} = \text{mod}(jc-1, nb_c)$ 、

$icrow = \text{indxg2p}(ic, mb_c, myrow, rsrc_c, nprow)$ 、

$iccol = \text{indxg2p}(jc, nb_c, mycol, csrc_c, npc01)$ 、

$Mpc0 = \text{numroc}(m+i\text{roffc}, mb_c, myrow, icrow, nprow)$ 、

$Nqc0 = \text{numroc}(n+i\text{coffc}, nb_c, mycol, iccol, npc01)$ 、

$ilcm$ 、 indxg2p 、および numroc は、ScaLAPACK ツール関数である。

$myrow$ 、 $mycol$ 、 $nprow$ 、および $npc01$ は、サブルーチン blacs_gridinfo を呼び出して求められる。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエンタリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

出力パラメーター

c	終了時に、 $\text{sub}(C)$ は、 $Q * \text{sub}(C)$ 、 $Q' * \text{sub}(C)$ 、 $\text{sub}(C) * Q'$ 、または $\text{sub}(C) * Q$ のいずれかで上書きされる。
$work$	終了時に、 $work(1)$ は、最小かつ最適な $lwork$ 値を返す。
$info$	(ローカル) INTEGER。 $= 0$ の場合、正常に終了したことを示す。 < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i * 100 + j)$ 、 i 番目の引数がスカラーで値が不正でならば $info = -i$ 。



注: 分散部分行列 $A(ia:*, ja:*)$ と $C(ic:ic+m-1, jc:jc+n-1)$ は、次の式を満たすようにアライメントされていなければならない。

$side = 'L'$ の場合、 $(nb_a.eq.mb_c .AND. icoffa.eq.iroffc)$

$side = 'R'$ の場合、 $(nb_a.eq.nb_c .AND. icoffa.eq.icoffc .AND. iacol.eq.iccol)$

p?pbtrsv

前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pbtrf で計算された帯行列の係数である。

構文

```
call pspbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pdpbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pcpbtrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
call pzpbttrsv(uplo, trans, n, bw, nrhs, a, ja, desca, b, ib, descb, af,
             laf, work, lwork, info)
```

説明

ルーチン p?pbtrsv は、次の三角帯行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1)*X = B(jb:jb+n-1, 1:nrhs)$$

または

$$A(1:n, ja:ja+n-1)^T * X = B(jb:jb+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H * X = B(jb:jb+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$ はコレスキー因子分解コード p?pbtrf によって生成される三角帯行列であり、 $A(1:n, ja:ja+n-1)$ と *af* に格納される。 $A(1:n, ja:ja+n-1)$ には、*uplo* の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$ または $A(1:n, ja:ja+n-1)^T$ (実数型の場合) と $A(1:n, ja:ja+n-1)^H$ (複素数型の場合) のどの式を解くのかは、パラメーター *trans* を介して、ユーザーによって決定される。

ルーチン [p?pbtrf](#) は、最初に呼び出さなければならない。

入力パラメーター

uplo (グローバル) CHARACTER。'U' または 'L' でなければならない。
uplo = 'U' の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。
uplo = 'L' の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。

trans (グローバル) CHARACTER。'N'、'T'、または 'C' のいずれかでなければならない。
trans = 'N' の場合、 $A(1:n, ja:ja+n-1)$ を解く。
trans = 'T' または 'C' (実数型) の場合、 $A(1:n, ja:ja+n-1)^T$ を解く。
trans = 'C' (複素数型) の場合、
 conjugate_transpose ($A(1:n, ja:ja+n-1)$) を解く。

<i>n</i>	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分行列 $A(1:n, ja:ja+n-1)$ の次数。 $n \geq 0$ 。
<i>bw</i>	(グローバル) INTEGER。'L' または 'U' の劣対角成分の数。 $0 \leq bw \leq n-1$ 。
<i>nrhs</i>	(グローバル) INTEGER。右辺の数。 分散部分行列 $B(jb:jb+n-1, 1:nrhs)$ の列数 ($nrhs \geq 0$)。
<i>a</i>	(ローカル)。 REAL (pspbtrsv の場合) DOUBLE PRECISION (pdpbtrsv の場合) COMPLEX (pcpbtrsv の場合) COMPLEX*16 (pzpbtrsv の場合) ローカルメモリーにある、第 1 次元が $lld_a \geq (bw+1)$ の配列へのポインター (<i>desca</i> に格納される)。 $n \times n$ の対称帯形式の分割コレスキー因子 L または $L^T A(1:n, ja:ja+n-1)$ のローカル部分を格納する。 このローカル部分は、LAPACK で使用される圧縮帯形式で格納される。分散行列の形式の詳細は、次の「アプリケーション・ノート」および ScaLAPACK マニュアルを参照。
<i>ja</i>	(グローバル) INTEGER。(A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 分散行列 A の配列ディスクリプター。 1D type (<i>dtype_a</i> = 501) の場合、 $dlen \geq 7$ 。 2D type (<i>dtype_a</i> = 1) の場合、 $dlen \geq 9$ 。 A のマッピング情報をメモリーに格納する。詳細は、ScaLAPACK マニュアルを参照。
<i>b</i>	(ローカル)。 REAL (pspbtrsv の場合) DOUBLE PRECISION (pdpbtrsv の場合) COMPLEX (pcpbtrsv の場合) COMPLEX*16 (pzpbtrsv の場合) ローカルメモリーにある、ローカル・リーディング・ディメンジョン $lld_b \geq nb$ の配列へのポインター。 右辺 $B(jb:jb+n-1, 1:nrhs)$ のローカル部分を格納する。
<i>ib</i>	(グローバル) INTEGER。(B のすべて、または B の部分行列で) 処理される行列の最初の行を指すグローバル配列 B の行インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen</i>)。 分散行列 B の配列ディスクリプター。 1D type (<i>dtype_b</i> = 502) の場合、 $dlen \geq 7$ 。 2D type (<i>dtype_b</i> = 1) の場合、 $dlen \geq 9$ 。 B のマッピング情報をメモリーに格納する。詳細は、ScaLAPACK マニュアルを参照。

<i>laf</i>	(ローカル) INTEGER。ユーザー入力の補助非零要素空間 <i>af</i> のサイズ。 $laf \geq (nb+2*bw)*bw$ でなければならない。 <i>laf</i> が小さすぎる場合、エラーコードが返され、エラーにならない最小のサイズが <i>af</i> (1) に返される。
<i>work</i>	(ローカル)。 REAL (<i>pspbtrsv</i> の場合) DOUBLE PRECISION (<i>pdpbtrsv</i> の場合) COMPLEX (<i>pcpbtrsv</i> の場合) COMPLEX*16 (<i>pzpbtrsv</i> の場合) 配列 <i>work</i> は、次元 <i>lwork</i> の一次ワークスペース配列。このスペースは、ルーチンの呼び出しの間に上書きされる。
<i>lwork</i>	(ローカルまたはグローバル) INTEGER。ユーザー入力の <i>work</i> のサイズは、 $lwork \geq bw*nrhs$ 以上でなければならない。 <i>lwork</i> が小さすぎる場合、エラーコードが戻され、 <i>work</i> (1) に最小許容サイズが格納される。

出力パラメーター

<i>af</i>	(ローカル)。 REAL (<i>pspbtrsv</i> の場合) DOUBLE PRECISION (<i>pdpbtrsv</i> の場合) COMPLEX (<i>pcpbtrsv</i> の場合) COMPLEX*16 (<i>pzpbtrsv</i> の場合) 配列 <i>af</i> の次元は <i>laf</i> 。補助非零要素空間が格納される。非零要素は、因子分解ルーチン <i>p?dttrf</i> で作成され、 <i>af</i> に格納される。因子分解ルーチンの後で、 <i>p?pbtrs</i> を使って連立1次方程式を解く場合、因子分解後に <i>af</i> を変更してはならない。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work</i> (1)	終了時に、 <i>lwork</i> の最小値が <i>work</i> (1) に格納される。
<i>info</i>	(ローカル) INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 <i>i</i> 番目の引数がスカラーで値が不正でならば $info = -i$ 。

アプリケーション・ノート

同じ係数行列を使用して右辺のさまざまなセットを解くために、因子分解ルーチンと算出ルーチンを別々に呼び出す場合、補助空間 *af* は因子分解ルーチンと算出ルーチンの呼び出しの間に変更してはならない。

帯行列と三重対角行列を係数行列とする連立1次方程式を解くための最良のアルゴリズムは、さまざまなパラメーター(特に、帯幅)に依存する。現時点では、N/P >> *bw* 用のアルゴリズムのみ実装されている。これらのアルゴリズムには、分割統治、分割、領域分解などがある。

アルゴリズムの説明：分割統治:*

分割統治アルゴリズムは、方程式の数と比較して、行列が綿密に帯格納されているとものと仮定する。このような場合、アトミックな列とプロセス間で分割された行とともに、入力行列 A を 1 次元で分配するのが最良である。基本アルゴリズムは、帯行列を 1 つのブロックが 1 つのプロセッサに格納されるよう P 個に分割し、フェーズ 2 の因子分解またはフェーズ 3 の連立 1 次方程式の算出に進む。

1. **ローカルフェーズ**: 各ブロックは、並行して個別に因子分解される。これらの係数は、補助空間 af に検査不可能な方法で格納された、非零要素を生成する行列に適用される。数学上、これは行列 A を $PA P^T$ に順序変更し、各プロセッサで因子分解された行列のサイズの合計と同じサイズの主要な先頭部分行列を因子分解することに等しい。これらの部分行列の係数は、メモリー中の A に対応する部分を上書きする。
2. **縮退された連立方程式フェーズ**: より大きなブロックの相互作用を表す小さな ($bw * (P-1)$) 連立方程式が形成され、(係数と同様に) 補助空間 af に格納される。並列ブロック・サイクリック分割が使用される。連立 1 次方程式に対して、並列前進解法とそれに続く後進解法が実行される。それぞれの解法は係数行列の構造を使用する。
3. **後退代入フェーズ**: 連立 1 次方程式に対して、各プロセッサでローカルの後退代入が並列に実行される。

p?pttrsv

前進解法または後退解法によって、単一の三角行列線形問題を解く。ここで、三角行列は p?pttrf で計算された三重対角行列の係数である。

構文

```
call psppttrsv(uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
              work, lwork, info)
call pdpttrsv(uplo, n, nrhs, d, e, ja, desca, b, ib, descb, af, laf,
              work, lwork, info)
call pcpttrsv(uplo, trans, n, nrhs, d, e, ja, desca, b, ib, descb, af,
              laf, work, lwork, info)
call pzpttrsv(uplo, trans, n, nrhs, d, e, ja, desca, b, ib, descb, af,
              laf, work, lwork, info)
```

説明

このルーチンは、次の三重対角三角行列を係数行列とする連立 1 次方程式を解く。

$$A(1:n, ja:ja+n-1)*X = B(jb:jb+n-1, 1:nrhs)$$

または

$$A(1:n, ja:ja+n-1)^T *X = B(jb:jb+n-1, 1:nrhs) \text{ (実数型の場合)}$$

$$A(1:n, ja:ja+n-1)^H *X = B(jb:jb+n-1, 1:nrhs) \text{ (複素数型の場合)}$$

ここで、 $A(1:n, ja:ja+n-1)$ はコレスキー因子分解コード `p?pttrf` によって生成される三角帯行列であり、 $A(1:n, ja:ja+n-1)$ と af に格納される。 $A(1:n, ja:ja+n-1)$ には、 $uplo$ の値に従って、上三角行列または下三角行列が格納される。 $A(1:n, ja:ja+n-1)$ または $A(1:n, ja:ja+n-1)^T$ (実数型の場合) と $A(1:n, ja:ja+n-1)^H$ (複素数型の場合) のどの式を解くのかは、パラメーター $trans$ を介して、ユーザーによって決定される。

ルーチン `p?pttrf` は、最初に呼び出さなければならない。

入カパラメーター

<code>uplo</code>	(グローバル) CHARACTER。'U' または 'L' でなければならない。 <code>uplo = 'U'</code> の場合、 $A(1:n, ja:ja+n-1)$ の上三角行列が格納される。 <code>uplo = 'L'</code> の場合、 $A(1:n, ja:ja+n-1)$ の下三角行列が格納される。
<code>trans</code>	(グローバル) CHARACTER。'N' または 'C' でなければならない。 <code>trans = 'N'</code> の場合、 $A(1:n, ja:ja+n-1)$ を解く。 <code>trans = 'C'</code> (複素数型) の場合、 <code>conjugate_transpose (A(1:n, ja:ja+n-1))</code> を解く。
<code>n</code>	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分行列 $A(1:n, ja:ja+n-1)$ の次数。 $n \geq 0$ 。
<code>nrhs</code>	(グローバル) INTEGER。右辺の数。 分散部分行列 $B(jb:jb+n-1, 1:nrhs)$ の列数 ($nrhs \geq 0$)。
<code>d</code>	(ローカル)。 REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合) 行列の主対角を格納するグローバルベクトルのローカル部分へのポインター。サイズは、 $\geq desca(nb_)$ でなければならない。
<code>e</code>	(ローカル)。 REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合) 行列の上対角を格納するグローバルベクトルのローカル部分へのポインター。サイズは、 $\geq desca(nb_)$ でなければならない。 全体的に、 $du(n)$ は参照されず、 du は d とアライメントされなければならない。
<code>ja</code>	(グローバル) INTEGER。 (A のすべて、または A の部分行列で) 処理される行列の先頭を指すグローバル配列 A のインデックス。
<code>desca</code>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。 1D type ($dtype_a = 501$ または 502) の場合、 $dlen \geq 7$ 。

- 2D type ($dtype_a = 1$) の場合、 $dlen \geq 9$ 。
 A のマッピング情報をメモリーに格納する。詳細は、
ScaLAPACK マニュアルを参照。
- b* (ローカル)。
REAL (pspttrsv の場合)
DOUBLE PRECISION (pdpttrsv の場合)
COMPLEX (pcpttrsv の場合)
COMPLEX*16 (pzpttrsv の場合)
ローカルメモリーにある、ローカル・リーディング・ディメン
ジョン
 $lld_b \geq nb$ の配列へのポインター。
右辺 $B(jb:jb+n-1, 1:nrhs)$ のローカル部分を格納する。
- ib* (グローバル) INTEGER。 (B のすべて、または B の部分行列で)
処理される行列の最初の行を指すグローバル配列 B の行イン
デックス。
- descb* (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。
分散行列 B の配列ディスクリプター。
1D type ($dtype_b = 502$) の場合、 $dlen \geq 7$ 。
2D type ($dtype_b = 1$) の場合、 $dlen \geq 9$ 。
 B のマッピング情報をメモリーに格納する。詳細は、
ScaLAPACK マニュアルを参照。
- laf* (ローカル) INTEGER。 ユーザー入力の補助非零要素空間 af のサ
イズ。
 $laf \geq (nb+2*bw)*bw$ でなければならない。
 laf が小さすぎる場合、エラーコードが返され、エラーにならな
い最小のサイズが $af(1)$ に返される。
- work* (ローカル)。
REAL (pspttrsv の場合)
DOUBLE PRECISION (pdpttrsv の場合)
COMPLEX (pcpttrsv の場合)
COMPLEX*16 (pzpttrsv の場合)
配列 $work$ は、次元 $lwork$ の一次ワークスペース配列。このス
ペースは、ルーチンの呼び出しの間に上書きされる。
- lwork* (ローカルまたはグローバル) INTEGER。 ユーザー入力の $work$ の
サイズは、 $lwork \geq (10+2*\min(100, nrhs))*nrcol+4*nrhs$ 以上
でなければならない。 $lwork$ が小さすぎる場合、エラーコードが
返され、エラーにならない最小のサイズが $work(1)$ に返される。

出力パラメーター

- d,e* (ローカル)。
REAL (pspttrsv の場合)
DOUBLE PRECISION (pdpttrsv の場合)
COMPLEX (pcpttrsv の場合)
COMPLEX*16 (pzpttrsv の場合)
終了時に、行列の係数を含む情報が格納される。

<i>af</i>	(ローカル)。 REAL (pspttrsv の場合) DOUBLE PRECISION (pdpttrsv の場合) COMPLEX (pcpttrsv の場合) COMPLEX*16 (pzpttrsv の場合) 配列 <i>af</i> の次元は <i>laf</i> 。補助非零要素空間が格納される。非零要素は、因子分解ルーチン p?pbtrf で作成され、 <i>af</i> に格納される。因子分解ルーチンの後で、 p?pttrs を使って連立 1 次方程式を解く場合、因子分解後に <i>af</i> を変更してはならない。
<i>b</i>	終了時に、解の分散行列 <i>X</i> のローカル部分が格納される。
<i>work(1)</i>	終了時に、 <i>lwork</i> の最小値が <i>work(1)</i> に格納される。
<i>info</i>	(ローカル) INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -(<i>i</i> *100+ <i>j</i>)、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> 。

p?potf2

対称/エルミート正定値行列のコレスキー因子分解を行う (ローカル非ブロック化アルゴリズム)。

構文

```
call pspotf2(uplo, n, a, ia, ja, desca, info)
call pdpotf2(uplo, n, a, ia, ja, desca, info)
call pcpotf2(uplo, n, a, ia, ja, desca, info)
call pzpotf2(uplo, n, a, ia, ja, desca, info)
```

説明

このルーチンは、実対称/複素エルミート正定値分散行列 $\text{sub}(A)=A(ia:ia+n-1, ja:ja+n-1)$ に対してコレスキー因子分解を行う。

因子分解の形式は次のとおりである。

$\text{sub}(A) = U' U$ (*uplo* = 'U' の場合)

$\text{sub}(A) = L L'$ (*uplo* = 'L' の場合)

ここで、*U* は上三角行列、*L* は下三角行列である。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER。 対称/エルミート行列 <i>A</i> の上三角または下三角部分のどちらを格納するか指定する。 'U' の場合、 $\text{sub}(A)$ の上三角部分が格納される。 'L' の場合、 $\text{sub}(A)$ の下三角部分が格納される。
<i>n</i>	(グローバル) INTEGER。演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。

<i>a</i>	(ローカル)。 REAL (pspotf2 の場合) DOUBLE PRECISION (pdpotf2 の場合) COMPLEX (pcpotf2 の場合) COMPLEX*16 (pzpotf2 の場合) ローカルメモリーにある、因子分解する $n \times n$ の対称分散行列 $\text{sub}(A)$ を格納する次元 ($11d_a$, $LOCc(ja+n-1)$) の配列へのポインター。 $uplo = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に、上三角行列を格納する。この行列の厳密な下三角部分は参照されない。 $uplo = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の下三角部分に、下三角行列を格納する。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。分散行列 A の配列ディスクリプター。

出力パラメーター

<i>a</i>	(ローカル)。終了時は、 $uplo = 'U'$ の場合、分散行列の上三角部分にコレスキー因子 U を格納する。 $uplo = 'L'$ の場合、分散行列の下三角部分にコレスキー因子 L を格納する。
<i>info</i>	(ローカル) INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100+j)$ 、 i 番目の引数がスカラーで値が不正でならば $info = -i$ 。 $INFO > 0$ の場合、 $info = k$ は次数 k の先頭の小行列式が正定値でないため因子分解を完了できなかったことを示す。

p?rscl

ベクトルに実スカラーの逆数を掛ける。

構文

```
call psrscl(n, sa, sx, ix, jx, descx, incx)
call pdrscl(n, sa, sx, ix, jx, descx, incx)
call pcsrscl(n, sa, sx, ix, jx, descx, incx)
call pzdrscl(n, sa, sx, ix, jx, descx, incx)
```

説明

このルーチンは、 n 成分の実/複素ベクトル $\text{sub}(x)$ に実スカラー $1/a$ を掛ける。最終結果 $\text{sub}(x)/a$ がオーバーフローまたはアンダーフローを起こさない限り、演算はオーバーフローまたはアンダーフローを起こすことなく実行される。

$\text{sub}(x)$ は $x(ix:ix+n-1, jx:jx)$ ($incx=1$ の場合)、または $x(ix:ix, jx:jx+n-1)$ ($incx=m_x$ の場合) である。

入力パラメーター

n	(グローバル) INTEGER。 乱数ベクトル $\text{sub}(x)$ の成分の個数。 $n \geq 0$ 。
sa	REAL (psrscl/pcsrsc1 の場合) DOUBLE PRECISION (pdrsc1/pzdrsc1 の場合) ベクトル x の各成分の除算に使用されるスカラー a 。この引数は ≥ 0 でなければならない。そうでないと、サブルーチンでゼロ除算が発生する。
sx	REAL (psrscl の場合) DOUBLE PRECISION (pdrsc1 の場合) COMPLEX (pcsrsc1 の場合) COMPLEX*16 (pzdrsc1 の場合) 次元が $((jx-1)*m_x + ix + (n-1)*\text{abs}(incx))$ 以上のローカル部分を格納する配列。 乱数ベクトル $\text{sub}(x)$ の成分を格納する。
ix	(グローバル) INTEGER。演算を行う分散行列 X の部分行列の行インデックス。
jx	(グローバル) INTEGER。演算を行う分散行列 X の部分行列の列インデックス。
$descx$	(グローバルおよびローカル)。INTEGER。 次元 8 の配列。分散行列 X の配列ディスクリプター。
$incx$	(グローバル) INTEGER。 X の成分に対する増分。このバージョンでは、1 と m_x の 2 つの $incx$ の値のみサポートされる。

出力パラメーター

sx	終了時に、結果 x/a で上書きされる。
------	------------------------

p?sygs2/p?hegs2

p?potrf で得られた因子分解の結果を用いて、対称/エルミート汎用固有値問題を標準形式に縮退させる(ローカル非ブロック化アルゴリズム)。

構文

```
call pssygs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
```

```
call pdsygs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
call pchehs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
call pzhehs2(ibtype, uplo, n, a, ia, ja, desca, b, ib, jb, descb, info)
```

説明

ルーチン `p?sygs2/p?hegs2` は、実対称 / 複素エルミートの汎用固有値問題を標準化形式に縮退させる。

$\text{sub}(A)$ は $A(ia:ia+n-1, ja:ja+n-1)$ 、 $\text{sub}(B)$ は $B(ib:ib+n-1, jb:jb+n-1)$ である。

$ibtype = 1$ の場合、問題は次のとおりである。

$$\text{sub}(A)x = \lambda \text{sub}(B)x$$

$\text{sub}(A)$ は次のように上書きされる。

$\text{inv}(U^T) * \text{sub}(A) * \text{inv}(U)$ または $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^T)$ (実数型の場合)
 $\text{inv}(U^H) * \text{sub}(A) * \text{inv}(U)$ または $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^H)$ (複素数型の場合)

$ibtype = 2$ または 3 の場合、問題は次のとおりである。

$$\text{sub}(A)\text{sub}(B)x = \lambda x \quad \text{または} \quad \text{sub}(B)\text{sub}(A)x = \lambda x$$

$\text{sub}(A)$ は次のように上書きされる。

$U * \text{sub}(A) * U^T$ または $L * T * \text{sub}(A) * L$ (実数型の場合)
 $U * \text{sub}(A) * U^H$ または $L * H * \text{sub}(A) * L$ (複素数型の場合)

$\text{sub}(B)$ は $U^T U$ 、 $L L^T$ (実数型の場合) または $U^H U$ 、 $L L^H$ (複素数型の場合) として、[p?potrf](#) によって事前に因子分解されていなければならない。

入力パラメーター

ibtype (グローバル) INTEGER。
 1 の場合、 $\text{inv}(U^T) * \text{sub}(A) * \text{inv}(U)$ または $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^T)$ (実数サブルーチンの場合)、 $\text{inv}(U^H) * \text{sub}(A) * \text{inv}(U)$ または $\text{inv}(L) * \text{sub}(A) * \text{inv}(L^H)$ (複素数サブルーチンの場合) を計算する。
 2 または 3 の場合、 $U * \text{sub}(A) * U^T$ または $L^T * \text{sub}(A) * L$ (実数サブルーチンの場合) または $U * \text{sub}(A) * U^H$ または $L^H * \text{sub}(A) * L$ (複素数サブルーチンの場合) を計算する。

uplo (グローバル) CHARACTER
 対称 / エルミート行列 $\text{sub}(A)$ の上三角部分または下三角部分のどちらが格納されているか、および $\text{sub}(B)$ がどのように因子分解されるかを指定する。
 'U' の場合、 $\text{sub}(A)$ の上三角を格納し、 $\text{sub}(B)$ を $U^T U$ (実数サブルーチンの場合) または $U^H U$ (複素数サブルーチン) として因数分解する。
 'L' の場合、 $\text{sub}(A)$ の下三角を格納し、 $\text{sub}(B)$ を $L L^T$ (実数サブルーチンの場合) または $L L^H$ (複素数サブルーチンの場合)

n (グローバル) INTEGER。
 行列 $\text{sub}(A)$ と $\text{sub}(B)$ の次数。 $n \geq 0$ 。

<i>a</i>	(ローカル)。 REAL (pssygs2 の場合) DOUBLE PRECISION (pdsygs2 の場合) COMPLEX (pcheys2 の場合) COMPLEX*16 (pzhegs2 の場合) ローカルメモリーにある 次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。 $n \times n$ の対称 / エルミート行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 <i>sub(A)</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、 <i>sub(A)</i> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。 <i>sub(A)</i> の厳密な上三角部分は参照されない。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。
<i>b</i>	(ローカル)。 REAL (pssygs2 の場合) DOUBLE PRECISION (pdsygs2 の場合) COMPLEX (pcheys2 の場合) COMPLEX*16 (pzhegs2 の場合) ローカルメモリーにある 次元 (<i>lld_b</i> , <i>LOCc(jb+n-1)</i>) の配列へのポインター。 p?potrf によって返された、 <i>sub(B)</i> のコレスキー因子分解で得られた三角係数を格納する。
<i>ib, jb</i>	(グローバル) INTEGER。それぞれ、 <i>sub(B)</i> の最初の行と最初の列を示す、グローバル配列 <i>B</i> の行インデックスと列インデックス。
<i>descb</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>B</i> の配列ディスクリプター。

出力パラメーター

<i>a</i>	(ローカル)。終了時に、 <i>info</i> = 0 の場合、変換後の行列が <i>sub(A)</i> と同じ形式で格納される。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -(<i>i</i> *100)、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> 。

p?sytd2/p?hetd2

直交/ユニタリー相似変換を用いて、対称/エルミート行列を実数対称三重対角形式に縮退させる (ローカル非ブロック化アルゴリズム)。

構文

```
call pssytd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pdsytd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pchetd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
call pzhetd2(uplo, n, a, ia, ja, desca, d, e, tau, work, lwork, info)
```

説明

ルーチン p?sytd2/p?hetd2 は、直交/ユニタリー相似変換 $Q' \text{sub}(A) Q = T$ を用いて、実対称/複素エルミート行列 $\text{sub}(A)$ を対称/エルミート三重対角形式 T に縮退させる。ここで、 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ である。

入力パラメーター

uplo (グローバル) CHARACTER。
対称/エルミート行列 $\text{sub}(A)$ の上三角部分と下三角部分のどちらが格納されるかを指定する。
'U' の場合、上三角。
'L' の場合、下三角部分

n (グローバル) INTEGER。
演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。
 $n \geq 0$ 。

a (ローカル)。
REAL (pssytd2 の場合)
DOUBLE PRECISION (pdsytd2 の場合)
COMPLEX (pchetd2 の場合)
COMPLEX*16 (pzhetd2 の場合)
ローカルメモリーにある
次元 (lld_a , $LOCc(ja+n-1)$) の配列へのポインター。
 $n \times n$ の対称/エルミート行列 $\text{sub}(A)$ のローカル部分を格納する。
 $uplo = 'U'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 $\text{sub}(A)$ の厳密な下三角部分は参照されない。
 $uplo = 'L'$ の場合、 $\text{sub}(A)$ の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。 $\text{sub}(A)$ の厳密な上三角部分は参照されない。

ia, ja (グローバル) INTEGER。それぞれ、部分行列 $\text{sub}(A)$ の最初の行と最初の列を示す、グローバル配列 A の行インデックスと列インデックス。

desca (グローバルおよびローカル) INTEGER 配列、次元は ($dlen_$)。
分散行列 A の配列ディスクリプター。

work (ローカル)。
 REAL (pssytd2 の場合)
 DOUBLE PRECISION (pdsytd2 の場合)
 COMPLEX (pchetd2 の場合)
 COMPLEX*16 (pzhetd2 の場合)
 配列 *work* は、次元 *lwork* の一次ワークスペース配列。

出力パラメーター

a 終了時に、*uplo* = 'U' の場合、*sub(A)* の対角成分と最初の優対角成分は、三重対角行列 *T* の対応する成分で上書きされる。最初の優対角成分より上の成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリー行列 *Q* を表現する。
uplo = 'L' の場合、*a* の対角成分と最初の劣対角成分は、三重対角行列 *T* の対応する成分で上書きされる。最初の劣対角成分より下の成分は、配列 *tau* とともに、基本リフレクターの積として直交/ユニタリー行列 *Q* を表現する。次の「アプリケーション・ノート」を参照。

d (ローカル)。
 REAL (pssytd2/pchetd2 の場合)
 DOUBLE PRECISION (pdsytd2/pzhetd2 の場合)
 配列、次元は (*LOCc(ja+n-1)*)。
 三重対角行列 *T* の対角成分。
 $d(i) = a(i,i)$; *d* は、分散行列 *A* に関連付けられる。

e (ローカル)。
 REAL (pssytd2/pchetd2 の場合)
 DOUBLE PRECISION (pdsytd2/pzhetd2 の場合)
 配列、次元は (*LOCc(ja+n-1)*) (*uplo* = 'U' の場合) または *LOCc(ja+n-2)* (それ以外の場合)。
 三重対角行列 *T* の非対角成分。
 $e(i) = a(i,i+1)$ (*uplo* = 'U' の場合)
 $e(i) = a(i+1,i)$ (*uplo* = 'L' の場合)
e は、分散行列 *A* に関連付けられる。

tau (ローカル)。
 REAL (pssytd2 の場合)
 DOUBLE PRECISION (pdsytd2 の場合)
 COMPLEX (pchetd2 の場合)
 COMPLEX*16 (pzhetd2 の場合)
 配列、次元は (*LOCc(ja+n-1)*)。
 基本リフレクターのスカラー係数。
tau は、分散行列 *A* に関連付けられる。

work(1) 終了時に、*work(1)* は、最小かつ最適な *lwork* 値を返す。

lwork (ローカルまたはグローバル) INTEGER。
 ワークスペース配列 *work* の次元。
lwork はローカル入力であり、 $lwork \geq 3n$ でなければならない。

$lwork = -1$ の場合、 $lwork$ はグローバル入力であり、ワークスペースのクエリーとみなされ、ルーチンはすべての配列の最小かつ最適なサイズだけを計算する。各値は該当する $work$ 配列の最初のエントリーとして返され、[pxerbla](#) はエラーメッセージを生成しない。

$info$ (ローカル) INTEGER。
 $= 0$ の場合、正常に終了したことを示す。
 < 0 の場合、 i 番目の引数が配列で j 番目の成分の値が不正ならば $info = -(i*100)$ 、 i 番目の引数がスカラーで値が不正でならば $info = -i$ 。

アプリケーション・ノート

$uplo = 'U'$ の場合、行列 Q は基本リフレクターの積として表現される。

$$Q = H(n-1) \dots H(2) H(1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(i+1:n) = 0$ および $v(i) = 1$ 。終了時に、 $v(1:i-1)$ は $A(ia:ia+i-2, ja+i)$ に、 τ は $TAU(ja+i-1)$ に格納される。

$uplo = 'L'$ の場合、行列 Q は基本リフレクターの積として表現される。

$$Q = H(1) H(2) \dots H(n-1)$$

それぞれの $H(i)$ は次のような形式を持つ。

$$H(i) = I - \tau * v * v'$$

ここで、 τ は実 / 複素スカラー、 v は実 / 複素ベクトルで $v(1:i) = 0$ および $v(i+1) = 1$ 。終了時に、 $v(i+2:n)$ は $A(ia+i+1:ia+n-1, ja+i-1)$ に、 τ は $TAU(ja+i-1)$ に格納される。

終了時の $sub(A)$ の内容を次に示す
($n = 5$ の場合)。

$uplo = 'U'$ の場合

$uplo = 'L'$ の場合

$$\begin{array}{c} \left[\begin{array}{ccccc} d & e & v_2 & v_3 & v_4 \\ & d & e & v_3 & v_4 \\ & & d & e & v_4 \\ & & & d & e \\ & & & & d \end{array} \right] & \left[\begin{array}{ccccc} d & & & & \\ e & d & & & \\ v_1 & e & d & & \\ v_1 & v_2 & e & d & \\ v_1 & v_2 & v_3 & e & d \end{array} \right] \end{array}$$

ここで、 d と e は T の対角成分と非対角成分である。 v_i は $H(i)$ を定義するベクトルの成分を表す。



注：分散部分行列 $\text{sub}(A)$ 次の式を満たすようにアライメントされていなければならない。

$(mb_a.eq.nb_a .AND.iroffa.eq.icoffa)$

ここで $iroffa = \text{mod}(ia - 1, mb_a)$ 、 $icoffa = \text{mod}(ja - 1, nb_a)$ である。

p?trti2

三角行列の逆行列を計算する (ローカル非ブロック化アルゴリズム)。

構文

```
call pstrti2(uplo, diag, n, a, ia, ja, desca, info)
call pdtrti2(uplo, diag, n, a, ia, ja, desca, info)
call pctrti2(uplo, diag, n, a, ia, ja, desca, info)
call pztrti2(uplo, diag, n, a, ia, ja, desca, info)
```

説明

このルーチンは、実 / 複素上三角 / 下三角ブロック行列 $\text{sub}(A) = A(ia:ia+n-1, ja:ja+n-1)$ の逆行列を計算する。

この行列は、1つのプロセスメモリーにのみ格納されなければならない (ローカル演算)。

入力パラメーター

<i>uplo</i>	(グローバル) CHARACTER*1。 行列 $\text{sub}(A)$ が上三角か下三角かを指定する。 'U' の場合、 $\text{sub}(A)$ は上三角である。 'L' の場合、 $\text{sub}(A)$ は下三角である。
<i>diag</i>	(グローバル) CHARACTER*1。 行列 A が単位三角かどうかを指定する。 'N' の場合、 $\text{sub}(A)$ は単位三角ではない。 'U' の場合、 $\text{sub}(A)$ は単位三角である。
<i>n</i>	(グローバル) INTEGER。 演算を行う行数と列数、すなわち、分散部分行列 $\text{sub}(A)$ の次数。 $n \geq 0$ 。

<i>a</i>	(ローカル)。 REAL (pstrti2 の場合) DOUBLE PRECISION (pdtrti2 の場合) COMPLEX (pctrti2 の場合) COMPLEX*16 (pztrti2 の場合) ローカルメモリーにある 次元 (<i>lld_a</i> , <i>LOCc(ja+n-1)</i>) の配列へのポインター。三角行列 <i>sub(A)</i> のローカル部分を格納する。 <i>uplo</i> = 'U' の場合、行列 <i>sub(A)</i> の先頭の $n \times n$ の上三角部分に行列の上三角部分を格納する。 <i>sub(A)</i> の厳密な下三角部分は参照されない。 <i>uplo</i> = 'L' の場合、行列 <i>sub(A)</i> の先頭の $n \times n$ の下三角部分に行列の下三角部分を格納する。 <i>sub(A)</i> の厳密な上三角部分は参照されない。 <i>diag</i> = 'U' の場合、 <i>sub(A)</i> の対角成分は参照されず 1 とみなされる。
<i>ia, ja</i>	(グローバル) INTEGER。それぞれ、部分行列 <i>sub(A)</i> の最初の行と最初の列を示す、グローバル配列 <i>A</i> の行インデックスと列インデックス。
<i>desca</i>	(グローバルおよびローカル) INTEGER 配列、次元は (<i>dlen_</i>)。分散行列 <i>A</i> の配列ディスクリプター。

出力パラメーター

<i>a</i>	終了時に、元の行列の (三角) 逆行列で、同じ格納形式で上書きされる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>i</i> 番目の引数が配列で <i>j</i> 番目の成分の値が不正ならば <i>info</i> = -(<i>i</i> *100)、 <i>i</i> 番目の引数がスカラーで値が不正でならば <i>info</i> = - <i>i</i> 。

?lamsh

送られるバルジの数を最大にするために、小さな (単一成分) 行列を介して複数のシフトを送る。

構文

```
call slamsh(s, lds, nbulge, jblk, h, ldh, n, ulp)
call dlamsh(s, lds, nbulge, jblk, h, ldh, n, ulp)
```

説明

このルーチンは、小さな (単一成分) 行列を介して複数のシフトを送り、送られるバルジの数が最大となるよう、小さな連続した対角成分が、続くシフトによってどのように変形されるかを検証する。複数のシフト / バルジ ($nbulge > 1$) があり、2 つ以上の小さな連続した対角成分のために、最初のシフトが縮退されていない Hessenberg 行列の途中から開始される場合のみ、サブルーチン呼び出すべきである。

入力パラメーター

<i>s</i>	(ローカル) INTEGER。 REAL (slamsh の場合) DOUBLE PRECISION (dlamsh の場合) 配列、次元は (<i>lds</i> , *)。 シフト行列。 <i>s</i> の 2×2 対角のみ参照される。 <i>s</i> は <i>jblk</i> 倍精度シフト (サイズ 2) とする。
<i>lds</i>	(ローカル) INTEGER。 <i>S</i> のリーディング・ディメンジョン。終了時に変更されない。 $1 < nbulge \leq jblk \leq lds/2$ 。
<i>nbulge</i>	(ローカル) INTEGER。 <i>h</i> (> 1) を介して送るバルジの数。 <i>nbulge</i> は、決定された (<i>jblk</i>) の最大値以下でなければならない。 $1 < nbulge \leq jblk \leq lds/2$ 。
<i>jblk</i>	(ローカル) INTEGER。 <i>S</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>h</i>	(ローカル) INTEGER。 REAL (slamsh の場合) DOUBLE PRECISION (dlamsh の場合) 配列、次元は (<i>lds</i> , <i>n</i>)。 シフトを適用するローカル行列。 <i>h</i> は、開始行が 2 となるようにアライメントされていなければならない。
<i>ldh</i>	(ローカル) INTEGER。 <i>H</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>n</i>	(ローカル) INTEGER。 <i>H</i> のサイズ。すべてのバルジが送られると見込まれる場合、 <i>n</i> は $4nbulge+2$ 以上でなければならない。それ以外の場合、 <i>nbulge</i> はこのルーチンによって縮小されることがある。
<i>ulp</i>	(ローカル)。 REAL (slamsh の場合) DOUBLE PRECISION (dlamsh の場合) マシン精度。終了時に変更されない。

出力パラメーター

<i>s</i>	終了時に、データは、適用するのに最適な順序に再配置される。
<i>nbulge</i>	終了時に、送信可能なバルジの最大値。

h 終了時に、データは破棄される。

?laref

Householder リフレクターを行列の行または列に適用する。

構文

```
call slaref(type, a, lda, wantz, z, ldz, block, irow1, icoll, istart, istop,
           itmp1, itmp2, lilo, lihiz, vecs, v2, v3, t1, t2, t3)
call dlaref(type, a, lda, wantz, z, ldz, block, irow1, icoll, istart, istop,
           itmp1, itmp2, lilo, lihiz, vecs, v2, v3, t1, t2, t3)
```

説明

このルーチンは、1つまたは複数のサイズ 3 の *Householder* リフレクターを 1つまたは2つの行列 (列が指定された場合) の行または列に適用する。

入力パラメーター

<i>type</i>	(グローバル) CHARACTER*1。 <i>type</i> = 'R' の場合、リフレクターを行列の行に左から適用する。それ以外の場合、リフレクターを行列の列に適用する。終了時に変更されない。
<i>a</i>	(グローバル)。 REAL (<i>slaref</i> の場合) DOUBLE PRECISION (<i>dlaref</i> の場合) 配列、次元は (<i>lda</i> , *)。リフレクターを受け取る行列。
<i>lda</i>	(ローカル) INTEGER。 <i>A</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>wantz</i>	(グローバル) LOGICAL。 <i>wantz</i> = .TRUE. の場合、任意の列リフレクターを <i>Z</i> にも適用する。 <i>wantz</i> = .FALSE. の場合、 <i>Z</i> はそのまま。
<i>z</i>	(グローバル)。 REAL (<i>slaref</i> の場合) DOUBLE PRECISION (<i>dlaref</i> の場合) 配列、次元は (<i>ldz</i> , *)。列リフレクターを受け取る 2 番目の行列。
<i>ldz</i>	(ローカル) INTEGER。 <i>Z</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>block</i>	(グローバル) LOGICAL。 .TRUE. の場合、直ちに複数のリフレクターを適用し、データを配列 <i>vecs</i> から読み込む。 .FALSE. の場合、 <i>v2</i> , <i>v3</i> , <i>t1</i> , <i>t2</i> , および <i>t3</i> で与えられる 1 つのリフレクターを適用する。

<i>irow1</i>	(ローカル) INTEGER。 行列 <i>A</i> のローカル行成分。
<i>icol1</i>	(ローカル) INTEGER。 行列 <i>A</i> のローカル列成分。
<i>istart</i>	(グローバル) INTEGER。 最初のリフレクターの " 番号 " を指定する。 <i>block</i> が設定されている場合、 <i>istart</i> は <i>vecs</i> へのインデックスとして使用される。 <i>block</i> が .FALSE. の場合、 <i>istart</i> は無視される。
<i>istop</i>	(グローバル) INTEGER。 最後のリフレクターの " 番号 " を指定する。 <i>block</i> が設定されている場合、 <i>istop</i> は <i>vecs</i> へのインデックスとして使用される。 <i>block</i> が .FALSE. の場合、 <i>istop</i> は無視される。
<i>itmp1</i>	(ローカル) INTEGER。 <i>A</i> への開始範囲。行に対する最初のローカル列。列に対する最初のローカル行。
<i>itmp2</i>	(ローカル) INTEGER。 <i>A</i> への終了範囲。行に対する最後のローカル列。列に対する最後のローカル行。
<i>liloz, lihiz</i>	(ローカル)。INTEGER。 <i>itmp1</i> 、 <i>itmp2</i> と同じ働きをする。ただし、 <i>Z</i> に対しては <i>wantz</i> が設定されている場合のみ。
<i>vecs</i>	(グローバル)。 REAL (<i>slaref</i> の場合) DOUBLE PRECISION (<i>dlaref</i> の場合) サイズ $3*n$ (行列のサイズ) の配列。サイズ 3 のリフレクターを次々と格納する。 <i>block</i> が .TRUE. の場合のみアクセスされる。
<i>v2,v3,t1,t2,t3</i>	(グローバル) INTEGER。 REAL (<i>slaref</i> の場合) DOUBLE PRECISION (<i>dlaref</i> の場合) これらのパラメーターは、1つのサイズ 3 の Householder リフレクターの情報を格納する。 <i>block</i> が .FALSE. の場合には読み込まれ、 <i>block</i> が .TRUE. の場合には上書きされる。

出力パラメーター

<i>a</i>	終了時に、更新された行列で上書きされる。
<i>z</i>	<i>wantz</i> が設定されている場合のみ変更される。 <i>wantz</i> が .FALSE. の場合、 <i>z</i> は参照されない。
<i>irow1</i>	定義されていない。
<i>icol1</i>	定義されていない。
<i>v2,v3,t1,t2,t3</i>	これらのパラメーターは、 <i>block</i> が .FALSE. の場合には読み込まれ、 <i>block</i> が .TRUE. の場合には上書きされる。

?lasorte

固有ペアを実数および複素数データ型でソートする。

構文

```
call slasorte(s, lds, j, out, info)
call dlasorte(s, lds, j, out, info)
```

説明

このルーチンは、固有ペアが同じデータ型 (実数または複素数) でまとまるようにソートする。これにより、第2劣対角成分が 0 であることが保証され、 2×2 シフトを容易に使用することができるようになる。このルーチンは、並列では動作しない。また、呼び出しも行わない。

入力パラメーター

<i>s</i>	(ローカル) INTEGER。 REAL (slasorte の場合) DOUBLE PRECISION (dlasorte の場合) 配列、次元は (<i>lds</i>)。 行列はすでに Schur 形式である。
<i>lds</i>	(ローカル) INTEGER。 配列 <i>s</i> のリーディング・ディメンジョン。終了時に変更されない。
<i>j</i>	(ローカル) INTEGER。 行列 <i>S</i> の次数。終了時に変更されない。
<i>out</i>	(ローカル) INTEGER。 REAL (slasorte の場合) DOUBLE PRECISION (dlasorte の場合) 配列、次元は (<i>jx2</i>)。 ルーチンに必要なワークバッファー。
<i>info</i>	(ローカル) INTEGER。 入力行列の実数固有値の数が奇数でペアにできない場合、または入力行列 <i>S</i> がもともと Schur 形式でない場合に設定する。0 は正常に終了したことを示す。

出力パラメーター

<i>s</i>	終了時に、 <i>S</i> の対角ブロックは、固有値をペアにするために書き直される。結果として得られる行列は、入力とは似ていない。
<i>out</i>	ワークバッファー。

?lasrt2

数を昇順または降順でソートする。

構文

```
call slasrt2(id, n, d, key, info)
call dlasrt2(id, n, d, key, info)
```

説明

このルーチンは、*d* に格納されている数を昇順 (*id* = 'I' の場合) または降順 (*ID* = 'D' の場合) でソートする、LAPACK ルーチン [?lasrt](#) の修正版である。このルーチンはクイックソートを使用するが、*size* ≤ 20 の配列では挿入ソートが使われる。スタックの次元によって *n* はおよそ 2^{32} に制限される。

入力パラメーター

<i>id</i>	CHARACTER*1。 = 'I' の場合、 <i>d</i> を昇順でソートする。 = 'D' の場合、 <i>d</i> を降順でソートする。
<i>n</i>	INTEGER。配列 <i>d</i> の長さ。
<i>d</i>	REAL (slasrt2 の場合) DOUBLE PRECISION (dlasrt2 の場合) 配列、次元は (<i>n</i>)。 ソートする配列を格納する。
<i>key</i>	INTEGER。 配列、次元は (<i>n</i>)。 <i>key</i> には <i>d</i> () の各成分のキーを格納する。 通常は、すべての <i>i</i> に対して <i>key</i> (<i>i</i>) = <i>i</i> 。

出力パラメーター

<i>d</i>	終了時に、 <i>d</i> は昇順 ($d(1) \leq \dots \leq d(n)$) または降順 ($d(1) \geq \dots \geq d(n)$) (<i>id</i> の設定による) となる。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。
<i>key</i>	終了時に、 <i>d</i> () が入力から出力に置換されたのと同様に、 <i>key</i> は置換される。したがって、入力時にすべての <i>i</i> に対して <i>key</i> (<i>i</i>) = <i>i</i> の場合、 $d_out(i) = d_in(key(i))$ である。

?stein2

逆反復法を用いて、実対称の三重対角行列の指定された固有値に対応する固有ベクトルを計算する。

構文

```
call sstein2(n, d, e, m, w, iblock, isplit, orfac, z, ldz,
            work, iwork, ifail, info)
call dstein2(n, d, e, m, w, iblock, isplit, orfac, z, ldz,
            work, iwork, ifail, info)
```

説明

このルーチンは、LAPACK ルーチン [?stein](#) の修正版である。このルーチンでは、実対称の三重対角行列 T の指定された固有値に対応する固有ベクトルを、反転 (逆行行列の計算) を繰り返して求める。

固有ベクトルごとの最大反復回数は、内部パラメータ `maxits` で指定する (現時点では、5 に設定されている)。

入力パラメーター

`n` INTEGER。行列 T の次数 ($n \geq 0$)。

`m` INTEGER。見つける固有値の数 ($0 \leq m \leq n$)。

`d, e, w` REAL (単精度の場合)
DOUBLE PRECISION (倍精度の場合)
配列:
`d(*)`、DIMENSION (n)。
三重対角行列 T の n 個の対角成分。
`e(*)`、次元は (n)。
三重対角行列 T の ($n-1$) 個の劣対角成分を成分 1 から $n-1$ に格納する。`e(n)` を設定する必要はない。
`w(*)`、次元は (n)。
`w` の最初の m 個の成分には固有ベクトルを計算する固有値を格納する。固有値は分割されたブロックごとにグループ分けし、ブロック内で最小から最大に並べられていなければならない
ORDER = 'B' である [?stebz](#) の出力配列 `w` がここで要求される)。
`w` の次元は、 $\max(1, n)$ 以上でなければならない。

`iblock` INTEGER。
配列、次元は (n)。
`w` 内の対応する固有値に関連した部分行列インデックス。固有値 `w(i)` が第 1 の部分行列の先頭から属する場合は `iblock(i) = 1`
固有値 `w(i)` が 2 番目の部分行列に属する場合、`iblock(i) = 2`。
([?stebz](#) の出力配列 `iblock` がここで要求される)。

<i>isplit</i>	INTEGER。 配列、次元は (n) 。 T を部分行列に分割した分割点。第 1 の部分行列は 1 から <i>isplit</i> (1) の行 / 列で構成され、第 2 の部分行列は <i>isplit</i> (1)+1 から <i>isplit</i> (2) の行 / 列で構成され、以下同様 (?stebz の出力配列 <i>isplit</i> がここで要求される)。
<i>orfac</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) <i>orfac</i> は、直交化される固有ベクトルを指定する。互いの $orfac * \ T\ $ 内にある固有値に対応する固有ベクトルは直交される。
<i>ldz</i>	INTEGER。出力配列 z のリーディング・ディメンジョン。 $ldz \geq \max(1, n)$ 。
<i>work</i>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) ワークスペース配列、次元は $(5n)$ 。
<i>iwork</i>	INTEGER。 ワークスペース配列、次元は (n) 。

出力パラメーター

z	REAL (<i>sstein2</i> の場合) DOUBLE PRECISION (<i>dstein2</i> の場合) 配列、次元は (ldz, m) 。計算された固有ベクトル。固有値 $w(i)$ に対応する固有ベクトルが z の i 番目の列に格納される。収束しなかったすべてのベクトルは、 <i>maxits</i> 回反復した後、現在の反復に設定される。
<i>ifail</i>	INTEGER。配列、次元は (m) 。 通常終了時は、 <i>ifail</i> のすべての成分はゼロになる。1 つまたは複数の固有ベクトルが <i>maxits</i> 回反復した後で収束に失敗した場合、その固有ベクトルのインデックスが配列 <i>ifail</i> に格納される。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - i は i 番目の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = i は i 番目の固有ベクトルが <i>maxits</i> 回目の反復で収束に失敗したことを示す。インデックスは、配列 <i>ifail</i> に格納される。

?dbtf2

一般帯行列の LU 因子分解を、ピボット演算を用いないで計算する (ローカル非ブロック化アルゴリズム)。

構文

```
call sdbtf2(m, n, kl, ku, ab, ldab, info)
call ddbtf2(m, n, kl, ku, ab, ldab, info)
call cdbtf2(m, n, kl, ku, ab, ldab, info)
call zdbtf2(m, n, kl, ku, ab, ldab, info)
```

説明

このルーチンは、 $m \times n$ の一般実 / 複素帯行列 A の LU 因子分解を、行交換を伴う部分ピボット演算を用いないで計算する。

このルーチンは、アルゴリズムの非ブロック化形式で、[BLAS のルーチンと関数](#) を呼び出す。

入力パラメーター

m	INTEGER。行列 A の行数 ($m \geq 0$)。
n	INTEGER。 A の列数 ($n \geq 0$)。
kl	INTEGER。 A の帯内の劣対角成分の数 ($kl \geq 0$)。
ku	INTEGER。 A の帯内の優対角成分の数 ($ku \geq 0$)。
ab	REAL (sdbtf2 の場合) DOUBLE PRECISION (ddbtf2 の場合) COMPLEX (cdbtf2 の場合) COMPLEX*16 (zdbtf2 の場合) 配列、次元は ($ldab, n$)。 行列 A を帯格納形式で行 $kl+1$ から $2kl+ku+1$ に格納する。配列の行 1 から kl を設定する必要はない。 A の j 番目の列は配列 ab の j 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(m, j+kl)$ に対して $ab(kl+ku+1+i-j, j) = A(i, j)$ 。
$ldab$	INTEGER。 ab のリーディング・ディメンジョン。 ($ldab \geq 2kl + ku + 1$)。

出力パラメーター

ab	終了時に、因子分解の詳細で上書きされる。 U は、 $kl+ku$ 個の優対角成分を持つ上三角帯行列として、行 1 から $kl+ku+1$ に格納される。因子分解に使用した乗数は、行 $kl+ku+2$ から $2*kl+ku+1$ に格納される。詳細は、次の「アプリケーション・ノート」を参照。
------	---

info INTEGER。
 = 0 の場合、正常に終了したことを示す。
info < 0 の場合、*info* = -*i* は *i* 番目の引数の値が不正だったことを示す。
info > 0 の場合、*info* = +*i* は *u*(*i*,*i*) が 0 であることを示す。因子分解は完了したが、係数 *U* は完全に特異である。連立 1 次方程式の解の算出に係数 *U* を使用すると、ゼロ除算が発生する。

アプリケーション・ノート

帯格納方式の例を以下に示す ($m = n = 6$ 、 $kl = 2$ 、 $ku = 1$ の場合)。

入力

$$\begin{bmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{bmatrix}$$

出力

$$\begin{bmatrix} * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\ u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\ m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\ m_{31} & m_{42} & m_{53} & m_{64} & * & * \end{bmatrix}$$

* でマークされた配列の成分はルーチンで使用されない。+ でマークされた成分は入力時に設定する必要はないが、行交換の結果として生じる非零要素のために、*U* の成分を格納するのにルーチンが必要となる。

?dbtrf

一般帯行列の LU 因子分解を、ピボット演算を用いないで計算する (ローカルブロック化アルゴリズム)。

構文

```
call sdbtrf(m, n, kl, ku, ab, ldab, info)
call ddbtrf(m, n, kl, ku, ab, ldab, info)
call cdbtrf(m, n, kl, ku, ab, ldab, info)
call zdbtrf(m, n, kl, ku, ab, ldab, info)
```

説明

このルーチンは、 $m \times n$ の一般実 / 複素帯行列 *A* の LU 因子分解を、行交換または部分ピボット演算を用いないで計算する。

このルーチンは、アルゴリズムのブロック化形式で、[BLAS のルーチンと関数](#)を呼び出す。

入力パラメーター

m INTEGER。行列 *A* の行数。 $m \geq 0$ 。

<i>n</i>	INTEGER。A の列数。 $n \geq 0$ 。
<i>k1</i>	INTEGER。A の帯内にある劣対角成分の数。 $k1 \geq 0$ 。
<i>ku</i>	INTEGER。A の帯内にあるの優対角成分の数。 $ku \geq 0$ 。
<i>ab</i>	REAL (sdbtrf の場合) DOUBLE PRECISION (ddbtrf の場合) COMPLEX (cdbtrf の場合) COMPLEX*16 (zdbtrf の場合) 配列、次元は (<i>ldab</i> , <i>n</i>)。 行列 A を帯格納形式で行 <i>k1+1</i> から $2k1+ku+1$ に格納する。配列の行 1 から <i>k1</i> を設定する必要はない。A の <i>j</i> 番目の列は配列 <i>ab</i> の <i>j</i> 番目の列に次のように格納する。 $\max(1, j-ku) \leq i \leq \min(m, j+k1)$ に対して $ab(k1+ku+1+i-j, j) = A(i, j)$ 。
<i>ldab</i>	INTEGER。 <i>ab</i> のリーディング・ディメンジョン。 ($ldab \geq 2k1 + ku + 1$)。

出力パラメーター

<i>ab</i>	終了時に、因子分解の詳細で上書きされる。U は、 <i>k1+ku</i> 個の優対角成分を持つ上三角帯行列として、行 1 から <i>k1+ku+1</i> に格納される。因子分解に使用した乗数は、行 <i>k1+ku+2</i> から $2*k1+ku+1$ に格納される。詳細は、次の「アプリケーション・ノート」を参照。
<i>info</i>	INTEGER。 = 0 の場合、正常に終了したことを示す。 <i>info</i> < 0 の場合、 <i>info</i> = - <i>i</i> は <i>i</i> 番目の引数の値が不正だったことを示す。 <i>info</i> > 0 の場合、 <i>info</i> = + <i>i</i> は $u(i, i)$ が 0 であることを示す。因子分解は完了したが、係数 U は完全に特異である。連立 1 次方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

アプリケーション・ノート

帯格納方式の例を以下に示す ($m = n = 6, k1 = 2, ku = 1$ の場合)。

入力

出力

$$\begin{array}{c}
 \left[\begin{array}{cccccc}
 * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\
 a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\
 a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\
 a_{31} & a_{42} & a_{53} & a_{64} & * & *
 \end{array} \right]
 \end{array}
 \qquad
 \begin{array}{c}
 \left[\begin{array}{cccccc}
 * & u_{12} & u_{23} & u_{34} & u_{45} & u_{56} \\
 u_{11} & u_{22} & u_{33} & u_{44} & u_{55} & u_{66} \\
 m_{21} & m_{32} & m_{43} & m_{54} & m_{65} & * \\
 m_{31} & m_{42} & m_{53} & m_{64} & * & *
 \end{array} \right]
 \end{array}$$

* でマークされた配列の成分はルーチンで使用されない。

?dtrf

一般三重対角行列のLU 因子分解を、ピボット演算を用いないで計算する(ローカルブロック化アルゴリズム)。

構文

```
call sdttrf(n, dl, d, du, info)
call ddttrf(n, dl, d, du, info)
call cdttrf(n, dl, d, du, info)
call zdttrf(n, dl, d, du, info)
```

説明

このルーチンは、実/複素帯行列 A の LU 因子分解を、部分ピボット演算を用いない消去を使用して計算する。

因子分解は次の形式を持つ。

$$A = LU$$

L 単位下二重対角行列の積で、 U は主対角成分と最初の優対角成分にのみ 0 でない値を持つ上三角行列である。

入力パラメーター

n	INTEGER。行列 A の次数。 $n \geq 0$
dl, d, du	REAL (sdttrf の場合) DOUBLE PRECISION (ddttrf の場合) COMPLEX (cdttrf の場合) COMPLEX*16 (zdttrf の場合) A の成分を格納する配列。 次元 $(n-1)$ の配列 dl は、 A の劣対角成分を格納する。 次元 n の配列 d は、 A の対角成分を格納する。 次元 $(n-1)$ の配列 du は、 A の優対角成分を格納する。

出力パラメーター

dl	A の LU 因子分解で得られた行列 L を定義する、 $(n-1)$ 個の乗数によって上書きされる。
d	A の LU 因子分解で得られた上三角行列 U の n 個の対角成分によって上書きされる。
du	U の最初の優対角成分の $(n-1)$ 個の成分によって上書きされる。
$info$	INTEGER。 = 0 の場合、正常に終了したことを示す。 $info < 0$ の場合、 $info = -i$ は i 番目の引数の値が不正だったことを示す。 $info > 0$ の場合、 $info = +i$ は $u(i,i)$ が 0 であることを示す。因子分解は完了したが、係数 U は完全に特異である。連立 1 次方程式の解の算出に係数 U を使用すると、ゼロ除算が発生する。

?dttrsv

?dttrf によって行われた LU 因子分解を使用して、一般三重対角行列を係数行列とする連立 1 次方程式を解く。

構文

```
call sdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
call ddttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
call cdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
call zdttrsv(uplo, trans, n, nrhs, dl, d, du, b, ldb, info)
```

説明

このルーチンは、次の連立 1 次方程式のいずれかを解く。

$$LX = B, \quad L^T X = B, \quad \text{または} \quad L^H X = B$$

$$UX = B, \quad U^T X = B, \quad \text{または} \quad U^H X = B$$

[?dttrf](#) による LU 因子分解で得られた三重対角行列 A の係数

入力パラメーター

<code>uplo</code>	CHARACTER*1。 L と U のどちらを解くのか指定する。
<code>trans</code>	CHARACTER。'N'、'T'、または 'C' のいずれかでなければならない。 方程式の形式を指定する。 <code>trans = 'N'</code> の場合、 $AX = B$ を X について解く (転置なし)。 <code>trans = 'T'</code> の場合、 $A^T X = B$ を X について解く (転置)。 <code>trans = 'C'</code> の場合、 $A^H X = B$ を X について解く (共役転置)。
<code>n</code>	INTEGER。行列 A の次数。 $n \geq 0$
<code>nrhs</code>	INTEGER。右辺の数、すなわち、行列 B の列数 ($nrhs \geq 0$)。
<code>dl, d, du, b</code>	REAL (sdttrsv の場合) DOUBLE PRECISION (ddttrsv の場合) COMPLEX (cdttrsv の場合) COMPLEX*16 (zdttrsv の場合) 配列、次元は $dl(n-1)$ 、 $d(n)$ 、 $du(n-1)$ 、 $b(ldb, nrhs)$ 。 配列 dl には、 A の LU 因子分解で得られた行列 L を定義する $(n-1)$ 個の乗数が格納される。 配列 d には、 A の LU 因子分解で得られた上三角行列 U の n 個の対角成分が格納される。 配列 du には、 U の最初の優対角成分の $(n-1)$ 個の成分が格納される。 配列 b には右辺の行列 B を格納する。
<code>ldb</code>	INTEGER。配列 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 X によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、実行は正常に終了したことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正であったことを示す。

?pttrsv

?pttrf によって行われた LDL^H 因子分解を使用して、対称(エルミート)正定値三重対角行列を係数行列とする連立1次方程式を解く。

構文

```
call spttrsv(trans, n, nrhs, d, e, b, ldb, info)
call dpttrsv(trans, n, nrhs, d, e, b, ldb, info)
call cpttrsv(uplo, trans, n, nrhs, d, e, b, ldb, info)
call zpttrsv(uplo, trans, n, nrhs, d, e, b, ldb, info)
```

説明

このルーチンは、次の三角法のいずれかを解く。

$$L^T X = B \text{ または } LX = B \quad (\text{実数型の場合})$$

または

$$LX = B \text{ または } L^H X = B,$$

$$UX = B \text{ または } U^H X = B \quad (\text{複素数型の場合})$$

L (複素数型の場合は U) は、以下の条件を満たすエルミート正定値三重対角行列 A のコレスキー因子である。

$$A = LDL^H \text{ (spttrf/dpttrf で計算)}$$

または

$$A = U^H DU \text{ または } A = LDL^H \text{ (cpttrf/zpttrf で計算)}$$

入力パラメーター

<i>uplo</i>	CHARACTER*1。'U' または 'L' でなければならない。 三重対角行列 A の優対角成分と劣対角成分のどちらを格納するかと、因子分解の形式を指定する。 <i>uplo</i> = 'U' の場合、 e は U の優対角成分で、 $A = U^H DU$ 。 <i>uplo</i> = 'L' の場合、 e は L の劣対角成分で、 $A = LDL^H$ 。 A が実数型の場合、2つの形式は等価である。
<i>trans</i>	CHARACTER。 連立方程式の形式を指定する。

	実数型の場合： $trans = 'N'$ の場合、 $LX = B$ (転置なし) $trans = 'T'$ の場合、 $L^T X = B$ (転置)
	複素数型の場合： $trans = 'N'$ の場合、 $LX = B$ (転置なし) $trans = 'N'$ の場合、 $LX = B$ (転置なし) $trans = 'C'$ の場合、 $U^H X = B$ (共役転置) $trans = 'C'$ の場合、 $L^H X = B$ (共役転置)
n	INTEGER。三重対角行列 A の次数。 $n \geq 0$ 。
$nrhs$	INTEGER。右辺の数、すなわち、行列 B の列数。 $nrhs \geq 0$ 。
d	REAL 配列、次元は (n) 。 ?pttrf による因子分解で得られた対角行列 D の対角成分 n 個を格納する。
e	COMPLEX 配列、次元は $(n-1)$ 。 ?pttrf による因子分解で得られた単位二重対角係数 U または L の $(n-1)$ 個の非対角成分を格納する。 $uplo$ を参照。
b	COMPLEX 配列、次元は $(ldb, nrhs)$ 。 右辺の行列 B を格納する。
ldb	INTEGER。 配列 b のリーディング・ディメンジョン。 $ldb \geq \max(1, n)$ 。

出力パラメーター

b	終了時に、解ベクトル X で上書きされる。
$info$	INTEGER。 $= 0$ の場合、正常に終了したことを示す。 < 0 の場合、 $info = -i$ は i 番目の引数の値が不正だったことを示す。

?steqr2

暗黙的 QL 法または QR 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。

構文

```
call ssteqr2(compz, n, d, e, z, ldz, nr, work, info)
call dsteqr2(compz, n, d, e, z, ldz, nr, work, info)
```

説明

このルーチンは、LAPACK ルーチン [?stegr](#) の修正版である。ルーチン `?stegr2` は、暗黙的 QL 法または QR 法を使用して、対称三重行列の固有値をすべて計算し、さらにオプションで、固有ベクトルも計算する。`?stegr2` は、`?stegr2` を実行する各 ScaLAPACK プロセスが分散行列 Q を更新できるように、`?stegr` を修正したものである。`?stegr2` の適切な使用方法については、ScaLAPACK ルーチン [p?syev](#) を参照。

入力パラメーター

<code>compz</code>	CHARACTER*1。'N' または 'I' でなければならない。 <code>compz = 'N'</code> の場合、固有値のみを計算する。 <code>compz = 'I'</code> の場合、三重対角行列 T の固有値と固有ベクトルを計算する。 このサブルーチンを実行する前に、 p?laset または ?laset で z を単位行列に初期化しなければならない。
<code>n</code>	INTEGER。行列 T の次数。 $n \geq 0$
<code>d,e,work</code>	REAL (単精度の場合) DOUBLE PRECISION (倍精度の場合) 配列: d には T の対角成分を格納する。 d の次元は、 $\max(1, n)$ 以上でなければならない。 e には、 T の $(n-1)$ 個の劣対角成分を格納する。 e の次元は、 $\max(1, n-1)$ 以上でなければならない。 $work$ は、ワークスペース配列。 $work$ の次元は $(1, 2*n-2)$ 。 <code>compz = 'N'</code> の場合、 $work$ は参照されない。
<code>z</code>	(ローカル) REAL (<code>sstegr2</code> の場合) DOUBLE PRECISION (<code>dstegr2</code> の場合) 配列、グローバル次元は (n, n) 、ローカル次元は (ldz, nr) 。 <code>compz = 'V'</code> の場合、 z は三重対角形式への縮退に使用された直交行列を格納する。
<code>ldz</code>	INTEGER。配列 z のリーディング・ディメンジョン。次の制約がある。 $ldz \geq 1$ $ldz \geq \max(1, n)$ (固有ベクトルを計算する場合)
<code>nr</code>	INTEGER。 $nr = \max(1, \text{numroc}(n, nb, \text{myprow}, 0, \text{nprocs}))$ <code>compz = 'N'</code> の場合、 nr は参照されない。

出力パラメーター

<code>d</code>	REAL 配列、次元は (n) (<code>sstegr2</code> の場合) DOUBLE PRECISION 配列、次元は (n) (<code>dstegr2</code> の場合) 終了時に、 <code>info = 0</code> の場合、固有値が昇順で格納される。 <code>info</code> も参照のこと。
----------------	--

- e* REAL 配列、次元は $(n-1)$ (*ssteqr2* の場合)
DOUBLE PRECISION 配列、次元は $(n-1)$ (*dsteqr2* の場合)
終了時に、*e* は削除される。
- z* (ローカル)
REAL (*ssteqr2* の場合)
DOUBLE PRECISION (*dsteqr2* の場合)
配列、グローバル次元は (n, n) 、ローカル次元は (ldz, nr) 。
終了時に、*info* = 0 の場合、*compz* = 'V' ならば、元の対称行列の正規直交固有ベクトルが *z* に格納され、*compz* = 'I' ならば、対称三重対角行列の正規直交固有ベクトルが *z* に格納される。
compz = 'N' の場合、*z* は参照されない。
- info* INTEGER。
info = 0 の場合、正常に終了したことを示す。
info < 0 の場合、*info* = -*i* は *i* 番目の値が不正だったことを示す。
info > 0 の場合、このアルゴリズムで $30n$ 回の処理を繰り返した結果、すべての固有値を見つけられなかった。*info* = *i* の場合、*e* の *i* 個の成分が、ゼロに収束しなかった。終了時に、*d* と *e* に、直交性が元の行列に近似している対称三重行列の成分を格納する。

ユーティリティー関数とルーチン

このセクションでは、ScaLAPACK のユーティリティー関数とルーチンについて説明する。次の表はこれらのルーチンについてまとめたものである。

表 7-2 ScaLAPACK ユーティリティー関数とルーチン

ルーチン名	データ型	説明
p?labad	s, d	指数範囲が極めて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。
p?lachkieee	s, d	IEEE 標準機能に対して簡単なチェックを実行する (C インターフェイス関数)。
p?lamch	s, d	浮動小数点演算に対するマシン・パラメーターを決定する。
p?lasnbt	s, d	浮動小数演算の符号ビットの位置を計算する。(C インターフェイス関数)。
pxerbla		ScaLAPACK ルーチンから呼び出されるエラー処理ルーチン。

p?labad

指数範囲が極めて大きい場合にアンダーフローおよびオーバーフローしきい値の平方根を返す。

構文

```
call pslabad(ictxt, small, large)
call pdlabad(ictxt, small, large)
```

説明

このルーチンは、アンダーフローとオーバーフローに対して [p?lamch](#) によって計算された値を入力として受け取り、*large* のログが十分に大きい場合にそれら値のそれぞれの平方根を返す。このサブルーチンは、Cray のような大きな指数範囲を持つマシンを判別するために使用し、アンダーフローとオーバーフローのリミット値を [p?lamch](#) で計算された値の平方根に再定義する。Cray に見られるような上半分の指数範囲における計算能力が弱い場合でも [p?lamch](#) は補正を行わないため、このサブルーチンが必要となる。

さらに、このルーチンは異種混合のコンピューティング・ネットワークをサポートするために、これらの値の最小化および最大化をグローバルに実行する。

入力パラメーター

ictxt (グローバル) INTEGER。
計算を行う BLACS コンテキスト・ハンドル。

small (ローカル)。
REAL PRECISION ([pslabad](#) の場合)
DOUBLE PRECISION ([pdlabad](#) の場合)
[p?lamch](#) によって計算されたアンダーフローしきい値。

large (ローカル)。
 REAL PRECISION (*pslabad* の場合)
 DOUBLE PRECISION (*pdlabad* の場合)
p?lamch によって計算されたオーバーフローしきい値。

出力パラメーター

small (ローカル)。
 終了時に、 $\log_{10}(\textit{large})$ が十分に大きい場合、*small* の平方根で上書きされ、そうでない場合は変更されない。

large (ローカル)。
 終了時に、 $\log_{10}(\textit{large})$ が十分に大きい場合、*large* の平方根で上書きされ、そうでない場合は変更されない。

p?lachkieee

IEEE 標準機能に対して簡単なチェックを実行する (C インターフェイス関数)。

構文

```
void pslachkieee(int *isieee, float *rmax, float *rmin);
void pdlachkieee(int *isieee, float *rmax, float *rmin);
```

説明

このルーチンは、簡単なチェックを実行し、IEEE 標準機能が実装されていることを確認する。一部の実装では、p?lachkieee が返されない場合がある。

すべての引数は参照による呼び出しであるため、このルーチンは Fortran コードから直接呼び出すことができる。

これは ScaLAPACK の内部サブルーチンであり、引数の妥当性は確認されない。

入力パラメーター

rmax (ローカル)。
 REAL (*pslachkieee* の場合)
 DOUBLE PRECISION (*pdlachkieee* の場合)
 オーバーフローしきい値 (= ?lamch('O'))。

rmin (ローカル)。
 REAL (*pslachkieee* の場合)
 DOUBLE PRECISION (*pdlachkieee* の場合)
 アンダーフローしきい値 (= ?lamch('U'))。

出力パラメーター

isieee (ローカル) INTEGER。
 終了時に、*isieee* = 1 の場合、依存するすべての IEEE 標準機能が実装されていることを意味する。
 終了時に、*isieee* = 0 の場合、依存する IEEE 標準機能のいくつかの実装されていないことを意味する。

p?lamch

浮動小数点演算に対するマシン・パラメーターを決定する。

構文

```
val = pslamch(ictxt, cmach)
```

```
val = pdlamch(ictxt, cmach)
```

説明

この関数は、単精度のマシン・パラメーターを決定する。

入力パラメーター

ictxt (グローバル) INTEGER。計算を行う BLACS コンテキスト・ハンドル。

cmach (グローバル) CHARACTER*1。
 ルーチン p?lamch が返す値を指定する。
 'E' または 'e' の場合、p?lamch := eps
 'S' または 's' の場合、p?lamch := sfmin
 'B' または 'b' の場合、p?lamch := base
 'P' または 'p' の場合、p?lamch := eps*base
 'N' または 'n' の場合、p?lamch := t
 'R' または 'r' の場合、p?lamch := rnd
 'M' または 'm' の場合、p?lamch := emin
 'U' または 'u' の場合、p?lamch := rmin
 'L' または 'l' の場合、p?lamch := emax
 'O' または 'o' の場合、p?lamch := rmax,

ここで、

eps = 相対マシン精度。

sfmin = 1/sfmin がオーバーフローしないような安全な最小値。

base = マシンの基数。

prec = eps*base

t = 仮数における (基数) 桁数。

rnd = 丸めが追加で発生した場合 1.0、それ以外では 0.0。

emin = (緩やかに) アンダーフローを起こす前の最小指数。

rmin = アンダーフローしきい値 - $\text{base}^{\text{emin}-1}$

emax = オーバーフローを起こす前の最大指数

rmax = オーバーフローしきい値 - $(\text{base}^{\text{emax}}) * (1 - \text{eps})$.

出力パラメーター

`val` 関数の戻り値。

p?lasnbt

浮動小数演算の符号ビットの位置を計算する。
(C インターフェイス関数)。

構文

```
void pslasnbt(int *ieflag);
void pdlasnbt(int *ieflag);
```

説明

このルーチンは、単精度 / 倍精度の浮動小数点値の符号ビットの位置を探す。このルーチンは IEEE 演算を仮定する。したがって、符号ビットの可能性のあるビット 32 (単精度の場合) またはビット 32 とビット 64 (倍精度の場合) のみテストする。sizeof(int) は 4 バイトとみなされる。

コンパイル・タイム・フラグ (NO_IEEE) が、マシンに IEEE 演算がないことを示す場合、`ieflag=0` が返される。

出力パラメーター

`ieflag` INTEGER。
このフラグは、任意の単精度 / 倍精度の浮動小数点値の符号ビットの位置を探す。
コンパイル・タイム・フラグ (NO_IEEE) が、マシンに IEEE 演算がないことを示す場合、または sizeof(int) が 4 バイトでない場合、`ieflag=0`。`ieflag=1` は、単精度ルーチンの場合の符号ビットがビット 32 であることを示す。
倍精度ルーチンの場合：
`ieflag=1` は、符号ビットがビット 32 (ビッグ・エンディアン) であることを示す。
`ieflag=2` は、符号ビットがビット 64 (リトル・エンディアン) であることを示す。

pxerbla

ScaLAPACK ルーチンから呼び出されるエラー処理ルーチン。

構文

```
call pxerbla( ictxt, srname, info )
```

説明

このルーチンは、ScaLAPACK ルーチンのエラー処理ルーチンである。ScaLAPACK ルーチンで不正な入力パラメーターの値があったときに呼び出される。

メッセージを出力する。プログラムの実行は停止しない。ScaLAPACK ドライバーおよび計算ルーチンに対して、`pxerbla` の呼び出しに続き、`RETURN` 文が生成される。高レベルの呼び出しルーチンに制御が返され、プログラムをどうするのかはユーザーによって決められる。ただし、特殊なローレベル ScaLAPACK ルーチン (レベル 2 の計算ルーチンと等価な補助ルーチン) では、`pxerbla()` を呼び出した直後に、プログラムの実行を停止する `BLACS_ABORT()` が呼び出される。これは、このレベルでエラーから計算を復元することは不可能なためである。

ScaLAPACK ルーチンから戻った際に、`info` の非ゼロ値を確認するとよい。LAPACK を導入する場合に、システム固有の例外処理機能呼び出すのであれば、このルーチンの変更を考慮するとよい。

入力パラメーター

<code>ictxt</code>	(グローバル) INTEGER。 演算のグローバル・コンテキストを示す BLACS コンテキスト・ハンドル。コンテキストそのものがグローバル。
<code>srname</code>	(グローバル) CHARACTER*6。 <code>pxerbla</code> を呼び出したルーチンの名前。
<code>info</code>	(グローバル) INTEGER。 呼び出しルーチンのパラメーター・リストにおいて、不正なパラメーターの位置。

スパース・ソルバー・ルーチン

8

インテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) には、ユーザーが直接呼び出すことができる、実数または複素数、対称、構造対称または非対称、正定値、不定値またはエルミートのスパース連立線形方程式を解くための線形スパース・ソルバー・ソフトウェアが用意されている。

インテル MKL のスパース・ソルバー・サブルーチンを使用するために必要な用語と概念は、「[線形ソルバーの基礎](#)」に記述されている。線形スパースソルバーおよびスパース行列 (疎行列) の格納スキームについて熟知している場合は、これらのセクションを省略し、直接、インターフェイスについて記述されているセクションに進んでよい。この後の節では、直接法スパースソルバー PARDISO* について説明する。続いて、ステップ・バイ・ステップの解法を用いた、いくつかのインテル MKL ルーチンからなる 2 つの代替インターフェイス ([直接法スパースソルバー](#)と[反復法スパースソルバー](#)) について説明する。

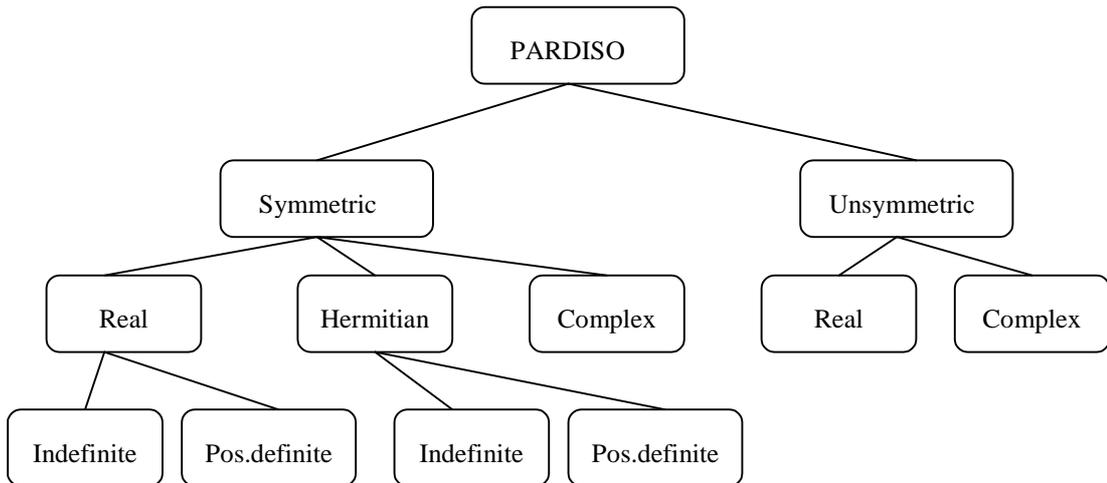
PARDISO - 並列化対応直接法スパース・ソルバー・インターフェイス

このセクションでは、PARDISO として知られる、メモリー共有型マルチプロセッシング並列化直接法スパースソルバーについて説明する。インターフェイスは Fortran で記述されているが、サポート対象のコンパイラーとオペレーティング・システムで使用される Fortran のパラメーターの受け渡しと命名規則に従うことにより、C プログラムから呼び出すこともできる。PARDISO で使用されているアルゴリズムについての説明およびソルバーについての詳細は、<http://www.computational.unibas.ch/cs/scicom> (英語) を参照のこと。

PARDISO パッケージは、共有メモリー型マルチプロセッサ上で大規模なスパース対称連立線形方程式とスパース非対称連立線形方程式を解くことができる、高性能で、安定した、メモリー効率が良く、使いやすいソフトウェアである。ソルバーは、レベル 3 BLAS の left-looking および right-looking スーパーノード手法 [[Schenk00-2](#)] を組み合わせて使用する。逐次および並列のスパース数値因子分解の性能を向上させるため、アルゴリズムはレベル 3 BLAS のアップデート版に基づき、パイプライン化並列処理は left-looking および right-looking スーパーノード手法が組み合わされている [[Schenk00](#)]、[\[Schenk01\]](#)、[\[Schenk02\]](#)、[\[Schenk03\]](#)。並列ピボット演算法は、因子分解中の数値計算の安定性とスケラビリティを両立させるために、完全なスーパーノード・ピボット演算を行うことができる。大規模な問題サイズの場合、並列アルゴリズムのスケラビリティは共有メモリー型マルチプロセッシング・アーキテクチャーにはほとんど依存せず、8 個のプロセッサを使用して 7 倍まで高速化されることが実証されている。

図 8-1 に示されるように、PARDISO は広範囲のスパース行列をサポートしており、共有メモリー型マルチプロセッシング・アーキテクチャーで、実数または複素数、対称、構造対称または非対称、正定値、不定値またはエルミートのスパース連立線形方程式を解くことができる。

図 8-1 PARDISO で解くことができるスパース行列



PARDISO インターフェイス・ルーチンを使用して連立線形方程式を解くコードの例は、付録 C の「[PARDISO のコード例](#)」のセクションを参照のこと。

pardiso

複数の右辺を持つスパース連立線形方程式の解を計算する。

構文

Fortran:

```
call pardiso(pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
            perm, nrhs, iparm, msglvl, b, x, error)
```

C:

```
pardiso(pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, perm, &nrhs,
        iparm, &msglvl, b, x, &error);
```

(“pardiso“ の後の下線は、OS とその OS 用のコンパイラーの規則によって、必須である場合と必須でない場合がある)。

インターフェイス:

```

SUBROUTINE pardiso(pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
                  perm, nrhs, iparm, msglvl, b, x, error)

INTEGER*4 pt(64)
INTEGER*4 maxfct, mnum, mtype, phase, n, nrhs, error,
          ia(*), ja(*), perm(*), iparm(*)
REAL*8 a(*), b(n,nrhs), x(n,nrhs)
    
```

注) 上記のインターフェイスは、32ビット・アーキテクチャー用である。64ビット・アーキテクチャーの場合は、変数 `pt(64)` を `INTEGER*8` 型で定義する。

説明

PARDISO は、複数の右边を持つスパース連立線形方程式の解を計算する。

$$AX = B$$

右边は複数列あり、 LU 、 LDL または LL^T 並列因子分解を用いる。 A は $n \times n$ の行列、 X と B は $n \times nrhs$ の行列である。PARDISO は、入力行列 A の構造に応じて、次の解析手順を実行する。

対称行列: 最初に METIS パッケージ [Karypis98] (インテル MKL に含まれている) から最小次数アルゴリズム [Liu85] または Nested Dissection アルゴリズムに基づいて対称非零要素低減置換 P を計算し、次に、対称正定値行列には $PAP^T = LL^T$ 、対称不定値行列には $PAP^T = LDL^T$ の並列 left-looking および right-looking コレスキー因子分解 [Schenk00-2] を計算する。対称不定値行列には部分ピボット演算または 1×1 および 2×2 の Bunch-Kaufman ピボット演算が使用され、 X の近似値は前方/後方代入と反復改善法によって得られる。

数値的に許容される 1×1 および 2×2 ピボットが対角スーパーノード・ブロック内に見つからない場合は常に、係数行列は摂動となる。反復改善法の 1 または 2 パスが摂動の効果を修正するために必要となる。この反復改善法によるピボット演算の制限の概念は、高度な不定値対称連立線形方程式で効果的である。さらに、この方法の精度は、完全なスパースピボット技法を使用する直接因子分解法 [Schenk04] と同程度に正確な異なるアプリケーション領域からの大きな行列のセット用である。ピボット演算の精度を改善する別の可能性は、対称重み付けマッチング・アルゴリズムを使用する方法である。これらの方法は、摂動が対角線に近い場合、因子分解プロセスがより許容されるピボットを識別し、より少数のピボット摂動を続けることを許可する、係数行列 A の大きな成分を識別する。これらの方法は、最大重み付けマッチングに基づいて、より完全なピボット計算技法を使用する代案への相補的な方法における要因の質を改善する。

慣性も実対称不定値行列用に計算される。

構造対称行列: 最初に対称非零要素低減置換 P を計算し、次に、 $PAP^T = QLU^T$ の並列数値因子分解を計算する。スーパーノードで部分ピボット演算が使用され、 X の近似値は前方/後方代入と反復改善法によって得られる。

非対称行列: 最初に、大きな値を持つ要素を対角に移すために非対称置換 P_{MPS} とスケールリング行列 D_r と D_c を計算する。これは数値因子分解の信頼性を大きく向上させる [Duff99]。次に、行列 $P_{MPS}A + (P_{MPS}A)^T$ に基づいて非零要素低減置換 P を計算し、スーパーノード・ピボット行列 Q および R を使用して並列数値因子分解 $QLUR = PP_{MPS}D_rAD_cP$ を計算する。因子分解アルゴリズムを実行していき、このピボット

ト演算法ではスーパーノードを因子分解できなくなったら、[Li99]のような摂動によるピボット演算法を使用する。ピボットの大きさは $\varepsilon = \alpha \cdot \|A_2\|_\infty$ の定数しきい値でテストされる。ここで、 ε は計算機の精度、 $A_2 = PP_{MPS}D_rAD_c$ と $\|A_2\|_\infty$ はスケーリングと置換が行われた行列 A の無限ノルムである。したがって、消去中に検出された小ピボットはすべて $\text{sign}(l_{ij}) \cdot \varepsilon \cdot \|A_2\|_\infty$ と設定される。ピボットが小さくなりすぎることを防止することで、数値計算が安定する。失敗が多いと本質的に役に立たない因子分解が返されるが、大規模な行列で対角成分が変更されることは稀である。このため、一般に、このピボット演算で得られる因子分解は不正確で、反復改善法が必要となる。

非対称連立線形方程式を解くための直接法と反復法の前処理： 過渡現象シミュレーション用の線形方程式を解くプロセスを加速するために、直接法と反復法を組み合わせることもできる [Som89]。スパースソルバーの多くのアプリケーションでは、非零係数行列の値が徐々に変化するが、スパースパターンが同一である方程式を解く必要がある。これらのアプリケーションでは、ソルバーの解析フェーズは一度しか実行されないため、数値因子分解がシミュレーションで最も時間を費やす重要なステップとなる。PARDISO は、最初の系に数値因子分解 $A = LU$ を使用して、得られた L と U を次の前処理付きクリロフ部分空間反復法のステップで適用する。反復計算が収束しない場合、ソルバーは自動的に数値因子分解に戻る。この手法は、PARDISO における非対称行列に適用できる。ユーザーは、1つの入力パラメーターのみを使用して、この手法を選択できる。詳細は、パラメーターの説明 (*iparm(4)*, *iparm(20)*) を参照のこと。

PARDISO のスパースデータ格納形式： 「[スパース行列の格納形式](#)」で記述されているように、*ja* が *columns*、*ia* が *rowIndex*、*a* が *values* を表す。対称行列および構造対称行列の場合は、行ごとに列番号 *ja* を増加して順序付けする必要がある。また、行ごとに対角成分が必要である。非対称行列の場合は、対角成分は必要ない。

PARDISO では、解析とシンボリック因子分解、数値因子分解、反復改善法を含む前方/後方代入、すべての内部ソルバーメモリーを解放する終了化処理の4つの処理を実行できる。入力データ構造が呼び出しでアクセスされない場合、NULL ポインターまたは有効なアドレスを引数のプレースホルダーとして渡すことができる。

入力パラメーター

pt INTEGER*4 (32 ビット のオペレーティング・システムの場合)
 INTEGER*8 (64 ビット のオペレーティング・システムの場合)。
 配列、DIMENSION は (64)。
 ソルバーの内部データのアドレスポインター。アドレスはソルバーに渡され、関連する内部メモリーの管理はすべて、このポインターにより行われる。



注： *pt* は、64 の成分を持つ整数配列である。PARDISO を最初に呼び出すとポインターがゼロに初期化されるが、これは非常に重要である。深刻なメモリーリークが発生する可能性があるため、呼び出した後は決してポインターを変更するべきではない。整数の長さは32ビットのオペレーティング・システムでは4バイト、64ビットのオペレーティング・システムでは8バイトでなければならない。

maxfct INTEGER。
 ユーザーがメモリー中に同時に保持する同じ非ゼロのスパース構造を持つ係数の最大数。同じ非ゼロの構造を持つ複数の異なる因子分解を

同時にソルバーの内部データ管理に格納することもできる。多くのアプリケーションで、この値は1である。

PARDISOは、行列のスパースパターンが同じである複数の行列を処理することができ、これらの行列の係数を同時に格納することが可能である。スパース構造が異なる行列は、それぞれ異なるメモリー・アドレス・ポインター (*pt*) でメモリーに保持できる。

- mnum* INTEGER。
解フェーズの実際の行列。このスカラーで、因子分解する行列を定義できる。値は、 $1 \leq mnum \leq maxfct$ でなければならない。
多くのアプリケーションで、この値は1である。
- mtype* INTEGER。
このスカラー値は行列のタイプを定義する。PARDISO ソルバーは次の行列をサポートしている。
- mtype* = 1 実構造対称行列
 - = 2 実対称正定値行列
 - = -2 実対称不定値行列
 - = 3 複素構造対称行列
 - = 4 複素エルミート正定値行列
 - = -4 複素エルミート不定値行列
 - = 6 複素対称行列
 - = 11 実非対称行列
 - = 13 複素非対称行列
- このパラメーターはピボット法に影響を与える。
- phase* INTEGER。
ソルバーの実行を制御する。
2桁の整数 *ij* ($10i + j$, $1 \leq i \leq 3$, $i < j \leq 3$ 。通常の実行モードの場合)。*i* 桁は実行の開始フェーズ、*j* 桁は終了フェーズを示す。PARDISO では次の実行フェーズがある。
- フェーズ 1: 非零要素低減解析とシンボリック因子分解
 - フェーズ 2: 数値因子分解
 - フェーズ 3: 順方向、逆方向算出 (反復改善法を含む)
 - 終了 / メモリー解放フェーズ ($phase \leq 0$)
- 前のフェーズからの計算情報が前のルーチンへの呼び出しにある場合、任意のフェーズで実行を開始できる。*phase* パラメーターには次の値を指定できる。

<i>phase</i>	ソルバーの実行ステップ
11	解析
12	解析、数値因子分解
13	解析、数値因子分解、解の算出、反復改善
22	数値因子分解
23	数値因子分解、解の算出、反復改善
33	解の算出、反復改善
0	L と U 行列の番号 <i>mnum</i> の内部メモリーの解放
-1	すべての行列のメモリー解放

<i>n</i>	INTEGER。 方程式の数。これはスパース連立線形方程式 $AX = B$ における方程式の数である。次の制約がある。 $n > 0$ 。
<i>a</i>	REAL/COMPLEX。 配列。 <i>ja</i> 内のインデックスに対応する係数行列 <i>A</i> の非ゼロの値を格納する。 <i>a</i> のサイズは、 <i>ja</i> のサイズと同じで、係数行列は実数または複素数のいずれかである。行列は、行ごとに <i>ja</i> の値が増加する行圧縮形式で格納されなければならない。詳細は、「 スパース行列の格納形式 」の <i>values</i> 配列の説明を参照のこと。



注： 行列 *A* の各行の非ゼロは、昇順で格納されなければならない。対称行列または構造対称行列の場合、対角成分を利用でき、行列に格納されることも重要である。行列が対称の場合、配列 *a* は、因子分解のフェーズ、三角分解および反復改善フェーズでのみアクセスされる。非対称行列は、解のプロセスにおけるすべてのフェーズでアクセスされる。

<i>ia</i>	INTEGER。 配列、次元は $(n+1)$ 。 $i \leq n$ の場合、 <i>ia</i> (<i>i</i>) は行圧縮形式で格納された配列 <i>ja</i> 内にある <i>i</i> 行の最初の列インデックスを指す。つまり、 <i>ia</i> (<i>i</i>) は、 <i>A</i> の <i>i</i> 行からの最初の非ゼロの成分を格納する配列 <i>a</i> にある成分のインデックスを提供する。最後の成分 <i>ia</i> ($n+1$) は、 <i>A</i> 内の非ゼロの数に 1 を加えたものに等しいとみなされる。詳細は、「 スパース行列の格納形式 」の <i>rowIndex</i> 配列の説明を参照のこと。 配列 <i>ia</i> は、解のプロセスのフェーズでアクセスされる。行番号と列番号は、1 から始まることに注意。
<i>ja</i>	INTEGER。 配列。 <i>ja</i> (*) は、行圧縮形式で格納されているスパース行列 <i>A</i> の列インデックスを含む。各行のインデックスは、昇順で格納する。 配列 <i>ia</i> は、解のプロセスのすべてのフェーズでアクセスされる。対称行列および構造対称行列の場合、ゼロ対角成分は <i>a</i> と <i>ja</i> の非ゼロのリストにも格納される。対称行列の場合、ソルバーは、「 スパース行列の格納形式 」の <i>columns</i> 配列のように、方程式の上三角部分だけを必要とする。
<i>perm</i>	INTEGER。 配列、次元は (n) 。サイズ n の置換ベクトルを格納する。 配列 <i>perm</i> は、次のように定義される。 <i>A</i> を元の行列とし、 $B = PAP^T$ を置換行列とする。 <i>A</i> の <i>i</i> 行(列)は、 <i>B</i> の <i>perm</i> (<i>i</i>) 行(列)である。 配列の番号付けは 1 を起点とし、置換を説明する。 独自の非零要素低減順序付けをソルバーに適用することができる。置換ベクトル <i>perm</i> は、 <i>iparm</i> (5) = 1 の場合にのみアクセスされる。
<i>nrhs</i>	INTEGER。 解を求める右辺の数。
<i>iparm</i>	INTEGER。 配列、次元は (64)。この配列はさまざまなパラメーターを PARDISO に渡すために使用され、ソルバーの実行後に有益な情報を返すためにも使用される。

$iparm(1) = 0$ の場合、PARDISO は $iparm(1)$ 、および $iparm(4)$ から $iparm(64)$ までをデフォルト値で埋め、それらの値を使用する。
 $iparm(3)$ のデフォルト値はなく、この値は $iparm(1)$ が 0 でも 1 でもユーザーが指定しなければならないことに注意。

$iparm$ 配列の個々の成分を以下に説明する (一部の成分は「[出力パラメーター](#)」のセクションで説明)。

$iparm(1)$ - デフォルト値の使用。

$iparm(1) = 0$ の場合、 $iparm(2)$ および $iparm(4)$ から $iparm(64)$ まではデフォルト値で埋められる。それ以外の場合、 $iparm(2)$ から $iparm(64)$ までの $iparm$ のすべての値をユーザーが指定しなければならない。

$iparm(2)$ - 非零要素低減順序付け。

$iparm(2)$ は、入力行列に対する非零要素低減順序付けを制御する。
 $iparm(2)$ が 0 の場合、最小次数アルゴリズムが適用される [Li99]。
 $iparm(2)$ が 2 の場合、ソルバーは METIS パッケージからの Nested Dissection アルゴリズムを使用する [Karypis98]。
 $iparm(2)$ のデフォルト値は 2 である。

$iparm(3)$ - プロセッサ数。

$iparm(3)$ には、並列実行の際に利用できるプロセッサ数を格納する。プロセッサ数は OpenMP 環境変数 OMP_NUM_THREADS と等しくなければならない。



注意: ユーザーが明示的に OMP_NUM_THREADS を設定していない場合、この値はオペレーティング・システムによりシステムの最大プロセッサ数に設定される。そのため、OMP_NUM_THREADS を明示的に設定して、ソルバーの並列実行を制御することを推奨する。指定されたプロセッサ数よりも少ない数のプロセッサしか利用できない場合、実行速度は向上せず、遅くなることもある。

$iparm(3)$ のデフォルト値はない。

$iparm(4)$ - 前処理付き CGS。

このパラメーターは、非対称行列または構造対称行列の前処理付き CGS [Sonn89] および対称行列の共役勾配を制御する。

$iparm(4)$ は次の形式を持つ。

$$iparm(4) = 10 * L + K$$

値 K と L の意味は次のとおりである。

値 K :

K の値	説明
0	$phase$ で必要なため、因子分解が常に行われる。
1	CGS 反復は LU の計算を置換する。前処理は、同一のスパースパターンに必要な一連の方法における、前のステップ (最初のステップまたは失敗を含む最後のステップ) で計算された LU である。
2	対称行列の CG 反復は LU の計算を置換する。前処理は、同一のスパースパターンに必要な一連の方法における、前のステップ (最初のステップまたは失敗を含む最後のステップ) で計算された LU である。

値 L :

クリロフ部分空間反復法の停止条件を制御する。

$\epsilon_{\text{CGS}} = 10^{-L}$ は次の停止条件で使用される。

$$\|dx_i\| / \|dx_1\| < \epsilon_{\text{CGS}}$$

$\|dx_i\| = \|(LU)^{-1}r_i\|$ で、 r_i は前処理付きクリロフ部分空間反復法における反復 i の際の誤差である。

方針: 因子分解時間の半分を費やす前に反復計算は収束するという予想から、反復最大回数は 150 に固定される。中間収束率および誤差の偏位が確認され、反復プロセスを終了することができる。

$phase = 23$ の場合、クリロフ部分空間反復法が失敗し、対応する直接解が返されるようなケースでは、 A の因子分解は自動で再計算される。それ以外では前処理付きクリロフ部分空間反復法からの解が返される。 $phase = 33$ を使用した場合、クリロフ部分空間反復法の停止条件を満たさないとエラーメッセージ ($error = 4$) が表示される。このエラーについての詳細は、 $iparm(20)$ から得ることができる。

デフォルトは $iparm(4) = 0$ で、上級ユーザーでない限り、その他の値を使用しないことを推奨する。 $iparm(4)$ は 0 以上でなければならない。

例:

$iparm(4)$	説明
31	非対称行列の場合は停止条件が 10^{-3} の LU-前処理付き CGS 反復法
61	非対称行列の場合は停止条件が 10^{-6} の LU-前処理付き CGS 反復法
62	対称行列の場合は停止条件が 10^{-6} の LU-前処理付き CGS 反復法

$iparm(5)$ - ユーザー置換。

このパラメーターは、統合 MMD (Multiple-Minimum Degree) アルゴリズムや Nested Dissection アルゴリズムの代わりにユーザー独自の非零要素低減置換を使用するかどうかを制御する。

このオプションは、順序付けアルゴリズムのテストや特殊なアプリケーション問題 (例えば、ゼロの対角成分を $end\ PAP^T$ に移動するなど) にコードを適応させる際に役立つ。置換の定義については、 $perm$ パラメーターの説明を参照のこと。

$iparm(5)$ のデフォルト値は 0 である。

$iparm(6)$ - x への解の格納。

$iparm(6)$ が 0 (デフォルト値) の場合、配列 x は解を格納し、 b の値は変わらない。 $iparm(6)$ が 1 の場合、ソルバーは右辺 b の解を格納する。

配列 x が常に使用されることに注意。 $iparm(6)$ のデフォルト値は 0 である。

$iparm(8)$ - 反復改善ステップ。

解の算出および反復改善ステップでは、 $iparm(8)$ はソルバーが実行する反復改善ステップの最大数に設定されなければならない。ソルバーは、反復改善の $iparm(8)$ ステップの絶対値を超えて実行されることはなく、後退誤差の点から満足できるレベルの解の精度に達した場合に停止する。

$iparm(8) < 0$ の場合、誤差の蓄積には拡張精度の実数データ型と複素数データ型を使用することに注意。摂動ピボットにより、反復改善法

(*iparm*(8)=0 とは関係なく) が行われ、実行された反復番号が *iparm*(7) に報告される。

摂動ピボットが数値因子分解中に取得され、*iparm*(8) が 0 だった場合、ソルバーは自動的に反復改善法の 2 つのステップを実行する。

実行された反復改善ステップの数は *iparm*(7) に報告される。

iparm(8) のデフォルト値は 0 である。

***iparm*(9)**

このパラメーターは将来用いることができるように予約されている。その値は、0 に設定しなければならない。

***iparm*(10) - ピボット演算摂動。**

このパラメーターは、PARDISO に対し、非対称行列 (*mtype* = 11 または *mtype* = 13) および対称行列 (*mtype* = -2, *mtype* = -4, または *mtype* = 6) の小さなピボットやゼロピボットの処理方法を指示する。これらの行列の場合、ソルバーは完全なスーパーノード・ピボット演算法を使用する。因子分解アルゴリズムを実行していき、このピボット演算法ではスーパーノードを因子分解できなくなったら、[Li99, [Schenk04] のような摂動によるピボット演算法を使用する。ピボットの大きさは、次の定数しきい値でテストされる。

$$\varepsilon = \alpha \cdot \|A_2\|_\infty$$

ここで $\varepsilon = 10^{-iparm(10)}$ 、 $\|PP_{MPS}D_rAD_cP\|_\infty$ はスケーリングと置換が行われた行列 *A* の無限ノルムである。したがって、消去中に検出された小ピボットはすべて $sign(l_{ij}) \cdot \varepsilon \cdot \|A_2\|_\infty$ と設定される。ピボットが小さくなりすぎることを防止することで、数値計算が安定する。それゆえ、小さなピボットは $\varepsilon = 10^{-iparm(10)}$ に置換される。

iparm(10) のデフォルト値は 13 であるため、非対称行列 (*mtype* = 11 または *mtype* = 13) では $\varepsilon = 10^{-13}$ となる。*iparm*(10) のデフォルト値は 8 であるため、対称不定値行列 (*mtype* = -2, *mtype* = -4, または *mtype* = 6) では $\varepsilon = 10^{-8}$ となる。

***iparm*(11) - ベクトルのスケーリング。**

PARDISO は、最大重みマッチング・アルゴリズムを用いて対角上の大きな成分を置換し、行列をスケーリングして、対角成分が 1 に等しく、非対角成分の絶対値が 1 以下になるようにする。このスケーリング手法は、非対称行列 (*mtype* = 11 または *mtype* = 13) にのみ適用される。このスケーリングは、対称重み付けマッチングが適用された (*iparm*(13)=1) 場合、対称不定値行列 (*mtype* = -2, *mtype* = -4, または *mtype* = 6) にも使用できる。

例えば、内点最適化や鞍点問題などの高度な不定値対称行列では、*iparm*(11)=1 (スケーリング) および *iparm*(13)=1 (マッチング) を使用することを推奨する。スケーリングと対称重み付けマッチングを使用する場合、ユーザーは解析フェーズ (*phase*=11) で行列 *A* の数値を指定しなければならないことに注意することも重要である。

iparm(11) のデフォルト値は、非対称行列 (*mtype* = 11 または *mtype* = 13) では 1 である。*iparm*(11) のデフォルト値は、対称行列 (*mtype* = -2, *mtype* = -4, または *mtype* = 6) では 0 である。

***iparm*(12)**

このパラメーターは将来用いることができるように予約されている。その値は、0 に設定しなければならない。

iparm(13) - (非)対称重み付けマッチングを使用した精度の改善。

PARDISO は、最大重み付けマッチング・アルゴリズムを使用して対角線に近い大きな成分を置換することができる。この手法は、因子分解法に新たなレベルの信頼性を追加し、数値因子分解中により完全なピボット計算技法を使用する代案への補足と見なすことができる。

例えば、内点最適化や鞍点問題などの高度な不定値対称行列では、*iparm(11)=1* (スケーリング) および *iparm(13)=1* (マッチング) を使用することを推奨する。スケーリングと対称重み付けマッチングを使用する場合、ユーザーは解析フェーズ (*phase=11*) で行列 *A* の数値を指定しなければならないことに注意することも重要である。

iparm(13) のデフォルト値は、非対称行列 (*mtype=11* または *mtype=13*) では 1 である。*iparm(13)* のデフォルト値は、対称行列 (*mtype=-2*、*mtype=-4*、または *mtype=6*) では 0 である。

iparm(18)

iparm(18) < 0 の場合、ソルバーは係数の非ゼロ数を報告する。

iparm(18) のデフォルト値は -1 である。

iparm(19) - 因子分解の MFlop。

iparm(19) < 0 の場合、ソルバーは行列 *A* の因子分解に必要な MFlop (10^6) を報告する。このため、順序付け時間が増加する。

iparm(19) のデフォルト値は 0 である。

iparm(21) - 対称不定値行列のピボット演算。

iparm(21) は、スパース対称不定値行列のピボット演算法を制御する。*iparm(21)* が 0 の場合、1x1 対角ピボット演算が使用される。*iparm(21)* が 1 の場合、1x1 および 2x2 Bunch-Kaufman ピボット演算が因子分解プロセス内で使用される。

例えば、内点最適化や鞍点問題などの高度な不定値対称行列では、*iparm(11)=1* (スケーリング) および *iparm(13)=1* (マッチング) を使用することも推奨する。

Bunch-Kaufman ピボット演算は、対称行列 (*mtype=-2*、*mtype=-4*、または *mtype=6*) で利用できる。

iparm(21) のデフォルト値は 0 である。

msglvl INTEGER。
メッセージレベルの情報。*msglvl=0* の場合、PARDISO は出力を生成しない。
msglvl=1 の場合、ソルバーは画面に統計情報を出力する。

b REAL/COMPLEX。
配列、次元は (*n, nrhs*)。右辺のベクトル / 行列 *B* を格納する。
b は、解フェーズでのみアクセスされる。

出力パラメーター

pt このパラメーターは内部アドレスポインターを格納する。

iparm 一部の *iparm* 値は有益な情報を報告する (例えば、係数にある非ゼロの数など)。

iparm(7) - 実行された反復改善ステップの数。

解の算出ステップ中に実際に実行された反復改善ステップの数。

iparm(14) - 摂動ピボットの数。

因子分解の後、*iparm(14)* は、*mtype = 11*、*mtype = 13*、*mtype = -2*、*mtype = -4*、または *mtype = 6* に対する消去プロセス中に摂動されたピボット数を格納する。

iparm(15) - シンボリック因子分解のピークメモリー。

パラメーター *iparm(15)* は、解析およびシンボリック因子分解フェーズにおいてソルバーが必要としたピークメモリーの総量を KB でユーザーに知らせる。この値はフェーズ 1 でのみ計算される。

iparm(16) - シンボリック因子分解の永久メモリー。

パラメーター *iparm(16)* は、因子分解フェーズと解フェーズにおける解析およびシンボリック因子分解フェーズでソルバーが必要とした永久メモリーの総量を KB でユーザーに知らせる。この値はフェーズ 1 でのみ計算される。

iparm(17) - 因子分解と解フェーズのメモリー。

パラメーター *iparm(17)* は、因子分解フェーズおよび解フェーズでソルバーが消費した倍精度のメモリー総量 (KB) をユーザーに知らせる。この値はフェーズ 2 でのみ計算される。

ソルバーが消費するピークメモリーの総量は $\max(iparm(15), iparm(16) + iparm(17))$ であることに注意。

iparm(18) - 係数の非ゼロの数。

iparm(18) < 0 の場合、ソルバーは係数の非ゼロ数を報告する。

iparm(19) - 因子分解の MFlop。

iparm(19) < 0 の場合、行列 *A* の因子分解に必要な演算回数を MFlop (10^6 回) でユーザーに返す。

iparm(20) - CG/CGS 診断。

この値は、CG/CGS 診断 (例えば、反復回数や失敗の原因など) に用いられる。

iparm(20) > 0 の場合、CGS が行われ、実行された反復回数が *iparm(20)* に報告される。

iparm(20) < 0 の場合、反復は実行されるが、CG/CGS は失敗する。

iparm(20) のエラーレポートの詳細は、次の形式をとる。

$iparm(20) = -it_cgs * 10 - cgs_error$

phase が 23 だった場合、行列 *A* に対して係数 *L*、*U* が再計算されており、因子分解が正常に行われた場合はエラーフラグの *error* はゼロである。

phase が 33 だった場合、*error = -4* はエラーを示す。

次の表は *cgs_error* の説明である。

cgs_error	説明
1	誤差の変動が大きすぎる
2	$\ dx_{\max_it_cgs/2}\ $ が大きすぎる (収束が遅い)

cgs_error	説明
3	停止条件が max_it_cgs に達しなかった
4	摂動ピボットにより反復改善法が行われた
5	因子分解がこの行列には速すぎる。iparm(4) = 0 で因子分解法を用いるほうが良い。

iparm(22) - 慣性：正の固有値の数。

パラメーター iparm(22) は、対称不定値行列の正の固有値の数を報告する。

iparm(23) - 慣性：負の固有値の数。

パラメーター iparm(23) は、対称不定値行列の負の固有値の数を報告する。

iparm(24) ~ iparm(64)

これらのパラメーターは将来用いることができるように予約されている。これらの値は、0 に設定しなければならない。

<i>b</i>	iparm(6) = 1 の場合、配列は解に置換される。
<i>x</i>	REAL/COMPLEX。 配列、次元は (n, nrhs)。iparm(6) = 0 の場合、解を格納する。 <i>x</i> は、解フェーズでのみアクセスされる。
<i>error</i>	INTEGER。 エラー・インジケータ :

error	情報
0	エラーなし
-1	入力が不整合
-2	十分なメモリーがない
-3	順序付けの問題
-4	ゼロピボット、数値因子分解または反復改善問題
-5	分類されない (内部) エラー
-6	事前順序付けに失敗 (行列タイプ 11、13 のみ)
-7	対角行列の問題

直接法スパースソルバー (DSS) インターフェイス・ルーチン

インテル MKL は、直接法スパースソルバー用の PARDISO インターフェイスに代わるインターフェイス (ここでは DSS インターフェイスと呼ぶ) をサポートしている。DSS インターフェイスは、ユーザーが呼び出すことのできるルーチングループを実装している。ステップ・バイ・ステップの解の算出プロセスで使用され、「[線形ソルバーの基礎](#)」で説明されている一般的なスキームを利用してスパース連立線形方程式を解く。このインターフェイスには、解プロセスに関連した統計を収集するルーチンや Fortran ルーチンから C ルーチンに文字列を渡す補助ルーチンも含まれている。

解のプロセスは、概念的に 6 つのフェーズに分けられる。[表 8-1](#) は、ルーチン名、フェーズごとのグループ、一般的な使用法の説明をまとめたものである。

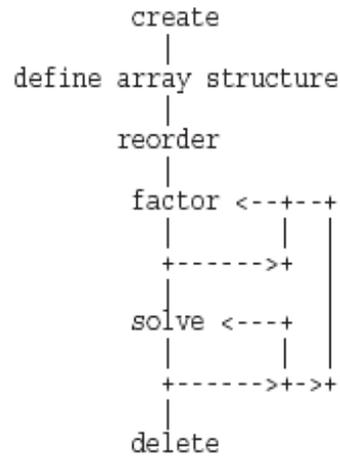
表 8-1 DSS インターフェイス・ルーチン

ルーチン	説明
dss_create	ソルバーを初期化し、ソルバーに必要な基本的なデータ構造を作成する。このルーチンは、他の DSS ルーチンが呼び出される前に呼び出される必要がある。
dss_define_structure	ソルバーに、配列内の非ゼロ成分の場所を知らせるために使用される。
dss_reorder	このルーチンは、配列の非ゼロ構造に基づいて、置換ベクトルを計算し、因子分解処理中の非零要素を低減させる。
dss_factor_real、 dss_factor_complex	実行列または複素行列の LU 、 LDT^t または LL^T 因子分解を計算する。
dss_solve_real、 dss_solve_complex	前のフェーズで計算された因子分解に基づき、連立方程式の解ベクトルを計算する。
dss_delete	解のプロセスで作成されたすべてのデータ構造を削除する。
dss_statistics	解のプロセスにおけるさまざまなフェーズについての統計を返す。順序付けに要した時間、因子分解に要した時間、求解時間、行列式、行列の慣性、因子分解中に行われた浮動小数点演算の回数などの統計を収集するために使用される。「順序付け」フェーズ後でかつ「削除」フェーズ前の解のプロセスにおける任意のフェーズで起動できる。このルーチンが起動されるフェーズに対応した適切な引数をルーチンに与える必要があることに注意。
mkl_cvt_to_null_terminated_str	Fortran ルーチンから C ルーチンに文字列を渡すために使用される。

右辺が 1 つの連立方程式の解ベクトルを求めるため、[表 8-1](#) のリスト順 (`dss_statistics` を除く。表の説明を参照のこと) でインテル MKL DSS インターフェイス・ルーチンが起動される。

ただし、一部のアプリケーションでは、複数の右辺に対して解ベクトルを作成したり、同じ非ゼロ構造で複数の行列を因子分解する必要がある。そのため、インテル MKL スパースルーチンを表とは別の順序で起動できるようにすることも必要である。[図 8-2](#) のダイアグラムは、DSS インターフェイス・ルーチンを起動する一般的な順序を示す。

図 8-2 DSS インターフェイス・ルーチンを起動する一般的な順序



DSS インターフェイス・ルーチンを使用して連立線形方程式を解くコードの例は、付録 C の「[直接法スパースソルバーのコード例](#)」のセクションを参照のこと。

インターフェイスの説明

「[メモリー割り当てとハンドル](#)」のセクションで注記したように、各 DSS ルーチンは、ハンドルと呼ばれるデータ・オブジェクトを記述する。ハンドルの宣言は言語ごとに異なるため、本書では MKL_DSS_HANDLE 型として宣言されることとする。ハンドル引数の正しい宣言方法については、「[メモリー割り当てとハンドル](#)」を参照のこと。

本書のその他すべての型については、標準 Fortran 型の INTEGER、REAL、COMPLEX、DOUBLE PRECISION、DOUBLE COMPLEX を参照のこと。

C プログラマーおよび C++ プログラマーは、Fortran 型から C/C++ 型へのマッピングについての情報が記述された「[C/C++ からのスパース・ソルバー・ルーチンの呼び出し](#)」を参照のこと。

ルーチンオプション

すべての DSS ルーチンは、整数引数 (以下、opt として参照する) を持ち、さまざまなオプションをルーチンに渡す。opt に許容される値は言語固有のヘッダーファイルで定義される (「[実装の詳細](#)」を参照)。すべてのルーチンは、[表 8-2](#) の説明のように、メッセージレベルや終了レベルの設定に使用されるオプションを受け入れる。さらに、すべてのルーチンは各 DSS ルーチンのデフォルトオプションを確立する MKL_DSS_DEFAULTS オプションを受け入れる。

表 8-2 **メッセージレベルと終了レベルの記号名**

メッセージレベル	終了レベル
MKL_DSS_MSG_LVL_SUCCESS	MKL_DSS_TERM_LVL_SUCCESS
MKL_DSS_MSG_LVL_INFO	MKL_DSS_TERM_LVL_INFO
MKL_DSS_MSG_LVL_WARNING	MKL_DSS_TERM_LVL_WARNING
MKL_DSS_MSG_LVL_ERROR	MKL_DSS_TERM_LVL_ERROR
MKL_DSS_MSG_LVL_FATAL	MKL_DSS_TERM_LVL_FATAL

メッセージレベルおよび終了レベルは、任意の DSS ルーチンの呼び出しで設定できる。ただし、いったん特定のレベルに設定されると、他の DSS ルーチンへの呼び出しで変更されるまで、そのレベルのままである。

オプションを同時に追加することによって、DSS ルーチンに複数のオプションを指定することができる。例えば、メッセージレベルをデバッグに、終了レベルをすべての DSS ルーチンのエラーに設定するには、次のようにする。

```
CALL dss_create( handle, MKL_DSS_MSG_LVL_INFO + MKL_DSS_TERM_LVL_ERROR)
```

ユーザーデータ配列

多くの DSS ルーチンは、ユーザーデータ配列を入力として受け取る。例えば、ユーザー配列は dss_define_structure ルーチンに渡され、行列内にある非ゼロの成分の位置について説明する。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL DSS ルーチンはユーザー入力配列のコピーを作成しない。



警告: ソルバールーチンに渡された後、ユーザーは、これらの配列の内容を編集することはできない。

DSS ルーチン

dss_create

ソルバーを初期化する。

構文

```
dss_create(handle, opt)
```

入力パラメーター

opt INTEGER。パラメーターを渡すオプション。デフォルト値は MKL_DSS_MSG_LVL_WARNING + MKL_DSS_TERM_LVL_ERROR。

出力パラメーター

handle MKL_DSS_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

説明

dss_create ルーチンは、ソルバーを初期化するために呼び出される。dss_create の呼び出し後に起動されるインテル MKL DSS ルーチンは、dss_create によって返されたハンドルの値を使用する。



警告: ハンドルの値を直接書き込まないこと。

戻り値

MKL_DSS_SUCCESS
MKL_DSS_INVALID_OPTION
MKL_DSS_OUT_OF_MEMORY

dss_define_structure

行列内の非ゼロ成分の位置をソルバーに知らせる。

構文

```
dss_define_structure(handle, opt, rowIndex, nRows, nCols, columns,  
                    nNonZeros)
```

入力パラメーター

<i>opt</i>	INTEGER。パラメーターを渡すオプション。行列構造のデフォルトオプションは、MKL_DSS_SYMMETRIC。
<i>rowIndex</i>	INTEGER。サイズ $\min(nRows, nCols)+1$ の配列。行列内の非ゼロ成分の場所を定義する。
<i>nRows</i>	INTEGER。行列の行数。
<i>nCols</i>	INTEGER。行列の列数。
<i>columns</i>	INTEGER。サイズ <i>nNonZeros</i> の配列。行列内の非ゼロ成分の場所を定義する。
<i>nNonZeros</i>	INTEGER。行列内の非ゼロ成分の数。

出力パラメーター

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「インターフェイスの説明」 を参照)。
---------------	---

説明

`dss_define_structure` ルーチンは、サイズが $nRows \times nCols$ の行列内の非ゼロ成分の数 *nNonZeros* をソルバーに知らせる。

現在のところ、インテル MKL DSS ソフトウェアは正方行列に対してのみ動作するので、*nRows* と *nCols* は等しくなければならない。

行列内の非ゼロ成分の位置を知らせるには、次の操作を行う。

1. オプション引数 *opt* に次の値のいずれかを指定して、行列の一般的な非ゼロの構造を定義する。

```
MKL_DSS_SYMMETRIC_STRUCTURE
MKL_DSS_SYMMETRIC
MKL_DSS_NON_SYMMETRIC
```

2. 配列 *rowIndex* と *columns* を用いて非ゼロの実際の位置を示す ([「スパース行列の格納形式」](#)を参照)。

```
MKL_DSS_SUCCESS
MKL_DSS_STATE_ERR
MKL_DSS_INVALID_OPTION
MKL_DSS_COL_ERR
MKL_DSS_NOT_SQUARE
MKL_DSS_TOO_FEW_VALUES
MKL_DSS_TOO_MANY_VALUES
```

dss_reorder

因子分解フェーズ中の非零要素を最小限に抑える置換ベクトルを計算する。

構文

```
dss_reorder(handle, opt, perm)
```

入力パラメーター

opt INTEGER。パラメーターを渡すオプション。置換型のデフォルトオプションは MKL_DSS_AUTO_ORDER。

perm INTEGER。長さ *nRows* の配列。ユーザー定義の置換ベクトル (*opt* が MKL_DSS_MY_ORDER を格納する場合にのみアクセスされる) を格納する。

出力パラメーター

handle MKL_DSS_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

説明

opt に MKL_DSS_AUTO_ORDER オプションが含まれている場合、*dss_reorder* は因子分解フェーズ中の非零要素を最小限に抑える置換ベクトルを計算する。このオプションでは、*perm* 配列はアクセスされない。

opt に MKL_DSS_MY_ORDER オプションが含まれている場合、配列 *perm* はユーザーが提供した置換ベクトルとみなされる。この場合、配列 *perm* は *nRows* 長である。ここで、*nRows* は前の [dss_define_structure](#) への呼び出しで定義された行列内の行数である。

戻り値

MKL_DSS_SUCCESS
MKL_DSS_STATE_ERR
MKL_DSS_INVALID_OPTION
MKL_DSS_OUT_OF_MEMORY

dss_factor_real、 dss_factor_complex

前に指定された位置にある行列の因子分解を計算する。

構文

```
dss_factor_real(handle, opt, rValues)  
dss_factor_complex(handle, opt, cValues)
```

入力パラメーター

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 インターフェイスの説明 」を参照)。
<i>opt</i>	INTEGER。パラメーターを渡すオプション。行列のデフォルトオプションは MKL_DSS_POSITIVE_DEFINITE。
<i>rValues</i>	DOUBLE PRECISION。サイズ <i>nNonZeros</i> の配列。行列内の実数の非ゼロ成分。
<i>cValues</i>	DOUBLE COMPLEX。サイズ <i>nNonZeros</i> の配列。行列内の複素数の非ゼロ成分。

説明

これらのルーチンは、前に [dss_define_structure](#) への呼び出しによって非ゼロの部分が指定された行列で、非ゼロの値が配列 *rValues* または *cValues* で与えられている行列の因子分解を行う。配列 *rValues* と *cValues* は前の [dss_define_structure](#) への呼び出しで定義されているように、*nNonZeros* 長とみなされる。

opt 引数は、次のいずれかのオプションを含む。

```
MKL_DSS_POSITIVE_DEFINITE、
MKL_DSS_INDEFINITE、
MKL_DSS_HERMITIAN_POSITIVE_DEFINITE、
MKL_DSS_HERMITIAN_INDEFINITE
```

rValues と *cValues* の非ゼロの値が、正定値行列、不定値行列、またはエルミート行列かによりオプションは異なる。

戻り値

```
MKL_DSS_SUCCESS
MKL_DSS_STATE_ERR
MKL_DSS_INVALID_OPTION
MKL_DSS_OPTION_CONFLICT
MKL_DSS_OUT_OF_MEMORY
MKL_DSS_ZERO_PIVOT
```

dss_solve_real、 dss_solve_complex

対応する解ベクトルを計算して、出力配列に配置する。

構文

```
dss_solve_real(handle, opt, rRhsValues, nRhs, rSolValues)
dss_solve_complex(handle, opt, cRhsValues, nRhs, cSolValues)
```

入カパラメーター

<i>handle</i>	MKL_DSS_HANDLE 型のデータ・オブジェクト (「 インターフェイスの説明 」を参照)。
<i>opt</i>	INTEGER。パラメーターを渡すオプション。
<i>nRhs</i>	INTEGER。線形方程式の右辺の数。
<i>rRhsValues</i>	DOUBLE PRECISION。サイズ <i>nRows</i> × <i>nRhs</i> の配列。実数の右辺ベクトルを格納する。
<i>cRhsValues</i>	DOUBLE COMPLEX。サイズ <i>nRows</i> × <i>nRhs</i> の配列。複素数の右辺ベクトルを格納する。

出カパラメーター

<i>rSolValues</i>	DOUBLE PRECISION。サイズ <i>nCols</i> × <i>nRhs</i> の配列。実数の解ベクトルを格納する。
<i>cSolValues</i>	DOUBLE COMPLEX。サイズ <i>nCols</i> × <i>nRhs</i> の配列。複素数の解ベクトルを格納する。

説明

?*RhsValues* (? は *r* または *c*) で定義される各右辺の列では、これらのルーチン是对応する解ベクトルを計算し、配列 ?*SolValues* に置く。

右辺の長さ と 解ベクトル、*nCols* と *nRows* は前の [dss_define_structure](#) への呼び出しで定義されているものとみなされる。

戻り値

MKL_DSS_SUCCESS
 MKL_DSS_STATE_ERR
 MKL_DSS_INVALID_OPTION
 MKL_DSS_OUT_OF_MEMORY

dss_delete

解のプロセスで作成されたすべてのデータ構造を削除する。

構文

```
dss_delete(handle, opt)
```

入カパラメーター

<i>opt</i>	INTEGER。パラメーターを渡すオプション。デフォルト値は MKL_DSS_MSG_LVL_WARNING + MKL_DSS_TERM_LVL_ERROR。
------------	--

出力パラメーター

handle MKL_DSS_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

説明

dss_delete ルーチンは、解プロセス中に作成されたすべてのデータ構造を削除するために呼び出される。

戻り値

MKL_DSS_SUCCESS
 MKL_DSS_INVALID_OPTION
 MKL_DSS_OUT_OF_MEMORY

dss_statistics

解のプロセスにおけるさまざまなフェーズについての統計を返す。

構文

dss_statistics(*handle*, *opt*, *statArr*, *retValues*)

入力パラメーター

handle MKL_DSS_HANDLE 型のデータ・オブジェクト (「[インターフェイスの説明](#)」を参照)。

opt INTEGER。パラメーターを渡すオプション。

statArr STRING。返す統計のタイプを定義する入力文字列。次の文字列定数のうち、1つまたは複数を含むことができる (入力文字列の大文字・小文字は区別しない)。

ReorderTime 順序付けに要した時間。

FactorTime 因子分解に要した時間。

SolveTime 因子分解後の求解に要した時間。

Determinant 行列 *A* の行列式。実行列の場合、2つの連続した戻り配列の位置で行列式は *det_pow*、*det_base* として返される。ここで、

$$1.0 \leq \text{abs}(\text{det_base}) < 10.0 \quad \text{および} \\ \text{determinant} = \text{det_base} \cdot 10^{\text{det_pow}}。$$

複素数行列の場合、3の連続した戻り配列の位置で行列式は *det_pow*、*det_re*、*det_im* が返される。ここで、

$$1.0 \leq \text{abs}(\text{det_re}) + \text{abs}(\text{det_im}) < 10.0 \quad \text{および} \\ \text{determinant} = (\text{det_re}, \text{det_im}) \cdot 10^{\text{det_pow}}。$$

Inertia	<p>実対称行列の慣性は、3つの非負整数 (p、n、z) として定義される。ここで、p は正の固有値の数、n は負の固有値の数、z はゼロの固有値の数。</p> <p>Inertia は、p、n、z と3つの連続した戻り配列の位置として返される。</p> <p>Inertia の計算は安定した行列に対してのみ行うことを推奨する。不安定な行列に対して行うと、誤った結果をもたらすことがある。</p> <p>実対称正定値行列 kxk の Inertia は常に $(k,0,0)$。そのため、Inertia は実対称不定値行列の場合のみ返される。その他の行列タイプでは、エラーメッセージが返される。</p>
Flops	因子分解中の浮動小数点演算の数。



注: Fortran から C に文字列を渡す際の問題を回避するため、Fortran では `dss_statistics` を呼び出す前に `mkl_cvt_to_null_terminated_str` ルーチン呼び出さなければならない。詳細は、「[mkl_cvt_to_null_terminated_str](#)」を参照のこと。

出力パラメーター

`retValues` DOUBLE PRECISION。統計値が返される。

説明

`dss_statistics` ルーチンは、解のプロセスのさまざまなフェーズについての統計を返す。

このルーチンを使用して次の分野の統計を収集できる。

- 順序付けに要した時間
- 因子分解に要した時間
- 求解時間
- 行列式
- 行列の慣性
- 因子分解中の浮動小数点演算の数

指定された入力文字列に対応する統計が返される。ユーザーにより割り当てられた戻り配列で統計の値が倍精度で返される。

複数の統計の場合、カンマで区切られた文字列定数を入力として使用できる。戻り値は、入力文字列で指定されたものと同じ順序で戻り配列に置かれる。

統計は、解のプロセス中の適切な段階でのみ要求されなければならない。例えば、行列が因子分解される前に `FactorTime` が要求されるとエラーとなる。

次の表は、各統計の呼び出しポイントを示す。

表 8-3 **統計呼び出しシーケンス**

統計のタイプ	呼び出しのタイミング
ReorderTime	dss_reorder が正常に完了した後。
FactorTime	dss_factor_real または dss_factor_complex が正常に完了した後。
SolveTime	dss_solve_real または dss_solve_complex が正常に完了した後。
Determinant	dss_factor_real または dss_factor_complex が正常に完了した後。
Inertia	dss_factor_real が正常に完了した後。行列が実数、対称、不定値の場合。
Flops	dss_factor_real または dss_factor_complex が正常に完了した後。

次は、dss_statistics ルーチンの使用例である。

例 8-1 **「順序付けに要した時間」と行列の「慣性」の求め方**

これらの値を求めるには、次のルーチン呼び出す。
dss_statistics(handle, opt, statArr, retValues)
ここで、statArr は "ReorderTime,Inertia"

この例では、retValues には次の値が含まれる。

```
retValue[0]            順序付け時間。
retValue[1]            正の固有値。
retValue[2]            負の固有値。
retValue[3]            ゼロの固有値。
```

戻り値

```
MKL_DSS_SUCCESS
MKL_DSS_STATISTICS_INVALID_MATRIX
MKL_DSS_STATISTICS_INVALID_STATE
MKL_DSS_STATISTICS_INVALID_STRING
```

mkl_cvt_to_null_terminated_str

Fortran ルーチンから C ルーチンに文字列を渡す。

構文

```
mkl_cvt_to_null_terminated_str(destStr, destLen, srcStr)
```

入力パラメーター

```
destLen            INTEGER。出力配列 destStr の長さ。
srcStr            STRING。入力文字列。
```

出力パラメーター

destStr INTEGER。1次元の整数配列。

説明

`mkl_cvt_to_null_terminated_str` ルーチンは、Fortran ルーチンから C ルーチンに文字列を渡すために使用される。文字列は C に渡される前に整数配列に変換される。このルーチンを使用すると、Fortran から C に文字列を渡す際に一部のプラットフォームで発生する問題を回避できる。このルーチンの使用を強く推奨する。

実装の詳細

インテル MKL DSS インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル MKL DSS の言語固有のヘッダーファイルを使用することを推奨する。現在、次の3つの言語固有のヘッダーファイルが用意されている。

- mkl_dss.f77 - F77 プログラム
- mkl_dss.f90 - F90 プログラム
- mkl_dss.h - C プログラム

これらの言語固有のヘッダーファイルは、エラーの戻り値の記号定数、関数オプション、定義されたデータ型、関数プロトタイプを定義する。



注: ヘッダーファイルで定義されている記号名でのみ、オプション、エラーの戻り値、メッセージ重要度の定数を参照することを推奨する。上記のヘッダーファイルのいずれかをインクルードしないでインテル MKL DSS ソフトウェアを使用することはできない。

メモリー割り当てとハンドル

インテル MKL DSS ルーチンをできるだけ簡単に使用できるように、ルーチンは、ユーザーが一時作業領域を割り当てる必要がないように設計されている。ソルバーに必要な領域（ユーザー入力ではない）は、ソルバー自身が割り当てる。複数のユーザーが同時にソルバーにアクセスできるように、ソルバーは**ハンドル**と呼ばれるデータ・オブジェクトを使用して特定のアプリケーションに割り当てられた領域の軌跡を記録する。

各インテル MKL DSS ルーチンは、ハンドルを作成、使用、または削除する。そのため、ユーザープログラムは、ハンドルに領域を割り当てることができなければならない。ハンドルに割り当てる領域の構文は言語間で異なる。ハンドル宣言の標準化のため、言語固有のヘッダはユーザーコードにハンドル・オブジェクトを宣言する際に使用するべき定数とデータ型を宣言する。

Fortran 90 プログラムでは、次のようにハンドルを宣言する。

```
INCLUDE "mkl_dss.f90"
TYPE(MKL_DSS_HANDLE) handle
```

C プログラムおよび C++ プログラムでは、次のようにハンドルを宣言する。

```
#include "mkl_dss.h"
_MKL_DSS_HANDLE_t handle;
```

8 バイト整数がサポートされたコンパイラーを使用する Fortran 77 プログラムでは、次のようにハンドルを宣言する。

```
INCLUDE "mkl_dss.f77"
INTEGER*8 handle
```

それ以外では、INTEGER*8 を DOUBLE PRECISION に置き換える。

インクルード・ファイルは、ハンドルの宣言を正しく定義することに加えて、次の項目も定義する。

- プロトタイプをサポートする言語の関数プロトタイプ
- エラーの戻り値に使用される記号定数
- ソルバールーチンのユーザーオプション
- メッセージの重要度

リバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復法スパースソルバー

インテル MKL は、PARDISO インターフェイスに加えて、RCI ベースの ISS インターフェイス (ここでは RCI ISS インターフェイスと呼ぶ) をサポートしている。RCI ISS インターフェイスは、ユーザーが呼び出すことが可能なルーチンのグループで、線形代数の対称正定値連立方程式 (RCI Conjugate Gradient ソルバー、または RCI CG)、および非対称不定値 (非退化) 連立方程式 (RCI Flexible Generalized Minimal RESidual ソルバー、または RCI FGMRES) をステップ・バイ・ステップの解の算出プロセスで解くために使用され、[\[Dong95\]](#) に説明されている一般的な RCI スキームを利用する。インテル MKL RCI ISS ルーチンを使用するために必要な用語と概念は、「[線形ソルバーの基礎](#)」に記載されている。このマニュアルでは、FORTRAN 形式の表記を使用している。しかし、すべての RCI ISS ルーチンは C から呼び出すことができる。詳細は、「[C/C++からのスパース・ソルバー・ルーチンの呼び出し](#)」を参照のこと。RCI とはユーザー自身が特定の演算をソルバー用に行うことを意味する (例えば、行列 - ベクトル演算など)。そのような演算結果をソルバーが必要とするとき、ユーザーは演算結果をソルバーに渡さなければならない。行列 - ベクトル乗算などの特定の演算実装に依存しないため、ソルバーに普遍性を持たせることができる。ただし、この方法ではユーザー側で若干の作業が必要となる。この作業を簡素化するため、ユーザーは、組み込みのスパース行列 - ベクトル乗算と三角ソルバールーチン (第 2 章の「[スパース BLAS レベル 2 およびレベル 3](#)」を参照) を使用することができる。



注 :CG 手法は、式の行列が対称、正定値でない場合に求解に失敗したり、誤った解を導くことがある。
FGMRES 手法は、行列が退化の場合に失敗する。

解のプロセスは、概念的に 4 ステップに分けられる。[表 8-4](#) は、ルーチン名と一般的な使用法の説明をまとめたものである。

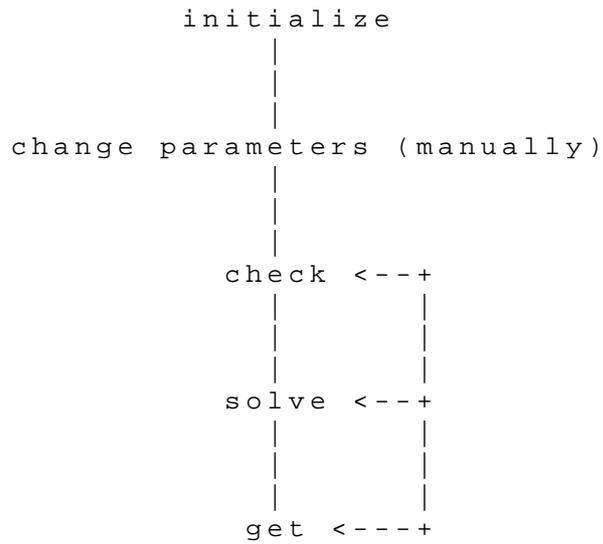
表 8-4 RCI ISS インターフェイス・ルーチン

ルーチン	説明
dcg_init 、 dfgmres_init	ソルバーを初期化する。
dcg_check 、 dfgmres_check	ユーザー定義のデータの一貫性や正確性を検証する。
dcg 、 dfgmres	近似解ベクトルを計算する。
dcg_get 、 dfgmres_get	現在の反復数を検索する。

右辺が 1 つの連立方程式の解ベクトルを求めるため、通常 [表 8-4](#) のリスト順 (コードの任意の場所で起動可能な `dcg_get` および `dfgmres_get` ルーチンを除く) でインテル MKL RCI ISS インターフェイス・ルーチンが起動される。しかし、この場合、間違った結果を回避するためにいくつかの予防措置を講じる必要がある。上級ユーザーは、必要に応じてこの順序を変更することができる。一般ユーザーは、この呼び出し順に従うことを推奨する。

図 8-3 のダイアグラムは、RCI ISS インターフェイス・ルーチンを起動する一般的な順序を示す。

図 8-3 RCI ISS インターフェイス・ルーチンを起動する一般的な順序



[図 8-4](#) と [図 8-5](#) は、それぞれ、RCI CG および RCI FGMRES ルーチンを使用する一般的なスキームを示す。

図 8-4 RCI CG ルーチンを使用する一般的なスキーム

```

...
行列 A を生成する
前処理 C を生成する ( オプション )
  call dcg_init(n, x, b, RCI_request, ipar, dpar, tmp)
  必要に応じて、ipar、dpar のパラメーターを変更する
  call dcg_check(n, x, b, RCI_request, ipar, dpar, tmp)
1  call dcg(n, x, b, RCI_request, ipar, dpar, tmp)
  if (RCI_request.eq.1) then
    行列 A に tmp(1:n,1) を掛け、結果を (1:n,2) に格納する
    この操作で、MKL スパース BLAS レベル 2 のサブルーチンを使用することもできる。
  c proceed with CG iterations
    goto 1
  endif
  if (RCI_request.eq.2) then
    停止テストを実行する
    if ( テストにパスしなかった ) then
  c proceed with CG iterations
    goto 1
  else
  c stop CG iterations
    goto 2
  endif
endif

  if (RCI_request.eq.3) then ( オプション )
    前処理 C の逆数を tmp(1:n,3) に適用し、結果を (1:n,4) に格納する
  c proceed with CG iterations
    goto 1
  end
2 call dcg_get(N, x, b, RCI_request, ipar, dpar, tmp, itercount)
  現在の反復数は itercount にある。
  計算された近似値は配列 x にある。

```

図 8-5 RCI FGMRES ルーチンを使用する一般的なスキーム

```

...
行列 A を生成する
前処理 C を生成する ( オプション )
  call dfgmres_init(n, x, b, RCI_request, ipar, dpar, tmp)
  必要に応じて、ipar、dpar のパラメーターを変更する
  call dfgmres_check(n, x, b, RCI_request, ipar, dpar, tmp)
1   call dfgmres(n, x, b, RCI_request, ipar, dpar, tmp)
   if (RCI_request.eq.1) then
     行列 A に tmp(ipar(22)) を掛け、結果を tmp(ipar(23)) に格納する
     この操作で、MKL スパース BLAS レベル 2 のサブルーチンを使用することもできる。
c proceed with FGMRES iterations
   goto 1
   endif
   if (RCI_request.eq.2) then
     停止テストを実行する
     if ( テストにパスしなかった ) then
c proceed with FGMRES iterations
       go to 1
     else
c stop FGMRES iterations
       goto 2
     endif
   endif
   if (RCI_request.eq.3) then ( オプション )
     前処理 C の逆数を tmp(ipar(22)) に適用し、結果を tmp(ipar(23)) に格納する
c proceed with FGMRES iterations
     goto 1
   endif
   if (RCI_request.eq.4) then
     dpar(7) に格納されている次の直交ベクトルのノルムを確認する
     if ( ノルムが丸め / 計算誤差によってゼロでない ) then
c proceed with FGMRES iterations
       goto 1
     else
c stop FGMRES iterations
       goto 2
     endif
   endif

2 call dfgmres_get(n, x, b, RCI_request, ipar, dpar, tmp, itercount)
   現在の反復数は itercount にある。
   計算された近似値は配列 x にある。

```

FGMRES 手法では、配列 x は、dfgmres ルーチンによって行われる計算に従って解を更新する dfgmres_get ルーチンと呼ぶことでのみ更新できる解の初期近似を最初に格納する。

これらの図の擬似コードは、RCI CG インターフェイスと RCI FGMRES インターフェイスの使用の 2 つの主な違いを示している。最初の違いは `RCI_request = 3` (`tmp` 配列の場所の違い。CG は 2 次元で FGMRES は 1 次元) であり、2 つ目の違いは `RCI_request = 4` (RCI CG インターフェイスは `RCI_request = 4` にならない) である。

RCI ISS インターフェイス・ルーチンを使用して連立線形方程式を解くコードの例は、付録 C の「[反復法スパースソルバーのコード例](#)」のセクションを参照のこと。

インターフェイスの説明

本書のすべての型については、標準 Fortran 型の INTEGER、DOUBLE PRECISION を参照のこと。

C プログラマーおよび C++ プログラマーは、Fortran 型から C/C++ 型へのマッピングについての情報が記述された「[C/C++からのスパース・ソルバー・ルーチンの呼び出し](#)」を参照のこと。

ルーチンオプション

すべての RCI ISS ルーチンはパラメーターを持ち、さまざまなオプションをルーチンに渡す。これらのパラメーター値は、注意深く指定する必要がある（「[CGの共通パラメーター](#)」および「[FGMRESの共通パラメーター](#)」を参照）、必要に応じて計算中に変更できるものでなければならない。



注：ユーザーは、計算を失敗したり、誤った結果にならないように、正確で一貫性のあるパラメーターをルーチンに渡さなければならない。

ユーザーデータ配列

多くの RCI ISS ルーチンは、ユーザーデータを入力として受け取る。例えば、ユーザー配列は `dcg` ルーチンに渡されて、連立線形代数方程式の解を計算する。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL RCI ISS ルーチンは、ユーザー入力配列のコピーを作成しない。

CGの共通パラメーター



注：次にリストされたデフォルト値と初期値は、`dcg_init` ルーチンを呼び出すことによってパラメーターに割り当てられる。

`n` - INTEGER。このパラメーターは、`dcg_init` ルーチンの問題サイズを設定する。他のルーチンは、代わりに `ipar(1)` を使用する。

`x` - DOUBLE PRECISION。配列。このパラメーターは、解ベクトルの近似値を格納する。`dcg` ルーチンを初めて呼び出す前に、解ベクトルの初期近似値を格納する。

`gb` - DOUBLE PRECISION。配列。このパラメーターは、右辺ベクトルを格納する。

`RCI_request` - INTEGER。このパラメーターは、RCI CG ルーチンの作業結果についての情報を知らせるために使用される。パラメーターの負の値は、ルーチンがエラーまたは警告により終了したことを示す。0 値は、作業が正常に完了したことを示す。正の値は、ユーザーが特定の動作を実行したことを示す。具体的には次のとおりである。

`RCI_request=1` - 行列に `tmp(1:n,1)` を掛け、結果を `tmp(1:n,2)` に格納し、`dcg` ルーチンに制御を戻す。

$RCI_request=2$ - 停止テストを実行する。失敗した場合、 dcg ルーチンに制御を戻す。それ以外の場合、解が得られ、ベクトル x に格納される。

$RCI_request=3$ - 前処理を $tmp(1:n,3)$ に適用し、結果を $tmp(1:n,4)$ に格納し、 dcg ルーチンに制御を戻す。

dcg_get ルーチンは $RCI_request$ パラメーターを変更しないことに注意。これにより、このルーチンをリバース・コミュニケーション計算内で使用することができる。

$ipar(128)$ - INTEGER。配列。このパラメーターは、 RCI CG 計算における整数データセットを指定するために用いられる。

$ipar(1)$ - 問題のサイズを指定する。 dcg_init ルーチンは $ipar(1)=n$ を割り当てる。他のすべてのルーチンは、 n の代わりにこのパラメーターを使用する。このパラメーターにはデフォルト値はない。

$ipar(2)$ - RCI CG ルーチンにより生成されるエラーメッセージと警告メッセージの出力タイプを指定する。デフォルト値の 6 は、すべてのメッセージが画面上に表示されることを意味する。それ以外の値の場合、エラーメッセージと警告メッセージは新しく作成された $dcg_errors.txt$ ファイルおよび $dcg_check_warnings.txt$ ファイルにそれぞれ書き込まれる。 $ipar(6)$ パラメーターと $ipar(7)$ パラメーターが 0 に設定された場合、エラーメッセージと警告メッセージは生成されないことに注意。

$ipar(3)$ - RCI CG 計算の現在の過程を格納する。初期値は 1。



注: 計算中は、この変数を変更しないことを強く推奨する。

$ipar(4)$ - 現在の反復番号を i 納する。初期値は 0。

$ipar(5)$ - 反復の最大数を指定する。デフォルト値は $\min\{150,n\}$ 。

$ipar(6)$ - 値が 0 でない場合、ルーチンはエラーメッセージを $ipar(2)$ パラメーターに沿って出力する。それ以外の場合、ルーチンはエラーメッセージを出力せず、 $RCI_request$ パラメーターの負の値を返す。デフォルト値は 1。

$ipar(7)$ - 値が 0 でない場合、ルーチンは警告メッセージを $ipar(2)$ パラメーターに沿って出力する。それ以外の場合、ルーチンは警告メッセージを出力せず、 $RCI_request$ パラメーターの負の値を返す。デフォルト値は 1。

$ipar(8)$ - 値が 0 でない場合、 dcg ルーチンは、反復の最大数を算出するために停止テストを行う。すなわち、 $ipar(4) \leq ipar(5)$ 。それ以外の場合、停止され、対応する値が $RCI_request$ に割り当てられる。値が 0 の場合、 dcg ルーチンは、この停止テストを行わない。デフォルト値は 1。

$ipar(9)$ - 値が 0 でない場合、 dcg ルーチンは、残差停止テストを行う。すなわち、 $dpar(5) \leq dpar(4) = dpar(1) \cdot dpar(3) + dpar(2)$ 。条件が満たされた場合、メソッドは停止され、対応する値が $RCI_request$ に割り当てられる。値が 0 の場合、 dcg ルーチンは、この停止テストを行わない。デフォルト値は 0。

ipar(10) - 値が 0 でない場合、*dcg* ルーチンは、*RCI_request=2* を設定してユーザー定義の停止テストを要求する。値が 0 の場合、*dcg* ルーチンは、このユーザー定義の停止テストを行わない。デフォルト値は 1。



注: *ipar(8)*-*ipar(10)* のパラメーターのうち少なくとも 1 つは 1 に設定しなければならない。

ipar(11) - 値が 0 の場合、*dcg* ルーチンは、共役勾配法の前処理なしバージョンを実行する。それ以外の場合、ルーチンは共役勾配法の前処理付きバージョンを実行し、*RCI_request=3* を設定して前処理のステップを実行するようにユーザーに要求する。デフォルト値は 0。

ipar(12:128) は予約されており、現在実行されている *RCI CG* ルーチンでは使用されない。



注: 上級ユーザーは、*RCI CG* を使用してコード内の配列を *INTEGER ipar(11)* のように定義することができる。しかし、インテル MKL の将来のリリースとの互換性を保証するために、長さ 128 の配列 *ipar* を宣言することを強く推奨する。

dpar(128) - *DOUBLE PRECISION*。配列。このパラメーターは、*RCI CG* 計算に倍精度のデータセットを指定するために用いられる。具体的には次のとおりである。

dpar(1) - 相対許容値を指定する。デフォルト値は *1.0D-6*。

dpar(2) - 絶対許容値を指定する。デフォルト値は *0.0D-0*。

dpar(3) - 初期残差 (*dcg* ルーチンで計算された場合) の二乗ノルムを指定する。初期値は 0.0。

dpar(4) - サービス変数。 *dpar(1)*dpar(3)+dpar(2)* と同じ (*dcg* ルーチンで計算された場合)。初期値は 0.0。

dpar(5) - 現在の残差の二乗ノルムを指定する。初期値は 0.0。

dpar(6) - 前の反復ステップ (該当する場合) からの残差の二乗ノルムを指定する。初期値は 0.0。

dpar(7) - *CG* 法の "alpha" パラメーターを格納する。初期値は 0.0。

dpar(8) - *CG* 法の "beta" パラメーターを格納する。 *dpar(5)/dpar(6)* と同じで、初期値は 0.0。

dpar(9:128) は予約されており、現在実行されている *RCI CG* ルーチンでは使用されない。



注: 上級ユーザーは、コード内の配列を *DOUBLE PRECISION dpar(8)* のように定義することができる。しかし、インテル MKL の将来のリリースとの互換性を保証するために、長さ 128 の配列 *dpar* を宣言することを強く推奨する。

tmp(n,4) - *DOUBLE PRECISION*。配列。このパラメーターは、*RCI CG* 計算に倍精度の一時スペースを与えるために用いられる。具体的には次のとおりである。

$tmp(:,1)$ - 現在の検索方向を指定する。初期値は 0.0。
 $tmp(:,2)$ - 現在の検索方向で乗算された行列を格納する。初期値は 0.0。
 $tmp(:,3)$ - 現在の残差を格納する。初期値は 0.0。
 $tmp(:,4)$ - 現在の残差に割り当てられた前処理の逆数を格納する。このパラメーターには初期値はない。



注: 上級ユーザーは、前処理なし CG 反復のみを行う場合は、コード内の配列を `DOUBLE PRECISION tmp(n,3)` のように定義することができる。

FGMRES の共通パラメーター



注: 次にリストされたデフォルト値と初期値は、`dfgmres_init` ルーチン呼び出すことによってパラメーターに割り当てられる。

n - INTEGER。このパラメーターは、`dfgmres_init` ルーチンの問題サイズを設定する。他のルーチンは、代わりに $ipar(1)$ を使用する。

x - DOUBLE PRECISION。配列。このパラメーターは、解ベクトルの近似値を格納する。`dfgmres` ルーチンを初めて呼び出す前に、解ベクトルの初期近似値を格納する。

gb - DOUBLE PRECISION。配列。このパラメーターは、右辺ベクトルを格納する。ユーザーの要求に応じて、後で近似解が格納される。

$RCI_request$ - INTEGER。このパラメーターは、**RCI FGMRES** ルーチンの作業結果についての情報を知らせるために使用される。パラメーターの負の値は、ルーチンがエラーまたは警告により終了したことを示す。0 値は、作業が正常に完了したことを示す。正の値は、ユーザーが特定の動作を実行したことを示す。具体的には次のとおりである。

$RCI_request=1$ - 行列に $tmp(ipar(22))$ を掛け、結果を $tmp(ipar(23))$ に格納し、`dfgmres` ルーチンに制御を戻す。

$RCI_request=2$ - 停止テストを実行する。失敗した場合、`dfgmres` ルーチンに制御を戻す。それ以外の場合、解は `dfgmres_get` ルーチンへの後の呼び出しで更新できる。

$RCI_request=3$ - 前処理を $tmp(ipar(22))$ に適用し、結果を $tmp(ipar(23))$ に格納し、`dfgmres` ルーチンに制御を戻す。

$RCI_request=4$ - 現在の直交ベクトルのノルムが丸め/計算誤差によってゼロでないかどうかを確認する。ゼロでない場合、`dfgmres` ルーチンに制御を戻す。他の場合には、`dfgmres_get` ルーチン呼び出して解プロセスを完了する。

$ipar(128)$ - INTEGER。配列。このパラメーターは、**RCI FGMRES** 計算における整数データセットを指定するために用いられる。

$ipar(1)$ - 問題のサイズを指定する。`dfgmres_init` ルーチンは $ipar(1)=n$ を割り当てる。他のすべてのルーチンは、 n の代わりにこのパラメーターを使用する。このパラメーターにはデフォルト値はない。

ipar(2) - RCI FGMRES ルーチンにより生成されるエラーメッセージと警告メッセージの出力タイプを指定する。デフォルト値の6は、すべてのメッセージが画面上に表示されることを意味する。それ以外の値の場合、エラーメッセージと警告メッセージは新しく作成される *MKL_RCI_FGMRES_Log.txt* ファイルに書き込まれる。*ipar*(6) パラメーターと *ipar*(7) パラメーターが0に設定された場合、エラーメッセージと警告メッセージは生成されないことに注意。

ipar(3) - RCI FGMRES 計算の現在の過程を格納する。初期値は1。



注: 計算中は、この変数を変更しないことを強く推奨する。

ipar(4) - 現在の反復番号を格納する。初期値は0。

ipar(5) - 反復の最大数を指定する。デフォルト値は $\min\{150, n\}$ 。

ipar(6) - 値が0でない場合、ルーチンはエラーメッセージを *ipar*(2) パラメーターに沿って出力する。それ以外の場合、ルーチンはエラーメッセージを出力せず、*RCI_request* パラメーターの負の値を返す。デフォルト値は1。

ipar(7) - 値が0でない場合、ルーチンは警告メッセージを *ipar*(2) パラメーターに沿って出力する。それ以外の場合、ルーチンは警告メッセージを出力せず、*RCI_request* パラメーターの負の値を返す。デフォルト値は1。

ipar(8) - 値が0でない場合、*dfgmres* ルーチンは、反復の最大数を算出するために停止テストを行う。すなわち、 $ipar(4) \leq ipar(5)$ 。それ以外の場合、停止され、対応する値が *RCI_request* に割り当てられる。値が0の場合、*dfgmres* ルーチンは、この停止テストを行わない。デフォルト値は1。

ipar(9) - 値が0でない場合、*dfgmres* ルーチンは、残差停止テストを行う。すなわち、 $dpar(5) \leq dpar(4) = dpar(1) \cdot dpar(3) + dpar(2)$ 。条件が満たされた場合、メソッドは停止され、対応する値が *RCI_request* に割り当てられる。値が0の場合、*dfgmres* ルーチンは、この停止テストを行わない。デフォルト値は0。

ipar(10) - 値が0でない場合、*dfgmres* ルーチンは、*RCI_request*=2を設定してユーザー定義の停止テストを要求する。値が0の場合、*dfgmres* ルーチンは、このユーザー定義の停止テストを行わない。デフォルト値は1。



注: *ipar*(8)-*ipar*(10) のパラメーターのうち少なくとも1つは1に設定しなければならない。

ipar(11) - 値が0の場合、*dfgmres* ルーチンは、FGMRES 法の前処理なしバージョンを実行する。それ以外の場合、ルーチンはFGMRES 法の前処理付きバージョンを実行し、*RCI_request*=3を設定して前処理のステップを実行するようにユーザーに要求する。デフォルト値は0。

ipar(12) - 値が0でない場合、*dfgmres* ルーチンは、現在生成されたベクトルのゼロノルムに対して自動テストを行う。すなわち、 $dpar(7) \leq dpar(8)$ 。ここで、*dpar*(8)には許容値を格納する。それ以外の場合、ルーチンはパラメーター *RCI_request*=4に設定してこの検証を実行するようにユーザーに要求する。デフォルト値は0。

ipar(13) - 値が 0 の場合、*dfgmres_get* ルーチンは、*dfgmres* ルーチンによって行われた計算に従ってベクトル x に対する解を更新する。値が正の場合、ルーチンはベクトル b の右辺に解を書き込む。値が負の場合、ルーチンは現在の反復数のみを返し、解を更新しない。デフォルト値は 0。



注： 上級ユーザーは、コードの任意の場所で *dfgmres_get* ルーチンを使用することができる。この場合、パラメーター *ipar(13)* に特別の注意を払う必要がある。RCI FGMRES 反復は、パラメーター *ipar(13)* がゼロでない場合にのみ、*dfgmres_get* ルーチンへの呼び出しの後も続行できる。*ipar(13)* が正の場合、更新された解はベクトル b の右辺に上書きされる。ユーザーが同じ右辺で FGMRES のリスタート・バージョンを実行する場合、正の *ipar(13)* で *dfgmres_get* ルーチンへの最初の呼び出しを行う前に右辺をメモリーの異なる場所に保存する必要がある。

ipar(14) - リスタート前の反復数をカウントする内部反復カウンターを格納する。初期値は 0。



注： 計算中は、この変数を変更しないことを強く推奨する。

ipar(15) - リスタートなし FGMRES 反復の長さを指定する。FGMRES 法のリスタート・バージョンを実行するには、ユーザーはリスタート前に反復数を *ipar(15)* に割り当てる必要がある。デフォルト値は $\min\{150, n\}$ 。つまり、デフォルトの場合は FGMRES 法のリスタートなしバージョンが使用される。

ipar(16) - サービス変数。 *tmp* 配列で圧縮形式 (詳細は、付録 B の「[行列引数](#)」を参照) で格納された行列が開始する回転 Hessenberg 行列の位置を指定する。

ipar(17) - サービス変数。 *tmp* 配列で余弦のベクトルが開始する回転余弦の位置を指定する。

ipar(18) - サービス変数。 *tmp* 配列で正弦のベクトルが開始する回転正弦の位置を指定する。

ipar(19) - サービス変数。 *tmp* 配列でベクトルが開始する回転残差ベクトルの位置を指定する。

ipar(20) - サービス変数。 *tmp* 配列でベクトルが開始する最小二乗解ベクトルの位置を指定する。

ipar(21) - サービス変数。 *tmp* 配列でセットが開始する前処理付きベクトルのセットの位置を指定する。 *ipar(21)* から開始する *tmp* 配列のメモリー位置は、前処理付き FGMRES 法でのみ使用される。

ipar(22) - *tmp* 配列で *RCI_request* によって要求された操作で使用される最初のベクトル (ソース) が開始するメモリー位置を指定する。

ipar(23) - *tmp* 配列で *RCI_request* によって要求された操作で使用される 2 番目のベクトル (ソース) が開始するメモリー位置を指定する。

ipar(24:128) は予約されており、現在実行されている RCI FGMRES ルーチンでは使用されない。



注: 上級ユーザーは、RCI FGMRES を使用してコード内の配列を INTEGER *ipar*(23) のように定義することができる。しかし、インテル MKL の将来のリリースとの互換性を保証するために、長さ 128 の配列 *ipar* を宣言することを強く推奨する。

dpar(128) - DOUBLE PRECISION。配列。このパラメーターは、RCI FGMRES 計算に倍精度のデータセットを指定するために用いられる。具体的には次のとおりである。

- dpar*(1) - 相対許容値を指定する。デフォルト値は 1.0D-6。
- dpar*(2) - 絶対許容値を指定する。デフォルト値は 0.0D-0。
- dpar*(3) - 初期残差 (*dfgmres* ルーチンで計算された場合) のユークリッド・ノルムを指定する。初期値は 0.0。
- dpar*(4) - サービス変数。*dpar*(1)**dpar*(3)+*dpar*(2) と同じ (*dfgmres* ルーチンで計算された場合)。初期値は 0.0。
- dpar*(5) - 現在の残差のユークリッド・ノルムを指定する。初期値は 0.0。
- dpar*(6) - 前の反復ステップ (該当する場合) からの残差のユークリッド・ノルムを指定する。初期値は 0.0。
- dpar*(7) - 生成されたベクトルのノルムを格納する。初期値は 0.0。



注: 参照のみ: [\[Saad03\]](#) の用語では、このパラメーターは Hessenberg 行列の係数 $h_{k+1,k}$ である。

dpar(8) - 現在生成されたベクトルの "zero" ノルムの許容値を格納する。デフォルト値は 1.0D-12。

dpar(9:128) は予約されており、現在実行されている RCI FGMRES ルーチンでは使用されない。



注: 上級ユーザーは、コード内の配列を DOUBLE PRECISION *dpar*(8) のように定義することができる。しかし、インテル MKL の将来のリリースとの互換性を保証するために、長さ 128 の配列 *dpar* を宣言することを強く推奨する。

tmp - DOUBLE PRECISION

サイズ $((2 * ipar(15) + 1) * n + ipar(15) * (ipar(15) + 9) / 2 + 1)$ の配列。このパラメーターは、RCI FGMRES 計算に倍精度の一時スペースを与えるために用いられる。具体的には次のとおりである。

- tmp*(1:*ipar*(16)-1) - FGMRES 法ベクトルによって生成された手順を格納する。長さ *n* のこのメモリーの最初の部分に対する初期値は 0.0。
- tmp*(*ipar*(16):*ipar*(17)-1) - 圧縮形式で格納された FGMRES 法によって生成された回転 Hessenberg 行列を格納する。*tmp* 配列のこの部分に対する初期値はない。
- tmp*(*ipar*(17):*ipar*(18)-1) - FGMRES 法によって生成された回転余弦ベクトルを格納する。*tmp* 配列のこの部分に対する初期値はない。

$tmp(ipar(18):ipar(19)-1)$ - FGMRES 法によって生成された回転正弦ベクトルを格納する。 tmp 配列のこの部分に対する初期値はない。

$tmp(ipar(19):ipar(20)-1)$ - FGMRES 法によって生成された回転残差ベクトルを格納する。 tmp 配列のこの部分に対する初期値はない。

$tmp(ipar(20):ipar(21)-1)$ - FGMRES 法によって生成された最小二乗問題に対する解ベクトルを格納する。 tmp 配列のこの部分に対する初期値はない。

$tmp(ipar(21):)$ - ユーザーによって FGMRES 法用に生成された前処理付きベクトルのセットを格納する。FGMRES 法の前処理なしバージョンが呼び出された場合、 tmp 配列のこの部分は使用されない。 tmp 配列のこの部分に対する初期値はない。



注: 上級ユーザーは、前処理なし FGMRES 反復のみを行う場合は、コード内の配列を `DOUBLE PRECISION tmp((ipar(15)+1)*n + ipar(15)*(ipar(15)+9)/2 + 1)` のように定義することができる。

RCI CG ルーチン

dcg_init

ソルバーを初期化する。

構文

```
dcg_init(n, x, b, RCI_request, ipar, dpar, tmp)
```

入力パラメーター

n	INTEGER。問題のサイズおよび配列 x と b のサイズを格納する。
x	DOUBLE PRECISION。サイズ n の配列。解ベクトルの初期近似値を格納する。通常、0 または b 。
b	DOUBLE PRECISION。サイズ n の配列。右辺ベクトルを格納する。

出力パラメーター

$RCI_request$	INTEGER。作業完了を知らせる。
$ipar$	INTEGER。サイズ 128 の配列。「 CG の共通パラメーター 」を参照のこと。
$dpar$	DOUBLE PRECISION。サイズ 128 の配列。「 CG の共通パラメーター 」を参照のこと。
tmp	DOUBLE PRECISION。サイズ $(n, 4)$ の配列。「 CG の共通パラメーター 」を参照のこと。

説明

dcg_init ルーチンは、ソルバーを初期化するために呼び出される。初期化の後、インテル MKL RCI CG ルーチンは、dcg_init によって返されたすべてのパラメーターの値を使用することができる。上級ユーザーは、このステップを省略して対応するルーチンのパラメーターに直接値を設定することができる。



警告: 値が正しく、一貫性があることが確実であれば、ソルバールーチンに配列を渡した後で、これらの配列の内容を編集できる。dcg_check ルーチンを呼び出すことにより、正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、方法が正しいことを保証するものではないことに注意。

戻り値

`RCI_request = 0` ルーチンは、作業を正常に完了した。

`RCI_request = -10000` ルーチンは、作業の完了に失敗した。

dcg_check

ユーザー定義のデータの一貫性や正確性を検証する。

構文

```
dcg_check(n, x, b, RCI_request, ipar, dpar, tmp)
```

入力パラメーター

`n` INTEGER。問題のサイズおよび配列 `x` と `b` のサイズを格納する。

`x` DOUBLE PRECISION。サイズ `n` の配列。解ベクトルの初期近似値を格納する。通常、0 または `b`。

`b` DOUBLE PRECISION。サイズ `n` の配列。右辺ベクトルを格納する。

出力パラメーター

`RCI_request` INTEGER。作業完了を知らせる。

`ipar` INTEGER。サイズ 128 の配列。「[CG の共通パラメーター](#)」を参照のこと。

`dpar` DOUBLE PRECISION。サイズ 128 の配列。「[CG の共通パラメーター](#)」を参照のこと。

`tmp` DOUBLE PRECISION。サイズ $(n, 4)$ の配列。「[CG の共通パラメーター](#)」を参照のこと。

説明

dcg_check ルーチンは、dcg ソルバールーチンに渡されるパラメーターの一貫性と正確性を検証する。ただし、この操作はこの方法が正確な結果をもたらすことを保証するものではない。用いられるパラメーターで誤った結果が導かれる率を減少させるだけである。上級ユーザーは、ソルバー・パラメーターに正しいデータが指定されていることがわかっているならば、このステップを省略することができる。



警告: 値が正しく、一貫性があることが確実であれば、ソルバールーチンに配列を渡した後で、これらの配列の内容を編集できる。dcg_check ルーチン呼び出すことにより、正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、方法が正しいことを保証するものではないことに注意。

すべてのベクトルの長さが dcg_init サブルーチンへの前の呼び出しで定義されているとみなされる。

戻り値

`RCI_request = 0` ルーチンは、作業を正常に完了した。

`RCI_request = -1100` ルーチンは、エラーが発生したために一時停止された。

`RCI_request = -1001` ルーチンは、警告メッセージを返した。

`RCI_request = -1010` ルーチンは、一貫性または正確性が保持されるように一部のパラメーターを変更した。

`RCI_request = -1011` ルーチンは、警告メッセージを返して一部のパラメーターを変更した。

dcg

近似解ベクトルを計算する。

構文

```
dcg(n, x, b, RCI_request, ipar, dpar, tmp)
```

入力パラメーター

n INTEGER。問題のサイズおよび配列 *x* と *b* のサイズを格納する。

x DOUBLE PRECISION。サイズ *n* の配列。解ベクトルの初期近似値を格納する。

b DOUBLE PRECISION。サイズ *n* の配列。右辺ベクトルを格納する。

tmp DOUBLE PRECISION。サイズ (*n*, 4) の配列。「[CGの共通パラメーター](#)」を参照のこと。

出力パラメーター

<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解ベクトルの更新された近似値を格納する。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 CGの共通パラメーター 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 CGの共通パラメーター 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ (<i>n</i> , 4) の配列。「 CGの共通パラメーター 」を参照のこと。

説明

dcg ルーチンは、CG 法 [Young71] を使用して、解ベクトルの近似を計算する。dcg ルーチンは、最初の呼び出しの前にベクトル *x* にある値を解の初期近似値として使用する。*RCI_request* パラメーターは、作業終了状況をユーザーに知らせ、ソルバーに必要な演算結果を要求する。

すべてのベクトルの長さが *dcg_init* ルーチンへの前の呼び出しで定義されているとみなされる。

戻り値

<i>RCI_request</i> = 0	ルーチンは作業を正常に完了し、求められた解をベクトル <i>x</i> に格納した。これは停止テストが完全に自動的に行われた場合のみ行われる。ユーザーが停止テストを定義した場合は、 <i>RCI_request</i> = 2 についての注釈を参照のこと。
<i>RCI_request</i> = -1	ルーチンは反復の最大数に達したが、停止条件が満たされなかったために一時停止された (これは、ユーザーが両方のテストを要求した場合にのみ発生する)。
<i>RCI_request</i> = -2	ルーチンはゼロの除算が起こったために一時停止された。(これは、行列のほとんどが 0 未満の正定値の場合に発生する)。
<i>RCI_request</i> = -10	ルーチンは、残差ノルムが妥当でないために一時停止された。(おそらく、 <i>dpar</i> (6) にあるデータがルーチンの外部で変更されたか、 <i>dcg_check</i> ルーチンは呼び出されなかった)。
<i>RCI_request</i> = -11	ルーチンは、無限サイクルに入ってしまったために一時停止された。(おそらく、 <i>ipar</i> (8)、 <i>ipar</i> (9)、 <i>ipar</i> (10) にあるデータがルーチンの外部で変更されたか、 <i>dcg_check</i> ルーチンが呼び出されなかった)。
<i>RCI_request</i> = 1	行列に <i>tmp</i> (1:N,1) を掛け、結果を <i>tmp</i> (1:N,2) に格納し、 <i>dcg</i> ルーチンに制御を戻すようにユーザーに要求する。
<i>RCI_request</i> = 2	停止テストを行うようにユーザーに要求する。失敗した場合、ユーザーは <i>dcg</i> ルーチンに制御を戻す必要がある。それ以外の場合、解が得られ、ベクトル <i>x</i> に格納される。
<i>RCI_request</i> = 3	前処理を <i>tmp</i> (:,3) に適用し、結果を <i>tmp</i> (:,4) に格納し、 <i>dcg</i> ルーチンに制御を戻す。

dcg_get

現在の反復数を検索する。

構文

```
dcg_get(n, x, b, RCI_request, ipar, dpar, tmp, itercount)
```

入力パラメーター

<i>n</i>	INTEGER。問題のサイズおよび配列 <i>x</i> と <i>b</i> のサイズを格納する。
<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解の近似値を格納する。
<i>b</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。右辺ベクトルを格納する。
<i>RCI_request</i>	INTEGER。このパラメーターは使用されない。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 CG の共通パラメーター 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 CG の共通パラメーター 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ (<i>n</i> , 4) の配列。「 CG の共通パラメーター 」を参照のこと。

出力パラメーター

<i>itercount</i>	INTEGER。この引数は、現在の反復番号の値を返す。
------------------	-----------------------------

説明

dcg_get ルーチンは、解プロセスの現在の反復番号を検索するために呼び出される。

戻り値

dcg_get ルーチンは、値を返さない。

RCI FGMRES ルーチン

dfgmres_init

ソルバーを初期化する。

構文

```
dfgmres_init(n, x, b, RCI_request, ipar, dpar, tmp)
```

入力パラメーター

<i>n</i>	INTEGER。問題のサイズおよび配列 <i>x</i> と <i>b</i> のサイズを格納する。
----------	--

<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解ベクトルの初期近似値を格納する。通常、0 または <i>b</i> 。
<i>b</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。右辺ベクトルを格納する。

出力パラメーター

<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>ipar</i>	INTEGER。サイズ 128 の配列。 「FGMRES の共通パラメーター」 を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。 「FGMRES の共通パラメーター」 を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ $((2 * ipar(15) + 1) * n + ipar(15) * (ipar(15) + 9) / 2 + 1)$ の配列。 「FGMRES の共通パラメーター」 を参照のこと。

説明

`dfgmres_init` ルーチンは、ソルバーを初期化するために呼び出される。初期化の後、インテル MKL RCI FGMRES ルーチンは、`dfgmres_init` によって返されたすべてのパラメーターの値を使用することができる。上級ユーザーは、このステップを省略して対応するルーチンのパラメーターに直接値を設定することができる。



警告： 値が正しく、一貫性があることが確実にあれば、ソルバールーチンに配列を渡した後で、これらの配列の内容を編集できる。`dfgmres_check` ルーチンを呼び出すことにより、正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、方法が正しいことを保証するものではないことに注意。

戻り値

`RCI_request = 0` ルーチンは、作業を正常に完了した。
`RCI_request = -10000` ルーチンは、作業の完了に失敗した。

dfgmres_check

ユーザー定義のデータの一貫性や正確性を検証する。

構文

```
dfgmres_check(n, x, b, RCI_request, ipar, dpar, tmp);
```

入力パラメーター

<i>n</i>	INTEGER。問題のサイズおよび配列 <i>x</i> と <i>b</i> のサイズを格納する。
<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。解ベクトルの初期近似値を格納する。通常、0 または <i>b</i> 。
<i>b</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。右辺ベクトルを格納する。

出力パラメーター

<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 FGMRES の共通パラメーター 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 FGMRES の共通パラメーター 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ $((2 * ipar(15) + 1) * n + ipar(15) * (ipar(15) + 9) / 2 + 1)$ の配列。「 FGMRES の共通パラメーター 」を参照のこと。

説明

dfgmres_check ルーチンは、dfgmres ソルバールーチンに渡されるパラメーターの一貫性と正確性を検証する。ただし、この操作はこの方法が正確な結果をもたらすことを保証するものではない。用いられるパラメーターで誤った結果が導かれる率を減少させるだけである。上級ユーザーは、ソルバー・パラメーターに正しいデータが指定されていることがわかっているならば、このステップを省略することができる。



警告： 値が正しく、一貫性があることが確実であれば、ソルバールーチンに配列を渡した後で、これらの配列の内容を編集できる。dfgmres_check ルーチン呼び出すことにより、正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、方法が正しいことを保証するものではないことに注意。

すべてのベクトルの長さが dfgmres_init サブルーチンへの前の呼び出しで定義されているとみなされる。

戻り値

- `RCI_request= 0` ルーチンは、作業を正常に完了した。
- `RCI_request= -1100` ルーチンは、エラーが発生したために一時停止された。
- `RCI_request= -1001` ルーチンは、警告メッセージを返した。
- `RCI_request= -1010` ルーチンは、一貫性または正確性が保持されるように一部のパラメーターを変更した。
- `RCI_request= -1011` ルーチンは、警告メッセージを返して一部のパラメーターを変更した。

dfgmres

FGMRES 反復を行う。

構文

`dfgmres(n, x, b, RCI_request, ipar, dpar, tmp)`

入力パラメーター

- `n` INTEGER。問題のサイズおよび配列 `x` と `b` のサイズを格納する。
- `x` DOUBLE PRECISION。サイズ `n` の配列。解ベクトルの現在の初期近似値を格納する。
- `b` DOUBLE PRECISION。サイズ `n` の配列。右辺ベクトルを格納する。
- `tmp` DOUBLE PRECISION。サイズ $((2*ipar(15)+1)*n + ipar(15)*(ipar(15)+9)/2 + 1)$ の配列。「[FGMRES の共通パラメーター](#)」を参照のこと。

出力パラメーター

- `RCI_request` INTEGER。作業完了を知らせる。
- `ipar` INTEGER。サイズ 128 の配列。「[FGMRES の共通パラメーター](#)」を参照のこと。
- `dpar` DOUBLE PRECISION。サイズ 128 の配列。「[FGMRES の共通パラメーター](#)」を参照のこと。
- `tmp` DOUBLE PRECISION。サイズ $((2*ipar(15)+1)*n + ipar(15)*(ipar(15)+9)/2 + 1)$ の配列。「[FGMRES の共通パラメーター](#)」を参照のこと。

説明

`dfgmres` ルーチンは、FGMRES 反復 [\[Saad03\]](#) を実行する。DFGMRES ルーチンは、最初の呼び出しの前にベクトル `x` にある値を解の初期近似値として使用する。解の現在の近似値を更新するには、ユーザーは `dfgmres_get` ルーチンを呼び出す必要がある。RCI FGMRES 反復は、パラメーター `ipar(13)` の値が 0 (デフォルト値) でない場合にのみ、

`dfgmres_get` ルーチンへの呼び出しの後も続行できる。`ipar(13)` が正の場合、更新された解はベクトル b の右辺に上書きされ、FGMRES 法のリスタート・バージョンは実行できないことに注意する。ユーザーが右辺を保持する場合、正の `ipar(13)` で `dfgmres_get` ルーチンへの最初の呼び出しを行う前に右辺をメモリーの異なる場所に保存する必要がある。

`RCI_request` パラメーターは、作業終了状況をユーザーに知らせ、ソルバーに必要な演算結果を要求する。

すべてのベクトルの長さが `dfgmres_init` ルーチンへの前の呼び出しで定義されているとみなされる。

戻り値

- | | |
|-------------------------------|--|
| <code>RCI_request= 0</code> | ルーチンは作業を正常に完了し、解が見つかった。これは停止テストが完全に自動的に行われた場合にのみ行われる。ユーザーが停止テストを定義した場合は、 <code>RCI_request= 2</code> または <code>4</code> についての注釈を参照のこと。 |
| <code>RCI_request= -1</code> | ルーチンは反復の最大数に達したが、停止条件が満たされなかったために一時停止された (これは、ユーザーが両方のテストを要求した場合にのみ発生する)。 |
| <code>RCI_request= -10</code> | ルーチンは、ゼロ除算が発生したために一時停止された。通常は、行列が (ほぼ) 退化の場合に発生する。しかし、パラメーター <code>dpar</code> が誤って変更された場合や、解が見つかったときにメソッドが停止していなかった場合にも発生することがある。 |
| <code>RCI_request= -11</code> | ルーチンは、無限サイクルに入ってしまったために一時停止された。(おそらく、 <code>ipar(8)</code> 、 <code>ipar(9)</code> 、 <code>ipar(10)</code> にあるデータがルーチンの外部で変更されたか、 <code>dfgmres_check</code> ルーチンが呼び出されなかった)。 |
| <code>RCI_request= -12</code> | ルーチンは、メソッド・パラメーターにエラーが見つかったために一時停止された。通常は、パラメーター <code>ipar</code> および <code>dpar</code> がルーチンの外部で誤って変更された場合に発生する。 |
| <code>RCI_request= 1</code> | 行列に <code>tmp(ipar(22))</code> を掛け、結果を <code>tmp(ipar(23))</code> に格納し、 <code>dfgmres</code> ルーチンに制御を戻すようにユーザーに要求する。 |
| <code>RCI_request= 2</code> | 停止テストを行うようにユーザーに要求する。失敗した場合、ユーザーは <code>dfgmres</code> ルーチンに制御を戻す必要がある。それ以外の場合、FGMRES の解が見つかる。ユーザーは <code>dfgmres_get</code> ルーチンを呼び出してベクトル x の算出された解を更新する必要がある。 |
| <code>RCI_request= 3</code> | 前処理の逆数を <code>tmp(ipar(22))</code> に適用し、結果を <code>tmp(ipar(23))</code> に格納し、 <code>dfgmres</code> ルーチンに制御を戻すようにユーザーに要求する。 |
| <code>RCI_request= 4</code> | 現在生成されたベクトルのノルムを検証するようにユーザーに要求する。ノルムが計算 / 丸め誤差によってゼロでない場合、ユーザーは <code>dfgmres</code> ルーチンに制御を戻す必要がある。それ以外の場合、FGMRES の解が見つかる。ユーザーは <code>dfgmres_get</code> ルーチンを呼び出してベクトル x の算出された解を更新する必要がある。 |

dfgmres_get

現在の反復数を検索して解を更新する。

構文

```
dfgmres_get(n, x, b, RCI_request, ipar, dpar, tmp, itercount)
```

入力パラメーター

<i>n</i>	INTEGER。問題のサイズおよび配列 <i>x</i> と <i>b</i> のサイズを格納する。
<i>ipar</i>	INTEGER。サイズ 128 の配列。「 FGMRES の共通パラメーター 」を参照のこと。
<i>dpar</i>	DOUBLE PRECISION。サイズ 128 の配列。「 FGMRES の共通パラメーター 」を参照のこと。
<i>tmp</i>	DOUBLE PRECISION。サイズ $((2 * ipar(15) + 1) * n + ipar(15) * (ipar(15) + 9) / 2 + 1)$ の配列。「 FGMRES の共通パラメーター 」を参照のこと。

出力パラメーター

<i>x</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。 <i>ipar</i> (13)=0 の場合、dfgmres ルーチンによって行われた計算に従って解の更新された近似値を格納する。それ以外の場合は変更されない。
<i>b</i>	DOUBLE PRECISION。サイズ <i>n</i> の配列。 <i>ipar</i> (13)>0 の場合、dfgmres ルーチンによって行われた計算に従って解の更新された近似値を格納する。それ以外の場合は変更されない。
<i>RCI_request</i>	INTEGER。作業完了を知らせる。
<i>itercount</i>	INTEGER。この引数は、現在の反復番号の値を返す。

説明

dfgmres_get ルーチンは、解プロセスの現在の反復番号を検索するため、および dfgmres ルーチンによって実行された計算に従って解を更新するために呼び出される。現在の反復番号のみを検索するには、パラメーター *ipar*(13)=-1 を前もって設定する。計算を引き続き行う場合、通常はこの処理を推奨する。中間の解が必要な場合、メソッド・パラメーターを適切に設定する必要がある。詳細は、付録 C の「[FGMRES の共通パラメーター](#)」および「[RCI\(前処理付き\)フレキシブル汎用最小残差ソルバーの使用例](#)」セクションを参照。

戻り値

<i>RCI_request</i> = 0	ルーチンは、作業を正常に完了した。
<i>RCI_request</i> = -12	ルーチンは、メソッド・パラメーターにエラーが見つかったために一時停止された。通常は、パラメーター <i>ipar</i> および <i>dpar</i> の一部がルーチンの外部で誤って変更された場合に発生する。
<i>RCI_request</i> = -10000	ルーチンは、作業の完了に失敗した。

実装の詳細

インテル MKL RCI ISS インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル MKL RCI ISS の言語固有のヘッダーファイルを使用することを推奨する。現在、C プログラム用の言語固有のヘッダーファイルが用意されている。

これらの言語固有のヘッダーファイルは、関数のプロトタイプおよび次のような項目を定義する。

```
void dcg_init(int *n, double *x, double *b, int *rci_request, int *ipar,
             double *dpar, double *tmp);
void dcg_check(int *n, double *x, double *b, int *rci_request, int *ipar,
              double *dpar, double *tmp);
void dcg(int *n, double *x, double *b, int *rci_request, int *ipar,
         double *dpar, double *tmp);
void dcg_get(int *n, double *x, double *b, int *rci_request, int *ipar,
            double *dpar, double *tmp, int *itercount);

void dfgmres_init(int *n, double *x, double *b, int *rci_request, int
                *ipar, double *dpar, double *tmp);
void dfgmres_check(int *n, double *x, double *b, int *rci_request, int
                  *ipar, double *dpar, double *tmp);
void dfgmres(int *n, double *x, double *b, int *rci_request, int *ipar,
            double *dpar, double *tmp);
void dfgmres_get(int *n, double *x, double *b, int *rci_request, int
                *ipar, double *dpar, double *tmp, int *itercount);
```



注意：言語固有のヘッダーファイルをインクルードしないでインテル MKL RCI ISS ソフトウェアを使用することはできない。

C/C++ からのスパース・ソルバー・ルーチンの呼び出し

インテル MKL スパース・ソルバー・ルーチンの呼び出しインターフェイスは、Fortran 77 または Fortran 90 から簡単に使用できるように設計されている。ただし、使用しているプラットフォームの異言語呼び出し規則について知識がある場合、これらのルーチンは C または C++ から直接起動することもできる。規則には、その言語での引数の渡し方、Fortran から C/C++ へのデータ型マッピング、およびプラットフォームにおける Fortran 外部名の修飾方法などが含まれる。

移植性を向上させ、ユーザーの負担を軽減させるため、C ヘッダーファイルでは、異言語呼び出し規則を隠蔽するためのマクロセットと型定義が提供され、C/C++ にとって自然なインターフェイスをインテル MKL スパース・ソルバー・ルーチンに提供する。

例えば、foo というライブラリー・ルーチンがあるとする。このルーチンは、長さが n の実ベクトルをとり、整数ステータスを返す。Fortran の場合、このような関数には次のようにアクセスする。

```

INTEGER n, status, foo
REAL x(*)
status = foo(x, n)
    
```

C 言語では、foo を起動する場合に、Fortran の INTEGER 型と REAL 型に対応する C のデータ型、Fortran コンパイラーが使用する引数の渡し方、そして(該当する場合は)外部シンボル foo の生成時に Fortran コンパイラーにより実行される名前修飾などを知っておく必要がある。

しかし、例えば mkl_solver.h などの C 固有のヘッダーファイルを使用すると、foo の起動は C プログラムでは次のようになる。

```

#include "mkl_solver.h"
_INTEGER_t i, status;
_REAL_t x[];
status = foo( x, i );
    
```

上記の例で分かるように、ヘッダーファイルの mkl_solver.h は、Fortran の INTEGER 型と REAL 型に対応する、_INTEGER_t 型と _REAL_t 型を提供する。

インテル MKL スパース・ソルバー・ルーチンを C および C++ から簡単に使用できるように、Fortran 型の C 定義がライブラリー全体で使用される。特に、スパースソルバーからの引数または結果が Fortran 言語固有の型である xxx の場合は、C ヘッダーファイルおよび C++ ヘッダーファイルは _xxx_t に適切な C 言語型の定義を提供する。

C ユーザーが気を付けるべきこと

C/C++ と Fortran の主な違いは、引数の渡し方が異なるということである。Fortran プログラムは参照渡しセマンティクスを使用するのに対し、C/C++ プログラムは値渡しセマンティクスを使用する。前のセクションの例では、ヘッダーファイルの mkl_solver.h は、適切な引数のアドレスをとるマクロ foo を定義することにより、この相違を隠蔽しようとした。例えば、Tru64 UNIX の場合、mkl_solver.h はマクロを次のように定義する。

```

#define foo(a,b) foo_(a), &(b))
    
```

注意すべき点は、foo というマクロ形式を使用する際、定数をどのように処理するかである。foo(x, 10) と記述すると、foo_(x, &10) と翻訳される。ANSI に厳密に準拠する C コンパイラーでは、定数のアドレスは許容されないため、厳密に準拠したプログラムは次のようになる。

```

_INTEGER_t iTen = 10;
_REAL_t * x;
status = foo( x, iTen );
    
```

ただし、ANSI 規格に準拠していない一部の C コンパイラーでは、Fortran プログラムでも簡単に使用できるように定数のアドレスをとることができる。このように、foo(x, 10) はこのようなコンパイラーでは許容される。

ベクトル数学関数

9

本章では、ベクトル引数に基づいて数学関数を計算するベクトル数学関数ライブラリー (VML) について説明する。VML は、インテル[®] マス・カーネル・ライブラリーの必須部分である。ここでは、VML に含まれる関数の説明を簡単にするために、VML 固有の用語を使用する。

VML には、ベクトルを演算するためのコアとなる数学関数の中でも非常に計算時間を要する一群の関数 (累乗、三角方程式、指数、双曲線など) について、高度に最適化された実装のセットが含まれる。

VML により、非線形プログラミング・ソフトウェアや積分計算などの多くのプログラムの性能が大幅に向上すると期待できる。VML は FORTRAN と C 言語のどちらのインターフェイスも備えている。

VML 関数は、実行する演算に応じて、以下のグループに分けられる。

- [VML 数学関数](#) : 単位増分のインデックスを使用するベクトルに基づいて、数学関数 (正弦、余弦、指数、対数など) の値を計算する。
- [VML Pack/Unpack 関数](#) : 正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する (ベクトル・インデックス方式の詳細は、[付録 B](#) を参照のこと)。
- [VML サービス関数](#) : 精度モードの設定 / 取得、エラーコードの設定 / 取得を行う。

VML 数学関数は、入力ベクトルを引数とし、それぞれの関数の値を成分ごとに計算して、結果を出力ベクトルに入れて戻す。

データ型と精度モード

VML の数学ベクトル関数と pack/unpack ベクトル関数は、単精度実数データと倍精度実数データのベクトル引数に対応するように実装されている。ライブラリーには、VML サービス関数を含むすべての関数に対する Fortran インターフェイスと C インターフェイスが用意されている。Fortran インターフェイスと C インターフェイスにおける関数の命名方法と呼び出し方法の違いは、この後の「[関数命名規則](#)」セクションで説明する。

VML の各ベクトル関数は、(各データ形式ごとに) 高精度 (HA) と低精度 (LA) の 2 つのモードで機能できる。多くの関数の場合、LA 版を使用すると、精度が低下するが処理速度が向上する。

しかし、精度を落としても処理速度がほとんど向上しない場合には、両方のモードに対して同じ関数が使用される。エラーの動作は、HA モードと LA モードのどちらを選択しているかだけでなく、ソフトウェアを実行するプロセッサによっても変わってくる。

また、特殊な値に対する動作は、関数で HA モードと LA モードのどちらを使用しているかにより異なる場合がある。精度に関係した事項は、インテル *MKL* リリースノート を参照のこと。

HA モードと LA モードを切り替えるには、`vmlSetMode(mode)` を使用する (表 9-12 を参照)。関数 `vmlGetMode()` は、現在使用しているモードを戻す。デフォルトでは、高精度 (HA) モードが使用される。

関数命名規則

Fortran インターフェイスの場合、VML 関数のフルネームには小文字しか含まれない。C インターフェイスの場合は、小文字と大文字が混在して含まれる。

VML 数学関数と pack/unpack 関数のフルネームは、以下の形式で指定する。

`v <p> <name> <mod>`

先頭文字の `v` は、関数が VML に属していることを示すプリフィックスである。

`<p>` フィールドは、以下のデータ型を指定する精度プリフィックスである。

<code>s</code>	REAL (Fortran インターフェイスの場合)、float (C インターフェイスの場合)
<code>d</code>	DOUBLE PRECISION (Fortran インターフェイスの場合)、double (C インターフェイスの場合)
<code>c</code>	COMPLEX (Fortran インターフェイスの場合)、MKL_Complex8 (C インターフェイスの場合)
<code>z</code>	DOUBLE COMPLEX (Fortran インターフェイスの場合)、MKL_Complex16 (C インターフェイスの場合)

`<name>` フィールドは関数の短縮名を示し、C インターフェイスの場合は一部大文字が含まれる (表 9-2 あるいは 表 9-11 の例を参照)。

`<mod>` フィールド (C インターフェイスの場合は大文字で表記) は、pack/unpack 関数でのみ指定する。次に示すインデックス方式を指定する。

<code>i</code>	正増分を使用するインデックス
<code>v</code>	インデックス・ベクトルを使用するインデックス
<code>m</code>	マスクベクトルを使用するインデックス

VML サービス関数のフルネームは、次の形式で指定する。

`vml <name>`

ここで `vml` は関数が VML に属しているのを示すプリフィックスであり、`<name>` は関数の短縮名である。C インターフェイスの場合は一部大文字が含まれる (表 9-11 を参照)。

アプリケーション・プログラムから VML 関数を呼び出すには、従来関数呼び出しを使用する。例えば、単精度実数データの VML 指数関数は、次のように呼び出せる。

```
call vsexp ( n, a, y ) (Fortran インターフェイスの場合)
vsExp ( n, a, y );    (C インターフェイスの場合)
```

関数インターフェイス

VML 関数のインターフェイスには、関数のフルネームと一連の引数が含まれる。それぞれの VML 関数グループにおける Fortran インターフェイスと C インターフェイスの説明を以下に示す。関数によっては、引数として a と b の 2 つの入力ベクトルを持つものがある (Div、Pow、Atan2)。一方、SinCos 関数は、 y と z の 2 つの出力ベクトルを持つ。

VML 数学関数

Fortran:

```
call v<p><name>( n, a, y )
call v<p><name>( n, a, b, y )
call v<p><name>( n, a, y, z )
```

C:

```
v<p><name>( n, a, y );
v<p><name>( n, a, b, y );
v<p><name>( n, a, y, z );
```

Pack 関数

Fortran:

```
call v<p>packi( n, a, inca, y )
call v<p>packv( n, a, ia, y )
call v<p>packm( n, a, ma, y )
```

C:

```
v<p>PackI( n, a, inca, y );
v<p>PackV( n, a, ia, y );
v<p>PackM( n, a, ma, y );
```

Pack 関数

Fortran:

```
call v<p>unpacki( n, a, y, incy )
call v<p>unpackv( n, a, y, iy )
call v<p>unpackm( n, a, y, my )
```

C:

```
v<p>UnpackI( n, a, y, incy );
v<p>UnpackV( n, a, y, iy );
v<p>UnpackM( n, a, y, my );
```

サービス関数 :

Fortran:

```

oldmode = vmlsetmode( mode )
mode     = vmlgetmode( )
olderr   = vmlseterrstatus ( err )
err      = vmlgeterrstatus( )
olderr   = vmlclearerrstatus( )
oldcallback = vmlseterrorcallback( callback )
callback = vmlgeterrorcallback( )
oldcallback = vmlclearerrorcallback( )

```

C:

```

oldmode = vmlSetMode( mode );
mode     = vmlGetMode( void );
olderr   = vmlSetErrStatus( err );
err      = vmlGetErrStatus( void );
olderr   = vmlClearErrStatus( void );
oldcallback = vmlSetErrorCallBack( callback );
callback = vmlGetErrorCallBack( void );
oldcallback = vmlClearErrorCallBack( void );

```

入力パラメーター

<i>n</i>	計算する成分の数
<i>a</i>	1 番目の入力ベクトル
<i>a</i>	1 番目の入力ベクトル
<i>inca</i>	入力ベクトル <i>a</i> のベクトル増分
<i>ia</i>	入力ベクトル <i>a</i> のインデックス・ベクトル
<i>ma</i>	入力ベクトル <i>a</i> のマスクベクトル
<i>incy</i>	出力ベクトル <i>y</i> のベクトル増分
<i>iy</i>	出力ベクトル <i>y</i> のインデックス・ベクトル
<i>my</i>	出力ベクトル <i>y</i> のマスクベクトル
<i>err</i>	エラーコード
<i>mode</i>	VML モード
<i>callback</i>	コールバック関数のアドレス

出力パラメーター

<i>y</i>	1 番目の出力ベクトル
<i>z</i>	2 番目の出力ベクトル
<i>err</i>	エラーコード

<code>mode</code>	VML モード
<code>olderr</code>	前のエラーコード
<code>oldmode</code>	前の VML モード
<code>callback</code>	コールバック関数のアドレス
<code>oldcallback</code>	前のコールバック関数のアドレス

各関数で使用するパラメーターのデータ型は、それぞれの関数の説明のセクションで記述する。VML 数学関数はすべて、インプレース演算を実行できる。つまり、同一のベクトルを入力パラメーターとしても出力パラメーターとしても使用可能である。これは、2つの入力ベクトルを持つ関数にも当てはまる。この場合、2つの入力パラメーターの一方が、出力ベクトルで上書きできる。2つの出力ベクトルを持つ関数の場合、出力ベクトルの一方は、入力ベクトルと一致させてもよい。しかし、部分的に入力ベクトルと出力ベクトルをオーバーラップさせることは、予測不能の結果につながる。

ベクトル・インデックス方式

現在の VML 数学関数は、単位増分インデックスでのみ機能する。他の増分を使用する配列、またはより複雑なインデックスを使用する配列を関数に適合させるには、成分を収集して連続する単位増分インデックスのベクトルを生成し、計算の完了後、それらを分散すればよい。

VML では、ベクトル成分の収集 / 分散に以下のインデックス方式を使用する。

- 正増分
- インデックス・ベクトル
- マスクベクトル

関数で使用するインデックス方式は、インデックス修飾子で指定する（「[関数命名規則](#)」の `<mod>` フィールドの説明を参照）。インデックス方式の詳細は、付録 B の「[VML のベクトル引数](#)」を参照のこと。

エラー診断

VML ライブラリーは、独自のエラーハンドラーを備えている。C インターフェイスと Fortran インターフェイスでは、異なる点が 1 つだけある。それは、Fortran インターフェイスでは、VML 関数がエラーを検出した後で、インテル MKL エラー報告ルーチン [xerbla](#) の呼び出しが可能なことである。このルーチンは、`VML_STATUS_BADSIZE` と `VML_STATUS_BADMEM` の入力エラーに関する情報を取得する（[表 9-14](#) を参照）。

VML エラーハンドラーは、以下のような機能を持つ。

1. エラーステータス (`vmlErrStatus`) グローバル変数が、VML 関数をそれぞれ呼び出した後で設定される。この変数が取り得る値を、[表 9-14](#) に示す。
2. VML モードに応じて、エラーハンドラー関数は以下を呼び出す。
 - `errno` 変数設定。 `errno` が取り得る値を、[表 9-1](#) に示す。
 - `stderr` ストリームへのエラーテキスト情報の書き込み。
 - エラー時の適切な例外処理の起動（必要な場合）

- 追加のエラー・ハンドラー・コールバック関数の呼び出し

表 9-1 **errno 変数の値の設定**

errno の値	説明
0	検出されたエラーはない。
EINVAL	配列の次元が正でない。
EACCES	NULL ポインタが渡された。
EDOM	配列の値の少なくとも 1 つが、定義の範囲外。
ERANGE	配列の値の少なくとも 1 つが、特異点、オーバーフロー、またはアンダーフローを引き起こした。

ベクトル数学関数のスレッド化

VML 関数は、OpenMP* で自動的にスレッド化することで、マルチコア・プロセッサ・システム、マルチプロセッサ・システム、インテル® ハイパースレッディング・テクノロジーをサポートするシステム上での性能を最大限に発揮する。

VML スレッド化ロジックは、`vmlSetMode` 関数で制御できる。デフォルトでは、次のようにスレッド化が行われる。まず、環境変数 `OPM_NUM_THREADS` によって、VML 関数のスレッド化で使用するスレッドの最大数が決定される。実際のスレッド数は、最大数よりも少なくなる場合がある。スレッド数は、ランタイム時に自動的に決定される。ユーザーは、OpenMP* の `omp_set_num_threads()` 関数を呼び出すことで、`OPM_NUM_THREADS` で指定されているスレッドの数を変更できる。

VML 数学関数

このセクションでは、単位増分を持つ実数および複素数のベクトル引数に基づいて数学関数の値を計算する VML 関数について説明する。それぞれの関数グループの説明では、短縮名と機能の簡単な説明を記載するとともに、Fortran インターフェイスと C インターフェイスの両方に対応した各データ型の呼び出しシーケンスを紹介し、入力/出力引数の説明も記載している。

すべての VML 数学関数において、パラメーターの入力範囲は、それぞれのデータ型に定められた一連の値を用いて定義した数学的な範囲に等しい。VML 関数の一部、具体的には、`Div`、`Exp`、`Sinh`、`Cosh`、`Pow` では、結果がオーバーフローする場合がある。これらの関数のそれぞれについて、オーバーフローとなるかどうかを振り分ける入力しきい値は、各関数の説明のセクションで指定する。その指定の際、`FLT_MAX` は単精度実数データ型で表現可能な最大数を表し、`DBL_MAX` は倍精度実数データ型で表現可能な最大数を表す。

表 9-2 は、使用可能な数学関数とそれに関連するデータタイプをまとめたものである。

表 9-2 VML 数学関数

関数	データ型	説明
累乗関数と累乗根関数		
Inv	s, d	ベクトル成分の逆数
Div	s, d	1つのベクトルの成分を2番目のベクトルの成分で割る
Sqrt	s, d, c, z	ベクトル成分の平方根
InvSqrt	s, d	ベクトル成分の平方根の逆数
Cbrt	s, d	ベクトル成分の立方根
InvCbrt	s, d	ベクトル成分の立方根の逆数
Pow	s, d, c, z	各ベクトル成分を指定された値で累乗する
Powx	s, d, c, z	各ベクトル成分を定数で累乗する
Hypot	s, d	二乗和の平方根
指数関数と対数関数		
Exp	s, d, c, z	ベクトル成分の指数
Ln	s, d, c, z	ベクトル成分の自然対数
Log10	s, d, c, z	ベクトル成分の10進対数
三角関数		
Cos	s, d, c, z	ベクトル成分の余弦
Sin	s, d, c, z	ベクトル成分の正弦
SinCos	s, d	ベクトル成分の正弦および余弦
Tan	s, d, c, z	ベクトル成分の正接
Acos	s, d, c, z	ベクトル成分の逆余弦
Asin	s, d, c, z	ベクトル成分の逆正弦
Atan	s, d, c, z	ベクトル成分の逆正接
Atan2	s, d	2つのベクトル成分の4象限逆正接
双曲線関数		
Cosh	s, d, c, z	ベクトル成分の双曲余弦
Sinh	s, d, c, z	ベクトル成分の双曲正弦
Tanh	s, d, c, z	ベクトル成分の双曲正接
Acosh	s, d, c, z	ベクトル成分の逆双曲余弦 (非負)
Asinh	s, d, c, z	ベクトル成分の逆双曲正弦
Atanh	s, d, c, z	ベクトル成分の逆双曲正接
特殊関数		
Erf	s, d	ベクトル成分の誤差関数値
Erfc	s, d	ベクトル成分の相補誤差関数値
ErfInv	s, d	ベクトル成分の逆誤差関数値
丸め関数		
Floor	s, d	マイナス無限大方向に丸める

表 9-2 VML 数学関数 (続き)

関数	データ型	説明
Ceil	s, d	プラス無限大方向に丸める
Trunc	s, d	ゼロ方向に丸める
Round	s, d	最も近い整数に丸める
NearbyInt	s, d	現在のモードに従って丸める
Rint	s, d	現在のモードに従って丸め、不正確な結果例外を引き上げる
Modf	s, d	整数と小数部

累乗関数と累乗根関数

Inv

ベクトルの成分単位で逆数を計算する。

構文

Fortran:

```
call vsinv( n, a, y )
call vdiv( n, a, y )
```

C:

```
vsInv( n, a, y );
vdInv( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsinv の場合) DOUBLE PRECISION, INTENT(IN) (vdiv の場合)	const float* (vsInv の場合) const double* (vdInv の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsinv の場合) DOUBLE PRECISION (vdiv の場合)	float* (vsInv の場合) double* (vdInv の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインター

Div

ベクトル a をベクトル b で対応成分ごとに割る。

構文

Fortran:

```
call vsdiv( n, a, b, y )
call vddiv( n, a, b, y )
```

C:

```
vsDiv( n, a, b, y );
vdDiv( n, a, b, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a, b	REAL, INTENT(IN) (vsdiv の場合) DOUBLE PRECISION, INTENT(IN) (vddiv の場合)	const float* (vsDiv の場合) const double* (vdDiv の場合)	Fortran: 入力ベクトル a と b を指定する配列 C: 入力ベクトル a と b を含む配列へのポインター

表 9-3 Div 関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$\text{abs}(a[i]) < \text{abs}(b[i]) * \text{FLT_MAX}$
倍精度	$\text{abs}(a[i]) < \text{abs}(b[i]) * \text{DBL_MAX}$

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsdiv の場合) DOUBLE PRECISION (vddiv の場合)	float* (vsDiv の場合) double* (vdDiv の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

Sqrt

ベクトル成分の平方根を計算する。

構文

Fortran:

```
call vssqrt( n, a, y )
call vdsqrt( n, a, y )
call vcsqrt( n, a, y )
call vzsqrt( n, a, y )
```

C:

```
vsSqrt( n, a, y );
vdSqrt( n, a, y );
vcSqrt( n, a, y );
vzSqrt( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vssqrt の場合) DOUBLE PRECISION, INTENT(IN) (vdsqrt の場合) COMPLEX, INTENT(IN) (vcsqrt の場合) DOUBLE COMPLEX, INTENT(IN) (vzsqrt の場合)	const float* (vsSqrt の場合) const double* (vdSqrt の場合) const MKL_Complex8* (vcSqrt の場合) const MKL_Complex16* (vzSqrt の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vssqrt の場合) DOUBLE PRECISION (vdsqrt の場合) COMPLEX (vcsqrt の場合) DOUBLE COMPLEX (vzsqrt の場合)	float* (vsSqrt の場 合) double* (vdSqrt の場 合) MKL_Complex8* (vcSqrt の場合) MKL_Complex16* (vzSqrt の場合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を含む配列へ のポインタ

InvSqrt

ベクトル成分の平方根の逆数を計算する。

構文

Fortran:

```
call vsinvsqrt( n, a, y )
call vdinvsqrt( n, a, y )
```

C:

```
vsInvSqrt( n, a, y );
vdInvSqrt( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsinvsqrt の場合) DOUBLE PRECISION, INTENT(IN) (vdinvsqrt の場合)	const float* (vsInvSqrt の場合) const double* (vdInvSqrt の場合)	Fortran: 入力ベクトル a を指 定する配列 C: 入力ベクトル a を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsinvsqrt の場合) DOUBLE PRECISION (vdinvsqrt の場合)	float* (vsInvSqrt の場合) double* (vdInvSqrt の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

Cbrt

ベクトル成分の立方根を計算する。

構文

Fortran:

```
call vsqrt( n, a, y )
```

```
call vdsqrt( n, a, y )
```

C:

```
vsqrt( n, a, y );
```

```
vdsqrt( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsqrt の場合) DOUBLE PRECISION, INTENT(IN) (vdsqrt の場合)	const float* (vsqrt の場合) const double* (vdsqrt の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsqrt の場合) DOUBLE PRECISION (vdsqrt の場合)	float* (vsqrt の場合) double* (vdsqrt の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

InvCbrt

ベクトル成分の立方根の逆数を計算する。

構文

Fortran:

```
call vsinvcbrt( n, a, y )
call vdinvcbrt( n, a, y )
```

C:

```
vsInvCbrt( n, a, y );
vdInvCbrt( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsinvcbrt の場合) DOUBLE PRECISION, INTENT(IN) (vdinvcbrt の場合)	const float* (vsInvCbrt の場合) const double* (vdInvCbrt の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vsinvcbrt の場合) DOUBLE PRECISION (vdinvcbrt の場合)	float* (vsInvCbrt の場合) double* (vdInvCbrt の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

Pow

2つのベクトルの成分について、 a の b 乗を計算する。

構文

Fortran:

```
call vspow( n, a, b, y )
call vdpow( n, a, b, y )
call vcpow( n, a, b, y )
call vzpow( n, a, b, y )
```

C:

```
vsPow( n, a, b, y );
vdPow( n, a, b, y );
vcPow( n, a, b, y );
vzPow( n, a, b, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a, b	REAL, INTENT(IN) (vspow の場合) DOUBLE PRECISION, INTENT(IN) (vdpow の場 合) COMPLEX, INTENT(IN) (vcpow の場合) DOUBLE COMPLEX, INTENT(IN) (vzpow の場 合)	const float* (vsPow の場合) const double* (vdPow の場合) const MKL_Complex8* (vcPow の場合) const MKL_Complex16* (vzPow の場合)	Fortran: 入力ベクトル a と b を指定する配列 C: 入力ベクトル a と b を含む 配列へのポインター

表 9-4 Pow 実数関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$\text{abs}(a[i]) < (\text{FLT_MAX})^{1/b[i]}$
倍精度	$\text{abs}(a[i]) < (\text{DBL_MAX})^{1/b[i]}$



注： Pow 複素数関数でオーバーフローが発生する可能性があるが、正確な式は本書の範囲外である。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vspow の場合) DOUBLE PRECISION (vdpow の場合) COMPLEX (vcpow の場合) DOUBLE COMPLEX (vzpow の場合)	float* (vsPow の場合) double* (vdPow の場 合) MKL_Complex8* (vcPow の場合) MKL_Complex16* (vzPow の場合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を含む配列へ のポインター

説明

実数関数 Pow には、パラメーター a と b の入力範囲について、一定の制限がある。具体的には、a[i] が正の場合、b[i] には任意の値を指定できる。しかし、a[i] が負の場合、b[i] の値は整数 (正または負) でなければならない。

複素数関数 Pow には、入力範囲の制限はない。

Powx

各ベクトル成分を定数で累乗する。

構文

Fortran:

```
call vspowx( n, a, b, y )
call vdpowx( n, a, b, y )
call vcpowx( n, a, b, y )
call vzpowx( n, a, b, y )
```

C:

```
vsPowx( n, a, b, y );
```

```
vdPowx( n, a, b, y );
```

```
vcPowx( n, a, b, y );
```

```
vzPowx( n, a, b, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vspowx の場合) DOUBLE PRECISION, INTENT(IN) (vdpowx の場 合) COMPLEX, INTENT(IN) (vcpowx の場合) DOUBLE COMPLEX, INTENT(IN) (vzpowx の場 合)	const float* (vsPowx の場合) const double* (vdPowx の場合) const MKL_Complex8* (vcPowx の場合) const MKL_Complex16* (vzPowx の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター
<i>b</i>	REAL, INTENT(IN) (vspowx の場合) DOUBLE PRECISION, INTENT(IN) (vdpowx の場 合) COMPLEX, INTENT(IN) (vcpowx の場合) DOUBLE COMPLEX, INTENT(IN) (vzpowx の場 合)	const float (vsPowx の場合) const double (vdPowx の場合) const MKL_Complex8* (vcPowx の場合) const MKL_Complex16* (vzPowx の場合)	Fortran: 定数の冪数として用 いるスカラー値 <i>b</i> C: 冪数として用いる定数 <i>b</i>

表 9-5 Powx 実数関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$\text{abs}(a[i]) < (\text{FLT_MAX})^{1/b}$
倍精度	$\text{abs}(a[i]) < (\text{DBL_MAX})^{1/b}$



注: Powx 複素数関数でオーバーフローが発生する可能性があるが、正確な式は本書の範囲外である。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vspowx の場合) DOUBLE PRECISION (vdpowx の場合) COMPLEX (vcpowx の場合) DOUBLE COMPLEX (vzpowx の場合)	float* (vsPowx の場 合) double* (vdPowx の場 合) MKL_Complex8* (vcPowx の場合) MKL_Complex16* (vzPowx の場合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を 含む配列へのポインタ

説明

実数関数 Powx には、パラメーター a と b の入力範囲について、一定の制限がある。具体的には、 $a[i]$ が正の場合、 $b[i]$ には任意の値を指定できる。しかし、 $a[i]$ が負の場合、 $b[i]$ の値は整数 (正または負) でなければならない。

複素数関数 Powx には、入力範囲の制限はない。

Hypot

2 つの二乗和の平方根を計算する。

構文

Fortran:

```
call vshypot( n, a, b, y )
call vdhypot( n, a, b, y )
```

C:

```
vsHypot( n, a, b, y );
vdHypot( n, a, b, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a, b	REAL, INTENT(IN) (vshypot の場合) DOUBLE PRECISION, INTENT(IN) (vdhypot の場 合)	const float* (vsHypot の場合) const double* (vdHypot の場合)	Fortran: 入力ベクトル a と b を指定する配列 C: 入力ベクトル a と b を含む 配列へのポインタ

表 9-6 Hypot 関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$\text{abs}(a[i]) < \text{sqrt}(\text{FLT_MAX})$ $\text{abs}(b[i]) < \text{sqrt}(\text{FLT_MAX})$
倍精度	$\text{abs}(a[i]) < \text{sqrt}(\text{DBL_MAX})$ $\text{abs}(b[i]) < \text{sqrt}(\text{DBL_MAX})$

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vshypot の場合) DOUBLE PRECISION (vdhypot の場合)	float* (vsHypot の場 合) double* (vdHypot の場 合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインター

指数関数と対数関数

Exp

ベクトル成分の指数を計算する。

構文

Fortran:

```
call vsexp( n, a, y )
call vdexp( n, a, y )
call vcexp( n, a, y )
call vzexp( n, a, y )
```

C:

```
vsExp( n, a, y );
vdExp( n, a, y );
vcExp( n, a, y );
vzExp( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsexp の場合) DOUBLE PRECISION, INTENT(IN) (vdexp の場合) COMPLEX, INTENT(IN) (vcexp の場合) DOUBLE COMPLEX, INTENT(IN) (vzexp の場合)	const float* (vsExp の場合) const double* (vdExp の場合) const MKL_Complex8* (vcExp の場合) const MKL_Complex16* (vzExp の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインタ

表 9-7 Exp 実数関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$a[i] < \text{Ln}(\text{FLT_MAX})$
倍精度	$a[i] < \text{Ln}(\text{DBL_MAX})$



注: Exp 複素数関数でオーバーフローが発生する可能性があるが、正確な式は本書の範囲外である。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsexp の場合) DOUBLE PRECISION (vdexp の場合) COMPLEX (vcexp の場合) DOUBLE COMPLEX (vzexp の場合)	float* (vsExp の場合) double* (vdExp の場合) MKL_Complex8* (vcExp の場合) MKL_Complex16* (vzExp の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインタ

Ln

ベクトル成分の自然対数を計算する。

構文

Fortran:

```
call vsln( n, a, y )
call vdln( n, a, y )
call vcln( n, a, y )
call vzln( n, a, y )
```

C:

```
vsLn( n, a, y );
vdLn( n, a, y );
vcLn( n, a, y );
vzLn( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsln の場合) DOUBLE PRECISION, INTENT(IN) (vdln の場合) COMPLEX, INTENT(IN) (vcln の場合) DOUBLE COMPLEX, INTENT(IN) (vzln の場合)	const float* (vsLn の場合) const double* (vdLn の場合) const MKL_Complex8* (vcLn の場合) const MKL_Complex16* (vzLn の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsln の場合) DOUBLE PRECISION (vdln の場合) COMPLEX (vcln の場合) DOUBLE COMPLEX (vzln の場合)	float* (vsLn の場合) double* (vdLn の場合) MKL_Complex8* (vcLn の場合) MKL_Complex16* (vzLn の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

Log10

ベクトル成分の10進対数を計算する。

構文

Fortran:

```
call vslog10( n, a, y )
call vdlog10( n, a, y )
call vclog10( n, a, y )
call vzlog10( n, a, y )
```

C:

```
vsLog10( n, a, y );
vdLog10( n, a, y );
vcLog10( n, a, y );
vzLog10( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vslog10 の場合) DOUBLE PRECISION, INTENT(IN) (vdlog10 の場 合) COMPLEX, INTENT(IN) (vclog10 の場合) DOUBLE COMPLEX, INTENT(IN) (vzlog10 の場 合)	const float* (vsLog10 の場合) const double* (vdLog10 の場合) const MKL_Complex8* (vcLog10 の場合) const MKL_Complex16* (vzLog10 の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vslog10 の場合) DOUBLE PRECISION (vdlog10 の場合) COMPLEX (vclog10 の場合) DOUBLE COMPLEX (vzlog10 の場合)	float* (vsLog10 の場 合) double* (vdLog10 の場 合) MKL_Complex8* (vcLog10 の場合) MKL_Complex16* (vzLog10 の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

三角関数

Cos

ベクトル成分の余弦を計算する。

構文

Fortran:

```
call vscol( n, a, y )
call vdcos( n, a, y )
call vccos( n, a, y )
call vzcos( n, a, y )
```

C:

```
vsCos( n, a, y );
vdCos( n, a, y );
vcCos( n, a, y );
vzCos( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数

名前	データ型		説明
	FORTRAN	C	
<i>a</i>	REAL, INTENT(IN) (<i>vscos</i> の場合) DOUBLE PRECISION, INTENT(IN) (<i>vdcos</i> の場合) COMPLEX, INTENT(IN) (<i>vccos</i> の場合) DOUBLE COMPLEX, INTENT(IN) (<i>vzcos</i> の場合)	const float* (<i>vsCos</i> の場合) const double* (<i>vdCos</i> の場合) const MKL_Complex8* (<i>vcCos</i> の場合) const MKL_Complex16* (<i>vzCos</i> の場合)	<i>Fortran</i> : 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (<i>vscos</i> の場合) DOUBLE PRECISION (<i>vdcos</i> の場合) COMPLEX (<i>vccos</i> の場合) DOUBLE COMPLEX (<i>vzcos</i> の場合)	float* (<i>vsCos</i> の場合) double* (<i>vdCos</i> の場合) MKL_Complex8* (<i>vcCos</i> の場合) MKL_Complex16* (<i>vzCos</i> の場合)	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインタ

Sin

ベクトル成分の正弦を計算する。

構文

Fortran:

```
call vssin( n, a, y )
call vdsin( n, a, y )
call vcsin( n, a, y )
call vzsine( n, a, y )
```

C:

```
vsSin( n, a, y );
vdSin( n, a, y );
vcSin( n, a, y );
vzSin( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vssin の場合) DOUBLE PRECISION, INTENT(IN) (vdsin の場合) COMPLEX, INTENT(IN) (vcsin の場合) DOUBLE COMPLEX, INTENT(IN) (vzsin の場合)	const float* (vsSin の場合) const double* (vdSin の場合) const MKL_Complex8* (vcSin の場合) const MKL_Complex16* (vzSin の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vssin の場合) DOUBLE PRECISION (vdsin の場合) COMPLEX (vcsin の場合) DOUBLE COMPLEX (vzsin の場合)	float* (vsSin の場合) double* (vdSin の場合) MKL_Complex8* (vcSin の場合) MKL_Complex16* (vzSin の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインター

SinCos

ベクトル成分の正弦および余弦を計算する。

構文

Fortran:

```
call vssincos( n, a, y, z )
call vdsincos( n, a, y, z )
```

C:

```
vsSinCos( n, a, y, z );
vdSinCos( n, a, y, z );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vssincos の場合) DOUBLE PRECISION, INTENT(IN) (vdsincos の 場合)	const float* (vsSinCos の場合) const double* (vdSinCos の場合)	Fortran: 入力ベクトル a を指 定する配列 C: 入力ベクトル a を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y, z	REAL (vssincos の場合) DOUBLE PRECISION (vdsincos の場合)	float* (vsSinCos の場 合) double* (vdSinCos の 場合)	Fortran: 出力ベクトル y (正弦 値) と出力ベクトル z (余弦値) を指定する配列 C: 出力ベクトル y (正弦値) と 出力ベクトル z (余弦値) を含む 配列へのポインタ

Tan

ベクトル成分の正接を計算する。

構文

Fortran:

```
call vstan( n, a, y )
call vdtan( n, a, y )
call vctan( n, a, y )
call vztan( n, a, y )
```

C:

```
vsTan( n, a, y );
vdTan( n, a, y );
vcTan( n, a, y );
vzTan( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vstan の場合) DOUBLE PRECISION, INTENT(IN) (vdtan の場合) COMPLEX, INTENT(IN) (vctan の場合) DOUBLE COMPLEX, INTENT(IN) (vztan の場合)	const float* (vsTan の場合) const double* (vdTan の場合) const MKL_Complex8* (vcTan の場合) const MKL_Complex16* (vzTan の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vstan の場合) DOUBLE PRECISION (vdtan の場合) COMPLEX (vctan の場合) DOUBLE COMPLEX (vztan の場合)	float* (vsTan の場合) double* (vdTan の場合) MKL_Complex8* (vcTan の場合) MKL_Complex16* (vzTan の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインター

Acos

ベクトル成分の逆余弦を計算する。

構文

Fortran:

```
call vsacos( n, a, y )
call vdacos( n, a, y )
call vcacos( n, a, y )
call vzacos( n, a, y )
```

C:

```
vsAcos( n, a, y );
vdAcos( n, a, y );
vcAcos( n, a, y );
vzAcos( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsacos の場合) DOUBLE PRECISION, INTENT(IN) (vdacos の場合) COMPLEX, INTENT(IN) (vcacos の場合) DOUBLE COMPLEX, INTENT(IN) (vzacos の場合)	const float* (vsAcos の場合) const double* (vdAcos の場合) const MKL_Complex8* (vcAcos の場合) const MKL_Complex16* (vzAcos の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsacos の場合) DOUBLE PRECISION (vdacos の場合) COMPLEX (vcacos の場合) DOUBLE COMPLEX (vzacos の場合)	float* (vsAcos の場合) double* (vdAcos の場合) MKL_Complex8* (vcAcos の場合) MKL_Complex16* (vzAcos の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインタ

Asin

ベクトル成分の逆正弦を計算する。

構文

Fortran:

```
call vsasin( n, a, y )
call vdasin( n, a, y )
call vcasin( n, a, y )
call vzasin( n, a, y )
```

C:

```
vsAsin( n, a, y );
vdAsin( n, a, y );
vcAsin( n, a, y );
```

```
vsAsin( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsasin の場合) DOUBLE PRECISION, INTENT(IN) (vdasin の場 合) COMPLEX, INTENT(IN) (vcasin の場合) DOUBLE COMPLEX, INTENT(IN) (vzasin の場 合)	const float* (vsAsin の 場合) const double* (vdAsin の場合) const MKL_Complex8* (vcAsin の場合) const MKL_Complex16* (vzAsin の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsasin の場合) DOUBLE PRECISION (vdasin の場合) COMPLEX (vcasin の場合) DOUBLE COMPLEX (vzasin の場合)	float* (vsAsin の場 合) double* (vdAsin の場 合) MKL_Complex8* (vcAsin の場合) MKL_Complex16* (vzAsin の場合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

Atan

ベクトル成分の逆正接を計算する。

構文

Fortran:

```
call vsatan( n, a, y )
call vdatan( n, a, y )
call vcatan( n, a, y )
call vzatan( n, a, y )
```

C:

```
vsAtan( n, a, y );
```

```
vdAtan( n, a, y );
vcAtan( n, a, y );
vzAtan( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsatan の場合) DOUBLE PRECISION, INTENT(IN) (vdatan の場 合) COMPLEX, INTENT(IN) (vcatan の場合) DOUBLE COMPLEX, INTENT(IN) (vzatan の場 合)	const float* (vsAtan の場合) const double* (vdAtan の場合) const MKL_Complex8* (vcAtan の場合) const MKL_Complex16* (vzAtan の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsatan の場合) DOUBLE PRECISION (vdatan の場合) COMPLEX (vcatan の場合) DOUBLE COMPLEX (vzatan の場合)	float* (vsAtan の場 合) double* (vdAtan の場 合) MKL_Complex8* (vcAtan の場合) MKL_Complex16* (vzAtan の場合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

Atan2

2 つのベクトル成分の 4 象限逆正接を計算する。

構文

Fortran:

```
call vsatan2( n, a, b, y )
call vdatan2( n, a, b, y )
```

C:

```
vsAtan2( n, a, b, y );
```

```
vdAtan2( n, a, b, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a, b</i>	REAL, INTENT(IN) (vsatan2 の場合) DOUBLE PRECISION, INTENT(IN) (vdatan2 の場 合)	const float* (vsAtan2 の場合) const double* (vdAtan2 の場合)	Fortran: 入力ベクトル <i>a</i> と <i>b</i> を指定する配列 C: 入力ベクトル <i>a</i> と <i>b</i> を含む 配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsatan2 の場合) DOUBLE PRECISION (vdatan2 の場合)	float* (vsAtan2 の場 合) double* (vdAtan2 の場 合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

説明

出力ベクトル *y* の各成分は、 $a[i] / b[i]$ の逆正接として計算される。入力の符号の組み合わせにより出力は 4 象限にわたる。

双曲線関数

Cosh

ベクトル成分の双曲余弦を計算する。

構文

Fortran:

```
call vscosh( n, a, y )
call vdcosh( n, a, y )
call vccosh( n, a, y )
call vzcosh( n, a, y )
```

C:

```
vsCosh( n, a, y );
vdCosh( n, a, y );
```

```
vcCosh( n, a, y );
vzCosh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vscosh の場合) DOUBLE PRECISION, INTENT(IN) (vdcosh の場 合) COMPLEX, INTENT(IN) (vccosh の場合) DOUBLE COMPLEX, INTENT(IN) (vzcosh の場 合)	const float* (vsCosh の 場合) const double* (vdCosh の場合) const MKL_Complex8* (vcCosh の場合) const MKL_Complex16* (vzCosh の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

表 9-8 Cosh 実数関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$-\text{Ln}(\text{FLT_MAX}) - \text{Ln}2 < a[i] < \text{Ln}(\text{FLT_MAX}) + \text{Ln}2$
倍精度	$-\text{Ln}(\text{DBL_MAX}) - \text{Ln}2 < a[i] < \text{Ln}(\text{DBL_MAX}) + \text{Ln}2$



注: Cosh 複素数関数でオーバーフローが発生する可能性があるが、正確な式は本書の範囲外である。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vscosh の場合) DOUBLE PRECISION (vdcosh の場合) COMPLEX (vccosh の場合) DOUBLE COMPLEX (vzcosh の場合)	float* (vsCosh の場 合) double* (vdCosh の場 合) MKL_Complex8* (vcCosh の場合) MKL_Complex16* (vzCosh の場合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

Sinh

ベクトル成分の双曲正弦を計算する。

構文

Fortran:

```
call vssinh( n, a, y )
call vdsinh( n, a, y )
call vcsinh( n, a, y )
call vzsinh( n, a, y )
```

C:

```
vsSinh( n, a, y );
vdSinh( n, a, y );
vcSinh( n, a, y );
vzSinh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vssinh の場合) DOUBLE PRECISION, INTENT(IN) (vdsinh の場 合) COMPLEX, INTENT(IN) (vcsinh の場合) DOUBLE COMPLEX, INTENT(IN) (vzsinh の場 合)	const float* (vsSinh の場合) const double* (vdSinh の場合) const MKL_Complex8* (vcSinh の場合) const MKL_Complex16* (vzSinh の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

表 9-9 Sinh 実数関数における各精度でのオーバーフローしきい値

データ型	入力パラメーターに対するしきい値制限
単精度	$-\ln(\text{FLT_MAX}) - \ln 2 < a[i] < \ln(\text{FLT_MAX}) + \ln 2$
倍精度	$-\ln(\text{DBL_MAX}) - \ln 2 < a[i] < \ln(\text{DBL_MAX}) + \ln 2$



注: Sinh 複素数関数でオーバーフローが発生する可能性があるが、正確な式は本書の範囲外である。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vssinh の場合) DOUBLE PRECISION (vdsinh の場合) COMPLEX (vcsinh の場合) DOUBLE COMPLEX (vzsinh の場合)	float* (vsSinh の場 合) double* (vdSinh の場 合) MKL_Complex8* (vcSinh の場合) MKL_Complex16* (vzSinh の場合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を含む配列へ のポインタ

Tanh

ベクトル成分の双曲正接を計算する。

構文

Fortran:

```
call vstanh( n, a, y )
call vdtanh( n, a, y )
call vctanh( n, a, y )
call vztanh( n, a, y )
```

C:

```
vsTanh( n, a, y );
vdTanh( n, a, y );
vcTanh( n, a, y );
vzTanh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数

名前	データ型		説明
	FORTRAN	C	
<i>a</i>	REAL, INTENT(IN) (vstanh の場合) DOUBLE PRECISION, INTENT(IN) (vdtanh の場 合) COMPLEX, INTENT(IN) (vctanh の場合) DOUBLE COMPLEX, INTENT(IN) (vztanh の場 合)	const float* (vsTanh の場 合) const double* (vdTanh の場合) const MKL_Complex8* (vcTanh の場合) const MKL_Complex16* (vzTanh の場合)	<i>Fortran</i> : 入力ベクトル <i>a</i> を指 定する配列 <i>C</i> : 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vstanh の場合) DOUBLE PRECISION (vdtanh の場合) COMPLEX (vctanh の場合) DOUBLE COMPLEX (vztanh の場合)	float* (vsTanh の場 合) double* (vdTanh の場 合) MKL_Complex8* (vcTanh の場合) MKL_Complex16* (vzTanh の場合)	<i>Fortran</i> : 出力ベクトル <i>y</i> を指定 する配列 <i>C</i> : 出力ベクトル <i>y</i> を含む配列へ のポインター

Acosh

ベクトル成分の逆双曲余弦を計算する。

構文

Fortran:

```
call vsacosh( n, a, y )
call vdacosh( n, a, y )
call vcacosh( n, a, y )
call vzacosh( n, a, y )
```

C:

```
vsAcosh( n, a, y );
vdAcosh( n, a, y );
vcAcosh( n, a, y );
vzAcosh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsacosh の場合) DOUBLE PRECISION, INTENT(IN) (vdacosh の場 合) COMPLEX, INTENT(IN) (vcacosh の場合) DOUBLE COMPLEX, INTENT(IN) (vzacosh の場 合)	const float* (vsAcosh の場合) const double* (vdAcosh の場合) const MKL_Complex8* (vcAcosh の場合) const MKL_Complex16* (vzAcosh の場合)	Fortran: 入力ベクトル a を指 定する配列 C: 入力ベクトル a を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsacosh の場合) DOUBLE PRECISION (vdacosh の場合) COMPLEX (vcacosh の場合) DOUBLE COMPLEX (vzacosh の場合)	float* (vsAcosh の場 合) double* (vdAcosh の場 合) MKL_Complex8* (vcAcosh の場合) MKL_Complex16* (vzAcosh の場合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を含む配列へ のポインター

Asinh

ベクトル成分の逆双曲正弦を計算する。

構文

Fortran:

```
call vsasinh( n, a, y )
call vdasinh( n, a, y )
call vcasinh( n, a, y )
call vzasinh( n, a, y )
```

C:

```
vsAsinh( n, a, y );
vdAsinh( n, a, y );
```

```
vcAsinh( n, a, y );
vzAsinh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsasinh の場合) DOUBLE PRECISION, INTENT(IN) (vdasinh の場 合) COMPLEX, INTENT(IN) (vcasinh の場合) DOUBLE COMPLEX, INTENT(IN) (vzasinh の場 合)	const float* (vsAsinh の場合) const double* (vdAsinh の場合) const MKL_Complex8* (vcAsinh の場合) const MKL_Complex16* (vzAsinh の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsasinh の場合) DOUBLE PRECISION (vdasinh の場合) COMPLEX (vcasinh の場合) DOUBLE COMPLEX (vzasinh の場合)	float* (vsAsinh の場 合) double* (vdAsinh の場 合) MKL_Complex8* (vcAsinh の場合) MKL_Complex16* (vzAsinh の場合)	Fortran: 出力ベクトル <i>y</i> を指 定する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

Atanh

ベクトル成分の逆双曲正接を計算する。

構文

Fortran:

```
call vsatanh( n, a, y )
call vdatanh( n, a, y )
call vcatanh( n, a, y )
call vzatanh( n, a, y )
```

C:

```
vsAtanh( n, a, y );
vdAtanh( n, a, y );
vcAtanh( n, a, y );
vzAtanh( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsatanh の場合) DOUBLE PRECISION, INTENT(IN) (vdatanh の場 合) COMPLEX, INTENT(IN) (vcatanh の場合) DOUBLE COMPLEX, INTENT(IN) (vzatanh の場 合)	const float* (vsAtanh の場合) const double* (vdAtanh の場合) const MKL_Complex8* (vcAtanh の場合) const MKL_Complex16* (vzAtanh の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsatanh の場合) DOUBLE PRECISION (vdatanh の場合) COMPLEX (vcatanh の場合) DOUBLE COMPLEX (vzatanh の場合)	float* (vsAtanh の場 合) double* (vdAtanh の場 合) MKL_Complex8* (vcAtanh の場合) MKL_Complex16* (vzAtanh の場合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインタ

特殊関数

Erf

ベクトル成分の誤差関数値を計算する。

構文

Fortran:

```
call vserf( n, a, y )
call vderf( n, a, y )
```

C:

```
vsErf( n, a, y );
vdErf( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vserf の場合) DOUBLE PRECISION, INTENT(IN) (vderf の場合)	const float* (vsErf の場合) const double* (vdErf の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vserf の場合) DOUBLE PRECISION (vderf の場合)	float* (vsErf の場合) double* (vdErf の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

説明

関数 Erf は、入力ベクトル *a* の成分に対して誤差関数値を計算し、結果を出力ベクトル *y* に書き出す。

誤差関数は次のように定義される。

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt .$$

Erfc

ベクトル成分の相補誤差関数値を計算する。

構文

Fortran:

```
call vserfc( n, a, y )
call vderfc( n, a, y )
```

C:

```
vsErfc( n, a, y );
vdErfc( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vserfc の場合) DOUBLE PRECISION, INTENT(IN) (vderfc の場 合)	const float* (vsErfc の場 合) const double* (vdErfc の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vserfc の場合) DOUBLE PRECISION (vderfc の場合)	float* (vsErfc の場 合) double* (vdErfc の場 合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインタ

説明

関数 Erfc は、入力ベクトル *a* の成分に対して相補誤差関数値を計算し、結果を出力ベクトル *y* に書き出す。

誤差関数は次のように定義される。

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$$

または書き換えると、

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt .$$

ErfInv

ベクトル成分の逆誤差関数値を計算する。

構文

Fortran:

```
call vserfinv( n, a, y )
call vderfinv( n, a, y )
```

C:

```
vsErfInv( n, a, y );
vdErfInv( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vserfinv の場合) DOUBLE PRECISION, INTENT(IN) (vderfinv の 場合)	const float* (vsErfInv の場合) const double* (vdErfInv の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vserfinv の場合) DOUBLE PRECISION (vderfinv の場合)	float* (vsErfInv の場 合) double* (vdErfInv の 場合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

説明

関数 ErfInv は、入力ベクトル *a* の成分に対して逆誤差関数値を計算し、結果を出力ベクトル *y* に書き出す。

逆誤差関数は次のように定義される。

$$\operatorname{erfinv}(x) = \operatorname{erf}^{-1}(x)$$

ここで、 $\operatorname{erf}(x)$ は、次のように定義された誤差関数を表す。

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

丸め関数

Floor

各ベクトル成分に対してマイナス無限大方向の整数に丸めた値を計算する。

構文

Fortran:

```
call vsfloor( n, a, y )
call vdfloor( n, a, y )
```

C:

```
vsFloor( n, a, y );
vdFloor( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsfloor の場合) DOUBLE PRECISION, INTENT(IN) (vdfloor の場合)	const float* (vsFloor の場合) const double* (vdFloor の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsfloor の場合) DOUBLE PRECISION (vdfloor の場合)	float* (vsFloor の場合) double* (vdFloor の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインター

Ceil

各ベクトル成分に対してプラス無限大方向の整数に丸めた値を計算する。

構文

Fortran:

```
call vsceil( n, a, y )
```

```
call vdceil( n, a, y )
```

C:

```
vsCeil( n, a, y );
```

```
vdCeil( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsceil の場合) DOUBLE PRECISION, INTENT(IN) (vdceil の場合)	const float* (vsCeil の場合) const double* (vdCeil の場合)	Fortran: 入力ベクトル <i>a</i> を指定する配列 C: 入力ベクトル <i>a</i> を含む配列へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>y</i>	REAL (vsceil の場合) DOUBLE PRECISION (vdceil の場合)	float* (vsCeil の場合) double* (vdCeil の場合)	Fortran: 出力ベクトル <i>y</i> を指定する配列 C: 出力ベクトル <i>y</i> を含む配列へのポインター

Trunc

各ベクトル成分に対してゼロ方向の整数に丸めた値を計算する。

構文

Fortran:

```
call vstrunc( n, a, y )
```

```
call vdtrunc( n, a, y )
```

C:

```
vsTrunc( n, a, y );
```

```
vdTrunc( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsTrunc の場合) DOUBLE PRECISION, INTENT(IN) (vdtrunc の場 合)	const float* (vsTrunc の場合) const double* (vdTrunc の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsTrunc の場合) DOUBLE PRECISION (vdtrunc の場合)	float* (vsTrunc の場 合) double* (vdTrunc の場 合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインタ

Round

各ベクトル成分に対して最も近い整数に丸めた値を計算する。

構文

Fortran:

```
call vsround( n, a, y )
```

```
call vdround( n, a, y )
```

C:

```
vsRound( n, a, y );
```

```
vdRound( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsround の場合) DOUBLE PRECISION, INTENT(IN) (vdround の場 合)	const float* (vsRound の場合) const double* (vdRound の場合)	Fortran: 入力ベクトル a を指 定する配列 C: 入力ベクトル a を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsround の場合) DOUBLE PRECISION (vdround の場合)	float* (vsRound の場 合) double* (vdRound の場 合)	Fortran: 出力ベクトル y を指定 する配列 C: 出力ベクトル y を含む配列へ のポインター

説明

0.5 や -1.5 のような中間の値は、ゼロから遠ざかって丸められる。つまり、0.5 は 1、-1.5 は -2 に丸められる。

NearbyInt

各ベクトル成分に対して現在の丸めモードで丸めた整数値を計算する。

構文

Fortran:

```
call vsnearbyint( n, a, y )
call vdnearbyint( n, a, y )
```

C:

```
vsNearbyInt( n, a, y );
vdNearbyInt( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsnearbyint の場合) DOUBLE PRECISION, INTENT(IN) (vdnearbyint の場合)	const float* (vsNearbyInt の場合) const double* (vdNearbyInt の場合)	Fortran: 入力ベクトル a を指定する配列 C: 入力ベクトル a を含む配列へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vsnearbyint の場合) DOUBLE PRECISION (vdnearbyint の場合)	float* (vsNearbyInt の場合) double* (vdNearbyInt の場合)	Fortran: 出力ベクトル y を指定する配列 C: 出力ベクトル y を含む配列へのポインタ

説明

0.5 や -1.5 のような中間の値は、ゼロ方向に (端数を切り捨てて) 丸められる。

Rint

各ベクトル成分に対して現在の丸めモードで丸めた整数値を計算し、変更された値に対して不正確な結果例外処理を起動する。

構文

Fortran:

```
call vsrint( n, a, y )
call vdrint( n, a, y )
```

C:

```
vsRint( n, a, y );
vdRint( n, a, y );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	計算する成分の数
<i>a</i>	REAL, INTENT(IN) (vsrint の場合) DOUBLE PRECISION, INTENT(IN) (vdrint の場 合)	const float* (vsRint の場合) const double* (vdRint の場合)	Fortran: 入力ベクトル <i>a</i> を指 定する配列 C: 入力ベクトル <i>a</i> を含む配列 へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsrint の場合) DOUBLE PRECISION (vdrint の場合)	float* (vsRint の場 合) double* (vdRint の場 合)	Fortran: 出力ベクトル <i>y</i> を指定 する配列 C: 出力ベクトル <i>y</i> を含む配列へ のポインター

説明

0.5 や -1.5 のような中間の値は、ゼロ方向に (端数を切り捨てて) 丸められる。変更された値に対して不正確な結果例外処理を起動する。

Modf

各ベクトル成分に対して切り捨てられた整数値と残りの小数部を計算する。

構文

Fortran:

```
call vsmodf( n, a, y, z )
call vdmodf( n, a, y, z )
```

C:

```
vsModf( n, a, y, z );
vdModf( n, a, y, z );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vsmodf の場合) DOUBLE PRECISION, INTENT(IN) (vdmodf の場 合)	const float* (vsModf の 場合) const double* (vdModf の場合)	Fortran: 入力ベクトル a を指 定する配列 C: 入力ベクトル a を含む配列 へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y, z	REAL (vsmodf の場合) DOUBLE PRECISION (vdmodf の場合)	float* (vsModf の場 合) double* (vdModf の場 合)	Fortran: 出力ベクトル y と z を 指定する配列 C: 出力ベクトル y と z を含む配 列へのポインタ

VML Pack/Unpack 関数

このセクションでは、単位増分を使用するベクトルから正増分インデックス、ベクトル・インデックス、マスク・インデックスを使用するベクトルへの変換、およびその逆の変換を実行する VML 関数について説明する（ベクトル・インデックス方式の詳細は、付録 B の「[ルーチンおよび関数の引数](#)」を参照のこと）。

[表 9-10](#) は、使用可能な VML Pack/Unpack 関数とそれに関連するデータ型とインデックス方式をまとめたものである。

表 9-10 VML Pack/Unpack 関数

関数の短縮名	データ型	インデックス		説明
		方式		
Pack	s, d	I, V, M		各種の方式でインデックス化されている配列の成分を収集する。
Unpack	s, d	I, V, M		各種のインデックスによって配列にベクトル成分を分散する。

Pack

指定されたインデックスを使用している配列の成分を、単位増分を使用するベクトルにコピーする。

構文

Fortran:

```
call vspacki( n, a, inca, y )
call vspackv( n, a, ia, y )
call vspackm( n, a, ma, y )
call vdpacki( n, a, inca, y )
call vdpackv( n, a, ia, y )
call vdpackm( n, a, ma, y )
```

C:

```
vsPackI( n, a, inca, y );
vsPackV( n, a, ia, y );
vsPackM( n, a, ma, y );
vdPackI( n, a, inca, y );
vdPackV( n, a, ia, y );
vdPackM( n, a, ma, y );
```

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数
a	REAL, INTENT(IN) (vspacki, vspackv, vspackm の場合) DOUBLE PRECISION, INTENT(IN) (vdpacki, vdpackv, vdpackm の場合)	const float* (vsPackI, vsPackV, vsPackM の場合) const double* (vdPackI, vdPackV, vdPackM の場合)	<i>Fortran</i> : 配列。次元は、 $(1 + (n-1) \cdot inca)$ 以上 (vspacki/vdpacki の場合) $\max(n, \max(ia[j]))$, $j=0, \dots, n-1$ 以上 (vspackv/vdpackv の場合) n 以上 (vspackm/vdpackm の場合) 入力ベクトル a を指定する <i>C</i> : 入力ベクトル a を含む配列へのポインタ。配列のサイズは、 $(1 + (n-1) \cdot inca)$ 以上 (vsPackI/vdPackI の場合) $\max(n, \max(ia[j]))$, $j=0, \dots, n-1$ 以上 (vsPackV/vdPackV の場合) n 以上 (vsPackM/vdPackM の場合)
$inca$	INTEGER, INTENT(IN) (vspacki, vdpacki の場合)	int (vsPackI, vdPackI の場合)	a の成分の増分
ia	INTEGER, INTENT(IN) (vspackv, vdpackv の場合)	const int* (vsPackV, vdPackV の場合)	<i>Fortran</i> : 配列。次元は n 以上。 a の成分に対するインデックス・ベクトルを指定する <i>C</i> : a の成分に対するインデックス・ベクトルを含むサイズが n 以上の配列へのポインタ
ma	INTEGER, INTENT(IN) (vspackm, vdpackm の場合)	const int* (vsPackM, vdPackM の場合)	<i>Fortran</i> : 配列。次元は n 以上。 a の成分に対するマスクベクトルを指定する <i>C</i> : a の成分に対するマスクベクトルを含むサイズが n 以上の配列へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
y	REAL (vspack i , vspack v , vspack m の場合) DOUBLE PRECISION (vdpack i , vdpack v , vdpack m の場合)	float* (vsPack I , vsPack V , vsPack M の場合) double* (vdPack I , vdPack V , vdPack M の場合)	Fortran: 配列。次元は n 以上。 出力ベクトル y を指定する C: 出力ベクトル y を含むサイズが n 以上の配列へのポインター

Unpack

単位増分を使用するベクトルの成分を、指定されたインデックスを使用した配列にコピーする。

構文

Fortran:

```
call vsunpacki( n, a, y, incy )
call vsunpackv( n, a, y, iy )
call vsunpackm( n, a, y, my )
call vdunpacki( n, a, y, incy )
call vdunpackv( n, a, y, iy )
call vdunpackm( n, a, y, my )
```

C:

```
vsUnpackI( n, a, y, incy );
vsUnpackV( n, a, y, iy );
vsUnpackM( n, a, y, my );
vdUnpackI( n, a, y, incy );
vdUnpackV( n, a, y, iy );
vdUnpackM( n, a, y, my );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
n	INTEGER, INTENT(IN)	int	計算する成分の数

名前	データ型		説明
	FORTRAN	C	
<i>a</i>	REAL, INTENT(IN) (vsunpacki, vsunpackv, vsunpackm の場合) DOUBLE PRECISION, INTENT(IN) (vdunpacki, vdunpackv, vdunpackm の場合)	const float* (vsUnpackI, vsUnpackV, vsUnpackM の場合) const double* (vdUnpackI, vdUnpackV, vdUnpackM の場合)	<i>Fortran</i> : 配列。次元は n 以上。入力ベクトル a を指定する <i>C</i> : 入力ベクトル a を含む配列へのポインタ
<i>incy</i>	INTEGER, INTENT(IN) (vsunpacki, vdunpacki の場合)	int (vsUnpackI, vdUnpackI の場合)	y の成分の増分を指定する
<i>iy</i>	INTEGER, INTENT(IN) (vsunpackv, vdunpackv の場合)	const int* (vsUnpackV, vdUnpackV の場合)	<i>Fortran</i> : 配列。次元は n 以上。 y の成分に対するインデックス・ベクトルを指定する <i>C</i> : y の成分に対するインデックス・ベクトルを含むサイズが n 以上の配列へのポインタ
<i>my</i>	INTEGER, INTENT(IN) (vsunpackm, vdunpackm の場合)	const int* (vsUnpackM, vdUnpackM の場合)。	<i>Fortran</i> : 配列。次元は n 以上。 y の成分に対するマスクベクトルを指定する <i>C</i> : y の成分に対するマスクベクトルを含むサイズが n 以上の配列へのポインタ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>y</i>	REAL (vsunpacki, vsunpackv, vsunpackm の場合) DOUBLE PRECISION (vdunpacki, vdunpackv, vdunpackm の場合)	float* (vsUnpackI, vsUnpackV, vsUnpackM の場合) double* (vdUnpackI, vdUnpackV, vdUnpackM の場合)	Fortran: 配列。次元は、 $(1 + (n-1)*incy)$ 以上 (vsunpacki/vdunpacki の場合) $\max(n, \max(iy[j]), j=0, \dots, n-1)$ 以上 (vsunpackv/vdunpackv の場合) n 以上 (vsunpackm/vdunpackm の場合) C: 出力ベクトル <i>y</i> を含む配列へのポインター。配列のサイズは、 $(1 + (n-1)*incy)$ 以上 (vsUnpackI/vdUnpackI の場合) $\max(n, \max(iy[j]), j=0, \dots, n-1)$ 以上 (vsUnpackV/vdUnpackV の場合) n 以上 (vsUnpackM/vdUnpackM の場合)

VML サービス関数

このセクションでは精度モードの設定/取得、エラーコードの設定/取得を行う VML 関数について説明する。これらの関数はすべて、Fortran インターフェイスと C インターフェイスの両方で使用できる。

表 9-11 は、使用可能な VML サービス関数とそれぞれについての簡単な説明をまとめたものである。

表 9-11 VML サービス関数

関数の短縮名	説明
SetMode	VML モードを設定する
GetMode	VML モードを取得する
SetErrStatus	VML エラーステータスを設定する
GetErrStatus	VML エラーステータスを取得する
ClearErrStatus	VML エラーステータスをクリアする
SetErrorCallBack	追加のエラー・ハンドラー・コールバック関数を設定する
GetErrorCallBack	追加のエラー・ハンドラー・コールバック関数を取得する
ClearErrorCallBack	追加のエラー・ハンドラー・コールバック関数を削除する

SetMode

mode パラメーターに従って VML 関数の新しいモードを設定し、前の VML モードを *oldmode* に保存する。

構文

Fortran:

```
oldmode = vmlsetmode( mode )
```

C:

```
oldmode = vmlSetMode( mode );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER, INTENT (IN)	int	設定する VML モード。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>oldmode</i>	INTEGER	int	前の VML モード

説明

mode パラメーターは、精度、FPU、エラー処理、スレッドオプションを制御する。[表 9-12](#) は、*mode* パラメーターの値をまとめたものである。*mode* パラメーターに指定できるその他の値はすべて、これらの値から得られる。その方法は、ビット単位 OR (|) 演算を使用して、精度用の値、FPU 用の値、エラー制御オプション用の値を各 1 つずつ組み合わせるものである。*mode* パラメーターのデフォルト値は、VML_HA | VML_ERRMODE_DEFAULT である。したがって、デフォルトでは、現在の FPU 制御ワード (FPU 精度および丸め方式) が使用される。

異なる FPU 精度または丸め方式が必要な場合、VML 数学関数は、これらのオプションを自動的に変更し、前の値を復元する。*mode* パラメーターを使用すると、同じ精度の設定で機能する各 VML 数学関数内の内部 FPU モードの切り替えを最小限に抑えられる。そのためには、*mode* パラメーターに VML_FLOAT_CONSISTENT (単精度実数および複素数関数の場合)、または VML_DOUBLE_CONSISTENT (倍精度実数および複素数関数の場合) を設定する。それぞれの関数グループで、これらの *mode* パラメーターの値は

最適な選択となる。なぜなら、これらの値は、ほとんどの VML 数学関数で必要だからである。前の FPU モードを復元する必要がある場合は、実行の完了後、*mode* に `VML_RESTORE` を設定する。

表 9-12 *mode* パラメーターの値

<i>mode</i> の値	説明
精度の制御	
<code>VML_HA</code>	VML 関数の高精度モードを使用する。
<code>VML_LA</code>	VML 関数の低精度モードを使用する。
追加 FPU モードの制御	
<code>VML_FLOAT_CONSISTENT</code>	単精度実数および複素数関数に最適な FPU モード (制御ワード) を設定し、前の FPU モードを保存する。
<code>VML_DOUBLE_CONSISTENT</code>	倍精度実数および複素数関数に最適な FPU モード (制御ワード) を設定し、前の FPU モードを保存する。
<code>VML_RESTORE</code>	前に保存した FPU モードを復元する。
エラーモードの制御	
<code>VML_ERRMODE_IGNORE</code>	計算エラーに対するアクションを設定しない。
<code>VML_ERRMODE_ERRNO</code>	エラー時に <code>errno</code> 変数を設定する。
<code>VML_ERRMODE_STDERR</code>	エラー時に、エラーテキスト情報を <code>stderr</code> に書き込む。
<code>VML_ERRMODE_EXCEPT</code>	エラー時に例外処理を起動する。
<code>VML_ERRMODE_CALLBACK</code>	エラー時に追加のエラーハンドラー関数を呼び出す。
<code>VML_ERRMODE_DEFAULT</code>	エラー時に <code>errno</code> 変数を設定し、例外処理を起動して、追加のエラーハンドラー関数を呼び出す。
スレッド化モードの制御	
<code>VML_NUM_THREADS_OMP_AUTO</code>	デフォルトの動作。スレッドの最大数は環境変数 <code>OMP_NUM_THREADS</code> によって決定される。OpenMP* 関数 <code>omp_set_num_threads()</code> で変更ができる。性能上の理由により、VML スレッド化ロジックは少ないスレッド数を使用する。
<code>VML_NUM_THREADS_OMP_FIXED</code>	スレッドの数は環境変数 <code>OMP_NUM_THREADS</code> によって決定される。OpenMP* 関数 <code>omp_set_num_threads()</code> で変更ができる。VML スレッド化ロジックを無効にするには、このモードを使用する。

例

mode パラメーターにさまざまな値を指定して関数 `vmlSetMode()` を呼び出す例をいくつか以下に示す。

Fortran:

```
oldmode = vmlsetmode( VML_LA )
call vmlsetmode( IOR(VML_LA, IOR(VML_FLOAT_CONSISTENT,
                               VML_ERRMODE_IGNORE )))
call vmlsetmode( VML_RESTORE )
call vmlsetmode( VML_NUM_THREADS_OMP_FIXED )
```

C:

```

vmlSetMode( VML_LA );
vmlSetMode( VML_LA | VML_FLOAT_CONSISTENT | VML_ERRMODE_IGNORE );
vmlSetMode( VML_RESTORE );
vmlSetMode( VML_NUM_THREADS_OMP_FIXED );

```

GetMode

VML モードを取得する。

構文

Fortran:

```
mod = vmlgetmode()
```

C:

```
mod = vmlGetMode( void );
```

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>mod</i>	INTEGER	int	合成された <i>mode</i> パラメーター

説明

関数 `vmlGetMode()` は、精度、FPU、エラー処理オプションを制御する VML *mode* パラメーターを戻す。*mod* 変数の値は、[表 9-12](#) で示した値の組み合わせで示される。[表 9-13](#) のそれぞれのマスクを使用すると、これらの値の一部を得られる。例を以下に示す。

Fortran:

```

mod = vmlgetmode()
accm = IAND(mod, VML_ACCURACY_MASK)
fpum = IAND(mod, VML_FPUMODE_MASK)
errm = IAND(mod, VML_ERRMODE_MASK)

```

C:

```

accm = vmlGetMode(void) & VML_ACCURACY_MASK;
fpum = vmlGetMode(void) & VML_FPUMODE_MASK;
errm = vmlGetMode(void) & VML_ERRMODE_MASK;

```

表 9-13 *mode* パラメーターに対するマスクの値

マスクの値	説明
VML_ACCURACY_MASK	精度 <i>mode</i> を選択する場合のマスクを指定する。
VML_FPUMODE_MASK	FPU <i>mode</i> を選択する場合のマスクを指定する。
VML_ERRMODE_MASK	エラー <i>mode</i> を選択する場合のマスクを指定する。

SetErrStatus

err に従って VML エラーステータスを設定し、前の VML エラーステータスを *olderr* に保存する。

構文

Fortran:

```
olderr = vmlseterrstatus( err )
```

C:

```
olderr = vmlSetErrStatus( err );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>err</i>	INTEGER, INTENT(IN)	int	設定する VML エラーステータス

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>olderr</i>	INTEGER	int	前の VML エラーステータス

表 9-14 は、*err* パラメーターの値をまとめたものである。

表 9-14 VML エラーステータスの値

エラーステータス	説明
VML_STATUS_OK	実行が正常に完了した。
VML_STATUS_BADSIZE	配列の次元が正でない。
VML_STATUS_BADMEM	NULL ポインターが渡された。
VML_STATUS_ERRDOM	配列の値の少なくとも 1 つが、定義の範囲外。
VML_STATUS_SING	配列の値の少なくとも 1 つが、特異点を引き起こした。
VML_STATUS_OVERFLOW	計算プロセスでオーバーフローが発生した。
VML_STATUS_UNDERFLOW	計算プロセスでアンダーフローが発生した。

例：

```
vmlSetErrStatus( VML_STATUS_OK );
vmlSetErrStatus( VML_STATUS_ERRDOM );
vmlSetErrStatus( VML_STATUS_UNDERFLOW );
```

GetErrStatus

VML エラーステータスを取得する。

構文**Fortran:**

```
err = vmlgeterrstatus( )
```

C:

```
err = vmlGetErrStatus( void );
```

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
err	INTEGER	int	VML エラーステータス

ClearErrStatus

VML エラーステータスに VML_STATUS_OK を設定し、前の VML エラーステータスを `olderr` に保存する。

構文

Fortran:

```
olderr = vmlclearerrstatus( )
```

C:

```
olderr = vmlClearErrStatus( void );
```

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<code>olderr</code>	INTEGER	int	前の VML エラーステータス

SetErrorCallback

追加のエラー・ハンドラー・コールバック関数を設定し、古いコールバック関数を取得する。

構文

Fortran:

```
oldcallback = vmlseterrorcallback( callback )
```

C:

```
oldcallback = vmlSetErrorCallBack( callback );
```

入力パラメーター

Fortran:

`callback` コールバック関数のアドレス。
コールバック関数は、以下の形式で示される。

```
INTEGER FUNCTION ERRFUNC(par)
TYPE (ERROR_STRUCTURE) par
! ...
! user error processing
! ...
ERRFUNC = 0
```

```

! if ERRFUNC = 0 - standard VML error handler
!   is called after the callback
! if ERRFUNC != 0 - standard VML error handler
!   is not called

END

```

渡されるエラー構造は、次のように定義される。

```

TYPE ERROR_STRUCTURE SEQUENCE
    INTEGER*4 ICODE
    INTEGER*4 IINDEX
    REAL*8 DBA1
    REAL*8 DBA2
    REAL*8 DBR1
    REAL*8 DBR2
    CHARACTER(64) CFUNCNAME
    INTEGER*4 IFUNCNAMELEN
END TYPE ERROR_STRUCTURE

```

C:

callback コールバック関数へのポインタ。
 コールバック関数は、以下の形式で示される。

```

static int __stdcall MyHandler(DefVmlErrorContext*
pContext)
{
    /* Handler body */
};

```

渡されるエラー構造は、次のように定義される。

```

typedef struct _DefVmlErrorContext
{
    int iCode;          /* Error status value */
    int iIndex;        /* Index for bad array
                        element, or bad array
                        dimension, or bad
                        array pointer */
    double dbA1;      /* Error argument 1 */
    double dbA2;      /* Error argument 2 */
    double dbR1;      /* Error result 1 */
    double dbR2;      /* Error result 2 */
    char cFuncName[64]; /* Function name */
    int iFuncNameLen; /* Length of function name*/
} DefVmlErrorContext;

```

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<code>oldcallback</code>	INTEGER	int	Fortran: 前のコールバック関数のアドレス C: 前のコールバック関数へのポインター

説明

VML_ERRMODE_CALLBACK エラーモードが設定されている場合、VML 数学関数のエラーが発生すると、そのたびにコールバック関数が呼び出される (表 9-12 を参照)。

デフォルトの空のコールバック関数ではなく、独自のコールバック関数を定義する必要がある場合は、`vmlSetErrorCallBack()` 関数を使用する。

コールバック関数の入力構造には、検出されたエラーに関して次の情報が含まれる。

- エラーを引き起こした入力値
- その値のロケーション (配列インデックス)
- 計算された結果値
- エラーコード
- エラーが発生した関数の名前

コールバック関数には、独自のエラー処理を組み込める。例えば、渡された結果値を訂正し、それを戻して計算を再開するような処理を組み込むことも可能である。コールバック関数の後に標準エラーハンドラーが呼び出されるのは、0 が戻された場合だけである。

GetErrorCallback

追加のエラー・ハンドラー・コールバック関数を取得する。

構文

Fortran:

```
callback = vmlgeterrorcallback( )
```

C:

```
callback = vmlGetErrorCallBack( void );
```

出力パラメーター

Fortran:

`callback` コールバック関数のアドレス。

C:
`callback` コールバック関数へのポインター。

ClearErrorCallback

追加のエラー・ハンドラー・コールバック関数を削除し、前のコールバック関数を回復する。

構文

Fortran:

```
oldcallback = vmlclearerrorcallback( )
```

C:

```
oldcallback = vmlClearErrorCallBack( void );
```

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<code>oldcallback</code>	INTEGER	int	Fortran: 前のコールバック関数のアドレス C: 前のコールバック関数へのポインター

本章では、ベクトル統計ライブラリー (VSL) として知られる、擬似乱数ベクトルの生成を目的として設計されたインテル[®] MKL の機能について説明する。

- 擬似乱数ベクトルと準乱数ベクトルの生成
- 畳み込み / 相関演算の実行

機能性については、「[乱数生成器](#)」セクションおよび「[畳み込みと相関](#)」セクションで説明する。

乱数生成器

VSL は基本的な連続分布または離散分布に対応した擬似 / 準乱数生成サブルーチン群で構成される。VSL ルーチンは高度に最適化された *基本乱数生成器* とベクトル数学関数ライブラリー VML を呼び出して最高レベルの性能を実現している (VML については、第 9 章の「[ベクトル数学関数](#)」を参照)。

VSL では Fortran インターフェイスと C インターフェイスの両方が用意されている。C および C++ 言語のユーザーには、`mk1_vsl.h` ヘッダーファイルが提供されている。FORTRAN-90 または FORTRAN-95 言語のユーザーには、`mk1_vsl.fi` ヘッダーファイルが提供されている。これらのヘッダーファイルは、次のディレクトリーにある。

```
$(MKL)/include
```

`mk1_vsl.fi` ヘッダーは FORTRAN の `include` 句から使用することを想定しており、F90/F95 ソースの両方の標準形式 (自由形式および 72 桁固定形式) と互換性がある。VSL インターフェイスを 80 桁または 132 桁固定形式ソースとともに使用する必要がある場合、新規ファイルをプロジェクトに追加する。このファイルは 72 桁固定形式ソースとして作成し、次のように単一の `include` 句を含む。

```
include 'mk1_vsl.fi'
```

この `include` 句により、コンパイラーは、モジュールファイル `mk1_vsl.mod` および `mk1_vsl_type.mod` を生成する。これらのファイルは、VSL インターフェイスを参照する FORTRAN の `use` 句を処理するために使用される。

```
use mk1_vsl_type
use mk1_vsl
```

この特殊な機能により、プロジェクトの各ソースに `mk1_vs1.fi` ヘッダーをインクルードする必要はない。ソースの一部にのみヘッダーをインクルードする必要がある。どのような場合でも、**VSL** インターフェイスに依存するソースがヘッダーをインクルードした後にコンパイルされている (モジュールファイル `mk1_vs1.mod` および `mk1_vs1_type.mod` がそれらを使用する前に生成されている) ことを確認すること。



注: FORTRAN インターフェイスでは、サブルーチン形式と関数形式のインターフェイスが提供される。デフォルトは関数形式である。サブルーチン形式は下位互換性のためにのみ提供されている。サブルーチン形式のインターフェイスを使用するには、手動で `include\mk1.fi` ファイル内の `include 'mk1_vs1.fi'` という行を `include 'mk1_vs1_subroutine.fi'` に変更し、`mk1_vs1.fi` ファイルの代わりに `mk1_vs1_subroutine.fi` ファイルを組み込む。

関数形式のインターフェイスでは、サブルーチン形式とは異なり、ユーザーは各ルーチンのエラーステータスを取得することができる。

VSL ルーチンは 3 種類のカテゴリーに分類できる。

- 一様分布、正規 (ガウス) 分布、二項分布など、さまざまなタイプの統計的分布に対応した変換ルーチン。これらのルーチンは、擬似乱数生成器または準乱数生成器を間接的に呼び出す。生成器の詳細は、「[分布生成器](#)」セクションで述べる。
- 乱数ストリームを取り扱うサービスルーチン。生成、初期化、削除、コピー、バイナリーファイルへの保存、バイナリーファイルからのロード、基本生成器のインデックス取得がある。ルーチンの詳細は、「[サービスルーチン](#)」セクションで述べる。
- 基本擬似乱数生成器を登録するルーチン、および登録された生成器のプロパティを取得するルーチン (詳細は、「[アドバンスト・サービス・ルーチン](#)」セクションを参照)。

カテゴリーの後半 2 つをサービスルーチンと呼ぶ。

規則

本章では、文脈上必要でない限り、真の乱数、擬似乱数、準乱数の間にも、真の乱数生成器、擬似乱数生成器、準乱数生成器の間にも、特定の相違はないものとする。詳細は、製品に付属の [VSL Notes](#) の「*Random Numbers*」セクションを参照のこと。

一様分布以外の各生成器は、離散分布と連続分布の両方とも、基本乱数生成器 (BRNG) と呼ばれる一様分布生成器を基に構築されている。一様分布以外の擬似乱数は、一様分布の擬似乱数にしかるべき変換を行って取得する。このような変換を *生成方法* と呼ぶ。一部の分布では複数の生成方法を選択できる。各生成器で利用可能な生成方法については、[VSL Notes](#) を参照のこと。

ストリーム・ディスクリプターは、与えられた変換方法で使用する BRNG を指定する。[VSL Notes](#) の「*Random Streams and RNGs in Parallel Computation*」セクションを参照のこと。

処理ノード は、データを並列処理する能力を持った論理的または物理的ユニットを意味する。

数学的表記

本文では次の表記を使用する。

N	自然数。 $N = \{1, 2, 3 \dots\}$ 。
Z	整数。 $Z = \{\dots -3, -2, -1, 0, 1, 2, 3 \dots\}$ 。
R	実数
$\lfloor a \rfloor$	a の切捨て値 (a よりも小さいか等しい最大の整数)
\oplus または xor	ビット単位の排他的 OR
C_{α}^k または $\binom{\alpha}{k}$	二項係数または組み合わせ ($\alpha \in R, \alpha \geq 0; k \in N \cup \{0\}$) $C_{\alpha}^0 = 1$ $\alpha \geq k$ の場合、二項係数は次のように定義される。

$$C_{\alpha}^k = \frac{\alpha(\alpha-1) \dots (\alpha-k+1)}{k!}$$

$\alpha < k$ の場合、次のようになる。

$$C_{\alpha}^k = 0$$

$\Phi(x)$	累積ガウス分布関数
-----------	-----------

$$\Phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right) dy$$

$$-\infty < x < +\infty$$

$$\Phi(-\infty) = 0, \Phi(+\infty) = 1$$

$\Gamma(\alpha)$	完全なガンマ関数
------------------	----------

$$\Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1} e^{-t} dt$$

$$\alpha > 0$$

$B(p, q)$	完全なベータ関数
-----------	----------

$$B(p, q) = \int_0^1 t^{p-1} (1-t)^{q-1} dt$$

$$p > 0 \text{ かつ } q > 0$$

$LCG(a, c, m)$	線形合同生成器 $x_{n+1} = (ax_n + c) \bmod m$ 。 ここで、 a は乗数、 c は加数、 m は生成器の係数である。
----------------	--

$MCG(a, m)$	乗算合同生成器 $x_{n+1} = (ax_n) \bmod m$ は、線形合同生成器の特別なケースである。ここで、加数 c は 0 である。
-------------	--

$GFSR(p, q)$	GFSR 乱数生成器 $x_n = x_{n-p} \oplus x_{n-q}$ 。
--------------	---

命名規則

FORTRAN 用の VSL 関数名はすべて小文字である。C 用の関数名は小文字と大文字で構成される。

生成器ルーチンの名前は、次の構造になっている。

`v<type of result>rng<distribution>` (FORTRAN インターフェイスの場合)

`v<type of result>Rng<distribution>` (C インターフェイスの場合)

ここで、`v` は VSL ベクトル関数のプリフィックスである。`<type of result>` フィールドは `s`、`d`、`i` のいずれかで、次の型から 1 つを指定する。

`s` REAL (FORTRAN インターフェイスの場合)

float (C インターフェイスの場合)

`d` DOUBLE PRECISION (FORTRAN インターフェイスの場合)

double (C インターフェイスの場合)

`i` INTEGER (FORTRAN インターフェイスの場合)

int (C インターフェイスの場合)

プリフィックス `s` と `d` は連続分布にのみ適用され、プリフィックス `i` は離散型にのみ適用される。プリフィックス `rng` はルーチンが乱数生成器であることを示し、`<distribution>` フィールドは統計的分布のタイプを示す。

サービス・サブルーチン名は次の規則に従う。

`vsl<name>`

ここで、`vsl` は VSL サービス関数のプリフィックスである。`<name>` フィールドには関数の短縮名が入る。サービスルーチンの詳細は、「[サービスルーチン](#)」セクションおよび「[アドバンスト・サービス・ルーチン](#)」セクションを参照のこと。

指定された確率分布に対応する各生成期ルーチンのプロトタイプは次の構造に適合する。

`<function name>(method, stream, n, r, [<distribution parameters>])`

ここで、

- `method` は生成方法を指定する数値である。このパラメーターの詳細は、「[分布生成器](#)」セクションに記載されている。`method` の名前の構造体定義については、次のページを参照のこと。
- `stream` はランダム・ストリーム・ディスクリプターを定義する。非ゼロでなければならない。ランダムストリームとその使い方は、後述する「[ランダムストリーム](#)」と「[サービスルーチン](#)」に記載されている。
- `n` は生成する乱数の個数を指定する。`n` がゼロかゼロ以下の場合、乱数は生成されない。また `n` が負の場合はエラー条件がセットされる。
- `r` は生成した乱数の出力先配列を指定する。配列の次元は、少なくとも `n` 個の乱数を格納できるだけの大きさがなくてはならない。

`<distribution parameters>` フィールドの追加パラメーターは生成器ルーチンごとに異なっており、詳細は「[分布生成器](#)」セクションに記載されている。

乱数生成器を呼び出すにはそれぞれの VSL ルーチンをコールする。例えば、平均値 `a`、標準偏差 `sigma` を持つ正規 (ガウス) 分布に従った `n` 個の独立かつ乱数としてみなされるベクトル `r` を取得するには、次のように記述する。

(FORTRAN インターフェイスの場合)

```
status = vsrnggaussian( method, stream, n, r, a, sigma )
```

(C インターフェイスの場合)

```
status = vsRngGaussian( method, stream, n, r, a, sigma )
```

method パラメーターの名前は、次の構造になっている。

```
VSL_METHOD_<precision><distribution>_<method>
VSL_METHOD_<precision><distribution>_<method>_ACCURATE
```

ここで、

```
<precision>      S  単精度連続分布
                  D  倍精度連続分布
                  I  離散分布

<distribution>   確率分布

<method>         生成方法の名前
```

method パラメーターの名前構造のタイプは、乱数生成の fast および accurate モードと対応する (詳細は、「[分布生成器](#)」セクションおよび [VSL Notes](#) を参照)。

VSL_METHOD_<precision><distribution>_<method> および
VSL_METHOD_<precision><distribution>_<method>_ACCURATE は、
vsl<precision>Rng<distribution> 関数とともにのみ使用する。

```
<precision>      s  単精度連続分布
                  d  倍精度連続分布
                  i  離散分布

<distribution>   確率分布
```

[表 10-1](#) は、定義済みの method の名前について説明する。3 番目の列は、その method を使用する関数の名前である。

表 10-1 method パラメーターの <method> の値

method	説明	関数
STD	標準生成法。現時点で、これらの関数に対する method は 1 つのみ。	Uniform (continuous)、 Uniform (discrete)、 UniformBits
BOXMULLER	BOXMULLER は、次の式に従い、一様分布の乱数 u_1 と u_2 のペアを介して正規分布に従った乱数 x を生成する。 $x = \sqrt{-2\ln u_1} \sin 2\pi u_2$	Gaussian 、 GaussianMV
BOXMULLER2	BOXMULLER2 は、次の式に従い、一様分布の乱数 u_1 と u_2 のペアを介して正規分布に従った乱数 x_1 と x_2 を生成する。 $x_1 = \sqrt{-2\ln u_1} \sin 2\pi u_2$ $x_2 = \sqrt{-2\ln u_1} \cos 2\pi u_2$	Gaussian 、 GaussianMV

表 10-1 *method* パラメーターの *<method>* の値 (続き)

method	説明	関数
ICDF	逆累積分布関数法	Exponential 、 Laplace 、 Weibull 、 Cauchy 、 Rayleigh 、 Lognormal 、 Gumbel 、 Bernoulli 、 Geometric
GNORM	$\alpha > 1$ の場合、ガンマ分布乱数は適切にスケールされた正規乱数の 3 乗として生成される。 $0.6 \leq \alpha < 1$ の場合、ガンマ分布乱数はワイブル分布からの棄却を使用して生成される。 $\alpha < 0.6$ の場合、ガンマ分布乱数は指数分布の変換を使用して取得される。 $\alpha = 1$ の場合、ガンマ分布は指数分布になる。	Gamma
CJA	$\min(p, q) > 1$ の場合、Cheng 法が使用される。 $\min(p, q) < 1$ の場合、Jöhnk 法 ($q + K \cdot p^2 + C \leq 0$ ($K = 0.852\dots$, $C = -0.956\dots$) の場合) または Atkinson 切り替えアルゴリズム (それ以外の場合) が使用される。 $\max(p, q) < 1$ の場合、Jöhnk 法が使用される。 $\min(p, q) < 1$ 、 $\max(p, q) > 1$ の場合、Atkinson 切り替えアルゴリズムが使用される (CJA は Cheng、Jöhnk、Atkinson の頭文字を示す)。 $p = 1$ または $q = 1$ の場合、逆累積分布関数法が使用される。 $p = 1$ かつ $q = 1$ の場合、ベータ分布は一様分布になる。	Beta
BTPE	$n_{\text{trial}} \cdot \min(p, 1 - p) \geq 30$ の場合の受容 / 棄却法。次の 4 つの領域に分解する。 - 2 つの平行四辺形 - 三角形 - 指数左裾 - 指数右裾	Binomial
H2PE	大規模な分布の受容 / 棄却法。次の 3 つの領域に分解する。 - 矩形 - 指数左裾 - 指数右裾	Hypergeometric
PTPE	$\lambda \geq 27$ の場合の受容 / 棄却法。次の 4 つの領域に分解する。 - 2 つの平行四辺形 - 三角形 - 指数左裾 - 指数右裾 それ以外の場合、テーブル・ルックアップ法が使用される。	Poisson
POISNORM	$\lambda \geq 1$ の場合、ガウス分布の逆累積分布関数によるポアソン分布の逆累積分布関数近似に基づく方法が使用される。 $\lambda < 1$ の場合、テーブル・ルックアップ法が使用される。	Poisson 、 PoissonV

表 10-1 *method* パラメーターの <method> の値 (続き)

method	説明	関数
NBAR	<p>以下の場合の受容 / 棄却法。 $\frac{(a-1) \cdot (1-p)}{p} \geq 100$</p> <p>次の 5 つの領域に分解する。 - 矩形 - 2 つの台形 - 指数左裾 - 指数右裾</p>	Negbinomial

基本生成器

VSL は、処理速度と特性の異なる次の基本乱数生成器 (BRNG) を提供する。

- 32 ビット乗算合同擬似乱数生成器 *MCG(1132489760, 2³¹ - 1)* [[L'Ecuyer99](#)]
- 32 ビット GFSR 擬似乱数生成器 *GFSR(250,103)* [[Kirkpatrick81](#)]
- 複合再帰擬似乱数生成器 *MRG-32k3a* [[L'Ecuyer99a](#)]
- NAG 数値ライブラリー [[NAG](#)] から 59 ビット乗算合同擬似乱数生成器 *MCG(13¹³, 2⁵⁹)*
- NAG 数値ライブラリー [[NAG](#)] から Wichmann-Hill 擬似乱数生成器 (実際は 273 個の基本生成器の集合)
- 生成されるシーケンスの周期の長さが 2¹⁹⁹³⁷-1 である Mersenne Twister 擬似乱数生成器 MT19937 [[Matsumoto98](#)]
- 1024 個の Mersenne Twister 擬似乱数生成器の集合 MT2203 [[Matsumoto98](#)]、[\[Matsumoto2000\]](#)。各生成器は、周期が 2²²⁰³-1 のシーケンスを生成する。生成器のパラメーターは、対応するシーケンスの相互独立を提供する。

これらの擬似乱数生成器に加え、VSL では基本準乱数生成器も 2 つ用意されている。

- 1 から 40 までの次元でデフォルトで機能する Sobol 準乱数生成器 [[Sobol76](#)]、[\[Bratley88\]](#)
- 1 から 318 までの次元でデフォルトで機能する Niederreiter 準乱数生成器 [[Bratley92](#)]

VSL は、外部的に定義された擬似乱数生成器の初期化パラメーターを登録し、ユーザーによって確立された次元で動作する機会も提供する。ライブラリーのパラメーターの登録に関するインターフェイスの詳細は、[VSL Notes](#) を参照のこと。

また、[VSL Notes](#) の生成器のテスト結果および

http://www.intel.com/software/products/mkl/data/vsl/vsl_performance_data.htm の相対性能データも参照のこと。

VSL では、「[アドバンスド・サービス・ルーチン](#)」セクションに記載のとおり、ユーザー定義の生成器を登録する手段を設けている。

一部の基本生成器では、マルチプロセッサ処理に対応して、複数の独立したランダムストリームを生成するリープフロッグ法とブロック分割法の 2 種類が提供される。これらのシーケンス分割法は、逐次モンテカルロ法にも役立つ。

また、MT2203 擬似乱数生成器は、1024 までの独立した乱数シーケンスを生成する、1024 個の生成器の集合である。この生成器は、並列モンテカルロ・シミュレーションで使用される。Wichmann-Hill 生成器も同様の機能を持つ。この生成器は、273 までの独立した乱数シーケンスを生成する。並列処理向けに設計された生成器の特性は、[\[Coddington94\]](#) に詳細が述べられている。

ユーザーは独自の基本生成器も作成できる。VSL では、「[アドバンスト・サービス・ルーチン](#)」セクションに記載のとおり、ユーザー定義の生成器を登録する手段を設けている。

また、外部で生成された乱数を VSL 分布生成器ルーチンで利用するためのオプションもある。この目的のために、VSL ではさらに 3 つの基本乱数生成器を提供している。

- 32 ビットの整数配列に格納された外部乱数データ用
- 倍精度の浮動小数点配列に格納された外部乱数データ用。データは、区間 (a,b) を持つ一様分布であるものとする。
- 単精度の浮動小数点配列に格納された外部乱数データ用。データは、区間 (a,b) を持つ一様分布であるものとする。

このような基本生成器を抽象基本生成器と呼ぶ。

生成器の特性に関する詳細は、[VSL Notes](#) を参照のこと。

BRNG パラメーターの定義

`brng` 入力パラメーターの定義済みの値を以下に示す。

表 10-2 `brng` パラメーターの値

値	説明
VSL_BRNG_MCG31	31 ビット乗算合同生成器
VSL_BRNG_R250	GFSR 乱数生成器。
VSL_BRNG_MRG32K3A	次数 3 のコンポーネントを 2 つ持つ複合再帰生成器
VSL_BRNG_MCG59	59 ビット乗算合同生成器
VSL_BRNG_WH	273 個の Wichmann-Hill 乗算合同生成器の集合
VSL_BRNG_MT19937	Mersenne Twister 擬似乱数生成器
VSL_BRNG_MT2203	1024 個の Mersenne Twister 擬似乱数生成器の集合
VSL_BRNG_SOBOL	不一致の少ないシーケンスを生成する 32 ビット Gray コードベース生成器。 次元は $1 \leq s \leq 40$ 。ユーザー定義の次元も利用できる。
VSL_BRNG_NIEDERR	不一致の少ないシーケンスを生成する 32 ビット Gray コードベース生成器。 次元は $1 \leq s \leq 318$ 。ユーザー定義の次元も利用できる。
VSL_BRNG_IABSTRACT	整数配列用の抽象乱数生成器
VSL_BRNG_DABSTRACT	倍精度の浮動小数点配列用の抽象乱数生成器
VSL_BRNG_SABSTRACT	単精度の浮動小数点配列用の抽象乱数生成器

詳細は、[VSL Notes](#) を参照のこと。

ランダムストリーム

ランダムストリーム (または ストリーム) は、一様分布の擬似乱数シーケンスおよび準乱数シーケンスの抽象的なソースである。これらのシーケンスに直接アクセスすることはできない。ストリーム・ステート・ディスクリプターを使用して操作する。ストリー

ム・ステート・ディスクリプターは、特定の BRNG のステート・ディスクリプティブ情報を格納し、分布生成器の各ルーチンで必須のパラメーターである。分布生成器ルーチンのみ、ランダムストリームで直接動作する。詳細は、[VSL Notes](#) を参照のこと。



注: 抽象乱数生成器に関連するランダムストリームを抽象ランダムストリームと呼ぶ。抽象ストリームとその使用に関する詳細は、[VSL Notes](#) を参照のこと。

[NewStream](#) のように、VSL [サービスルーチン](#) によって作成できるランダムストリームの数に制限はない。また、指定された確率分布の乱数ストリームを取得するために、任意の分布生成器で作成したランダムストリームを利用することができる。ストリームが必要でなくなった場合は、サービスルーチン [DeleteStream](#) を呼び出して、ストリームを削除しなければならない。

VSL では、ランダム・ストリーム・ディスクリプティブ・データをバイナリーファイルに保存したり、バイナリーファイルから読み取るために、サービスルーチン [SaveStreamF](#) と [LoadStreamF](#) を提供している。詳細は、[VSL Notes](#) を参照のこと。

データ型

FORTTRAN:

```
TYPE VSL_STREAM_STATE
    INTEGER*4 descriptor1
    INTEGER*4 descriptor2
END TYPE VSL_STREAM_STATE
```

C:

```
typedef (void*) VSLStreamStatePtr;
```

ユーザー定義生成器のストリームステート構造体のフォーマットは、「[アドバンスト・サービス・ルーチン](#)」セクションを参照のこと。

エラー報告

VSL ルーチンは、呼び出し元のプログラムにエラーおよび警告を報告するために、実行した演算のステータスコードを返す。このため、エラーに関連するアクションやエラーからの復元を実行するかどうかはアプリケーションによって決定される。ステータスコードは整数型で以下の形式で示される。

VSL_ERROR_<ERROR_NAME> - VSL エラーを示す。

VSL_WARNING_<WARNING_NAME> - VSL 警告を示す。

VSL エラーは負の値で、警告は正の値である。ゼロの場合は、演算が正常に終了したことを示す (VSL_ERROR_OK または VSL_STATUS_OK)。

表 10-3 ステータスコードとメッセージ

ステータスコード	メッセージ
VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	入力引数の値が不正である。
VSL_ERROR_NULL_PTR	入力ポインター引数が NULL である。
VSL_ERROR_MEM_FAILURE	システムがメモリーを割り当てることができない。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが不正である。
VSL_ERROR_BRNGS_INCOMPATIBLE	2 つの BRNG は、この演算では互換性がない。
VSL_ERROR_LEAPFROG_UNSUPPORTED	BRNG がリープフロッグ法をサポートしていない。
VSL_ERROR_SKIPAHEAD_UNSUPPORTED	BRNG が Skip-Ahead 法をサポートしていない。
VSL_ERROR_BAD_STREAM	ランダムストリームが無効である。
VSL_ERROR_FILE_OPEN	ファイルを開く際にエラーが発生したことを示す。
VSL_ERROR_FILE_READ	ファイルを読み取る際にエラーが発生したことを示す。
VSL_ERROR_FILE_WRITE	ファイルへ書き込む際にエラーが発生したことを示す。
VSL_ERROR_FILE_CLOSE	ファイルを閉じる際にエラーが発生したことを示す。
VSL_ERROR_BAD_FILE_FORMAT	ファイル形式が不明である。
VSL_ERROR_UNSUPPORTED_FILE_VER	サポートされていないファイル形式のバージョンである。
VSL_ERROR_BRNG_TABLE_FULL	登録されている BRNG の表にフリー成分が不足しているため、登録を完了することができない。
VSL_ERROR_BAD_STREAM_STATE_SIZE	StreamStateSize フィールドの値が不正である。
VSL_ERROR_BAD_WORD_SIZE	WordSize フィールドの値が不正である。
VSL_ERROR_BAD_NSEEDS	NSeeds フィールドの値が不正である。
VSL_ERROR_BAD_NBITS	NBits フィールドの値が不正である。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。
VSL_ERROR_INVALID_ABSTRACT_STREAM	抽象ランダムストリームが無効である。

サービスルーチン

ストリームを取り扱うルーチン群であり、ストリームの作成、削除、コピー、基本生成器のインデックス取得がある。ランダムストリームは、バイナリーファイルへ保存、またはバイナリーファイルから読み出すこともできる。[表 10-4](#) は、利用可能なサービス・サブルーチンの全リストを示す。

表 10-4 サービスルーチン

ルーチン	説明
NewStream	ランダムストリームを作成し初期化する。
NewStreamEx	複数の初期化条件を必要とする生成器に対し、ランダムストリームを作成し初期化する。
iNewAbstractStream	整数配列の抽象ランダムストリームを作成し、初期化する。
dNewAbstractStream	倍精度の浮動小数点配列の抽象ランダムストリームを作成し、初期化する。
sNewAbstractStream	単精度の浮動小数点配列の抽象ランダムストリームを作成し、初期化する。
DeleteStream	以前に生成したストリームを削除する。
CopyStream	別のストリームにコピーする。
CopyStreamState	ランダム・ストリーム・ステートのコピーを作成する。
SaveStreamF	ストリームをバイナリーファイルに書き込む。
LoadStreamF	ストリームをバイナリーファイルから読み取る。
LeapfrogStream	元のシーケンスのサブシーケンスを生成するため、ストリームをリープフロッグ法で初期化する。
SkipAheadStream	ストリームを Skip-Ahead 法で初期化する。
GetStreamStateBrng	指定のランダムストリームを生成した基本生成器のインデックスを取得する。
GetNumRegBrngs	現在登録されている基本生成器の数を取得する。



注：上記の表では関数名の vs1 プリフィックスを省略した。このプリフィックスは、関数のリファレンスでは関数プロトタイプとコード例とともに常に使用される。

生成器を用いた処理は、基本的な 3 つのステップで構成する。

1. ストリームを作成し、初期化する ([NewStream](#), [NewStreamEx](#), [CopyStream](#), [CopyStreamState](#), [LeapfrogStream](#), [SkipAheadStream](#))。
2. 指定された分布を用いて乱数を生成する。「[分布生成器](#)」を参照。
3. ストリームを削除する ([DeleteStream](#))。

ストリームは同時に複数を作成可能で、1 つまたは複数の生成器からストリームステートを用いて乱数データを取得できる。最後に [DeleteStream](#) 関数を用いてすべてのストリームを削除しなければならない。

NewStream

ランダムストリームを作成し初期化する。

構文

Fortran:

```
status = vslnewstream( stream, brng, seed )
```

C:

```
status = vslNewStream( &stream, brng, seed );
```

説明

この関数は、*brng* 値を持つ基本生成器のストリームを新規に作成し、32ビットの種 (シード) を使用して初期化を行う。種とは、基本生成器 *brng* によって生成された特定のシーケンスを選択するのに使用する初期値である。また、この関数は、複数の初期化条件を伴う生成器にも適用可能である。基本生成器ごとのストリーム初期化の詳細は、[VSL Notes](#) を参照のこと。



注: この関数は、抽象基本乱数生成器には適用できない。整数、単精度、倍精度の外部乱数データを利用するには、それぞれ [vslNewAbstractStream](#)、[vslsNewAbstractStream](#)、または [vslNewAbstractStream](#) を使用する。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT(IN)	int	ストリームを初期化する基本生成器のインデックス値。特定の値については、 表 10-2 を参照のこと。
<i>seed</i>	INTEGER, INTENT(IN)	unsigned int	ストリームの初期条件。準乱数生成器では、 <i>seed</i> パラメーターは次元を設定するのに使用される。次元が <i>brng</i> のサポートする次元よりも大きい場合、または 1 よりも小さい場合、次元は 1 とみなされる。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリーム・ステート・ディスクリプター

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリーを割り当てることができない。

NewStreamEx

複数の初期化条件を使用する生成器でランダムストリームを作成し初期化する。

構文

Fortran:

```
status = vslnewstreamex( stream, brng, n, params )
```

C:

```
status = vslNewStreamEx( &stream, brng, n, params );
```

説明

この関数は、入力引数が複数の初期化パラメーターで構成される基本生成器に複数の初期条件を設定する。初期値は、基本生成器 *brng* によって生成された特定のシーケンスを選択するのに使用される。可能であれば、32 ビットの単一初期条件のみを扱う [NewStream](#) を、[vslNewStreamEx](#) に相当する関数として使用する。ただし、GFSR のステートテーブルの初期化には [vslNewStreamEx](#) を使用する。この問題の詳細な説明は、[VSL Notes](#) を参照のこと。

この関数は、ユーザーが定義した擬似乱数生成器の初期化パラメーターをライブラリーに渡すためにも使用できる。パラメーターを渡すための書式および VSL における登録の詳細は、[VSL Notes](#) を参照のこと。



注: この関数は、抽象基本乱数生成器には適用できない。整数、単精度、倍精度の外部乱数データを利用するには、それぞれ [vslNewAbstractStream](#)、[vslsNewAbstractStream](#)、または [vslNewAbstractStream](#) を使用する。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT(IN)	int	ストリームを初期化する基本生成器のインデックス値。特定の値については、 表 10-2 を参照のこと。

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	unsigned int	<i>params</i> に含まれる初期条件の数
<i>params</i>	INTEGER, INTENT(IN)	const unsigned int	基本生成器 <i>brng</i> がストリームの初期化に必要とする初期条件の配列。準乱数生成器では、次元を設定するのに <i>params</i> パラメーターの最初の成分が使用される。次元が <i>brng</i> のサポートする次元よりも大きい場合、または 1 よりも小さい場合、次元は 1 とみなされる。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリーム・ステート・ディスクリプター

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリーを割り当てることができない。

iNewAbstractStream

整数配列の抽象ランダムストリームを作成し、初期化する。

構文

Fortran:

```
status = vslinewabstractstream( stream, n, ibuf, icallback )
```

C:

```
status = vslNewAbstractStream( &stream, n, ibuf, icallback );
```

説明

この関数は、新規の抽象ストリームを作成し、整数配列 *ibuf* と、*ibuf* の内容を更新するユーザーのコールバック関数 *icallback* に関連付ける。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	配列 <i>ibuf</i> のサイズ
<i>ibuf</i>	INTEGER, INTENT(IN)	unsigned int*	<i>n</i> 個の 32 ビット整数の配列
<i>icallback</i>	次のノート を参照	次のノート を参照	Fortran: <i>ibuf</i> の更新に使用するコールバック関数のアドレス C: <i>ibuf</i> の更新に使用するコールバック関数へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリームステート構造体のディスクリプター

ノート:

Fortran におけるコールバック関数の形式:

```
INTEGER FUNCTION IUPDATEFUNC[C]( stream, n, ibuf, nmin, nmax, idx )
```

```
TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
```

```
INTEGER(KIND=4), INTENT( IN) :: n[reference]
```

```
INTEGER(KIND=4), INTENT( OUT) :: ibuf[reference](0:n-1)
```

```
INTEGER(KIND=4), INTENT( IN) :: nmin[reference]
```

```
INTEGER(KIND=4), INTENT( IN) :: nmax[reference]
```

```
INTEGER(KIND=4), INTENT( IN) :: idx[reference]
```

C におけるコールバック関数の形式:

```
int iUpdateFunc( VSLStreamStatePtr stream, int* n, unsigned int ibuf[],  
int* nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-5](#) は、コールバック関数のパラメーターの説明である。

表 10-5 *icallback* コールバック関数のパラメーター

パラメーター	説明
<i>stream</i>	抽象ストリーム・ディスクリプター
<i>n</i>	<i>ibuf</i> のサイズ
<i>ibuf</i>	ストリーム <i>stream</i> に関連付けられた乱数の配列

表 10-5 `icallback` コールバック関数のパラメーター

パラメーター	説明
<code>nmin</code>	更新する乱数の最小個数
<code>nmax</code>	更新できる乱数の最大個数
<code>idx</code>	更新を開始するサイクリック・バッファー <code>ibuf</code> の位置。 $0 \leq idx < n$ 。

戻り値

<code>VSL_ERROR_OK</code> , <code>VSL_STATUS_OK</code>	正常に終了したことを示す。
<code>VSL_ERROR_BAD_ARG</code>	パラメーター <code>n</code> が正ではない。
<code>VSL_ERROR_MEM_FAILURE</code>	システムが、 <code>stream</code> に対してメモリーを割り当てることができない。
<code>VSL_ERROR_NULL_PTR</code>	バッファーまたはコールバック関数のパラメーターが <code>NULL</code> ポインターである。

dNewAbstractStream

倍精度の浮動小数点配列の抽象ランダム
ストリームを作成し、初期化する。

構文**Fortran:**

```
status = vsldnewabstractstream( stream, n, dbuf, a, b, dcallback )
```

C:

```
status = vsldNewAbstractStream( &stream, n, dbuf, a, b, dcallback );
```

説明

この関数は、区間 (a,b) を持つ一様分布に従った乱数を格納する、倍精度の浮動小数点配列に対する新規の抽象ストリームを作成する。この関数は、ストリームを倍精度の配列 `dbuf` と、`dbuf` の内容を更新するユーザーのコールバック関数 `dcallback` に関連付ける。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<code>n</code>	INTEGER, INTENT(IN)	int	配列 <code>dbuf</code> のサイズ
<code>dbuf</code>	DOUBLE PRECISION, INTENT(IN)	double*	区間 (a,b) を持つ一様分布に従った、 <code>n</code> 個の倍精度の浮動小数点乱数の配列

名前	データ型		説明
	FORTTRAN	C	
<i>a</i>	DOUBLE PRECISION, INTENT(IN)	double	区間の開始 <i>a</i>
<i>b</i>	DOUBLE PRECISION, INTENT(IN)	double	区間の終了 <i>b</i>
<i>dcallback</i>	次のノート を参照	次のノート を参照	Fortran: 配列 <i>dbuf</i> の更新に使用したコールバック関数の アドレス C: 配列 <i>dbuf</i> の更新に使用したコールバック関数への ポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリーム状態構造体の ディスクリプター

ノート:

Fortran におけるコールバック関数の形式:

```
INTEGER FUNCTION DUPDATEFUNC[C]( stream, n, dbuf, nmin, nmax, idx )
```

```
TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
```

```
INTEGER(KIND=4), INTENT(IN) :: n[reference]
```

```
REAL(KIND=8), INTENT(OUT) :: dbuf[reference](0:n-1)
```

```
INTEGER(KIND=4), INTENT(IN) :: nmin[reference]
```

```
INTEGER(KIND=4), INTENT(IN) :: nmax[reference]
```

```
INTEGER(KIND=4), INTENT(IN) :: idx[reference]
```

C におけるコールバック関数の形式:

```
int dUpdateFunc( VSLStreamStatePtr stream, int* n, double dbuf[], int*  
nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-6](#) は、コールバック関数のパラメーターの説明である。

表 10-6 *dcallback* コールバック関数のパラメーター

パラメーター	説明
<i>stream</i>	抽象ストリーム・ディスクリプター
<i>n</i>	<i>dbuf</i> のサイズ
<i>dbuf</i>	ストリーム <i>stream</i> に関連付けられた乱数の配列
<i>nmin</i>	更新する乱数の最小個数
<i>nmax</i>	更新できる乱数の最大個数

表 10-6 dcallback コールバック関数のパラメーター

パラメーター	説明
<i>idx</i>	更新を開始するサイクリック・バッファー <i>dbuf</i> の位置。 $0 \leq idx < n$ 。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	パラメーター <i>n</i> が正ではない。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリーを割り当てることができない。
VSL_ERROR_NULL_PTR	バッファーまたはコールバック関数のパラメーターが NULL ポインターである。

sNewAbstractStream

単精度の浮動小数点配列の抽象ランダム
ストリームを作成し、初期化する。

構文

Fortran:

```
status = vslnewabstractstream( stream, n, sbuf, a, b, scallback )
```

C:

```
status = vslnNewAbstractStream( &stream, n, sbuf, a, b, scallback );
```

説明

この関数は、区間 (a,b) を持つ一様分布に従った乱数を格納する、単精度の浮動小数点配列に対する新規の抽象ストリームを作成する。この関数は、ストリームを単精度の配列 *sbuf* と、*sbuf* の内容を更新するユーザーのコールバック関数 *scallback* に関連付ける。

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>n</i>	INTEGER, INTENT(IN)	int	配列 <i>sbuf</i> のサイズ
<i>sbuf</i>	REAL, INTENT(IN)	float*	区間 (a,b) を持つ一様分布に従った、 <i>n</i> 個の単精度の浮動小数点乱数の配列
<i>a</i>	REAL, INTENT(IN)	float	区間の開始 <i>a</i>

名前	データ型		説明
	FORTRAN	C	
<i>b</i>	REAL, INTENT(IN)	float	区間の終了 <i>b</i>
<i>scallback</i>	次のノート を参照	次のノート を参照	Fortran: 配列 <i>sbuf</i> の更新に使用したコールバック関数のアドレス C: 配列 <i>sbuf</i> の更新に使用したコールバック関数へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE(VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	ストリームステート構造体のディスクリプター

ノート:

Fortran におけるコールバック関数の形式:

```
INTEGER FUNCTION SUPDATEFUNC[C]( stream, n, sbuf, nmin, nmax, idx )
```

```
TYPE(VSL_STREAM_STATE), POINTER :: stream[reference]
INTEGER(KIND=4), INTENT( IN)      :: n[reference]
REAL(KIND=4), INTENT( OUT)        :: sbuf[reference](0:n-1)
INTEGER(KIND=4), INTENT( IN)      :: nmin[reference]
INTEGER(KIND=4), INTENT( IN)      :: nmax[reference]
INTEGER(KIND=4), INTENT( IN)      :: idx[reference]
```

C におけるコールバック関数の形式:

```
int sUpdateFunc( VSLStreamStatePtr stream, int* n, float sbuf[], int*
    nmin, int* nmax, int* idx );
```

コールバック関数は、この関数によって実際に更新される配列の成分の数を返す。[表 10-7](#) は、コールバック関数のパラメーターの説明である。

表 10-7 *scallback* コールバック関数のパラメーター

パラメーター	説明
<i>stream</i>	抽象ストリーム・ディスクリプター
<i>n</i>	<i>sbuf</i> のサイズ
<i>sbuf</i>	ストリーム <i>stream</i> に関連付けられた乱数の配列
<i>nmin</i>	更新する乱数の最小個数
<i>nmax</i>	更新できる乱数の最大個数
<i>idx</i>	更新を開始するサイクリック・バッファー <i>sbuf</i> の位置。 $0 \leq idx < n$ 。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BAD_ARG	パラメーター <i>n</i> が正ではない。
VSL_ERROR_MEM_FAILURE	システムが、 <i>stream</i> に対してメモリーを割り当てることができない。
VSL_ERROR_NULL_PTR	バッファまたはコールバック関数のパラメーターが NULL ポインターである。

DeleteStream

ランダムストリームを削除する。

構文

Fortran:

```
status = vsldeletestream( stream )
```

C:

```
status = vslDeleteStream( &stream );
```

説明

この関数は、いずれかの初期化関数で作成されたランダムストリームを削除する。

入力/出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	<i>Fortran</i> : ストリーム・ステート・ディスクリプター。非ゼロであること。ストリームの削除に成功すると、ディスクリプターは無効になる。 <i>C</i> : ストリーム・ステート・ディスクリプター。非ゼロであること。ストリームの削除に成功するとポインターには NULL がセットされる。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> パラメーターが NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。

CopyStream

ランダムストリームをコピーする。

構文

Fortran:

```
status = vslcopystream( newstream, srcstream )
```

C:

```
status = vslCopyStream( &newstream, srcstream );
```

説明

この関数は、*srcstream* の完全なコピーを作成し、*newstream* のディスクリプターに格納する。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>srcstream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: コピーするストリームの ディスクリプター C: コピーするストリームステート 構造体へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>newstream</i>	TYPE (VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	コピーされたストリーム・ディス クリプター

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

srcstream パラメーターが NULL ポインターである。

VSL_ERROR_BAD_STREAM

srcstream が有効なランダムストリームでない。

VSL_ERROR_MEM_FAILURE

システムが、*newstream* に対してメモリーを割り当てることができない。

CopyStreamState

ランダム・ストリーム・ステートのコピーを作成する

構文

Fortran:

```
status = vslcopystreamstate( deststream, srcstream )
```

C:

```
status = vslCopyStreamState( deststream, srcstream );
```

説明

この関数は *srcstream* から既存の *deststream* ストリームへストリームステートをコピーする。両方のストリームは同一の基本生成器を使って作成されていなければならない。*deststream* ストリームを作成した BRNG インデックスが *srcstream* ストリームを作成した BRNG インデックスと異なっている場合はエラーメッセージが出力される。

[CopyStream](#) 関数は新規のストリームを作成して *srcstream* からストリームステートと他のデータをコピーするが、*CopyStreamState* 関数は *srcstream* ストリーム・ステート・データのみ作成済みの *deststream* ストリームにコピーする。

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>srcstream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : <i>srcstream</i> ストリームのステートがコピーされるコピー先ストリームのディスクリプター <i>C</i> : ステート構造体がコピーされるストリームステート構造体へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>deststream</i>	TYPE (VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr	<i>Fortran</i> : ステートがコピーされるストリームのディスクリプター <i>C</i> : ストリームステートがコピーされるストリームステート構造体へのポインター

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>srcstream</i> または <i>deststream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>srcstream</i> または <i>deststream</i> が有効なランダムストリームではない。
VSL_ERROR_BRNGS_INCOMPATIBLE	<i>srcstream</i> に関連する BRNG と <i>deststream</i> に関連する BRNG との間に互換性がない。

SaveStreamF

ランダム・ストリーム・ディスクリプティブ・データをバイナリーファイルに書き込む。

構文

Fortran:

```
errstatus = vslsavestreamf( stream, fname )
```

C:

```
errstatus = vslSaveStreamF( stream, fname );
```

説明

この関数は、ランダム・ストリーム・ディスクリプティブ・データをバイナリーファイルに書き込む。ランダム・ストリーム・ディスクリプティブ・データは、*fname* という名前バイナリーファイルに保存される。ランダムストリーム *stream* は、[NewStream](#) や [CopyStream](#) などのサービスルーチンによって作成された、有効なストリームでなければならない。ストリームをファイルに保存できない場合、*errstatus* は非ゼロである。[LoadStreamF](#) 関数を使用して、バイナリーファイルからランダムストリームを読み取ることができる。

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	ファイルへ書き込むランダムストリーム
<i>fname</i>	CHARACTER(*), INTENT(IN)	char*	<i>Fortran</i> : C 形式の NULL で終了する文字列として指定されたファイル名 <i>C</i> : Fortran 形式の文字列として指定されたファイル名

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<code>errstatus</code>	INTEGER	int	演算のエラーステータス

戻り値

<code>VSL_ERROR_OK, VSL_STATUS_OK</code>	正常に終了したことを示す。
<code>VSL_ERROR_NULL_PTR</code>	<code>fname</code> または <code>stream</code> が NULL ポインターである。
<code>VSL_ERROR_BAD_STREAM</code>	<code>stream</code> が有効なランダムストリームでない。
<code>VSL_ERROR_FILE_OPEN</code>	ファイルを開く際にエラーが発生したことを示す。
<code>VSL_ERROR_FILE_WRITE</code>	ファイルへ書き込む際にエラーが発生したことを示す。
<code>VSL_ERROR_FILE_CLOSE</code>	ファイルを閉じる際にエラーが発生したことを示す。
<code>VSL_ERROR_MEM_FAILURE</code>	システムが、内部ニーズに対してメモリーを割り当てることができない。

LoadStreamF

新規のストリームを作成して、ストリーム・ディスクリプティブ・データをバイナリーファイルから読み取る。

構文

Fortran:

```
errstatus = vslloadstreamf( stream, fname )
```

C:

```
errstatus = vslLoadStreamF( &stream, fname );
```

説明

この関数は、新規のストリームを作成して、ストリーム・ディスクリプティブ・データをバイナリーファイルから読み取る。`fname` という名前のバイナリーファイルから、ストリーム・ディスクリプティブ・データを使用して、新規のランダムストリームが作成される。(I/O エラーや無効ファイル形式などが原因で) ストリームが読み取れない場合、`errstatus` は非ゼロである。ランダムストリームをファイルに保存するには、[SaveStreamF](#) 関数を使用する。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>fname</i>	CHARACTER(*), INTENT(IN)	char*	<i>Fortran</i> : C 形式の NULL で終了する文字列として指定されたファイル名 <i>C</i> : Fortran 形式の文字列として指定されたファイル名

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(OUT)	VSLStreamStatePtr*	<i>Fortran</i> : 新規ランダムストリームのディスクリプター <i>C</i> : 新規ランダムストリームへのポインター
<i>errstatus</i>	INTEGER	int	演算のエラーステータス

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>fname</i> が NULL ポインターである。
VSL_ERROR_FILE_OPEN	ファイルを開く際にエラーが発生したことを示す。
VSL_ERROR_FILE_WRITE	ファイルへ書き込む際にエラーが発生したことを示す。
VSL_ERROR_FILE_CLOSE	ファイルを閉じる際にエラーが発生したことを示す。
VSL_ERROR_MEM_FAILURE	システムが、内部ニーズに対してメモリーを割り当てることができない。
VSL_ERROR_BAD_FILE_FORMAT	ファイル形式が不明である。
VSL_ERROR_UNSUPPORTED_FILE_VER	サポートされていないファイル形式のバージョンである。

LeapfrogStream

リープフロッグ法を用いてストリームを初期化する。

構文

Fortran:

```
status = vsllleapfrogstream( stream, k, nstreams )
```

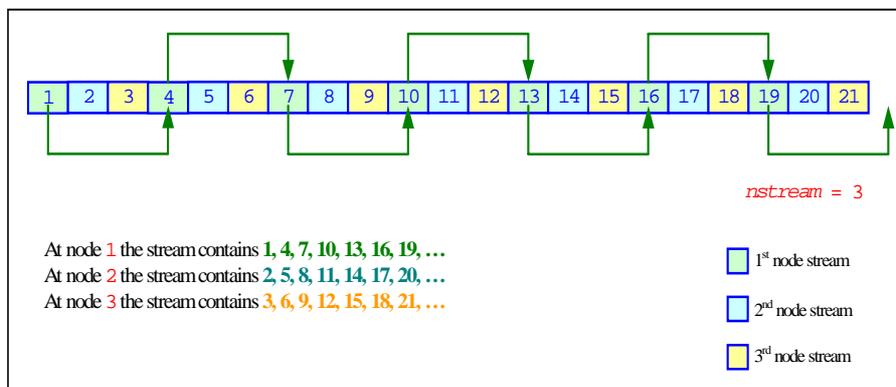
C:

```
status = vslLeapfrogStream( stream, k, nstreams );
```

説明

この関数は、単位ストライドではないランダムストリームで乱数を生成する。この機能は、元のランダムシーケンスの生成とそれに続くマニュアル分配をせずに、元のストリームから *nstreams* バッファに乱数を分配するのに特に役立つ。リープフロッグ法の重要な適用方法の1つは、元のシーケンスを *nstreams* 処理ノードにまたがるオーバーラップしないサブシーケンスに分割することである。この関数は、元のランダムストリーム ([図 10-1](#) を参照) を初期化し、処理ノード *k* ($0 \leq k < nstreams$) に対して乱数を生成する。ここで、*nstreams* は対象の処理ノードの最大値である。

図 10-1 リープフロッグ法



リープフロッグ法は、リープフロッグ法で成分の分割が可能な基本生成器に対してのみサポートされている。リープフロッグ法を使用すると、処理ノードへのマニュアル分配が伴う、生成器による生成よりも効率が良い。詳細は、[VSL Notes](#) を参照のこと。

準乱数基本生成器では、リープフロッグ法は準乱数ベクトル全体ではなく、個別のコンポーネントを生成することができる。この場合、*nstreams* パラメーターは準乱数ベクトルの次元に等しく、*k* パラメーターは生成されるコンポーネントのインデックスでなければならない ($0 \leq k < nstreams$)。他のパラメーター値は認められない。

以下に、リープフロッグ法を用いて3つの独立したストリームを初期化するコード例を示す。

例 10-1 リープフロッグ法の FORTRAN コード

```
...
TYPE(VSL_STREAM_STATE)      ::stream1
TYPE(VSL_STREAM_STATE)      ::stream2
TYPE(VSL_STREAM_STATE)      ::stream3

! Creating 3 identical streams
status = vslnewstream(stream1, VSL_BRNG_MCG31, 174)
status = vslcopystream(stream2, stream1)
status = vslcopystream(stream3, stream1)

! Leapfrogging the streams
status = vslleapfrogstream(stream1, 0, 3)
status = vslleapfrogstream(stream2, 1, 3)
status = vslleapfrogstream(stream3, 2, 3)

! Generating random numbers
...
! Deleting the streams
status = vsldeletestream(stream1)
status = vsldeletestream(stream2)
status = vsldeletestream(stream3)
...
```

例 10-2 リープフロッグ法の C コード

```
...
VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating 3 identical streams */
status = vslNewStream(&stream1, VSL_BRNG_MCG31, 174);
status = vslCopyStream(&stream2, stream1);
status = vslCopyStream(&stream3, stream1);

/* Leapfrogging the streams */
status = vslLeapfrogStream(stream1, 0, 3);
status = vslLeapfrogStream(stream2, 1, 3);
status = vslLeapfrogStream(stream3, 2, 3);

/* Generating random numbers */
...
/* Deleting the streams */
status = vslDeleteStream(&stream1);
status = vslDeleteStream(&stream2);
status = vslDeleteStream(&stream3);
...
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : リープフロッグ法を適用するストリームのディスクリプター <i>C</i> : リープフロッグ法を適用するストリームステート構造体へのポインター
<i>k</i>	INTEGER, INTENT(IN)	int	処理ノードのインデックス、またはストリーム番号
<i>nstreams</i>	INTEGER, INTENT(IN)	int	処理ノードまたはストライドの最大値

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_LEAPFROG_UNSUPPORTED	BRNG がリープフロッグ法をサポートしていない。

SkipAheadStream

ブロック分割法を用いてストリームを初期化する。

構文

Fortran:

```
status = vslskipaheadstream( stream, nskip )
```

C:

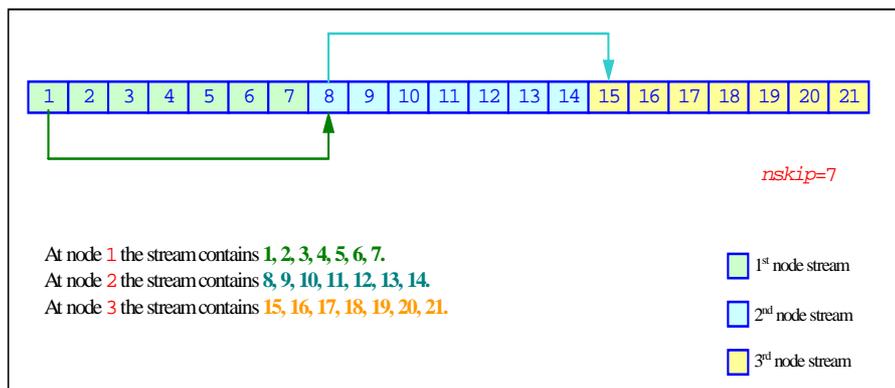
```
status = vslSkipAheadStream( stream, nskip );
```

説明

この関数は、ランダムストリームで指定された数の成分をスキップする。この機能は、元のランダムストリームから異なる処理ノードに乱数を分配するのに特に役立つ。処理ノードで使用される乱数の最大値が *nskip* の場合、元のランダムシーケンスは、各ブ

ロックがそれぞれの処理ノードに対応するように、SkipAheadStream によってサイズ n_{skip} のオーバーラップしないブロックに分割される。処理ノードの値に制限はない。これは、ブロック分割法または Skip-Ahead 法と呼ばれている (図 10-2 を参照)。

図 10-2 ブロック分割法



Skip-Ahead 法は、Skip-Ahead 法で成分をスキップすることができる基本生成器に対してのみサポートされている。Skip-Ahead 法を使用すると、処理ノードへのマニュアルスキップが伴う、生成器による生成よりも効率が良い。詳細は、[VSL Notes](#) を参照のこと。

準乱数基本生成器では、Skip-Ahead 法は準乱数ベクトル全体ではなく、コンポーネントとともに動作する点に注意する。このため、NS 準乱数ベクトルをスキップするには、 n_{skip} パラメーターを $NS \cdot DIMEN$ と等しくなるように設定する ($DIMEN$ は準乱数ベクトルの次元)。

以下に、SkipAheadStream 関数を用いて 3 つの独立したストリームを初期化するコード例を示す。

例 10-3 ブロック分割法の FORTRAN コード

```
...
TYPE(VSL_STREAM_STATE)    ::stream1
TYPE(VSL_STREAM_STATE)    ::stream2
TYPE(VSL_STREAM_STATE)    ::stream3

! Creating the 1st stream
status = vslnewstream(stream1, VSL_BRNG_MCG31, 174)

! Skipping ahead by 7 elements the 2nd stream
status = vslcopystream(stream2, stream1);
status = vslskipaheadstream(stream2, 7);

! Skipping ahead by 7 elements the 3rd stream
status = vslcopystream(stream3, stream2);
status = vslskipaheadstream(stream3, 7);

! Generating random numbers
...
! Deleting the streams
status = vsldeletestream(stream1)
status = vsldeletestream(stream2)
status = vsldeletestream(stream3)
...
```

例 10-4 ブロック分割法の C コード

```

VSLStreamStatePtr stream1;
VSLStreamStatePtr stream2;
VSLStreamStatePtr stream3;

/* Creating the 1st stream */
status = vslNewStream(&stream1, VSL_BRNG_MCG31, 174);

/* Skipping ahead by 7 elements the 2nd stream */
status = vslCopyStream(&stream2, stream1);
status = vslSkipAheadStream(stream2, 7);

/* Skipping ahead by 7 elements the 3rd stream */
status = vslCopyStream(&stream3, stream2);
status = vslSkipAheadStream(stream3, 7);

/* Generating random numbers */
...
/* Deleting the streams */
status = vslDeleteStream(&stream1);
status = vslDeleteStream(&stream2);
status = vslDeleteStream(&stream3);
...

```

入カパラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ブロック分割法を適用する ストリームのディスクリプター <i>C</i> : ブロック分割法を適用するスト リームステート構造体へのポイン ター
<i>nskip</i>	INTEGER, INTENT(IN)	int	スキップする成分数

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_SKIPAHEAD_UNSUPPORTED	BRNG が Skip-Ahead 法をサポートしていない。

GetStreamStateBrng

指定されたランダムストリームを生成した基本生成器のインデックス値を返す。

構文

Fortran:

```
brng = vslgetstreamstatebrng( stream )
```

C:

```
brng = vslGetStreamStateBrng( stream );
```

説明

この関数は、指定されたランダムストリームの生成に使用された基本生成器のインデックスを返す。

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: ストリームステート構造体のディスクリプター C: ストリームステート構造体へのポインター

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>brng</i>	INTEGER	int	<i>stream</i> の生成に使用された基本生成器のインデックス値。値が負の場合はエラーを表す。

戻り値

VSL_ERROR_NULL_PTR
VSL_ERROR_BAD_STREAM

stream が NULL ポインターである。
stream が有効なランダムストリームでない。

GetNumRegBrngs

現在登録されている基本生成器の数を取得する。

構文

Fortran:

```
nregbrngs = vslgetnumregbrngs( )
```

C:

```
nregbrngs = vslGetNumRegBrngs( void );
```

説明

この関数は現在登録されている基本生成器の数を取得する。ユーザーがユーザー定義生成器を登録すると、登録されている基本生成器の数はインクリメントされる。登録できる基本生成器の上限は `VSL_MAX_REG_BRNGS` パラメーターによって決まる。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>nregbrngs</i>	INTEGER	int	関数呼び出しを発生した時点で登録されている基本生成器の数

分布生成器

ここでは複数の分布タイプに対応した乱数生成 VSL ルーチンについて説明する。それぞれの関数は基本となる分布タイプによってグループに分けられ、機能概要に加えて FORTRAN と C インターフェイスの呼び出し手順仕様および入力と出力パラメーターの説明が記載される。乱数生成器ルーチンを、データ型、出力分布、生成器ルーチンのデータ型と呼び出された基本乱数生成器の対応のセットと併せて、[表 10-8](#) と [表 10-9](#) に一覧表示した。

表 10-8 連続分布生成器

分布のタイプ	データ型	BRNG データ型	説明
Uniform	s, d	s, d	区間 [a,b] に対する連続一様分布
Gaussian	s, d	s, d	正規 (ガウス) 分布
GaussianMV	s, d	s, d	他変量正規 (ガウス) 分布
Exponential	s, d	s, d	指数分布
Laplace	s, d	s, d	ラプラス分布 (両側指数分布)
Weibull	s, d	s, d	ワイブル分布
Cauchy	s, d	s, d	コーシー分布
Rayleigh	s, d	s, d	レイリー分布
Lognormal	s, d	s, d	対数正規分布
Gumbel	s, d	s, d	ガンベル (極値) 分布
Gamma	s, d	s, d	ガンマ分布
Beta	s, d	s, d	ベータ分布

表 10-9 離散分布生成器

分布のタイプ	データ型	BRNG データ型	説明
Uniform	i	d	区間 [a,b] に対する離散一様分布
UniformBits	i	i	一様ビット分布を持つ整数乱数生成器
Bernoulli	i	s	ベルヌーイ分布
Geometric	i	s	幾何分布
Binomial	i	d	二項分布
Hypergeometric	i	d	超幾何分布
Poisson	i	s (VSL_METHOD_IPOISSON_POISNORM の場合) s (分布パラメーターが $\lambda \geq 27$ の場合)、 d ($\lambda < 27$ の場合) (VSL_METHOD_IPOISSON_PTPE の場合)	ポアソン分布
PoissonV	i	s	多様な平均値を持つポアソン分布
Negbinomial	i	d	負の二項分布、またはパスカル分布

ライブラリーは、`accurate` と `fast` の 2 つの乱数生成モードを提供する。`accurate` 生成モードは、計算の正確性が高度に要求されるアプリケーション用に用意されている。このモードを使用すると、生成器は分布パラメーターのすべての値が完全に定義領域内に位置する乱数を生成する。例えば、区間 $[a,b]$ で一様な連続する分布の生成器から得られた乱数は、 a と b の値に関係なくこの区間に属する。`fast` モードは高性能な生成を提供し、生成された乱数が一部の特定の分布パラメーター値を除いて定義領域に属することを保証する。生成モードは、生成器ルーチンで適切な `method` パラメーターの値を指定して設定する。`accurate` モードをサポートしている分布の一覧を次の表に示す。

表 10-10 **accurate モードをサポートしている分布生成器**

分布のタイプ	データ型
Uniform	s, d
Exponential	s, d
Weibull	s, d
Rayleigh	s, d
Lognormal	s, d
Gamma	s, d
Beta	s, d

乱数番号生成の `accurate` および `fast` モードに関する詳細は、[VSL Notes](#) を参照のこと。

連続分布

連続分布の乱数を生成するルーチンについて説明する。

Uniform

一様分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnguniform( method, stream, n, r, a, b )
```

```
status = vdrnguniform( method, stream, n, r, a, b )
```

C:

```
status = vsRngUniform( method, stream, n, r, a, b );
```

```
status = vdRngUniform( method, stream, n, r, a, b );
```

説明

この関数は、区間 $[a, b]$ を持つ一様分布の乱数を生成する。ここで、 a, b はそれぞれ区間の始まりと終わりであり、 $a, b \in \mathbb{R}$ 、 $a > b$ 。

確率密度関数は次のように示される。

$$f_{a,b}(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & x \notin [a, b] \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}, \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SUNIFORM_STD VSL_METHOD_DUNIFORM_STD VSL_METHOD_SUNIFORM_STD _ACCURATE VSL_METHOD_DUNIFORM_STD _ACCURATE 標準生成法。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリーム状態構造体のディスクリプター <i>C</i> : ストリーム状態構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrnguniform の場合) DOUBLE PRECISION, INTENT(IN) (vdrnguniform の場合)	float (vsRngUniform の場合) double (vdRngUniform の場合)	区間の開始 <i>a</i>
<i>b</i>	REAL, INTENT(IN) (vsrnguniform の場合) DOUBLE PRECISION, INTENT(IN) (vdrnguniform の場合)	float (vsRngUniform の場合) double (vdRngUniform の場合)	区間の終了 <i>b</i>

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnguniform の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnguniform の場合)	float* (vsRngUniform の場合) double* (vdRngUniform の場合)	区間 [<i>a</i> , <i>b</i>] を持つ一様分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Gaussian

正規分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnggaussian( method, stream, n, r, a, sigma )
status = vdrnggaussian( method, stream, n, r, a, sigma )
```

C:

```
status = vsRngGaussian( method, stream, n, r, a, sigma );
status = vdRngGaussian( method, stream, n, r, a, sigma );
```

説明

この関数は、平均値 *a*、標準偏差 σ の正規 (ガウス) 分布の乱数を生成する。ここで、 $a, \sigma \in \mathbb{R}$ 、 $\sigma > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-a)^2}{2\sigma^2}\right), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\sigma}(x) = \int_{-\infty}^x \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2\sigma^2}\right) dy, \quad -\infty < x < +\infty$$

累積分布関数 $F_{a,\sigma}(x)$ は、標準的な正規分布 $\Phi(x)$ を用いて次のようにも表せる。

$$F_{a,\sigma}(x) = \Phi((x-a)/\sigma)$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAUSSIAN_BOX MULLER VSL_METHOD_SGAUSSIAN_BOX MULLER2 VSL_METHOD_DGAUSSIAN_BOX MULLER VSL_METHOD_DGAUSSIAN_BOX MULLER2 表 10-1 の生成方法 BOXMULLER と BOXMULLER2 の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)	float (vsRngGaussian の場合) double (vdRngGaussian の場合)	平均値 a
<i>sigma</i>	REAL, INTENT(IN) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussian の場合)	float (vsRngGaussian の場合) double (vdRngGaussian の場合)	標準偏差 σ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggaussian の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggaussian の場合)	float* (vsRngGaussian の場合) double* (vdRngGaussian の場合)	正規分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

GaussianMV

多変量正規分布から乱数を生成する。

構文

Fortran:

```
status = vsrnggaussianmv( method, stream, n, r, dimen, mstorage, a, t )
status = vdrnggaussianmv( method, stream, n, r, dimen, mstorage, a, t )
```

C:

```
status = vsRngGaussianMV( method, stream, n, r, dimen, mstorage, a, t );
status = vdRngGaussianMV( method, stream, n, r, dimen, mstorage, a, t );
```

説明

この関数は、平均値 *a*、分散共分散行列 *C* の *d*-変量正規 (ガウス) 分布の乱数を生成する。ここで、 $a \in \mathbb{R}^d$ 。 *C* は *d* x *d* 対称正定値行列。

確率密度関数は次のように示される。

$$f_{a,C}(x) = \frac{1}{\sqrt{\det(2\pi C)}} \exp(-1/2(x-a)^T C^{-1}(x-a))$$

ここで、 $x \in R^d$ 。

行列 C は $C = TT^T$ として表すことができる。ここで、 T は C のコレスキー因子である下三角行列。

分散共分散行列 C の代わりに、生成ルーチンは入力で C のコレスキー因子を要求する。行列 C のコレスキー因子を計算するには、行列の因子分解用の MKL LAPACK ルーチン呼び出すとよい。v?RngGaussianMV/v?rnggaussianmv ルーチンの場合、[?potrf](#) または [?pptrf](#) (? は、s (単精度) または d (倍精度) を意味する)。詳細は、「[アプリケーション・ノート](#)」を参照。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAUSSIANMV_BOXMULLER VSL_METHOD_SGAUSSIANMV_BOXMULLER2 VSL_METHOD_DGAUSSIANMV_BOXMULLER VSL_METHOD_DGAUSSIANMV_BOXMULLER2 表 10-1 の生成方法 BOXMULLER と BOXMULLER2 の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>dimen</i>	INTEGER, INTENT(IN)	int	出力乱数ベクトルの次元 d ($d \geq 1$)

名前	データ型		説明
	FORTRAN	C	
<i>mstorage</i>	INTEGER, INTENT(IN)	int	<p><i>Fortran</i>: 上三角行列 T^T の行列格納形式。このルーチンは 3 つの行列格納形式をサポートしている。</p> <ul style="list-style-type: none"> VSL_MATRIX_STORAGE_FULLL – 行列 T^T のすべての $d \times d$ 成分が渡されるが、実際にルーチンで使用されるのは上三角部分のみ。 VSL_MATRIX_STORAGE_PACKED – T^T の上三角成分が、行ごとに 1 次元配列に圧縮される。 VSL_MATRIX_STORAGE_DIAGONAL – T^T の対角成分のみ渡される。 <p>C: 下三角行列 T の行列格納形式。このルーチンは 3 つの行列格納形式をサポートしている。</p> <ul style="list-style-type: none"> VSL_MATRIX_STORAGE_FULLL – 行列 T のすべての $d \times d$ 成分が渡されるが、実際にルーチンで使用されるのは下三角部分のみ。 VSL_MATRIX_STORAGE_PACKED – T の下三角成分が、行ごとに 1 次元配列に圧縮される。 VSL_MATRIX_STORAGE_DIAGONAL – T の対角成分のみ渡される。
<i>a</i>	REAL, INTENT(IN) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の場合) double* (vdRngGaussianMV の場合)	次元 d の平均値ベクトル a
<i>t</i>	REAL, INTENT(IN) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の場合) double* (vdRngGaussianMV の場合)	<p><i>Fortran</i>: 上三角行列の成分は、行列 T^T の格納形式 <i>mstorage</i> に従って渡される。</p> <p>C: 下三角行列の成分は、行列 T の格納形式 <i>mstorage</i> に従って渡される。</p>

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggaussianmv の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggaussianmv の場合)	float* (vsRngGaussianMV の場合) double* (vdRngGaussianMV の場合)	次元 <i>dimen</i> の <i>n</i> 個の乱数ベクトルの配列

アプリケーション・ノート

行列は Fortran では列ごとに、C では行ごとに格納されるため、MKL 因子分解ルーチン (Fortran 行列格納形式) を多変量正規 RNG (C 行列格納形式) と一緒に使用する場合、C と Fortran では使用方法がやや異なる。次の表は、C と Fortran でこれらのルーチンを使用する際に役立つ。詳細は、適切な VSL example ファイルを参照のこと。

表 10-11 Fortran におけるコレスキー因子分解ルーチンの使用

行列の格納形式	分散共分散 行列引数	因子分解ルーチン	因子分解 ルーチンに おける UPLO パラメーター	RNG の入力 引数としての 因子分解の 結果
VSL_MATRIX_STORAGE_FULL	Fortran の 2 次元配列で の C	spotrf (vsrnggaussianmv の場合) dpotrf (vdrnggaussianmv の場合)	'U'	T^T の上三角。 下三角は使用 されない。
VSL_MATRIX_STORAGE_PACKED	列ごとに 1 次元配列に 圧縮された C の下三角 部分	spptrf (vsrnggaussianmv の場合) dpptrf (vdrnggaussianmv の場合)	'L'	列ごとに 1 次 元配列に圧縮 された T^T の 上三角部分

表 10-12 C におけるコレスキー因子分解ルーチンの使用

行列の格納形式	分散共分散 行列引数	因子分解ルーチン	因子分解 ルーチンに おける UPLO パラメーター	RNG の入力 引数としての 因子分解の 結果
VSL_MATRIX_STORAGE_FULLL	C の 2 次元 配列での C	spotrf (vsRngGaussianMV の場合) dpotrf (vdRngGaussianMV の場合)	'U'	T^T の上三角。 下三角は使用 されない。
VSL_MATRIX_STORAGE_PACKED	列ごとに 1 次元配列に 圧縮された C の下三角 部分	sppturf (vsRngGaussianMV の場合) dppturf (vdRngGaussianMV の場合)	'L'	列ごとに 1 次 元配列に圧縮 された T^T の 上三角部分

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Exponential

指数分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrngexponential( method, stream, n, r, a, beta )
```

```
status = vdrngexponential( method, stream, n, r, a, beta )
```

C:

```
status = vsRngExponential( method, stream, n, r, a, beta );
```

```
status = vdRngExponential( method, stream, n, r, a, beta );
```

説明

この関数は、位置母数 a 、尺度母数 β を持つ指数分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \beta}(x) = \begin{cases} \frac{1}{\beta} \exp(-(x-a)/\beta), & x \geq a \\ 0 & x < a \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a, \beta}(x) = \begin{cases} 1 - \exp(-(x-a)/\beta), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SEXPONENTIAL_ICDF VSL_METHOD_DEXPONENTIAL_ICDF VSL_METHOD_SEXPONENTIAL_ICDF _ACCURATE VSL_METHOD_DEXPONENTIAL_ICDF _ACCURATE 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体の ディスクリプター <i>C</i> : ストリームステート構造体へのポイン ター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrngexponential の場合) DOUBLE PRECISION, INTENT(IN) (vdrngexponential の場合)	float (vsRngExponential の場合) double (vdRngExponential の場合)	位置母数 a
<i>beta</i>	REAL, INTENT(IN) (vsrngexponential の場合) DOUBLE PRECISION, INTENT(IN) (vdrngexponential の場合)	float (vsRngExponential の場合) double (vdRngExponential の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngexponential の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngexponential の場合)	float* (vsRngExponential の場合) double* (vdRngExponential の場合)	指数分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Laplace

ラプラス分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnglaplace( method, stream, n, r, a, beta )
```

```
status = vdrnglaplace( method, stream, n, r, a, beta )
```

C:

```
status = vsRngLaplace( method, stream, n, r, a, beta );
```

```
status = vdRngLaplace( method, stream, n, r, a, beta );
```

説明

この関数は、平均値 a 、尺度母数 β を持つラプラス分布の乱数を生成する。ここで、 $a, \beta \in R$ 、 $\beta > 0$ 。
尺度母数から標準偏差が次のように決まる。

$$\sigma = \beta\sqrt{2}$$

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \frac{1}{\sqrt{2\beta}} \exp\left(-\frac{|x-a|}{\beta}\right), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} \frac{1}{2} \exp\left(-\frac{|x-a|}{\beta}\right), & x < a \\ 1 - \frac{1}{2} \exp\left(-\frac{|x-a|}{\beta}\right), & x \geq a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SLAPLACE_ICDF VSL_METHOD_DLAPLACE_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合)	float (vsRngLaplace の場合) double (vdRngLaplace の場合)	平均値 <i>a</i>
<i>beta</i>	REAL, INTENT(IN) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglaplace の場合)	float (vsRngLaplace の場合) double (vdRngLaplace の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnglaplace の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnglaplace の場合)	float* (vsRngLaplace の場合) double* (vdRngLaplace の場合)	ラプラス分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Weibull

ワイブル分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrngweibull( method, stream, n, r, alpha, a, beta )
status = vdrngweibull( method, stream, n, r, alpha, a, beta )
```

C:

```
status = vsRngWeibull( method, stream, n, r, alpha, a, beta );
status = vdRngWeibull( method, stream, n, r, alpha, a, beta );
```

説明

この関数は、位置母数 a 、尺度母数 β 、形状母数 α を持つワイブル分布の乱数を生成する。ここで、 $\alpha, \beta, a \in \mathbb{R}$ 、 $\alpha > 0$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \alpha, \beta}(x) = \begin{cases} \frac{\alpha}{\beta^\alpha} (x-a)^{\alpha-1} \exp\left(-\left(\frac{x-a}{\beta}\right)^\alpha\right), & x \geq a \\ 0, & x < a \end{cases}$$

累積分布関数は次のように示される。

$$F_{a, \alpha, \beta}(x) = \begin{cases} 1 - \exp\left(-\left(\frac{x-a}{\beta}\right)^\alpha\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SWEIBULL_ICDF VSL_METHOD_DWEIBULL_ICDF VSL_METHOD_SWEIBULL_ICDF _ACCURATE VSL_METHOD_DWEIBULL_ICDF _ACCURATE 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>alpha</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場合)	形状母数 α
<i>a</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場合)	位置母数 a
<i>beta</i>	REAL, INTENT(IN) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(IN) (vdrngweibull の場合)	float (vsRngWeibull の場合) double (vdRngWeibull の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngweibull の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngweibull の場合)	float* (vsRngWeibull の場合) double* (vdRngWeibull の場合)	ワイブル分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Cauchy

コーシー分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrngcauchy( method, stream, n, r, a, beta )
status = vdrngcauchy( method, stream, n, r, a, beta )
```

C:

```
status = vsRngCauchy( method, stream, n, r, a, beta );
status = vdRngCauchy( method, stream, n, r, a, beta );
```

説明

この関数は、位置母数 *a*、尺度母数 β を持つコーシー分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \frac{1}{\pi\beta\left(1 + \left(\frac{x-a}{\beta}\right)^2\right)}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a, \beta}(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x-a}{\beta}\right), \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SCAUCHY_ICDF VSL_METHOD_DCAUCHY_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: ストリームステート構造体のディスクリプター C: ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合)	float (vsRngCauchy の場合) double (vdRngCauchy の場合)	位置母数 <i>a</i>
<i>beta</i>	REAL, INTENT(IN) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(IN) (vdrngcauchy の場合)	float (vsRngCauchy の場合) double (vdRngCauchy の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngcauchy の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngcauchy の場合)	float* (vsRngCauchy の場合) double* (vdRngCauchy の場合)	コーシー分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Rayleigh

レイリー分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrngrayleigh( method, stream, n, r, a, beta )
status = vdrnggrayleigh( method, stream, n, r, a, beta )
```

C:

```
status = vsRngRayleigh( method, stream, n, r, a, beta );
status = vdRngRayleigh( method, stream, n, r, a, beta );
```

説明

この関数は、位置母数 a 、尺度母数 β を持つレイリー分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}$ 、 $\beta > 0$ 。

レイリー分布は [Weibull](#) 分布の一種であり、形状母数 $\alpha = 2$ のときである。

確率密度関数は次のように示される。

$$f_{a,\beta}(x) = \begin{cases} \frac{2(x-a)}{\beta^2} \exp\left(-\frac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = \begin{cases} 1 - \exp\left(-\frac{(x-a)^2}{\beta^2}\right), & x \geq a \\ 0, & x < a \end{cases}, \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SRAYLEIGH_ICDF VSL_METHOD_DRAYLEIGH_ICDF VSL_METHOD_SRAYLEIGH_ICDF_ACCURATE VSL_METHOD_DRAYLEIGH_ICDF_ACCURATE 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrngrayleigh の場合)	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合)	位置母数 a
<i>beta</i>	REAL, INTENT(IN) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(IN) (vdrngrayleigh の場合)	float (vsRngRayleigh の場合) double (vdRngRayleigh の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngrayleigh の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngrayleigh の場合)	float* (vsRngRayleigh の場合) double* (vdRngRayleigh の場合)	レイリー分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK
VSL_ERROR_NULL_PTR

正常に終了したことを示す。
stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Lognormal

対数正規分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnglognormal( method, stream, n, r, a, sigma, b, beta )
```

```
status = vdrnglognormal( method, stream, n, r, a, sigma, b, beta )
```

C:

```
status = vsRngLognormal( method, stream, n, r, a, sigma, b, beta );
```

```
status = vdRngLognormal( method, stream, n, r, a, sigma, b, beta );
```

説明

この関数は、分布の平均 a 、正規分布の標準偏差 σ 、位置母数 b 、尺度母数 β を持つ対数正規分布の乱数を生成する。ここで、 $a, \sigma, b, \beta \in \mathbb{R}$ 、 $\sigma > 0$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \sigma, b, \beta}(x) = \begin{cases} \frac{1}{\sigma(x-b)\sqrt{2\pi}} \exp\left(-\frac{[\ln((x-b)/\beta) - a]^2}{2\sigma^2}\right), & x > b \\ 0, & x \leq b \end{cases}$$

累積分布関数は次のように示される。

$$F_{a, \sigma, b, \beta}(x) = \begin{cases} \Phi((\ln((x-b)/\beta) - a)/\sigma), & x > b \\ 0, & x \leq b \end{cases}$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SLOGNORMAL_ICDF VSL_METHOD_DLOGNORMAL_ICDF VSL_METHOD_SLOGNORMAL_ICDF_ACCURATE VSL_METHOD_DLOGNORMAL_ICDF_ACCURATE 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	正規分布の平均 a
<i>sigma</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	正規分布の標準偏差 σ
<i>b</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	位置母数 b
<i>beta</i>	REAL, INTENT(IN) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(IN) (vdrnglognormal の場合)	float (vsRngLognormal の場合) double (vdRngLognormal の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnglognormal の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnglognormal の場合)	float* (vsRngLognormal の場合) double* (vdRngLognormal の場合)	対数正規分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Gumbel

ガンベル分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnggumbel( method, stream, n, r, a, beta )
status = vdrnggumbel( method, stream, n, r, a, beta )
```

C:

```
status = vsRngGumbel( method, stream, n, r, a, beta );
status = vdRngGumbel( method, stream, n, r, a, beta );
```

説明

この関数は、位置母数 a 、尺度母数 β を持つガンベル分布の乱数を生成する。ここで、 $a, \beta \in \mathbb{R}$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{a, \beta}(x) = \frac{1}{\beta} \exp\left(\frac{x-a}{\beta}\right) \exp(-\exp((x-a)/\beta)), \quad -\infty < x < +\infty$$

累積分布関数は次のように示される。

$$F_{a,\beta}(x) = 1 - \exp(-\exp((x-a)/\beta)), \quad -\infty < x < +\infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGUMBEL_ICDF VSL_METHOD_DGUMBEL_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGumbel の場合) double (vdRngGumbel の場合)	位置母数 <i>a</i>
<i>beta</i>	REAL, INTENT(IN) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggumbel の場合)	float (vsRngGumbel の場合) double (vdRngGumbel の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggumbel の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggumbel の場合)	float* (vsRngGumbel の場合) double* (vdRngGumbel の場合)	ガンベル分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK
VSL_ERROR_NULL_PTR
VSL_ERROR_BAD_STREAM

正常に終了したことを示す。
stream が NULL ポインターである。
stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Gamma

ガンマ分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrnggamma( method, stream, n, r, alpha, a, beta )
```

```
status = vdrnggamma( method, stream, n, r, alpha, a, beta )
```

C:

```
status = vsRngGamma( method, stream, n, r, alpha, a, beta );
```

```
status = vdRngGamma( method, stream, n, r, alpha, a, beta );
```

説明

この関数は、形状母数 α 、位置母数 a 、尺度母数 β を持つガンマ分布の乱数を生成する。ここで、 α, β , および $a \in \mathbb{R}$ 、 $\alpha > 0$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{\alpha, a, \beta}(x) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta^\alpha} (x-a)^{\alpha-1} e^{-(x-a)/\beta}, & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < \infty$$

ここで、 $\Gamma(\alpha)$ 完全ガンマ関数である。

累積分布関数は次のように示される。

$$F_{\alpha, a, \beta}(x) = \begin{cases} \int_a^x \frac{1}{\Gamma(\alpha)\beta^\alpha} (y-a)^{\alpha-1} e^{-(y-a)/\beta} dy, & x \geq a \\ 0, & x < a \end{cases}, -\infty < x < \infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SGAMMA_GNORM VSL_METHOD_DGAMMA_GNORM VSL_METHOD_SGAMMA_GNORM_ACCURATE VSL_METHOD_DGAMMA_GNORM_ACCURATE ガウス分布に従った乱数を使用する受容/棄却法。 表 10-1 の生成方法GNORMの説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>alpha</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggamma の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	形状母数 α
<i>a</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggamma の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	位置母数 a
<i>beta</i>	REAL, INTENT(IN) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(IN) (vdrnggamma の場合)	float (vsRngGamma の場合) double (vdRngGamma の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrnggamma の場合) DOUBLE PRECISION, INTENT(OUT) (vdrnggamma の場合)	float* (vsRngGamma の場合) double* (vdRngGamma の場合)	ガンマ分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Beta

ベータ分布に従う乱数を生成する。

構文

Fortran:

```
status = vsrngbeta( method, stream, n, r, p, q, a, beta )
status = vdrngbeta( method, stream, n, r, p, q, a, beta )
```

C:

```
status = vsRngBeta( method, stream, n, r, p, q, a, beta );
status = vdRngBeta( method, stream, n, r, p, q, a, beta );
```

説明

この関数は、形状母数 p と q 、位置母数 a 、尺度母数 β を持つベータ分布の乱数を生成する。ここで、 p, q, a , および $\beta \in \mathbb{R}$ 、 $p > 0$ 、 $q > 0$ 、 $\beta > 0$ 。

確率密度関数は次のように示される。

$$f_{p, q, a, \beta}(x) = \begin{cases} \frac{1}{B(p, q)\beta^{p+q-1}}(x-a)^{p-1}(\beta+a-x)^{q-1}, & a \leq x < a+\beta \\ 0, & x < a, x \geq a+\beta \end{cases}, \quad -\infty < x < \infty$$

ここで、 $B(p, q)$ は完全ベータ関数である。

累積分布関数は次のように示される。

$$F_{p, q, a, \beta}(x) = \begin{cases} 0, & x < a \\ \int_a^x \frac{1}{B(p, q)\beta^{p+q-1}}(y-a)^{p-1}(\beta+a-y)^{q-1} dy, & a \leq x < a+\beta \\ 1, & x \geq a+\beta \end{cases}, \quad -\infty < x < \infty$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_SBETA_CJA VSL_METHOD_DBETA_CJA VSL_METHOD_SBETA_CJA_ACCURATE VSL_METHOD_DBETA_CJA_ACCURATE 表 10-1 の生成方法 CJA の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>p</i>	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	形状母数 p
<i>q</i>	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	形状母数 q
<i>a</i>	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	位置母数 a
<i>beta</i>	REAL, INTENT(IN) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(IN) (vdrngbeta の場合)	float (vsRngBeta の場合) double (vdRngBeta の場合)	尺度母数 β

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	REAL, INTENT(OUT) (vsrngbeta の場合) DOUBLE PRECISION, INTENT(OUT) (vdrngbeta の場合)	float* (vsRngBeta の場合) double* (vdRngBeta の場合)	ベータ分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

離散分布

離散分布の乱数を生成するルーチンについて説明する。

Uniform

区間 $[a, b)$ を持つ一様離散分布に従った乱数を生成する。

構文

Fortran:

```
status = virnguniform( method, stream, n, r, a, b )
```

C:

```
status = viRngUniform( method, stream, n, r, a, b );
```

説明

この関数は、区間 $[a, b)$ を持つ一様離散分布に従った乱数を生成する。ここで、 a, b は、それぞれ区間の始まりと終わりであり、 $a, b \in \mathbb{Z}$ 、 $a < b$ 。

確率分布は次のように示される。

$$P(X = k) = \frac{1}{b - a}, \quad k \in \{a, a + 1, \dots, b - 1\}$$

累積分布関数は次のように示される。

$$F_{a,b}(x) = \begin{cases} 0, & x < a \\ \frac{\lfloor x - a + 1 \rfloor}{b - a}, & a \leq x < b \\ 1, & x \geq b \end{cases}, x \in \mathbb{R}$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。このパラメーターはダミーで一樣分布の場合は0を設定する。特定の値は次のとおりである。 VSL_METHOD_IUNIFORM_STD 標準生成法。現時点で、この分布生成器に対する <i>method</i> は1つのみ。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	INTEGER, INTENT(IN)	int	区間の開始 <i>a</i>
<i>b</i>	INTEGER, INTENT(IN)	int	区間の終了 <i>b</i>

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	区間 [<i>a</i> , <i>b</i>) を持つ一樣分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

UniformBits

一様ビット分布に従う整数乱数を生成する。

構文

Fortran:

```
status = virnguniformbits( method, stream, n, r )
```

C:

```
status = viRngUniformBits( method, stream, n, r );
```

説明

この関数は一様ビット分布に従う整数の乱数を生成する。一様分布の乱数生成器は、整数値に対する剰余算の漸化式として表せる。各整数は、それぞれ複数のビットを持つベクトルとして扱えることは明らかである。真の乱数を発する生成器ではこれらビットはランダムであるが、擬似乱数生成器ではランダムさは失われる。例えば、線形合同生成器には、上位ビットに比べて下位ビットのランダムさが少ないというよく知られた欠点がある(例えば、[\[Knuth81\]](#)を参照)。この理由から、この関数を使用する場合は注意が必要である。一般に、32ビット LCG では、整数値の上位 24 ビットのみがランダムであるとみなされる。詳細は、[VSL Notes](#) を参照のこと。

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。ダミーで 0 を設定する。特定の値は次のとおりである。 VSL_METHOD_IUNIFORMBITS_STD 標準生成法。現時点で、この分布生成器に対する <i>method</i> は 1 つのみ。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	unsigned int*	<p><i>Fortran</i>: <i>n</i> 個の乱数のベクトル。 <i>stream</i> が 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の <i>r</i> の成分でそれぞれ表される。各整数が占有するバイト数は VSL_BRNG_PROPERTIES 構造体の <i>wordsize</i> フィールドで指定される。値を格納するために実際に使用される合計ビット数は同じ構造体の <i>nbits</i> フィールドで指定される。VSLBrngProperties の詳細は、「アドバンスド・サービス・ルーチン」を参照のこと。</p> <p><i>C</i>: <i>n</i> 個の乱数のベクトル。 <i>stream</i> が 64 ビットまたは 128 ビットの生成器で生成された場合、各整数値は 2 個または 4 個の <i>r</i> の成分でそれぞれ表される。各整数が占有するバイト数は VSLBrngProperties 構造体の <i>WordSize</i> フィールドで指定される。値を格納するために実際に使用される合計ビット数は同じ構造体の <i>NBits</i> フィールドで指定される。VSLBrngProperties の詳細は、「アドバンスド・サービス・ルーチン」を参照のこと。</p>

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Bernoulli

ベルヌーイ分布に従う乱数を生成する。

構文

Fortran:

```
status = virngbernoulli( method, stream, n, r, p )
```

C:

```
status = viRngBernoulli( method, stream, n, r, p );
```

説明

この関数は、単一の試行成功確率を p とするベルヌーイ分布に従った乱数を生成する。ここで、 $p \in R$ 、 $0 \leq p \leq 1$ 。

変数はベルヌーイ分布と呼ばれ、試行の結果 1 となる成功確率は p 、0 となる確率は $1-p$ である。

確率分布は次のように示される。

$$P(X = 1) = p,$$

$$P(X = 0) = 1 - p.$$

累積分布関数は次のように示される。

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1 - p, & 0 \leq x < 1 \\ 1, & x \geq 1 \end{cases}, x \in R.$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IBERNOULLI_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	試行の成功確率 p

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ベルヌーイ分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または > nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Geometric

幾何分布に従う乱数を生成する。

構文

Fortran:

```
status = virnggeometric( method, stream, n, r, p )
```

C:

```
status = viRngGeometric( method, stream, n, r, p );
```

説明

この関数は、試行の成功確率を p とする幾何分布に従った乱数を生成する。ここで、 $p \in R$ 、 $0 < p < 1$ 。

幾何分布の変数は最初の成功が得られるまでの独立したベルヌーイ試行の回数である。ベルヌーイ試行の成功確率は p である。

確率分布は次のように示される。

$$P(X = k) = p \cdot (1 - p)^k, \quad k \in \{0, 1, 2, \dots\}.$$

累積分布関数は次のように示される。

$$F_p(x) = \begin{cases} 0, & x < 0 \\ 1 - (1 - p)^{\lfloor x + 1 \rfloor}, & x \geq 0 \end{cases}, \quad x \in R.$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IGEOMETRIC_ICDF 逆累積分布関数法
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	試行の成功確率 p

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	幾何分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Binomial

二項分布に従う乱数を生成する。

構文

Fortran:

```
status = virngbinomial( method, stream, n, r, ntrial, p )
```

C:

```
status = viRngBinomial( method, stream, n, r, ntrial, p );
```

説明

この関数は、独立したベルヌーイ試行回数 m と単一の試行成功確率 p を持つ二項分布に従った乱数を生成する。ここで、 $p \in R$ 、 $0 \leq p \leq 1$ 、 $m \in N$ 。

二項分布の変数は、単一の試行成功確率 p を持つベルヌーイ試行を、独立に m 回行ったときの成功回数として表される。

確率分布は次のように示される。

$$P(X = k) = C_m^k p^k (1-p)^{m-k}, \quad k \in \{0, 1, \dots, m\}.$$

累積分布関数は次のように示される。

$$F_{m,p}(x) = \begin{cases} 0, & x < 0 \\ \sum_{k=0}^{\lfloor x \rfloor} C_m^k p^k (1-p)^{m-k}, & 0 \leq x < m \\ 1, & x \geq m \end{cases}, \quad x \in R.$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IBINOMIAL_BTPE 表 10-1 の生成方法 BTPE の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: ストリームステート構造体のディスクリプター C: ストリームステート構造体へのポインタ
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>ntrials</i>	INTEGER, INTENT(IN)	int	独立した試行回数 m
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	単一試行の成功確率 p

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	二項分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Hypergeometric

超幾何分布に従う乱数を生成する。

構文

Fortran:

```
status = virnghypergeometric( method, stream, n, r, l, s, m )
```

C:

```
status = viRngHypergeometric( method, stream, n, r, l, s, m );
```

説明

この関数は、母集団の大きさ *l*、抽出の大きさ *s*、母集団内の「あたり」の成分数 *m* を持つ超幾何分布に従った乱数を生成する。ここで、*l, m, s* ∈ $N \cup \{0\}$ 、 $l \geq \max(s, m)$ 。

母集団 *l* の成分は、*m* 個の「あたり」成分と *l-m* 個の「はずれ」成分で構成されているとする。この母集団から復元なしで *s* 個の成分を試行抽出すると、*s* 個の成分の中に *k* 個の「あたり」を含んでいる確率を示す超幾何分布が得られる。

確率分布は次のように示される。

$$P(X = k) = \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, \quad k \in \{\max(0, s+m-l), \dots, \min(s, m)\}$$

累積分布関数は次のように示される。

$$F_{l,s,m}(x) = \begin{cases} 0, & x < \max(0, s+m-1) \\ \sum_{k=\max(0, s+m-1)}^{\lfloor x \rfloor} \frac{C_m^k C_{l-m}^{s-k}}{C_l^s}, & \max(0, s+m-1) \leq x \leq \min(s, m) \\ 1, & x > \min(s, m) \end{cases}$$

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IHYPERGEOMETRIC_H2PE 表 10-1 の生成方法 H2PE の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: ストリームステート構造体のディスクリプター C: ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>l</i>	INTEGER, INTENT(IN)	int	母集団の大きさ <i>l</i>
<i>s</i>	INTEGER, INTENT(IN)	int	復元なしに抽出する個数 <i>s</i>
<i>m</i>	INTEGER, INTENT(IN)	int	「あたり」成分の個数 <i>m</i>

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	超幾何分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Poisson

ポアソン分布に従う乱数を生成する。

構文

Fortran:

```
status = virngpoisson( method, stream, n, r, lambda )
```

C:

```
status = viRngPoisson( method, stream, n, r, lambda );
```

説明

この関数は、分布パラメーターを λ とするポアソン分布に従った乱数を生成する。ここで、 $\lambda \in R$ 、 $\lambda > 0$ 。

確率分布は次のように示される。

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, k \in \{0, 1, 2, \dots\}.$$

累積分布関数は次のように示される。

$$F_{\lambda}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda^k e^{-\lambda}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R.$$

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IPOISSON_PTPE VSL_METHOD_IPOISSON_POISNORM 表 10-1 の生成方法 PTPE と POISNORM の説明を参照。

名前	データ型		説明
	FORTRAN	C	
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	Fortran: ストリームステート構造体のディスクリプター C: ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>lambda</i>	DOUBLE PRECISION, INTENT(IN)	double	分布パラメーター λ

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ポアソン分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

PoissonV

多様な平均値を持つポアソン分布に従った乱数を生成する。

構文

Fortran:

```
status = virngpoissonv( method, stream, n, r, lambda )
```

C:

```
status = viRngPoissonV( method, stream, n, r, lambda );
```

説明

この関数は、分布パラメーターを λ_i とするポアソン分布に従った乱数 $x_i (i = 1, \dots, n)$ を n 個生成する。ここで、 $\lambda_i \in R$ 、 $\lambda_i > 0$ 。

確率分布は次のように示される。

$$P(X_i = k) = \frac{\lambda_i^k \exp(-\lambda_i)}{k!}, k \in \{0, 1, 2, \dots\}.$$

累積分布関数は次のように示される。

$$F_{\lambda_i}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} \frac{\lambda_i^k e^{-\lambda_i}}{k!}, & x \geq 0 \\ 0, & x < 0 \end{cases}, x \in R.$$

入力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_IPOISSONV_POISNORM 表 10-1 の生成方法 POISNORM の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>lambda</i>	DOUBLE PRECISION, INTENT(IN)	double*	n 個の分布パラメーター λ_i の配列

出力パラメーター

名前	データ型		説明
	FORTTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	ポアソン分布に従った n 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_NULL_PTR	<i>stream</i> が NULL ポインターである。
VSL_ERROR_BAD_STREAM	<i>stream</i> が有効なランダムストリームでない。
VSL_ERROR_BAD_UPDATE	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (<0 または >nmax) を返した。
VSL_ERROR_NO_NUMBERS	抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

Negbinomial

負の二項分布に従う乱数を生成する。

構文

Fortran:

```
status = virngnegbinomial( method, stream, n, r, a, p )
```

C:

```
status = viRngNegbinomial( method, stream, n, r, a, p );
```

説明

この関数は、分布パラメーター a と p を持つ負の二項分布に従った乱数を生成する。ここで、 $p, a \in \mathbb{R}$, $0 < p < 1$, $a > 0$ 。

第 1 の分布パラメーターが $a \in \mathbb{N}$ なら、この分布はパスカル分布と等しくなる。この分布は、 $a \in \mathbb{N}$ では、成功確率 p のベルヌーイ試行のときに、 a 番目の成功を得るまでの見込まれる試行回数として解釈できる。

確率分布は次のように示される。

$$P(X = k) = C_{a+k-1}^k p^a (1-p)^k, \quad k \in \{0, 1, 2, \dots\}.$$

累積分布関数は次のように示される。

$$F_{a,p}(x) = \begin{cases} \sum_{k=0}^{\lfloor x \rfloor} C_{a+k-1}^k p^a (1-p)^k, & x \geq 0 \\ 0, & x < 0 \end{cases}, \quad x \in \mathbb{R}.$$

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>method</i>	INTEGER, INTENT(IN)	int	生成方法。特定の値は次のとおりである。 VSL_METHOD_INEBINOMIAL_NBAR 表 10-1 の生成方法 NBAR の説明を参照。
<i>stream</i>	TYPE (VSL_STREAM_STATE), INTENT(IN)	VSLStreamStatePtr	<i>Fortran</i> : ストリームステート構造体のディスクリプター <i>C</i> : ストリームステート構造体へのポインター
<i>n</i>	INTEGER, INTENT(IN)	int	生成する乱数の個数
<i>a</i>	DOUBLE PRECISION, INTENT(IN)	double	第 1 の分布パラメーター <i>a</i>
<i>p</i>	DOUBLE PRECISION, INTENT(IN)	double	第 2 の分布パラメーター <i>p</i>

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>r</i>	INTEGER, INTENT(OUT)	int*	負の二項分布に従った <i>n</i> 個の乱数のベクトル

戻り値

VSL_ERROR_OK, VSL_STATUS_OK

正常に終了したことを示す。

VSL_ERROR_NULL_PTR

stream が NULL ポインターである。

VSL_ERROR_BAD_STREAM

stream が有効なランダムストリームでない。

VSL_ERROR_BAD_UPDATE

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数として無効な値 (< 0 または > nmax) を返した。

VSL_ERROR_NO_NUMBERS

抽象 BRNG のコールバック関数が、バッファ内の更新された成分の数としてゼロを返した。

アドバンスト・サービス・ルーチン

このセクションでは、ユーザー定義基本生成器の登録を行うルーチン ([RegisterBrng](#)) と、登録済み基本生成器のプロパティを取得するルーチン ([GetBrngProperties](#)) について説明する。ユーザー定義の BRNG を含む基本生成器の必要性については、[VSL Notes](#) (VSL Structure の章の「Basic Generators」セクション) を参照。

データ型

ここで記載のルーチン群は、基本生成器のプロパティを定義する構造体を参照する。この構造体は、Fortran では次のようになっている。

```
TYPE VSL_BRNG_PROPERTIES
    INTEGER      streamstatesize
    INTEGER      nseeds
    INTEGER      includeszero
    INTEGER      wordsize
    INTEGER      nbits
    INTEGER      initstream
    INTEGER      sbrng
    INTEGER      dbrng
    INTEGER      ibrng
END TYPE VSL_BRNG_PROPERTIES
```

C では次のようになっている。

```
typedef struct _VSLBRngProperties {
    int      StreamStateSize;
    int      NSeeds;
    int      IncludesZero;
    int      WordSize;
    int      NBits;
    IntPtr  InitStream;
    IntPtr  sBRng;
    IntPtr  dBRng;
    IntPtr  iBRng;
} VSLBRngProperties;
```

上記の構造体に関与しているフィールドの説明を次の表に示す。

表 10-13 **フィールドの説明**

フィールド	説明
FORTRAN: streamstatesize C: StreamStateSize	バイトを単位とする、指定された基本生成器に対するストリームステート構造体のサイズ。
FORTRAN: nseeds C: NSeeds	指定された基本生成器に対するストリームステート構造体の初期化に必要な、32 ビット初期条件 (種) の個数。

表 10-13 フィールドの説明 (続き)

フィールド	説明
FORTRAN: includeszero C: IncludesZero	生成器が乱数 0 を生成可能であることを示すフラグ値。
FORTRAN: wordsize C: WordSize	バイトを単位とする、整数値計算で 사용되는マシン・ワード・サイズ。取り得る値は、32 ビット、64 ビット、128 ビット生成器で、それぞれ 4、8、16。
FORTRAN: nbits C: NBits	整数算で乱数を表すために必要となるビット数。例えば、48 ビット乱数は 64 ビット (8 バイト) メモリー・ロケーションに格納される点に注意する。この場合を例に取ると、wordsize/WordSize (その乱数の格納に使用されるバイト数) は 8 となるが、nbits/NBits はその値によって占有される実際のビット数であるから 48 となる。
FORTRAN: initstream C: InitStream	指定された基本生成器の初期化ルーチンに対するポインター。
FORTRAN: sbrng C: sBRng	区間 (a, b) を持つ一様分布を単精度実数で処理する基本生成器に対するポインター (FORTRAN では REAL、C では float)。
FORTRAN: dbrng C: dBRng	区間 (a, b) を持つ一様分布を倍精度実数で処理する基本生成器に対するポインター (FORTRAN では DOUBLE PRECISION、C では double)。
FORTRAN: ibrng C: iBRng	一様ビット分布を整数で処理する基本生成器に対するポインター (FORTRAN では INTEGER、C では unsigned int) ¹ 。

1. 乱数の単一ビットとビットグループとの演算を許可している特有の生成器である。

RegisterBrng

ユーザー定義の基本生成器を登録する。

構文

Fortran:

```
brng = vslregisterbrng( properties )
```

C:

```
brng = vslRegisterBrng( &properties );
```

説明

登録手順の例は各システムの VSL examples ディレクトリーにある。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
properties	TYPE (VSL_BRNG_PROPERTIES), INTENT(IN)	VSLBrngProperties*	登録する基本生成器のプロパティーを含む構造体へのポインター

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT(OUT)	int	登録された基本生成器の番号 (インデックス値) で、識別に使用する。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_BRNG_TABLE_FULL	登録されている BRNG の表にフリー成分が不足しているため、登録を完了することができない。
VSL_ERROR_BAD_STREAM_STATE_SIZE	StreamStateSize フィールドの値が不正である。
VSL_ERROR_BAD_WORD_SIZE	WordSize フィールドの値が不正である。
VSL_ERROR_BAD_NSEEDS	NSeeds フィールドの値が不正である。
VSL_ERROR_BAD_NBITS	NBits フィールドの値が不正である。
VSL_ERROR_NULL_PTR	iBrng、dBrng、sBrng、または InitStream の少なくとも 1 つが NULL ポインターである。

GetBrngProperties

指定した基本生成器のプロパティを構造体で返す。

構文

Fortran:

```
status = vslgetbrngproperties( brng, properties )
```

C:

```
status = vslGetBrngProperties( brng, &properties );
```

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>brng</i>	INTEGER, INTENT(IN)	int	登録された基本生成器の番号 (インデックス値) で、識別に使用する。表 10-2 の特定の値を参照。負の値が返ってきた場合は登録時にエラーが発生したことを示す。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>properties</i>	TYPE (VSL_BRNG_PROPERTIES), INTENT(OUT)	VSLBrngProperties*	番号 <i>brng</i> を持つ生成器のプロパティーを含む構造体へのポインター

戻り値

VSL_ERROR_OK, VSL_STATUS_OK	正常に終了したことを示す。
VSL_ERROR_INVALID_BRNG_INDEX	BRNG のインデックスが無効である。

ユーザー定義生成器のフォーマット

[RegisterBrng](#) 関数を用いてユーザー定義基本生成器を登録するには、整数値で実装されている生成器のポインターを *iBrng* に、単精度値および倍精度値で実装されている生成器のポインターをそれぞれ *sBrng* と *dBrng* に、ストリーム初期化ルーチンのポインターを *InitStream* に、それぞれ渡さなければならない。以下のセクションに、入力引数と出力引数の使い方と合わせ各関数定義の推奨例を記載している。ユーザー定義生成器の登録手順例は各システムの VSL examples ディレクトリーにある。

それぞれのポインターは次のように定義される。

```
typedef      int (*InitStreamPtr)( int method, void * stream, int n,
                                const unsigned int params[] );

typedef      int (*sBrngPtr)( void * stream, int n, float r[], float
                                a, float b );

typedef      int (*dBrngPtr)( void * stream, int n, double r[], double
                                a, double b );

typedef      int (*iBrngPtr)( void * stream, int n, unsigned int
                                r[] );
```

InitStream

```
C:
int MyBrngInitStream( int method, VSLStreamStatePtr stream,
int n, const unsigned int params[] );
{
/* Initialize the stream */
...
} /* MyBrngInitStream */
```

説明

ユーザー定義生成器の初期化ルーチンは、指定された初期化 *method*、初期条件 *params*、引数 *n* に従って *stream* を初期化しなければならない。*method* の値によって初期化方法が指定される。

- *method* が 0 の場合、すべての基本生成器がサポートしなければならない標準的な生成方法を用いて初期化を行う。ここで関数は、*stream* 構造体は過去に初期化されていないと仮定する。*n* は、初期化条件として *params* を介して渡される 32 ビット値の個数を表す。関数に渡される初期条件の個数が生成器の初期化に足りない場合でもエラーではないことに注意が必要である。このような状況が起こった場合、基本生成器は、不足している初期条件に代わってデフォルト値を用いて初期化を行わなければならない。
- *method* が 1 の場合はリープフロッグ法による生成となり、*n* は処理ノード (独立ストリーム) 数を指定する。*stream* は標準生成法で初期化されていると仮定する。*params* は処理ノードを指定する単一成分のみで構成される。生成器がリープフロッグ法をサポートしない場合、関数はエラーコード `VSL_ERROR_LEAPFROG_UNSUPPORTED` を返さなければならない。
- *method* が 2 の場合はブロック分割法による生成となる。上記と同様に、*stream* は標準生成法で初期化されていると仮定する。*params* は使用されない。*n* はスキップする成分数を指定する。生成器がブロック分割法をサポートしていない場合、関数はエラーコード `VSL_ERROR_SKIPAHEAD_UNSUPPORTED` を返さなければならない。

リープフロッグ法とブロック分割法の詳細は、[LeapfrogStream](#) および [SkipAheadStream](#) をそれぞれ参照のこと。

ストリーム状態構造体はそれぞれの生成器ごとに固有である。ただし、すべての生成器で共通なフィールドはすべての構造体を持っている。

```
C:
typedef struct
{
    unsigned int Reserved1[2];
    unsigned int Reserved2[2];
    [fields specific for the given generator]
} MyStreamState;
```

フィールド *Reserved1* と *Reserved2* は非公開目的のために予約されており、ユーザーは変更してはならない。構造体に専用のフィールドを追加する場合は次の規則に従うこと。

- フィールドは生成器の最新ステートを完全に記述しなくてはならない。例えば、線形合同生成器のステータは単一の初期条件しか持たない。
- 生成器がリープフロッグ法とブロック分割法の両方に対応するのであれば、独立したストリームを識別するために追加フィールドを導入すべきである。例えば、 $LCG(a, c, m)$ では、初期条件とは別に、乗数 a^k と、加数 $(a^k-1)c/(a-1)$ の2つの追加フィールドを指定する必要がある。

詳細は、[\[Knuth81\]](#) と [\[Gentle98\]](#) を参照のこと。登録手順の例は各システムの VSL examples ディレクトリーにある。

iBRng

C:

```
int iMyBrng( VSLStreamStatePtr stream, int n,
            unsigned int r[] )
{
    int i;      /* Loop variable */
    /* Generating integer random numbers */
    /* Pay attention to word size needed to
       store only random number */
    for( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
    /* Update stream state */
    ...
    return errcode;
} /* iMyBrng */
```



注: 64 ビットと 128 ビットの生成器を使用する場合、乱数ベクトル r に値を正しく格納するために桁容量を考慮すること。例えば、1 個の 64 ビット値は r の 2 個の成分を必要とし、1 番目の成分に下位 32 ビット、2 番目の成分に上位 32 ビットが格納される。同様に 128 ビット値では 4 個の r 成分が使われる。

sBRng

```
C:
int sMyBrng( VSLStreamStatePtr stream, int n, float r[],
            float a, float b )
{
    int i; /* Loop variable */
    /* Generating float (a,b) random numbers */
    for ( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
    /* Update stream state */
    ...
return errcode;
} /* sMyBrng */
```

dBRng

```
C:
int dMyBrng( VSLStreamStatePtr stream, int n, double r[],
            double a, double b )
{
    int i; /* Loop variable */
    /* Generating double (a,b) random numbers */
    for ( i = 0; i < n; i++ )
    {
        r[i] = ...;
    }
    /* Update stream state */
    ...
return errcode;
} /* dMyBrng */
```

畳み込みと相関

概要

VSL では、単精度および倍精度データの線形畳み込み / 相関変換を実行するためのルーチン群が用意されている。

実装されている演算の正しい定義については、「[数学表記と定義](#)」セクションを参照のこと。

現時点では、次の実装が提供されている。

- 1次元の単精度および倍精度データ用フーリエ・アルゴリズム
- 多次元の単精度および倍精度データ用フーリエ・アルゴリズム
- 1次元の単精度および倍精度データ用直接アルゴリズム
- 多次元の単精度および倍精度データ用直接アルゴリズム

1次元用のアルゴリズムは、IBM ESSL ライブラリーの次の関数を備えている。

SCONF, SCORF

SCOND, SCORD

SDCON, SDCOR

DDCON, DDCOR

SDDCON, SDDCOR

これらの ESSL 関数をシミュレートするために特殊なラッパーが用意されている。ラッパーは、FORTRAN および C のサンプルソースとして提供されている。それらを利用するには、次のディレクトリーを使用する。

```
`${MKL}/examples/vslc/essl/vsl_wrappers
```

```
`${MKL}/examples/vslf/essl/vsl_wrappers
```

また、次のディレクトリーでは、ラッパーによる ESSL 関数の計算例を参照することもできる。

```
`${MKL}/examples/vslc/essl
```

```
`${MKL}/examples/vslf/essl
```

畳み込み / 相関 API は、FORTRAN-90 インターフェイスと C/89 インターフェイスを提供する。C/89 インターフェイスは、C/C++ のそれ以降のバージョンで使用することもできる。また、FORTRAN-90 は FORTRAN-95 のプログラムと使用することができる。

FORTRAN-77 はサポートされない。

C/C++ および FORTRAN 言語のユーザー用に、mkl_vsl.h および mkl_vsl.fi ヘッダーファイルが提供されている。これらのヘッダーファイルは、次のディレクトリーにある。

```
`${MKL}/include
```

FORTRAN ヘッダーに関する詳細は、本章の始めにある、「[乱数生成器](#)」セクションを参照。

畳み込み / 相関 API は、タスク・オブジェクト やタスクによって実装される。タスク・オブジェクトは、特定の畳み込み / 相関演算を決定するパラメーターを格納するデータ構造またはディスクリプターである。これらのパラメーターは、精度、データ型、ユーザーデータの次元、使用する計算アルゴリズムの識別子、データ配列の形状などである。

すべてのインテル MKL の畳み込み / 相関ルーチンは、タスク・ディスクリプターを新規に作成してパラメーターの設定を変更し、格納されているパラメーターを用いて畳み込みや相関の数値演算結果を計算するか、またはそのほかの演算を実行してタスク・オブジェクトを処理する。すべてのルーチンは以下のグループに分類される。

[タスク・コンストラクター](#) - タスク・オブジェクト・ディスクリプターを作成して共通のパラメーターを設定するルーチン。

[タスクエディター](#) - 既存のタスク・ディスクリプターのパラメーター設定を設定または変更するルーチン。

[タスク実行ルーチン](#) - タスク・ディスクリプターに格納されている演算パラメーターを用いて、実際の入力データに対する畳み込み / 相関演算の結果を計算する。

[タスクコピー](#) - タスク・ディスクリプターのコピーを作成するのに使用するルーチン。

[タスク・ディストラクター](#) - タスク・オブジェクトを削除してメモリーを開放するルーチン。

タスクを最初に実行またはコピーする場合、タスクの遂行と呼ばれる特殊処理を実行する。この処理では、タスク・パラメーターの一貫性がチェックされ、必要な作業データが準備される。パラメーターが一貫している場合、タスクが正常に遂行されたことを示すタグが付けられる。タスクは、パラメーターが編集されるまで、遂行されたままである。そのため、1 回の遂行処理でタスクを複数回実行することが可能である。タスクの遂行処理には、入力データのフーリエ変換の準備などコストのかかる中間計算が含まれるため、処理を一度だけ開始することにより全体の性能を向上させることができる。

命名規則

畳み込み / 相関 API では、FORTRAN ルーチン名は小文字で記述され、FORTRAN データ型と定数は大文字で記述される。これらの名前は大文字と小文字を区別しない。

C では、ルーチン、データ型、および定数の名前は大文字と小文字を区別し、大文字と小文字のどちらも使用することができる。

ルーチンの名前は、次の構造になっている。

```
vs1[datatype]{Conv|Corr}<base name>           (C インターフェイスの場合)
vs1[datatype]{conv|corr}<base name>           (FORTRAN インターフェイスの場合)
```

vs1 は、ルーチンがインテル MKL のベクトル統計ライブラリーに属していることを示すプリフィックスである。

[datatype] はオプションである。存在する場合、記号は入力データおよび出力データのデータ型を指定する。s (単精度実数型の場合) または d (倍精度実数型の場合) のいずれか。

プリフィックス Conv と Corr は、ルーチンが畳み込みタスクを参照するのか、または相関タスクを参照するのかを指定する。

<base name> フィールドは、NewTask や DeleteTask など、ルーチンの特性を示す。



注：本章では、曖昧にならない場合は、ルーチンの基本名を使用する。ルーチンのリファレンスでは、プロトタイプやコード例で常にフルネームを使用する。

データ型

すべての畳み込み / 関連ルーチンは、データ・オブジェクトの指定に次のデータ型を使用する。

データ型		データ・オブジェクト
FORTRAN	C	
TYPE(VSL_CONV_TASK)	VSLConvTaskPtr	畳み込みのタスク・ディスクリプターへのポインター
TYPE(VSL_CORR_TASK)	VSLCorrTaskPtr	関連のタスク・ディスクリプターへのポインター
REAL(KIND=4)	float	単精度の入力 / 出力ユーザーデータ
REAL(KIND=8)	double	倍精度の入力 / 出力ユーザーデータ
INTEGER	int	他のすべてのデータ

すべての整数データには一般整数型 (バイトサイズの指定なし) を使用する。



注：一般整数型の実際のサイズはプラットフォームに依存する。整数型の適切なバイトサイズは、ソフトウェアをコンパイルする段階で選択しなければならない。

パラメーター

タスク・ディスクリプターに格納される基本パラメーターは、タスク・オブジェクトがタスクエディターによって作成、コピー、または変更された際に割り当てられた値である。

畳み込み / 関連タスクのパラメーターは、タスク・オブジェクトが作成される際に、タスク・コンストラクターによって最初に設定される。パラメーターの変更や追加設定はタスクエディターによって行われる。タスク実行ルーチン呼び出す前に、畳み込みを行うデータの位置を定義するパラメーターを指定する必要がある。

引き渡し方法や割り当てられた値に従って、すべてのパラメーターは明示的 (タスク・オブジェクトが作成時または実行時に、ルーチン・パラメーターとして直接渡される) あるいはオプション (タスク構成時にデフォルトまたは暗黙的な値が割り当てられる) のいずれかのカテゴリーに分類される。

インテル MKL の畳み込み / 相関 API で使用される、すべての適用可能なパラメーターを次の表に示す。

表 10-14 畳み込み / 相関タスクのパラメーター

名前	カテゴリー	データ型	デフォルト値ラベル	説明
<i>job</i>	明示的	整数	コンストラクターの名前によって示される	タスクを畳み込みまたは相関のどちらに関連付けるかを指定する。
<i>type</i>	明示的	整数	コンストラクターの名前によって示される	入力 / 出力データのデータ型 (実数または複素数) を指定する。現在のバージョンでは実数に設定する。
<i>precision</i>	明示的	整数	コンストラクターの名前によって示される	配列 x , y , z で与えられる入力 / 出力データの精度 (単精度または倍精度) を指定する。
<i>kind</i>	オプション	整数	"linear"	タスクを線形または巡回のどちらの畳み込み / 相関演算に関連付けるかを指定する。
<i>mode</i>	明示的	整数	なし	畳み込み / 相関演算の実行方法 (フーリエ変換、直接法、あるいは 2 つの間で自動的に選択) を指定する。このパラメーターで使用される名前付き定数のリストは、 SetMode を参照のこと。
<i>method</i>	オプション	整数	"auto"	与えられた <i>mode</i> に対して複数の方法が利用可能な場合、特定の計算方法を示す。このパラメーターを "auto" に設定すると、ソフトウェアは利用可能な最良の方法を選択する。
<i>internal_precision</i>	オプション	整数	<i>precision</i> の値と等しくなるように設定する。	内部計算の精度を指定する。入力 / 出力データが単精度の場合でも倍精度計算を強制することができる。このパラメーターで使用される名前付き定数のリストは、 SetInternalPrecision を参照のこと。
<i>dims</i>	明示的	整数	なし	配列 x , y , z で与えられるユーザーデータの階数 (次元) を指定する。範囲は 1 ~ 7。
x, y	明示的	実数配列	なし	入力データ配列を指定する。詳細は、「 データの割り当て 」を参照。
z	明示的	実数配列	なし	出力データ配列を指定する。詳細は、「 データの割り当て 」を参照。
<i>xshape</i> , <i>yshape</i> , <i>zshape</i>	明示的	整数配列	なし	配列 x , y , z の形状を定義する。詳細は、「 データの割り当て 」を参照。
<i>xstride</i> , <i>ystride</i> , <i>zstride</i>	明示的	整数配列	なし	配列 x , y , z 内のストライドを定義する。ストライドは、それぞれの配列における入力 / 出力データの物理的な位置を指定する。詳細は、「 データの割り当て 」を参照。
<i>start</i>	オプション	整数配列	未定義	数値演算の結果の最初の成分を定義する。これは、出力配列 z に格納される。詳細は、 SetStart と「 データの割り当て 」を参照。

表 10-14 畳み込み / 関連タスクのパラメーター (続き)

名前	カテゴリ	データ型	デフォルト値 ラベル	説明
<i>decimation</i>	オプション	整数配列	未定義	数値演算の結果を減少させる方法を定義する。これは、出力配列 <i>z</i> に格納される。詳細は、 SetDecimation と「 データの割り当て 」を参照。

C または C++ 言語のユーザーは、多次元計算で *xstride*、*ystride*、*zstride* パラメーターのいずれか、またはすべての代わりに NULL ポインタを渡してもよい。この場合、ソフトウェアは、多次元配列の FORTRAN 形式の「列」表現により、配列 *x*、*y* または *z* に密データ割り当てを仮定する。

タスクステータスとエラー報告

タスクステータスは整数値である。タスク処理中にエラーが検出されなかった場合はゼロに、それ以外の場合は特定の非ゼロのエラーコードになる。負の値はエラーに使用され、正の値は警告に使用される。

エラーは、不正なパラメーター値、メモリ割り当ての失敗などのシステム障害によって引き起こされる。また、ソフトウェアによって検出される内部エラーのこともある。

タスク・ディスクリプターはそれぞれ、タスクの現状ステータスを含んでいる。タスク・オブジェクトを作成する際、コンストラクターはタスクに `VSL_STATUS_OK` ステータスを割り当てる。後でタスクを処理する際にエラーが発生した場合、実行エディターなど他のルーチンは、タスクステータスを変更し、タスク・ステータス・フィールドに対応するエラーコードを書き込むことができる。

タスクの作成またはそのパラメーターの編集段階では、パラメーターのセットが一貫しないことがある。パラメーターの一貫性のチェックは、タスクの実行またはコピー前に暗黙的に呼び出され、タスク遂行処理中にのみ実行される。この段階でエラーが検出されると、タスクの実行またはコピーは終了され、タスク・ディスクリプターは対応するエラーコードを保存する。いったんエラーが発生すると、そのタスク・ディスクリプターを処理するそれ以上の試みは終了され、タスクは同じエラーコードを保持する。

通常、それぞれの畳み込み / 関連関数 (`DeleteTask` を除く) は、関数を実行中にタスクに割り当てられたステータスを返す。

ステータスコードは、それぞれのヘッダーファイルで定義された記号名を与えられる。これらの名前は、C/C++ では `#define` 構文によってマクロとして定義され、FORTRAN では `PARAMETER` 演算子によって整数定数として定義される。

エラーがない場合、`VSL_STATUS_OK` (ゼロとして定義される) が返される。

```
C/C++:          #define VSL_STATUS_OK 0
```

```
F90/F95:        INTEGER,PARAMETER:: VSL_STATUS_OK = 0
```

エラーの場合、エラーの原因を知らせる非ゼロのエラーコードが返される。畳み込み / 関連のエラーコード用に、以下のステータスコードが C/C++ と FORTRAN 言語両方のヘッダーファイルであらかじめ定義されている。

表 10-15 畳み込み / 関連のステータスコードとメッセージ

ステータスコード	メッセージ
VSL_CC_ERROR_NOT_IMPLEMENTED	要求された機能は実装されていない。
VSL_CC_ERROR_ALLOCATION_FAILURE	メモリー割り当てに失敗した。
VSL_CC_ERROR_BAD_DESCRIPTOR	タスク・ディスクリプターが壊れている。
VSL_CC_ERROR_SERVICE_FAILURE	サービス関数に失敗した。
VSL_CC_ERROR_EDIT_FAILURE	タスクの編集中にエラーが発生した。
VSL_CC_ERROR_EDIT_PROHIBITED	このパラメーターは編集できない。
VSL_CC_ERROR_COMMIT_FAILURE	タスク遂行に失敗した。
VSL_CC_ERROR_COPY_FAILURE	タスクのコピー中にエラーが発生した。
VSL_CC_ERROR_DELETE_FAILURE	タスクの削除中にエラーが発生した。
VSL_CC_ERROR_BAD_ARGUMENT	不正な引数またはタスク・パラメーター。
VSL_CC_ERROR_JOB	不正なパラメーター: job。
VSL_CC_ERROR_KIND	不正なパラメーター: kind。
VSL_CC_ERROR_MODE	不正なパラメーター: mode。
VSL_CC_ERROR_METHOD	不正なパラメーター: method。
VSL_CC_ERROR_TYPE	不正なパラメーター: type。
VSL_CC_ERROR_EXTERNAL_PRECISION	不正なパラメーター: external_precision。
VSL_CC_ERROR_INTERNAL_PRECISION	不正なパラメーター: internal_precision。
VSL_CC_ERROR_PRECISION	外部 / 内部精度が非互換。
VSL_CC_ERROR_DIMS	不正なパラメーター: dims。
VSL_CC_ERROR_XSHAPE	不正なパラメーター: xshape。
VSL_CC_ERROR_YSHAPE	不正なパラメーター: yshape。
VSL_CC_ERROR_ZSHAPE	不正なパラメーター: zshape。
VSL_CC_ERROR_XSTRIDE	不正なパラメーター: xstride。
VSL_CC_ERROR_YSTRIDE	不正なパラメーター: ystride。
VSL_CC_ERROR_ZSTRIDE	不正なパラメーター: zstride。
VSL_CC_ERROR_X	不正なパラメーター: x。
VSL_CC_ERROR_Y	不正なパラメーター: y。
VSL_CC_ERROR_Z	不正なパラメーター: z。
VSL_CC_ERROR_START	不正なパラメーター: start。
VSL_CC_ERROR_DECIMATION	不正なパラメーター: decimation。
VSL_CC_ERROR_OTHER	その他のエラー。

タスク・コンストラクター

タスク・コンストラクターは、タスク・ディスクリプターを新規に作成して、基本パラメーターを設定するためのルーチンである。これは、追加でパラメーターを調整する必要がなく、他のルーチンがタスク・オブジェクトを使用できることを意味する。

インテル MKL の畳み込み / 相関 API では、一般形式と X 形式の 2 つの異なる形式のコンストラクターを用意している。X 形式のコンストラクターは一般形式のコンストラクターと同じ働きをするが、畳み込み / 相関演算で使用される最初の演算子ベクトル (配列 x に格納される) に特定のデータも割り当てる。

配列 y に格納された異なるベクトルに対して、配列 x に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、X 形式のコンストラクターの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

各コンストラクター・ルーチンに対して、関連する 1 次元のバージョンが存在する。このバージョンは、1 次元のデータ構造の単純性によってもたらされる、アルゴリズムおよび計算上の利点を活用する。



注: コンストラクターはタスク・ディスクリプターの作成に失敗すると、NULL タスクポインターを返す。

表 10-16 は、利用可能なタスク・コンストラクターをまとめたものである。

表 10-16 **タスク・コンストラクター**

ルーチン	説明
NewTask	多次元の畳み込み / 相関タスク・ディスクリプターを新規に作成する。
NewTask1D	1 次元の畳み込み / 相関タスク・ディスクリプターを新規に作成する。
NewTaskX	多次元の畳み込み / 相関タスク・ディスクリプターを X 形式で新規に作成する。
NewTaskX1D	1 次元の畳み込み / 相関タスク・ディスクリプターを X 形式で新規に作成する。

NewTask

多次元の畳み込み / 相関タスク・ディスクリプターを新規に作成する。

構文

Fortran:

```
status = vslsconvnewtask(task, mode, dims, xshape, yshape, zshape)
status = vsldconvnewtask(task, mode, dims, xshape, yshape, zshape)
status = vslscorrnewtask(task, mode, dims, xshape, yshape, zshape)
status = vsldcorrnewtask(task, mode, dims, xshape, yshape, zshape)
```

C:

```

status = vsIsConvNewTask(task, mode, dims, xshape, yshape, zshape);
status = vsLdConvNewTask(task, mode, dims, xshape, yshape, zshape);
status = vsIsCorrNewTask(task, mode, dims, xshape, yshape, zshape);
status = vsLdCorrNewTask(task, mode, dims, xshape, yshape, zshape);

```

説明

各 NewTask コンストラクターは、明示的パラメーターにユーザー指定の値を使用した新規の畳み込み / 相関タスク・ディスクリプターを作成する。オプション・パラメーターはデフォルト値に設定される ([表 10-14](#) を参照)。

パラメーター *xshape*、*yshape*、*zshape* はそれぞれ、配列 *x*、*y*、*z* で与えられる入力 / 出力データの形状を定義する。各形状パラメーターは、*dims* の値と同じ長さの整数配列である。

コンストラクターを呼び出す際に形状パラメーターを明示的に割り当てる。パラメーター *dims* の値が 1 の場合、*xshape*、*yshape*、*zshape* は配列 *x* と *y* から読み取られる成分の個数、または配列 *z* に格納される成分の個数に等しい。ストライドが割り当てられた場合、形状パラメーターの値は配列 *x*、*y*、*z* の物理的形状とは異なることがある。

コンストラクターはタスク・ディスクリプターの作成に失敗すると、NULL タスクポインターを返す。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 表 10-18 は指定可能な値をまとめたものである。
<i>dims</i>	INTEGER	int	ユーザーデータの階数 実行段階で使用される入力 / 出力配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> の次元を指定する。範囲は 1 ~ 7 でなければならない。値はコンストラクターによって明示的に割り当てられる。
<i>xshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>x</i> の入力データの形状を定義する。詳細は、「 データの割り当て 」を参照。
<i>yshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>y</i> の入力データの形状を定義する。詳細は、「 データの割り当て 」を参照。
<i>zshape</i>	INTEGER, DIMENSION(*)	int[]	配列 <i>z</i> に格納される出力データの形状を定義する。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (<i>vslsconvnewtask</i> , <i>vsldconvnewtask</i> の場合) TYPE(VSL_CORR_TASK) (<i>vsldcorrnewtask</i> , <i>vsldcorrnewtask</i> の場合)	VSLConvTaskPtr* (<i>vslsConvNewTask</i> , <i>vsldConvNewTask</i> の場合) VSLCorrTaskPtr* (<i>vsldCorrNewTask</i> , <i>vsldCorrNewTask</i> の場合)	正常に作成された場合は、タスク・ディスクリプターへのポインター。それ以外の場合は、NULL ポインター。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

NewTask1D

1次元の畳み込み / 関連タスク・ディスクリプターを新規に作成する。

構文

Fortran:

```

status = vslsconvnewtask1d(task, mode, xshape, yshape, zshape)
status = vsldconvnewtask1d(task, mode, xshape, yshape, zshape)
status = vsldcorrnewtask1d(task, mode, xshape, yshape, zshape)
status = vsldcorrnewtask1d(task, mode, xshape, yshape, zshape)

```

C:

```

status = vslsConvNewTask1D(task, mode, xshape, yshape, zshape);
status = vsldConvNewTask1D(task, mode, xshape, yshape, zshape);
status = vsldCorrNewTask1D(task, mode, xshape, yshape, zshape);
status = vsldCorrNewTask1D(task, mode, xshape, yshape, zshape);

```

説明

各 NewTask1D コンストラクターは、明示的パラメーターにユーザー指定の値を使用した新規の畳み込み / 関連タスク・ディスクリプターを作成する。オプション・パラメーターはデフォルト値に設定される (表 10-14 を参照)。

[NewTask](#) と異なり、これらのルーチンは、パラメーター *dims* の値が 1 であると仮定するコンストラクターの特殊な 1 次元バージョンを表す。

パラメーター *xshape*, *yshape*, *zshape* は、配列 *x* と *y* から読み出される成分の個数、または配列 *z* に格納される成分の個数に等しい。コンストラクターを呼び出す際に形状パラメーターを明示的に割り当てる。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 表 10-18 は指定可能な値をまとめたものである。
<i>xshape</i>	INTEGER	int	ソース配列 <i>x</i> の入力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。
<i>yshape</i>	INTEGER	int	ソース配列 <i>y</i> の入力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。
<i>zshape</i>	INTEGER	int	配列 <i>z</i> に格納される出力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE (VSL_CONV_TASK) (<i>vslsconvnewtaskld</i> , <i>vsldconvnewtaskld</i> の場合)	VSLConvTaskPtr* (<i>vslsConvNewTask1D</i> , <i>vsldConvNewTask1D</i> の場合)	正常に作成された場合は、タスク・ディスクリプターへのポインター。それ以外の場合は、NULL ポインター。
	TYPE (VSL_CORR_TASK) (<i>vsldcorrnewtaskld</i> , <i>vsldcorrnewtaskld</i> の場合)	VSLCorrTaskPtr* (<i>vsldCorrNewTask1D</i> , <i>vsldCorrNewTask1D</i> の場合)	
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

NewTaskX

多次元の畳み込み / 相関タスク・ディスクリプターを新規に作成して、最初の演算子ベクトルにソースデータを割り当てる。

構文

Fortran:

```
status = vslsconvnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
status = vsldconvnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
```

```
status = vslscorrnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx(task, mode, dims, xshape, yshape, zshape, x, xstride)
```

C:

```
status = vslsConvNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
    xstride);
status = vsldConvNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
    xstride);
status = vslsCorrNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
    xstride);
status = vsldCorrNewTaskX(task, mode, dims, xshape, yshape, zshape, x,
    xstride);
```

説明

各 `NewTaskX` コンストラクターは、明示的パラメーターにユーザー指定の値を使用した新規の畳み込み/相関タスク・ディスクリプターを作成する。オプション・パラメーターはデフォルト値に設定される ([表 10-14](#) を参照)。

[NewTask](#) と異なり、これらのルーチンは、コンストラクターの X 形式バージョンを表す。X 形式バージョンは、タスク・ディスクリプターの作成に加え、畳み込み/相関演算で使用される配列 `x` の最初の演算子ベクトルに特定のデータを割り当てる。`NewTaskX` コンストラクターによって作成されたタスク・ディスクリプターは、デストラクター・ルーチンによってタスク・オブジェクトが削除されるまで常に配列 `x` へのポインターを保持する ([DeleteTask](#) を参照)。

配列 `y` に格納された異なるベクトルに対して、配列 `x` に格納された同じデータベクトルを使用して複数の畳み込み/相関演算を行う必要がある場合、この形式のコンストラクターの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

パラメーター `xshape`、`yshape`、`zshape` はそれぞれ、配列 `x`、`y`、`z` で与えられる入力/出力データの形状を定義する。各形状パラメーターは、`dims` の値と同じ長さの整数配列である。コンストラクターを呼び出す際に形状パラメーターを明示的に割り当てる。パラメーター `dims` の値が 1 の場合、`xshape`、`yshape`、`zshape` は配列 `x` と `y` から読み取られる成分の個数、または配列 `z` に格納される成分の個数に等しい。ストライドが割り当てられた場合、形状パラメーターの値は配列 `x`、`y`、`z` の物理的形状とは異なることがある。

ストライド・パラメーター `xstride` は、配列 `x` における入力データの物理的な位置を指定する。1 次元の場合、ストライドは配列の続く成分間の区間である。例えば、パラメーター `xstride` の値が `s` の場合、配列 `x` の `s` 番目ごとの成分のみ、入力シーケンスを形成するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。表 10-18 は指定可能な値をまとめたものである。
<i>dims</i>	INTEGER	int	ユーザーデータの階数 実行段階で使用される入力 / 出力配列 <i>x</i> 、 <i>y</i> 、 <i>z</i> の次元を指定する。範囲は 1 ~ 7 でなければならない。値はコンストラクターによって明示的に割り当てられる。
<i>xshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>x</i> の入力データの形状を定義する。詳細は、「データの割り当て」を参照。
<i>yshape</i>	INTEGER, DIMENSION(*)	int[]	ソース配列 <i>y</i> の入力データの形状を定義する。詳細は、「データの割り当て」を参照。
<i>zshape</i>	INTEGER, DIMENSION(*)	int[]	配列 <i>z</i> に格納される出力データの形状を定義する。詳細は、「データの割り当て」を参照。
<i>x</i>	REAL(KIND=4), DIMENSION(*) (単精度の場合) REAL(KIND=8), DIMENSION(*) (倍精度の場合)	float[] (単精度の場合) double[] (倍精度の場合)	最初の演算子ベクトルの入力データを含む配列へのポインター。詳細は、「データの割り当て」を参照。
<i>xstride</i>	INTEGER, DIMENSION(*)	int[]	配列 <i>x</i> の入力データのストライド

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (<i>vslsconvnewtaskx</i> , <i>vsldconvnewtaskx</i> の場合) TYPE(VSL_CORR_TASK) (<i>vsldcorrnewtaskx</i> , <i>vsldcorrnewtaskx</i> の場合)	VSLConvTaskPtr* (<i>vslsConvNewTaskX</i> , <i>vsldConvNewTaskX</i> の場合) VSLCorrTaskPtr* (<i>vsldCorrNewTaskX</i> , <i>vsldCorrNewTaskX</i> の場合)	正常に作成された場合は、タスク・ディスクリプターへのポインター。それ以外の場合は、NULL ポインター。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

NewTaskX1D

1次元の畳み込み / 相関タスク・ディスクリプターを新規に作成して、最初の演算子ベクトルにソースデータを割り当てる。

構文

Fortran:

```
status = vslsconvnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vsldconvnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vslscorrnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
status = vsldcorrnewtaskx1d(task, mode, xshape, yshape, zshape, x, xstride)
```

C:

```
status = vslsConvNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vsldConvNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vslsCorrNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
status = vsldCorrNewTaskX1D(task, mode, xshape, yshape, zshape, x, xstride);
```

説明

各 NewTaskX1D コンストラクターは、明示的パラメーターにユーザー指定の値を使用した新規の畳み込み / 相関タスク・ディスクリプターを作成する。オプション・パラメーターはデフォルト値に設定される ([表 10-14](#) を参照)。

これらのルーチンは、コンストラクターの X 形式の特殊な 1 次元バージョンを表す。パラメーター [dims](#) の値は 1 であると仮定し、タスク・ディスクリプターの作成に加え、コンストラクター・ルーチンは畳み込み / 相関演算で使用される配列 x の最初の演算子ベクトルに特定のデータを割り当てる。NewTaskX1D コンストラクターによって作成されたタスク・ディスクリプターは、デストラクター・ルーチンによってタスク・オブジェクトが削除されるまで常に配列 x へのポインターを保持する ([DeleteTask](#) を参照)。

配列 y に格納された異なるベクトルに対して、配列 x に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、この形式のコンストラクターの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

パラメーター $xshape$ 、 $yshape$ 、 $zshape$ は、配列 x と y から読み出される成分の個数、または配列 z に格納される成分の個数に等しい。コンストラクターを呼び出す際に形状パラメーターを明示的に割り当てる。

ストライド・パラメーター $xstride$ は、配列の続く成分間の区間であり、配列 x における入力データの物理的な位置を指定する。例えば、パラメーター $xstride$ の値が s の場合、配列 x の s 番目ごとの成分のみ、入力シーケンスを形成するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>mode</i>	INTEGER	int	畳み込み / 相関演算を直接法を使用して実行すべきか、または入力データのフーリエ変換によって実行すべきかを指定する。 表 10-18 は指定可能な値をまとめたものである。
<i>xshape</i>	INTEGER	int	ソース配列 <i>x</i> の入力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。
<i>yshape</i>	INTEGER	int	ソース配列 <i>y</i> の入力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。
<i>zshape</i>	INTEGER	int	配列 <i>z</i> に格納される出力データシーケンスの長さを定義する。詳細は、「 データの割り当て 」を参照。
<i>x</i>	REAL (KIND=4), DIMENSION (*) (単精度の場合) REAL (KIND=8), DIMENSION (*) (倍精度の場合)	float[] (単精度の場合) double[] (倍精度の場合)	最初の演算子ベクトルの入力データを含む配列へのポインタ。詳細は、「 データの割り当て 」を参照。
<i>xstride</i>	INTEGER	int	配列 <i>x</i> の入力データシーケンスのストライド。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE (VSL_CONV_TASK) (<i>vslsconvnewtaskx1d</i> , <i>vsldconvnewtaskx1d</i> の場合) TYPE (VSL_CORR_TASK) (<i>vsldcorrnewtaskx1d</i> , <i>vsldcorrnewtaskx1d</i> の場合)	VSLConvTaskPtr* (<i>vslsConvNewTaskX1D</i> , <i>vsldConvNewTaskX1D</i> の場合) VSLCorrTaskPtr* (<i>vsldCorrNewTaskX1D</i> , <i>vsldCorrNewTaskX1D</i> の場合)	正常に作成された場合は、タスク・ディスクリプターへのポインタ。それ以外の場合は、NULLポインタ。
<i>status</i>	INTEGER	int	タスクが正常に作成された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

タスクエディター

インテル MKL の畳み込み / 相関 API のタスクエディターは、次のタスク・パラメーターの設定または変更を行うためのルーチンである (表 10-14 を参照)。

- `mode`
- `internal_precision`
- `start`
- `decimation`

各パラメーターの設定または変更には、別々のルーチンが存在する。



注: タスク・ディスクリプター構造のフィールドは、ソフトウェアに備えられたタスク・エディター・ルーチンのセットによってのみアクセス可能である。

タスク・ディスクリプターの設定を変更するためにエディタールーチンを適用すると、タスクの遂行ステータスが失われ、次回の実行またはコピー時に再度、完全な遂行処理が行われる。これは、前回の遂行処理中に格納された現在の作業データは、新しいパラメーター設定に対して無効となることがあるためである。タスク遂行に関する詳細は、「概要」を参照のこと。

表 10-17 は利用可能なタスクエディターをまとめたものである。

表 10-17 **タスクエディター**

ルーチン	説明
SetMode	畳み込み / 相関演算のためにパラメーター <code>mode</code> の値を変更する。
SetInternalPrecision	畳み込み / 相関演算のためにパラメーター <code>internal_precision</code> の値を変更する。
SetStart	畳み込み / 相関演算のためにパラメーター <code>start</code> の値を設定する。
SetDecimation	畳み込み / 相関演算のためにパラメーター <code>decimation</code> の値を設定する。



注: エディタールーチンの呼び出しに NULL タスクポインターを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

SetMode

畳み込み / 相関演算タスク・ディスクリプターのパラメーター *mode* の値を変更する。

構文

Fortran:

```
status = vslconvsetmode(task, newmode)
status = vslcorrsetmode(task, newmode)
```

C:

```
status = vslConvSetMode(task, newmode);
status = vslCorrSetMode(task, newmode);
```

説明

このルーチンは、畳み込み / 相関演算のためにパラメーター *mode* の値を変更する。このパラメーターは、入力 / 出力データのフーリエ変換によって計算を行うか、または直接法を使用して計算を行うかを定義する。*mode* の初期値はタスク・コンストラクターによって割り当てられる。

mode パラメーターの定義済みの値を以下に示す。

表 10-18 *mode* パラメーターの値

値	目的
VSL_CONV_MODE_FFT	高速フーリエ変換により畳み込み演算を行う。
VSL_CORR_MODE_FFT	高速フーリエ変換により相関演算を行う。
VSL_CONV_MODE_DIRECT	直接畳み込み演算を行う。
VSL_CORR_MODE_DIRECT	直接相関演算を行う。
VSL_CONV_MODE_AUTO	畳み込み演算方法 (直接またはフーリエ) を自動的に選択する。
VSL_CORR_MODE_AUTO	相関演算方法 (直接またはフーリエ) を自動的に選択する。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE (VSL_CONV_TASK) (vslconvsetmode の場合)	VSLConvTaskPtr (vslConvSetMode の場合)	タスク・ディスクリプターへのポインター。
	TYPE (VSL_CORR_TASK) (vslcorrsetmode の場合)	VSLCorrTaskPtr (vslCorrSetMode の場合)	
<i>newmode</i>	INTEGER	int	パラメーター <i>mode</i> の新しい値。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

SetInternalPrecision

畳み込み / 相関演算タスク・ディスクリプターのパラメーター *internal_precision* の値を変更する。

構文

Fortran:

```
status = vslconvsetinternalprecision(task, precision)
status = vslcorrsetinternalprecision(task, precision)
```

C:

```
status = vslConvSetInternalPrecision(task, precision);
status = vslCorrSetInternalPrecision(task, precision);
```

説明

このルーチンは、畳み込み / 相関演算のためにパラメーター *internal_precision* の値を変更する。このパラメーターは、畳み込み / 相関結果の中間計算を単精度で行うべきか、または倍精度で行うべきか定義する。*internal_precision* の初期値はタスク・コンストラクターによって割り当てられ、使用されるコンストラクターのデータ型に従って「単精度」または「倍精度」のいずれかに設定される。

internal_precision を変更すると、デフォルト設定が「単精度」で、入力 / 出力データが単精度の場合でも倍精度の計算結果を求めることができる。

internal_precision 入力パラメーターの定義済みの値を以下に示す。

表 10-19 *internal_precision* パラメーターの値

値	目的
VSL_CONV_PRECISION_SINGLE	単精度の畳み込み演算を行う。
VSL_CORR_PRECISION_SINGLE	単精度の相関演算を行う。
VSL_CONV_PRECISION_DOUBLE	倍精度の畳み込み演算を行う。
VSL_CORR_PRECISION_DOUBLE	倍精度の相関演算を行う。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (<i>vslconvsetinternalp</i> <i>recision</i> の場合)	VSLConvTaskPtr (<i>vslConvSetInternalP</i> <i>recision</i> の場合)	タスク・ディスクリプターへの ポインター。
	TYPE(VSL_CORR_TASK) (<i>vslcorrsetinternalp</i> <i>recision</i> の場合)	VSLCorrTaskPtr (<i>vslCorrSetInternalP</i> <i>recision</i> の場合)	
<i>precision</i>	INTEGER	int	パラメーター <i>internal_precision</i> の新 しい値。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

SetStart

畳み込み / 相関演算タスク・ディスクリプター
のパラメーター *start* の値を変更する。

構文

Fortran:

```
status = vslconvsetstart(task, start)
status = vslcorrsetstart(task, start)
```

C:

```
status = vslConvSetStart(task, start);
status = vslCorrSetStart(task, start);
```

説明

このルーチンは、畳み込み / 相関演算のためにパラメーター *start* の値を設定する。1次元の場合、このパラメーターは出力配列に格納される数値演算結果の最初の成分を指す。多次元の場合、*start* はインデックスの配列であり、その長さはパラメーター *dims* で指定された次元に等しい。このパラメーターの定義と効果に関する詳細は、「[データの割り当て](#)」を参照のこと。

タスク・ディスクリプターの初期構成時、*start* のデフォルト値は未定義で、このパラメーターは使用されていない。そのため、*start* パラメーターを設定して使用する唯一の方法は、SetStart ルーチンを使用して値を割り当てることである。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (vslconvsetstart の場合) TYPE(VSL_CORR_TASK) (vslcorrsetstart の場合)	VSLConvTaskPtr (vslConvSetStart の 場合) VSLCorrTaskPtr (vslCorrSetStart の 場合)	タスク・ディスクリプターへのポ インター。
<i>start</i>	INTEGER, DIMENSION (*)	int[]	パラメーター <i>start</i> の新しい値。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

SetDecimation

畳み込み / 相関演算タスク・ディスクリプター
のパラメーター *decimation* の値を変更する。

構文

Fortran:

```
status = vslconvsetdecimation(task, decimation)
status = vslcorrsetdecimation(task, decimation)
```

C:

```
status = vslConvSetDecimation(task, decimation);
status = vslCorrSetDecimation(task, decimation);
```

説明

このルーチンは、畳み込み / 相関演算のためにパラメーター *decimation* の値を設定する。

このパラメーターは、畳み込み / 相関の数値演算結果を、出力データ配列に書き込む前に減少させる方法を定義する。例えば、1次元の場合、*decimation* = $d > 1$ ならば数値演算結果の d 番目ごとの成分のみ出力配列 z に書き込まれる。

多次元の場合、*decimation* はインデックスの配列であり、その長さはパラメーター *dims* で指定された次元に等しい。このパラメーターの定義と効果に関する詳細は、「[データの割り当て](#)」を参照のこと。

タスク・ディスクリプターの初期構成時、*decimation* のデフォルト 値は未定義で、このパラメーターは使用されていない。そのため、*decimation* パラメーターを設定して使用する唯一の方法は、*SetDecimation* ルーチンを使用して値を割り当てることである。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (<i>vslconvsetdecimation</i> の場合)	VSLConvTaskPtr (<i>vslConvSetDecimation</i> の場合)	タスク・ディスクリプターへのポインター。
	TYPE(VSL_CORR_TASK) (<i>vslcorrsetdecimation</i> の場合)	VSLCorrTaskPtr (<i>vslCorrSetDecimation</i> の場合)	
<i>decimation</i>	INTEGER, DIMENSION(*)	int[]	パラメーター <i>decimation</i> の新しい値。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>status</i>	INTEGER	int	タスクの現在のステータス。

タスク実行ルーチン

タスク実行ルーチンは、タスク・ディスクリプターに格納されたパラメーターと入力ベクトルに対して与えられたユーザーデータを基に畳み込み / 相関結果を計算する。

いったん作成および調整されると、タスクは同じデータ型、精度、および形状の異なる入力 / 出力データに適用することによって複数回実行することができる。

インテル MKL の畳み込み / 相関 API では、一般形式と X 形式の 2 つの異なる形式の実行ルーチンを用意している。一般形式の実行ルーチンは、一般形式のコンストラクターによって作成されたタスク・ディスクリプターを使用し、入力時に 2 つのソースデータ配列 *x* と *y* を取得する。一方、X 形式の実行ルーチンは、X 形式のコンストラクターによって作成されたタスク・ディスクリプターを使用する。最初の配列 *x* は構成時にすでに指定されているため、入力時には 1 つのソースデータ配列 *y* のみを取得する。

タスクを最初に実行する際、実行ルーチンはタスク遂行処理を行う。この処理は、パラメーターの一貫性のチェックと補助データの準備 (例えば、入力データのフーリエ変換の計算など) の基本的な 2 つのステップで構成される。

各実行ルーチンに対して、関連する 1 次元のバージョンが存在する。このバージョンは、1 次元のデータ構造の単純性によってもたらされる、アルゴリズムおよび計算上の利点を活用する。



注: 実行ルーチンの呼び出しに NULL タスクポインターを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

タスクが正常に実行されると、実行ルーチンはゼロ・ステータス・コードを返す。エラーが検出されると、実行ルーチンは特定のエラーの発生を知らせるエラーコードを返す。次の場合、エラー・ステータス・コードが返される。

- タスクポインターが NULL の場合
- タスク・ディスクリプターが壊れている場合
- その他の理由で計算に失敗した場合



注: インテル MKL は、オーバーフローや漸次アンダーフロー、NaN による演算、その他の浮動小数点エラーは制御しない。

エラーが発生した場合、タスク・ディスクリプターはエラーコードを格納する。

すべてのタスク実行ルーチンを次の表に示す。

表 10-20 **タスク実行ルーチン**

ルーチン	説明
Exec	多次元の畳み込み / 相関演算を行う。
Exec1D	1 次元の畳み込み / 相関演算を行う。
ExecX	多次元の畳み込み / 相関演算を X 形式として行う。
ExecX1D	1 次元の畳み込み / 相関演算を X 形式として行う。

Exec

多次元の畳み込み / 相関演算を行う。

構文

Fortran:

```
status = vslsconvexec(task, x, xstride, y, ystride, z, zstride)
```

```
status = vsldconvexec(task, x, xstride, y, ystride, z, zstride)
```

```
status = vslscorexec(task, x, xstride, y, ystride, z, zstride)
```

```
status = vsldcorexec(task, x, xstride, y, ystride, z, zstride)
```

C:

```

status = vslsConvExec(task, x, xstride, y, ystride, z, zstride);
status = vsldConvExec(task, x, xstride, y, ystride, z, zstride);
status = vslsCorrExec(task, x, xstride, y, ystride, z, zstride);
status = vsldCorrExec(task, x, xstride, y, ystride, z, zstride);

```

説明

各 Exec ルーチンは、配列 x と y で与えられたデータの畳み込み / 相関を計算し、その結果を配列 z に格納する。処理のパラメーターは、対応する [NewTask](#) コンストラクターによって以前に作成され、 $task$ によってポイントされたタスク・ディスクリプターから読み出される。 $task$ が NULL の場合、処理は行われない。

ストライド・パラメーター $xstride$ 、 $ystride$ 、 $zstride$ はそれぞれ、配列 x 、 y 、 z の入力 / 出力データの物理的な位置を指定する。1次元の場合、ストライドは配列の続く成分間の区間である。例えば、パラメーター $zstride$ の値が s の場合、配列 z の s 番目ごとの成分のみ、出力データを格納するのに使用される。ストライド値は、正または負でなければならない。ゼロであってはならない。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
$task$	TYPE(VSL_CONV_TASK) (vslsconvexec、 vsldconvexec の場合) TYPE(VSL_CORR_TASK) (vslsCorrexec、 vsldCorrexec の場合)	VSLConvTaskPtr (vslsConvExec、 vsldConvExec の場合) VSLCorrTaskPtr (vslsCorrExec、 vsldCorrExec の場合)	タスク・ディスクリプターへのポインター。
x , y	REAL(KIND=4), DIMENSION(*) (vslsconvexec、 vslsCorrexec の場合) REAL(KIND=8), DIMENSION(*) (vsldconvexec、 vsldCorrexec の場合)	float[] (vslsConvExec、 vslsCorrExec の場合) double[] (vsldConvExec、 vsldCorrExec の場合)	入力データを含む配列へのポインター。詳細は、「 データの割り当て 」を参照。
$xstride$, $ystride$, $zstride$	INTEGER, DIMENSION(*)	int[]	入力 / 出力データのストライド。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>z</i>	REAL(KIND=4), DIMENSION(*) (<i>vslsconvexec</i> 、 <i>vslscorrexec</i> の場合) REAL(KIND=8), DIMENSION(*) (<i>vsldconvexec</i> 、 <i>vsldcorrexec</i> の場合)	float[] (<i>vslsConvExec</i> 、 <i>vslsCorrExec</i> の場合) double[] (<i>vsldConvExec</i> 、 <i>vsldCorrExec</i> の場合)	出力データを格納する配列へのポインター。詳細は、「 データの割り当て 」を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

Exec1D

1 次元の畳み込み / 相関演算を行う。

構文

Fortran:

```
status = vslsconvexeclD(task, x, xstride, y, ystride, z, zstride)
status = vsldconvexeclD(task, x, xstride, y, ystride, z, zstride)
status = vslscorrexeclD(task, x, xstride, y, ystride, z, zstride)
status = vsldcorrexeclD(task, x, xstride, y, ystride, z, zstride)
```

C:

```
status = vslsConvExeclD(task, x, xstride, y, ystride, z, zstride);
status = vsldConvExeclD(task, x, xstride, y, ystride, z, zstride);
status = vslsCorrExeclD(task, x, xstride, y, ystride, z, zstride);
status = vsldCorrExeclD(task, x, xstride, y, ystride, z, zstride);
```

説明

各 Exec1D ルーチンは、配列 *x* と *y* で与えられたデータの畳み込み / 相関を計算し、その結果を配列 *z* に格納する。これらのルーチンは、パラメーター [dims](#) の値が 1 であると仮定し、演算の特殊な 1 次元のバージョンを表す。実行ルーチンのこのバージョンを使用すると、1 次元データにおける性能を向上させることができる。

処理のパラメーターは、対応する [NewTask1D](#) コンストラクターによって以前に作成され、*task* によってポイントされたタスク・ディスクリプターから読み出される。*task* が NULL の場合、処理は行われない。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE(VSL_CONV_TASK) (<i>vslsconvexeclD</i> 、 <i>vsldconvexeclD</i> の場合) TYPE(VSL_CORR_TASK) (<i>vsldcorrexeclD</i> 、 <i>vsldcorrexeclD</i> の場合)	VSLConvTaskPtr (<i>vslsConvExec1D</i> 、 <i>vsldConvExec1D</i> の場合) VSLCorrTaskPtr (<i>vslsCorrExec1D</i> 、 <i>vsldCorrExec1D</i> の場合)	タスク・ディスクリプターへのポインター。
<i>x</i> , <i>y</i>	REAL(KIND=4) , DIMENSION(*) (<i>vslsconvexeclD</i> 、 <i>vsldconvexeclD</i> の場合) REAL(KIND=8) , DIMENSION(*) (<i>vsldconvexeclD</i> 、 <i>vsldcorrexeclD</i> の場合)	float[] (<i>vslsConvExec1D</i> 、 <i>vsldConvExec1D</i> の場合) double[] (<i>vsldConvExec1D</i> 、 <i>vsldCorrExec1D</i> の場合)	入力データを含む配列へのポインター。詳細は、「 データの割り当て 」を参照。
<i>xstride</i> , <i>ystride</i> , <i>zstride</i>	INTEGER , DIMENSION(*)	int[]	入力 / 出力データのストライド。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>z</i>	REAL(KIND=4) , DIMENSION(*) (<i>vslsconvexeclD</i> 、 <i>vsldconvexeclD</i> の場合) REAL(KIND=8) , DIMENSION(*) (<i>vsldconvexeclD</i> 、 <i>vsldcorrexeclD</i> の場合)	float[] (<i>vslsConvExec1D</i> 、 <i>vsldConvExec1D</i> の場合) double[] (<i>vsldConvExec1D</i> 、 <i>vsldCorrExec1D</i> の場合)	出力データを格納する配列へのポインター。詳細は、「 データの割り当て 」を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

ExecX

最初の演算子ベクトルが固定である多次元の
畳み込み / 相関演算を行う。

構文

Fortran:

```
status = vslsconvexec(task, y, ystride, z, zstride)
status = vsldconvexec(task, y, ystride, z, zstride)
status = vslscorexec(task, y, ystride, z, zstride)
status = vsldcorexec(task, y, ystride, z, zstride)
```

C:

```
status = vslsConvExecX(task, y, ystride, z, zstride);
status = vsldConvExecX(task, y, ystride, z, zstride);
status = vslsCorrExecX(task, y, ystride, z, zstride);
status = vsldCorrExecX(task, y, ystride, z, zstride);
```

説明

各 ExecX ルーチンは、配列 x と y で与えられたデータの畳み込み / 相関を計算し、その結果を配列 z に格納する。これらのルーチンは、最初の演算子ベクトルはタスク構造時に設定され、タスクポインターは配列 x へのポインターを保持すると仮定する、演算の特殊なバージョンを表す。

処理のパラメーターは、対応する [NewTaskX](#) コンストラクターによって以前に作成され、 $task$ によってポイントされたタスク・ディスクリプターから読み出される。 $task$ が NULL の場合、処理は行われない。

配列 y に格納された異なるベクトルに対して、配列 x に格納された同じデータベクトルを使用して複数の畳み込み / 相関演算を行う必要がある場合、この形式の実行ルーチンの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
$task$	TYPE(VSL_CONV_TASK) (vslsconvexec、 vsldconvexec の場合) TYPE(VSL_CORR_TASK) (vslscorexec、 vsldcorexec の場合)	VSLConvTaskPtr (vslsConvExecX、 vsldConvExecX の場合) VSLCorrTaskPtr (vslsCorrExecX、 vsldCorrExecX の場合)	タスク・ディスクリプターへの ポインター。

名前	データ型		説明
	FORTRAN	C	
<i>x</i> , <i>y</i>	REAL(KIND=4), DIMENSION(*) (<i>vslsconvexecx</i> 、 <i>vslsruorexecx</i> の場合) REAL(KIND=8), DIMENSION(*) (<i>vsldconvexecx</i> 、 <i>vsldcorrexecx</i> の場合)	float[] (<i>vslsConvExecX</i> 、 <i>vslsruCorrExecX</i> の場合) double[] (<i>vsldConvExeX</i> 、 <i>vsldCorrExecX</i> の場合)	2番目の演算子ベクトルの入力データを含む配列へのポインタ。詳細は、「 データの割り当て 」を参照。
<i>xstride</i> , <i>ystride</i> , <i>zstride</i>	INTEGER, DIMENSION(*)	int[]	入力/出力データのストライド。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>z</i>	REAL(KIND=4), DIMENSION(*) (<i>vslsconvexecx</i> 、 <i>vslsruorexecx</i> の場合) REAL(KIND=8), DIMENSION(*) (<i>vsldconvexecx</i> 、 <i>vsldcorrexecx</i> の場合)	float[] (<i>vslsConvExecX</i> 、 <i>vslsruCorrExecX</i> の場合) double[] (<i>vsldConvExecX</i> 、 <i>vsldCorrExecX</i> の場合)	出力データを格納する配列へのポインタ。詳細は、「 データの割り当て 」を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合はVSL_STATUS_OKに設定し、それ以外の場合は非ゼロのエラーコードに設定する。

ExecX1D

最初の演算子ベクトルが固定である1次元の畳み込み/相関演算を行う。

構文

Fortran:

```
status = vslsconvexecx1d(task, y, ystride, z, zstride)
status = vsldconvexecx1d(task, y, ystride, z, zstride)
status = vslsruorexecx1d(task, y, ystride, z, zstride)
status = vsldcorrexecx1d(task, y, ystride, z, zstride)
```

C:

```
status = vslsConvExecX1D(task, y, ystride, z, zstride);
```

```
status = vsldConvExecX1D(task, y, ystride, z, zstride);
status = vslsCorrExecX1D(task, y, ystride, z, zstride);
status = vsldCorrExecX1D(task, y, ystride, z, zstride);
```

説明

各 ExecX1D ルーチンは、配列 x と y で与えられたデータの畳み込み / 相関を ($dims=1$ と仮定して) 計算し、その結果を配列 z に格納する。これらのルーチンは、最初の演算子ベクトルがタスク構造時に設定されることを前提とする、演算の特殊なバージョンを表す。

処理のパラメーターは、対応する [NewTaskX1D](#) コンストラクターによって以前に作成され、 $task$ によってポイントされたタスク・ディスクリプターから読み出される。 $task$ が NULL の場合、処理は行われぬ。

配列 y に格納された異なるベクトルに対して、配列 x に格納された同じデータベクトルを使用して複数の 1 次元畳み込み / 相関演算を行う必要がある場合、この形式の実行ルーチンの使用が推奨される。これは、演算に必要な中間データの繰り返し計算で発生する不必要なオーバーヘッドを排除し、性能を向上させるためである。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
$task$	TYPE(VSL_CONV_TASK) (vslsconvexecx1d、 vsldconvexecx1d の場合) TYPE(VSL_CORR_TASK) (vsldcorrexecx1d、 vsldcorrexecx1d の場合)	VSLConvTaskPtr (vslsConvExecX1D、 vsldConvExecX1D の場合) VSLCorrTaskPtr (vsldCorrExecX1D、 vsldCorrExecX1D の場合)	タスク・ディスクリプターへのポインター。
x, y	REAL(KIND=4), DIMENSION(*) (vslsconvexecx1d、 vsldconvexecx1d の場合) REAL(KIND=8), DIMENSION(*) (vsldconvexecx1d、 vsldcorrexecx1d の場合)	float[] (vsldConvExecX1D、 vsldCorrExecX1D の場合) double[] (vsldConvExecX1D、 vsldCorrExecX1D の場合)	2 番目の演算子ベクトルの入力データを含む配列へのポインター。詳細は、「 データの割り当て 」を参照。
$xstride, ystride, zstride$	INTEGER, DIMENSION(*)	int[]	入力 / 出力データのストライド。詳細は、「 データの割り当て 」を参照。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>z</i>	REAL(KIND=4), DIMENSION(*), (<i>vslsconvexecx1d</i> , <i>vslscorreexecx1d</i> の場合) REAL(KIND=8), DIMENSION(*), (<i>vsldconvexecx1d</i> , <i>vsldcorreexecx1d</i> の場合)	float[] (<i>vslsConvExecX1D</i> , <i>vslsCorrExecX1D</i> の場合) double[] (<i>vsldConvExecX1D</i> , <i>vsldCorrExecX1D</i> の場合)	出力データを格納する配列へのポインター。詳細は、「 データの割り当て 」を参照。
<i>status</i>	INTEGER	int	タスクが正常に実行された場合は VSL_STATUS_OK に設定し、それ以外の場合は非ゼロのエラーコードに設定する。

タスク・ディストラクター

タスク・ディストラクターは、タスク・オブジェクトを削除し、メモリーを開放するためのルーチンである。

DeleteTask

タスク・オブジェクトを削除し、メモリーを開放する。

構文

Fortran:

```
errcode = vslconvdeletetask(task)
errcode = vslcorrdeletetask(task)
```

C:

```
errcode = vslConvDeleteTask(task);
errcode = vslCorrDeleteTask(task);
```

説明

タスク・ディスクリプターへのポインターを与えられると、このルーチンはタスク・ディスクリプター・オブジェクトを削除し、データ構造に割り当てられていたメモリーを開放する。タスクに作業メモリーがある場合、作業メモリーも開放される。タスクポインターは NULL に設定される。

何らかの理由でタスクが正常に削除されない場合、ルーチンはエラーコードを返す。このエラーコードは、タスクステータスとは関係がなく、タスクステータスを変更しない。



注: デストラクター・ルーチンの呼び出しに NULL タスクポインターを使用することができる。この場合、ルーチンは終了し、システム障害は発生しない。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>task</i>	TYPE (VSL_CONV_TASK) (vslconvdeletetask の場合) TYPE (VSL_CORR_TASK) (vslcorrdeletetask の場合)	VSLConvTaskPtr* (vslConvDeleteTask の場合) VSLCorrTaskPtr* (vslCorrDeleteTask の場合)	タスク・ディスクリプターへのポインター。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>errcode</i>	INTEGER	int	タスク・オブジェクトが正常に削除された場合は 0。エラーが発生した場合は、エラーコードを含む。

タスクコピー

このルーチンは、畳み込み / 関連タスク・ディスクリプターをコピーする。

CopyTask

畳み込み / 関連タスクのディスクリプターをコピーする。

構文

Fortran:

```
status = vslconvcopytask(newtask, srctask)
```

```
status = vslcorrcopytask(newtask, srctask)
```

C:

```
status = vslConvTaskCopy(newtask, srctask);
status = vslCorrTaskCopy(newtask, srctask);
```

説明

タスク・オブジェクト *srctask* がすでに存在する場合、適切な CopyTask ルーチンを使用して *newtask* にコピーを作成することができる。コピー後、元のタスク・オブジェクトと新しいタスク・オブジェクトは遂行される（タスク遂行については、「[概要](#)」を参照）。元のタスクが過去に遂行されていない場合、コピーを開始する前に遂行処理が暗黙的に呼び出される。元のタスクを遂行中にエラーが発生すると、ステータスフィールドにエラーコードが格納される。コピー中にエラーが発生すると、このルーチンは新しいタスク・オブジェクトへの参照の代わりに NULL ポインターを返す。

入力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>srctask</i>	TYPE(VSL_CONV_TASK) (vslconvcopytask の場合) TYPE(VSL_CORR_TASK) (vslcorrcopytask の場合)	VSLConvTaskPtr (vslConvCopyTask の場合) VSLCorrTaskPtr (vslCorrCopyTask の場合)	元のタスク・ディスクリプターへのポインター。

出力パラメーター

名前	データ型		説明
	FORTRAN	C	
<i>newtask</i>	TYPE(VSL_CONV_TASK) (vslconvcopytask の場合) TYPE(VSL_CORR_TASK) (vslcorrcopytask の場合)	VSLConvTaskPtr* (vslConvCopyTask の場合) VSLCorrTaskPtr* (vslCorrCopyTask の場合)	新しいタスク・ディスクリプターへのポインター。
<i>status</i>	INTEGER	int	元のタスクの現在のステータス。

使用法の例

このセクションでは、インテル MKL を使用した一般的な畳み込み / 相関演算 (シングルスレッド化計算およびマルチスレッド化計算) の実行方法を示す。次の 2 つの関数例 `scond1` と `sconf1` は、IBM ESSL* ライブラリーの畳み込み / 相関関数 `SCOND` と `SCONF` をシミュレートする。これらの関数はシングルスレッド化計算を前提としており、C/C++ コンパイラーで使用することができる。

例 10-5 シングルスレッド化計算用関数 `scond1`

```
#include "mkl_vsl.h"

int scond1(
    float h[], int inch,
    float x[], int incx,
    float y[], int incy,
    int nh, int nx, int iy0, int ny)
{
    int status;
    VSLConvTaskPtr task;
    vslsConvNewTask1D(&task, VSL_CONV_MODE_DIRECT, nh, nx, ny);
    vslConvSetStart(task, &iy0);
    status = vslsConvExec1D(task, h, inch, x, incx, y, incy);
    vslConvDeleteTask(&task);
    return status;
}
```

例 10-6 シングルスレッド化計算関数 sconfl

```
#include "mkl_vsl.h"
int sconfl(
    int init,
    float h[], int inclh,
    float x[], int inclx, int inc2x,
    float y[], int incly, int inc2y,
    int nh, int nx, int m, int iy0, int ny,
    void* aux1, int naux1, void* aux2, int naux2)
{
    int status;
    /* assume that aux1!=0 and naux1 is big enough */
    VSLConvTaskPtr* task = (VSLConvTaskPtr*)aux1;
    if (init != 0)
        /* initialization: */
        status = vslsConvNewTaskX1D(task, VSL_CONV_MODE_FFT,
            nh, nx, ny, h, inclh);
    if (init == 0) {
        /* calculations: */
        int i;
        vslConvSetStart(*task, &iy0);
        for (i=0; i<m; i++) {
            float* xi = &x[inc2x * i];
            float* yi = &y[inc2y * i];
            /* task is implicitly committed at i==0 */
            status = vslsConvExecX1D(*task, xi, inclx, yi, incly);
        };
    };
    vslConvDeleteTask(task);
    return status;
}
```

マルチスレッドの使用

前述の `sconfl` のような関数では、循環よりも並列計算のほうが適している。 $m > 1$ の場合、マルチスレッドを使用して異なるデータシーケンスに対してタスクの実行を呼び出すことができる。この場合、計算する前にタスク・コピー・ルーチンを使用してタスク・オブジェクトのコピーを m 個作成し、これらのコピーを異なるスレッドで実行する。コピーする前に、([タスクエディター](#)を使用して)タスクに必要なパラメーター調整がすべて行われていることを確認する。

コード例を次に示す。

```
if (init == 0) {
    int i, status, ss[M];
    VSLConvTaskPtr tasks[M];
    /* assume that M is big enough */
    ...
    vslConvSetStart(*task, &iy0);
    ...
    for (i=0; i<m; i++)
        /* implicit commitment at i==0 */
        vslConvCopyTask(&tasks[i],*task);
    ...
}
```

ここで、タスクの異なるコピーを実行するために、m 個のスレッドが開始される。

```
...
    float* xi = &x[inc2x * i];
    float* yi = &y[inc2y * i];
    ss[i]=vslsConvExecX1D(tasks[i], xi,inc1x, yi,inc1y);
    ...
}
```

最後に (すべてのスレッドが計算を完了した後で)、全体のステータスがすべてのタスク・オブジェクトから回収される。次のコードは最初に検出されたエラー (存在する場合) を知らせる。

```
...
for (i=0; i<m; i++) {
    status = ss[i];
    if (status != 0) /* 0 means "OK" */
        break;
};
return status;
}; /* end if init==0 */
```

実行ルーチンはタスク内部ステート (タスク構造のフィールド) を変更する。異なるスレッドが同じタスク・オブジェクトを処理する場合、そのような変更は互いに矛盾を引き起こすことがある。このため、異なるスレッドはタスクの異なるコピーを使用しなければならない。

数学表記と定義

次の表記は、本文で使用されている基本となる数学定義を説明するのに必要である。

$\mathbf{R} = (-\infty, +\infty)$	実数の集合
$\mathbf{Z} = \{0, \pm 1, \pm 2, \dots\}$	整数の集合
$\mathbf{Z}^N = \mathbf{Z} \times \dots \times \mathbf{Z}$	N 次元の整数の級数のセット。
$p = (p_1, \dots, p_N) \in \mathbf{Z}^N$	N 次元の整数の級数。
$u: \mathbf{Z}^N \rightarrow \mathbf{R}$	\mathbf{Z}^N からの引数と \mathbf{R} からの値を持つ関数 u 。
$u(p) = u(p_1, \dots, p_N)$	引数 $p = (p_1, \dots, p_N)$ に対する関数 u の値。
$w = u * v$	関数 w は、関数 u 、 v の畳み込み。
$w = u \bullet v$	関数 w は、関数 u 、 v の相関。

級数 $p, q \in \mathbf{Z}^N$ が与えられたとき、

- 級数 $r = p + q$ は各 $n = 1, \dots, N$ に対して $r_n = p_n + q_n$ と定義される。
- 級数 $r = p - q$ は各 $n = 1, \dots, N$ に対して $r_n = p_n - q_n$ と定義される。
- 級数 $r = \sup\{p, q\}$ は各 $n = 1, \dots, N$ に対して $r_n = \max\{p_n, q_n\}$ と定義される。
- 級数 $r = \inf\{p, q\}$ は各 $n = 1, \dots, N$ に対して $r_n = \min\{p_n, q_n\}$ と定義される。
- 不等式 $p \leq q$ は各 $n = 1, \dots, N$ に対して $p_n \leq q_n$ を示す。

次の条件を満たす級数 $P^{\min}, P^{\max} \in \mathbf{Z}^N$ が存在する場合、関数 $u(p)$ は有限関数と呼ばれる。

$$u(p) \neq 0 \text{ は } P^{\min} \leq p \leq P^{\max} \text{ を意味する。}$$

畳み込み / 相関演算は有限関数に対してのみ定義される。

次の条件を満たす関数 u, v と級数 $P^{\min}, P^{\max}, Q^{\min}, Q^{\max} \in \mathbf{Z}^1$ について考える。

$$u(p) \neq 0 \text{ は } P^{\min} \leq p \leq P^{\max}$$

$$v(q) \neq 0 \text{ は } Q^{\min} \leq q \leq Q^{\max}$$

関数 u と v の線形畳み込み / 線形相関の定義は以下のとおりである。

線形畳み込み

関数 $w = u * v$ が u と v の畳み込みの場合：

$$w(r) \neq 0 \text{ は } R^{\min} \leq r \leq R^{\max}$$

$$\text{ここで、 } R^{\min} = P^{\min} + Q^{\min} \text{ および } R^{\max} = P^{\max} + Q^{\max} \text{。}$$

$R^{\min} \leq r \leq R^{\max}$ の場合：

$$w(r) = u(t) \cdot v(r - t) \text{ は、 } T^{\min} \leq t \leq T^{\max} \text{ を満たすすべての } t \in \mathbf{Z}^N \text{ の合計}$$

$$\text{ここで、 } T^{\min} = \sup\{P^{\min}, r - Q^{\max}\} \text{ および } T^{\max} = \inf\{P^{\max}, r - Q^{\min}\} \text{。}$$

線形相関

関数 $w = u \bullet v$ が u と v の相関の場合:

$$w(r) \neq 0 \text{ は } R^{\min} \leq r \leq R^{\max}$$

$$\text{ここで、 } R^{\min} = Q^{\min} - P^{\max} \text{ および } R^{\max} = Q^{\max} - P^{\min}。$$

$R^{\min} \leq r \leq R^{\max}$ の場合:

$$w(r) = u(t) \cdot v(r+t) \text{ は、 } T^{\min} \leq t \leq T^{\max} \text{ を満たすすべての } t \in \mathbf{Z}^N \text{ の合計}$$

$$\text{ここで、 } T^{\min} = \sup\{P^{\min}, Q^{\min} - r\} \text{ および } T^{\max} = \inf\{P^{\max}, Q^{\max} - r\}。$$

インテル MKL の畳み込み / 相関関数の入力 / 出力データとしての関数 u 、 v 、 w の表現は、次の「[データの割り当て](#)」セクションで説明する。

データの割り当て

このセクションでは次の関係について説明する。

- 「[数学表記と定義](#)」セクションの数学有限関数 u 、 v 、 w 。
- 関数 u 、 v 、 w を表す多次元入力 / 出力データベクトル。
- コンピューター・メモリーに入力 / 出力データベクトルを格納するのに使用する配列 x 、 y 、 z 。

入力 / 出力データの割り当てを決定する畳み込み / 相関ルーチン・パラメーターは次のとおりである。

- データ配列 x 、 y 、 z
- 形状配列 $xshape$ 、 $yshape$ 、 $zshape$
- 配列内のストライド $xstride$ 、 $ystride$ 、 $zstride$
- パラメーター $start$ 、 $decimation$

有限関数とデータベクトル

前述の有限関数 $u(p)$ 、 $v(q)$ 、 $w(r)$ は、入力 / 出力データの多次元ベクトルとして表される。

$\text{inputu}(i_1, \dots, i_{\text{dims}})$ ($u(p_1, \dots, p_N)$ の場合)

$\text{inputv}(j_1, \dots, j_{\text{dims}})$ ($v(q_1, \dots, q_N)$ の場合)

$\text{output}(k_1, \dots, k_{\text{dims}})$ ($w(r_1, \dots, r_N)$ の場合)

パラメーター $dims$ は次元を表し、 N に等しい。

パラメーター $xshape$ 、 $yshape$ 、 $zshape$ は入力 / 出力ベクトルの形状を定義する。

各 $n=1, \dots, dims$ に対して $1 \leq i_n \leq xshape(n)$ の場合、 $\text{inputu}(i_1, \dots, i_{\text{dims}})$ が定義される

各 $n=1, \dots, dims$ に対して $1 \leq j_n \leq yshape(n)$ の場合、 $\text{inputv}(j_1, \dots, j_{\text{dims}})$ が定義される

各 $n=1, \dots, dims$ に対して $1 \leq k_n \leq zshape(n)$ の場合、 $\text{output}(k_1, \dots, k_{\text{dims}})$ が定義される

入力ベクトルと関数 u 、 v の関係は次の式によって定義される。

$$\text{inputu}(i_1, \dots, i_{\text{dims}}) = u(p_1, \dots, p_N), \text{ 各 } n \text{ に対して } p_n = p_n^{\min} + (i_n - 1)$$

$$\text{inputv}(j_1, \dots, j_{\text{dims}}) = v(q_1, \dots, q_N), \text{ 各 } n \text{ に対して } q_n = q_n^{\min} + (j_n - 1)$$

出力ベクトルと関数 $w(r)$ の関係も似ている (ただし、パラメーター $start$ と $decimation$ が定義されていない場合のみ)。

$output(k_1, \dots, k_{dims}) = w(r_1, \dots, r_N)$ 、各 n に対して $r_n = R_n^{\min} + (k_n - 1)$

パラメーター $start$ が定義されている場合、区間 $R_n^{\min} \leq start(n) \leq R_n^{\max}$ 内でなければならない。定義されている場合、 $start$ パラメーターは次の式で R^{\min} を置き換える。

$output(k_1, \dots, k_{dims}) = w(r_1, \dots, r_N)$ 。ここで、 $r_n = start(n) + (k_n - 1)$

パラメーター $decimation$ が定義されている場合は、次の式に従って関係を変更する。

$output(k_1, \dots, k_{dims}) = w(r_1, \dots, r_N)$ 。ここで、 $r_n = R_n^{\min} + (k_n - 1) * decimation(n)$

パラメーター $start$ と $decimation$ の両方が定義されている場合の式は次のとおりである。

$output(k_1, \dots, k_{dims}) = w(r_1, \dots, r_N)$ 。

ここで、 $r_n = start(n) + (k_n - 1) * decimation(n)$

畳み込み / 相関ソフトウェアは、タスク遂行時に $zshape$ 、 $start$ 、 $decimation$ の値をチェックする。 $k_n, n=1, \dots, dims$ 、に対して r_n が R_n^{\max} を超えると、エラーが発生する。

データベクトルの割り当て

パラメーター配列 x と y はメモリーに入力データベクトルを格納し、配列 z は出力データベクトルを格納する。メモリーにアクセスするのに、畳み込み / 相関ソフトウェアはこれらの配列への唯一のポインターを使用し、配列の形状は無視する。

パラメーター x 、 y 、 z に対して、これらの配列の実際の長さがデータベクトルを格納するのに十分であることを条件とする 1次元の配列を与えることができる。

配列 x 、 y 、 z 内に入力 / 出力データの割り当てについて以下に示す。ここで配列は 1次元であると仮定する。多次元インデックス $i, j, k \in \mathbf{Z}^N$ が与えられると、1次元インデックス $e, f, g \in \mathbf{Z}$ は次のように定義される。

$inputu(i_1, \dots, i_{dims})$ は $x(e)$ で割り当てられる。

$inputv(j_1, \dots, j_{dims})$ は $y(f)$ で割り当てられる。

$output(k_1, \dots, k_{dims})$ は $z(g)$ で割り当てられる。

インデックス e 、 f 、 g は次のように定義される。

$e = 1 + xstride(n) \cdot dx(n)$ (すべての $n=1, \dots, dims$ に対する合計)

$f = 1 + ystride(n) \cdot dy(n)$ (すべての $n=1, \dots, dims$ に対する合計)

$g = 1 + zstride(n) \cdot dz(n)$ (すべての $n=1, \dots, dims$ に対する合計)

距離 $dx(n)$ 、 $dy(n)$ 、 $dz(n)$ はストライドの符号に依存する。

$xstride(n) > 0$ の場合は $dx(n) = i_n - 1$ 、

$xstride(n) < 0$ の場合は $dx(n) = i_n - xshape(n)$

$ystride(n) > 0$ の場合は $dy(n) = j_n - 1$ 、

$ystride(n) < 0$ の場合は $dy(n) = j_n - yshape(n)$

$zstride(n) > 0$ の場合は $dz(n) = k_n - 1$ 、
 $zstride(n) < 0$ の場合は $dz(n) = k_n - zshape(n)$

インデックス e 、 f 、 g の定義は、配列 x 、 y 、 z のインデックスが 1 から開始されることを前提としている。

$x(e)$ は $e = 1, \dots, \text{length}(x)$ に対して定義される。

$y(f)$ は $f = 1, \dots, \text{length}(y)$ に対して定義される。

$z(g)$ は $g = 1, \dots, \text{length}(z)$ に対して定義される。

1 次元または 2 次元の場合、多次元出力ベクトルの成分が配列 z にどのように格納されるのかを以下に説明する。

1 次元の場合： $dims = 1$ の場合、 $zshape$ は配列 z に格納される出力値の個数。配列 z の実際の長さは $zshape$ 個の成分よりも大きいことがある。

$zstride > 1$ の場合、出力値はストライドとともに格納される。 $output(1)$ は $z(1)$ に、 $output(2)$ は $z(1+zstride)$ に格納される。そのため、 z の実際の長さは $1+zstride*(zshape-1)$ 個の成分以上でなければならない。

$zstride < 0$ の場合も配列 z の成分間のストライドを定義する。ただし、使用される成分の次数は逆。 k 番目の出力値では、 $output(k)$ は $z(1+|zstride|*(zshape-k))$ に格納される。 $|zstride|$ は $zstride$ の絶対値。配列 z の実際の長さは、 $1+|zstride|*(zshape - 1)$ 個の成分以上でなければならない。

2 次元の場合： $dims = 2$ の場合、出力データは 2 次元の行列である。 $zstride(1)$ の値は、行列の列内のストライド、つまりインデックス k_1 、 k_2 それぞれの組み合わせに対する $output(k_1, k_2)$ と $output(k_1+1, k_2)$ の間のストライドを定義する。一方、 $zstride(2)$ は列間のストライド、つまり $output(k_1, k_2)$ と $output(k_1, k_2+1)$ の間のストライドを定義する。

$zstride(2)$ が $zshape(1)$ よりも大きい場合、列のスパース割り当てを引き起こす。 $zstride(2)$ の値が $zshape(1)$ よりも小さい場合、出力行列が転置されることがある。例えば、 $zshape = (2, 3)$ の場合、形状が 3×2 の転置行列のような出力値を割り当てるために $zstride = (3, 1)$ を定義することができる。

$zstride$ がこのような転置を前提としているかどうかに関わらず、異なる成分 $output(k_1, \dots, k_{dims})$ が異なる場所 $z(g)$ に格納されることを確認する必要がある。

フーリエ変換関数

11

本章では、インテル[®] MKL で利用可能な、次の離散フーリエ変換関数について説明する。

- シングル・プロセッサまたは共有メモリーシステムにおける離散フーリエ変換 (DFT) 関数 (以下の「[DFT 関数](#)」を参照)。
- 分散メモリー・アーキテクチャーにおける[クラスター DFT 関数](#) (インテル[®] クラスター・エディション製品でのみ利用可能)。

これらの DFT 関数は、使い勝手のよいアプリケーション・プログラム・インターフェイスとして提供され、高速フーリエ変換 (FFT) アルゴリズムによって高速に DFT 計算を実行する。



注: DFT 関数は任意の長さをサポートしている。これらのルーチンは、基数 2 だけでなく、基数 3、5、7、11 に対しても高性能で広範な機能性を提供する。対応する基数の一覧は、使用しているライブラリー・バージョンの「インテル MKL テクニカル・ユーザー・ノート」を参照のこと。

DFT 関数

インテル MKL の離散フーリエ変換関数ライブラリーは、1 次元、2 次元、多次元 (最大 7 次元) のルーチン、およびすべての変換関数で Fortran インターフェイスと C インターフェイスの両方を提供する。

インテル MKL に実装されているすべての DFT 関数を次の表に示す。

表 11-1 インテル MKL の DFT 関数

関数名	演算
ディスクリプター操作関数	
DftiCreateDescriptor	ディスクリプター・データ構造にメモリーを割り当て、デフォルト構成設定で具体化する。
DftiCommitDescriptor	実際の DFT 計算を実行可能にするすべての初期化を実行する。
DftiCopyDescriptor	既存のディスクリプターをコピーする。
DftiFreeDescriptor	ディスクリプターに割り当てられていたメモリーを解放する。

表 11-1 インテル MKL の DFT 関数 (続き)

関数名	演算
DFT 計算関数	
DftiComputeForward	順方向 DFT を計算する。
DftiComputeBackward	逆方向 DFT を計算する。
ディスクリプター構成関数	
DftiSetValue	特定の 1 つの構成パラメーターを指定された構成値で設定する。
DftiGetValue	特定の 1 つの構成パラメーターの構成値を取得する。
ステータス確認関数	
DftiErrorClass	ステータスが定義済みクラスのエラーを反映しているか確認する。
DftiErrorMessage	エラーメッセージを生成する。

DFT 関数の説明に続いて、構成設定 (「[構成設定](#)」 を参照) および使用されるさまざまな構成パラメーターを説明する。

DFT の計算

本章で後述する DFT 関数は Fortran インターフェイスと C インターフェイスで実装される。Fortran とは Fortran 95 を意味する。DFT インターフェイスは、Fortran 77 にはない



注: Fortran では明示的な関数インターフェイスに続くデータ配列は、どの変換タイプに対しても 1 次元として定義されなければならない。

Fortran 95 のさまざまな新機能に強く依存している。

本章で示される資料では、C のネイティブ複素タイプの有効性は C9X で規定されているものと仮定している。

DFT インターフェイスを使用して変換結果を計算するコードの例については、付録の「[フーリエ変換関数のコード例](#)」のセクションを参照のこと。

一般的な状況では、1 回の DFT 計算は 4 つの関数呼び出しで成立すると考えられる。インテル MKL では、1 つの単一データ構造、すなわちディスクリプターを使用して、パラメーターを個別に変更できるようにした自由度の高い手法を DFT 計算に適用している。これによって機能の拡張性と使いやすさを実現した。

生成された DFTI_DESCRIPTOR のレコードタイプは、計算される DFT の長さや領域の情報だけでなく、構成パラメーターの多くの設定も含んでいる。これらすべてのパラメーターのデフォルト設定には、例えば以下の項目が含まれる。

- 計算される DFT は倍率因子を持っていない
- 変換されるデータは 1 セットのみ
- データはメモリーに連続して格納
- 計算結果は入力データに上書き (インプレース) される、ほか

これらのデフォルト設定が適当でない場合は、[例 C-20](#) と [例 C-21](#) で示すように、[DftiSetValue](#) 関数を用いて 1 つずつ変更できる。

DFT インターフェイス

DFT 関数を使用するには、Fortran では "use" 構文を使用して MKL_DFTI モジュールにアクセスする必要がある。C では "include" 構文を使用してヘッダーファイル `mkl_dfti.h` にアクセスする必要がある。

Fortran インターフェイスでは、派生型 `DFTI_DESCRIPTOR`、さまざまな構成パラメータの名称とそれらの取り得る値を表す名前付き定数、Fortran 95 の汎用機能性を介したさまざまなオーバーロード関数が利用できる。

C インターフェイスでは、構造型 `DFTI_DESCRIPTOR`、マクロ定義

```
#define DFTI_DESCRIPTOR_HANDLE DFTI_DESCRIPTOR *
```

`DFTI_CONFIG_PARAM` と `DFTI_CONFIG_VALUE` の 2 つの列挙型の名前付き定数、関数によっては異なる個数の入力引数を受け付ける種々の関数が与えられる。



注: 現時点のライブラリー実装では、本章の続くセクションに記載されている関数あるいは機能の一部がサポートされていない場合がある。実装に依存して除外される項目については、使用しているライブラリー・バージョンのリリースノートを参照のこと。

インテル MKL では DFT 関数を 4 種類の大きなカテゴリに分けている。

- 1. ディスクリプター操作。** このカテゴリは 4 つの関数で構成される。1 つ目の [DftiCreateDescriptor](#) は DFT ディスクリプターを生成しストレージを動的に割り当てる関数である。この関数はユーザーが与える入力値によってディスクリプターをデフォルト設定に構成する。

2 つ目の [DftiCommitDescriptor](#) はディスクリプターを設定に基づいて「遂行」させる関数である。実際には、すべての必要な事前計算が実行されることを意味する。この事前計算には、入力長さの因子分解と必要なすべての回転因子の計算が含まれる。3 つ目の [DftiCopyDescriptor](#) はディスクリプターのコピーを作成する関数、4 つ目の [DftiFreeDescriptor](#) はディスクリプター情報に割り当てられていたすべてのメモリーを解放する関数である。
- 2. DFT 計算。** このカテゴリは 2 つの関数で構成される。1 つ目の [DftiComputeForward](#) は順方向 DFT 計算を実行する関数で、2 つ目の [DftiComputeBackward](#) は逆方向 DFT 計算を実行する関数である。
- 3. ディスクリプター構成。** このカテゴリは 2 つの関数で構成される。1 つ目の [DftiSetValue](#) 関数は、特定の 1 つの値を多くの構成パラメータのうち変更可能な (わずかにあるか、全くない) パラメータの 1 つに設定する。2 つ目の [DftiGetValue](#) 関数は、これら構成パラメータの任意の 1 つの現在値を取得する (すべてが読み取り可能)。読み取り可能なパラメータ数は多いが、扱えるのは 1 回の関数呼び出しにつき 1 つである。
- 4. ステータス確認。** 上記の 3 種類のカテゴリに記述されている各関数は演算のステータスを表す整数値を返す。戻り値が非ゼロの場合はある種の問題が起きたことを意味する。インテル MKL で

は将来のリリースにおける拡張を想定して、現在の DFT インターフェイスは単一の論理ステータスクラス関数 [DftiErrorClass](#) および単純なステータスメッセージ生成関数 [DftiErrorMessage](#) のみを提供している。

ステータス確認関数

ディスクリプター操作、DFT 計算、ディスクリプター構成のカテゴリに含まれる各関数は、演算のステータスを示す整数値を返す。ここで述べる 2 つの関数は、このステータスを確認する関数である。1 つ目の関数は論理関数で定義済みクラスのエラーをステータスが反映しているか確認し、2 つ目の関数はエラーメッセージ関数で文字列を返す。

ErrorClass

ステータスが定義済みクラスのエラーを反映しているか確認する。

構文

Fortran:

```
Predicate = DftiErrorClass( Status, Error_Class )
```

C:

```
predicate = DftiErrorClass( status, error_class );
```

説明

インテル MKL の DFT インターフェイスには [表 11-2](#) に示す定義済みエラークラスが与えられている。これらは名前付き定数で、Fortran では INTEGER 型、C では long 型である。

表 11-2 定義済みエラークラス

名前付き定数	意味
DFTI_NO_ERROR	エラーなし。
DFTI_MEMORY_ERROR	通常、メモリー割り当てに関係する。
DFTI_INVALID_CONFIGURATION	構成パラメーターの 1 つ以上の設定が無効。
DFTI_INCONSISTENT_CONFIGURATION	構成パラメーターまたは入力パラメーターが不整合。
DFTI_NUMBER_OF_THREADS_ERROR	計算関数の OMP スレッド数は、(遂行関数の) 初期化段階の OMP スレッド数と等しくない。
DFTI_MULTITHREADED_ERROR	通常、OMP ルーチンのエラー戻り値に関係する。
DFTI_BAD_DESCRIPTOR	ディスクリプターを計算に使用できない。
DFTI_UNIMPLEMENTED	正当な設定が未実装、実装依存。
DFTI_MKL_INTERNAL_ERROR	ライブラリー内部エラー。
DFTI_1D_LENGTH_EXCEEDS_INT32	1 つの次元の長さが $2^{32} - 1$ (4 バイト) を超えている。

定義済みエラークラスの確認では、指定したエラークラスに対して `DftiErrorClass` を適用して、その結果ステータス戻り値が `.TRUE.` または `.FALSE.` かを確認することが正しい使用方法である。ステータスと定義済みクラスの直接比較は使用方法としては適切ではない。ステータス確認関数の正しい使用法は[例 C-22](#) を参照。

インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorClass
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_8( Status, Error_Class )
    LOGICAL some_actual_function_8
    INTEGER, INTENT(IN) :: Status, Error_Class
END FUNCTION some_actual_function_8
END INTERFACE DftiErrorClass

/* C prototype */
long DftiErrorClass( long , long );
```

ErrorMessage

エラーメッセージを生成する。

構文

Fortran:

```
ERROR_MESSAGE = DftiErrorMessage( Status )
```

C:

```
error_message = DftiErrorMessage( status );
```

説明

このエラーメッセージ関数はエラーメッセージ文字列を生成する。文字列の最大長は Fortran では名前付き定数 `DFTI_MAX_MESSAGE_LENGTH` で与えられる。実際のエラーメッセージは実装依存である。Fortran の場合、ユーザーは `DFTI_MAX_MESSAGE_LENGTH` を最大長とした文字列を使用する。C ではこの関数は文字列へのポインターを返す。すなわちデリミター '0' を持つ文字配列である。

[例 C-22](#) に、この関数の実装例を示す。

インターフェイスとプロトタイプ

```
//Fortran interface
INTERFACE DftiErrorMessage
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_9( Status, Error_Class )
    CHARACTER(LEN=DFTI_MAX_MESSAGE_LENGTH) some_actual_function_9( Status )
    INTEGER, INTENT(IN) :: Status
END FUNCTION some_actual_function_9
END INTERFACE DftiErrorMessage

/* C prototype */
char *DftiErrorMessage( long );
```

ディスクリプター操作

このカテゴリーは、ディスクリプターの生成、ディスクリプターの遂行、ディスクリプターのコピー、ディスクリプターの解放、の4つの関数で構成される。

CreateDescriptor

ディスクリプター・データ構造にメモリーを割り当て、デフォルト構成設定で具体化する。

構文

Fortran:

```
Status = DftiCreateDescriptor( Desc_Handle, &
    Precision, &
    Forward_Domain, &
    Dimension, &
    INTEGER length
```

C:

```
status = DftiCreateDescriptor( &desc_handle );
    precision
    forward_domain,
    dimension,
    INTEGER length
```

説明

この関数はディスクリプター・データ構造にメモリーを割り当て、変換対象の精度、領域、次元、長さをそれぞれのデフォルト構成設定を使って具体化する。領域は順方向変換での領域として解釈される。メモリーは動的に割り当てられるため、出力結果として生成されたディスクリプターのポインターが得られる。この関数は、DFT を計算する従来型の多くのソフトウェア・パッケージやライブラリーに見られる「初期化」ルーチンとはやや異なっている。ユーザーの要求によって値設定関数 [DftiSetValue](#) を通してデフォルト構成設定を変更できることから、この関数は回転因子計算のような計算量の多い計算は実行しない。

精度と(順方向)領域は DFT インターフェイスで構成値に与えられる名前付き定数によって指定される。精度の選択肢は DFTI_SINGLE か DFTI_DOUBLE である。(順方向)領域の選択肢は DFTI_COMPLEX または DFTI_REAL である。[表 11-5](#) に構成値に対する名前付き定数の全リストを示す。

次元は変換の次元を表す単純な正の整数である。長さは、1 次元変換では単純な正の整数、多次元変換では長さ次元に対応した正の整数を含む整数配列(C ではポインター)である。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

!Fortran interface.

```
INTERFACE DftiCreateDescriptor
```

```
!Note that the body provided here is to illustrate the different
```

```
!argument list and types of dummy arguments. The interface
```

```
!does not guarantee what the actual function names are.
```

```
!Users can only rely on the function name following the keyword INTERFACE
```

```
FUNCTION some_actual_function_1D( Desc_Handle, Prec, Dom, Dim, Length )
```

```
  INTEGER :: some_actual_function_1D
```

```
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
```

```
  INTEGER, INTENT(IN) :: Prec, Dom
```

```
  INTEGER, INTENT(IN) :: Dim, Length
```

```
END FUNCTION some_actual_function_1D
```

```
FUNCTION some_actual_function_HIGHD( Desc_Handle, Prec, Dom, Dim, Length )
```

```
  INTEGER :: some_actual_function_HIGHD
```

```
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
```

```
  INTEGER, INTENT(IN) :: Prec, Dom
```

```
  INTEGER, INTENT(IN) :: Dim, Length(*)
```

```
END FUNCTION some_actual_function_HIGHD
```

```
END INTERFACE DftiCreateDescriptor
```

関数はオーバーロードであるとともに、長さの実際の引数はスカラーまたは階数 1 配列を取り得る点に注意する。

```

/* C prototype */
long DftiCreateDescriptor( DFTI_DESCRIPTOR_HANDLE * );
                        DFTI_CONFIG_PARAM ,
                        DFTI_CONFIG_PARAM ,
                        long
                        ... );

```

可変引数機能は、スカラー (long) または配列 (long *) で表現される長さの引数に対処するために使用される。

CommitDescriptor

実際の DFT 計算を実行可能にするすべての初期化を実行する。

構文

Fortran:

```
Status = DftiCommitDescriptor( Desc_Handle )
```

C:

```
status = DftiCommitDescriptor( desc_handle );
```

説明

DFT インターフェイスでは、ディスクリプターを使って DFT 計算を実行する前に、生成されたディスクリプターを遂行する関数を呼び出す必要がある。通常この遂行処理では、実際の DFT 計算を実行可能にするすべての初期化が実行される。近代的な実装では、高度に効率化された計算方法を検索するために、さまざまな長さの入力を用いた因子分解を行う調査が含まれる場合がある。

遂行されたディスクリプターの構成パラメーターを値設定関数 (「[ディスクリプター構成](#)」を参照) を介して変更した場合は、計算関数を呼び出す前にディスクリプターの再遂行が必要となる。通常は、この遂行関数の呼び出しのすぐ後に計算関数の呼び出しを続ける (「[DFT 計算](#)」を参照)。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```

! Fortran interface
INTERFACE DftiCommitDescriptor
!Note that the body provided here is to illustrate the different
!argument list and types of dummy arguments. The interface
!does not guarantee what the actual function names are.
!Users can only rely on the function name following the
!keyword INTERFACE
    FUNCTION some_actual_function_1 ( Desc_Handle )

```

```

    INTEGER :: some_actual function_1
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    END FUNCTION some_actual function_1
END INTERFACE DftiCommitDescriptor

/* C prototype */
long DftiCommitDescriptor( DFTI_DESCRIPTOR_HANDLE );

```

CopyDescriptor

既存のディスクリプターをコピーする。

構文

Fortran:

```
Status = DftiCopyDescriptor( Desc_Handle_Original,
                             Desc_Handle_Copy )
```

C:

```
status = DftiCopyDescriptor( desc_handle_original,
                             &desc_handle_copy );
```

説明

この関数は既存ディスクリプターのコピーを作成し、それに対するポインターを与える。この関数の目的は、ディスクリプター解放関数 `DftiFreeDescriptor` によって元のディスクリプターが破壊された場合でも、元のディスクリプターの全情報を維持することである。

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiCopyDescriptor
! Note that the body provided here is to illustrate the different
! argument list and types of dummy arguments. The interface
! does not guarantee what the actual function names are.
! Users can only rely on the function name following the
! keyword INTERFACE
    FUNCTION some_actual_function_2( Desc_Handle_Original,
                                     Desc_Handle_Copy )

        INTEGER :: some_actual_function_2
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Original, Desc_Handle_Copy
    END FUNCTION some_actual_function_2

```

```
END INTERFACE DftiCopyDescriptor

/* C prototype */
long DftiCopyDescriptor( DFTI_DESCRIPTOR_HANDLE, DFTI_DESCRIPTOR_HANDLE * );
```

FreeDescriptor

ディスクリプターに割り当てられていたメモリーを解放する。

構文

Fortran:

```
Status = DftiFreeDescriptor( Desc_Handle )
```

C:

```
status = DftiFreeDescriptor( &desc_handle );
```

説明

この関数はディスクリプターに割り当てられているすべてのメモリーを解放する。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiFreeDescriptor
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_3( Desc_Handle )
    INTEGER :: some_actual_function_3
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
END FUNCTION some_actual_function_3
END INTERFACE DftiFreeDescriptor

/* C prototype */
long DftiFreeDescriptor( DFTI_DESCRIPTOR_HANDLE * );
```

DFT 計算

このカテゴリーは、順方向変換の計算、逆方向変換の計算の2つの関数で構成される。

ComputeForward

順方向 DFT を計算する。

構文

Fortran:

```
Status = DftiComputeForward( Desc_Handle, X_inout )
Status = DftiComputeForward( Desc_Handle, X_in, X_out )
```

C:

```
status = DftiComputeForward( desc_handle, x_inout );
status = DftiComputeForward( desc_handle, x_in, x_out );
```

説明

実際の DFT 計算はディスクリプターの生成と遂行を完了した後に実行する。
DftiComputeForward 関数は順方向 DFT を計算する。

これは、因子 $e^{-i2\pi/n}$ を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメーターの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメーターは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言される。

この関数は正常に終了すると DFTI_NO_ERROR を返す。

戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeFoward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
// One argument single precision complex
FUNCTION some_actual_function_4_C( Desc_Handle, X )
  INTEGER :: some_actual_function_4_C
  TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
  COMPLEX, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_4_C
// One argument double precision complex
FUNCTION some_actual_function_4_Z( Desc_Handle, X )
  INTEGER :: some_actual_function_4_Z
```

```

        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
        COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
    END FUNCTION some_actual_function_4_Z
    // One argument single precision real
    FUNCTION some_actual_function_4_R( Desc_Handle, X )
        INTEGER :: some_actual_function_4_R
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
        REAL, INTENT(INOUT) :: X(*)
    END FUNCTION some_actual_function_4_R
    // One argument double precision real
    ...
    // Two argument single precision complex
    ...
    ...
    FUNCTION some_actual_function_4_CC( Desc_Handle, X_In, Y_Out )
        INTEGER :: some_actual_function_4_CC
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
        COMPLEX, INTENT(IN) :: X_In
        COMPLEX, INTENT(OUT) :: Y_Out(*)
    END FUNCTION some_actual_function_4_CC
END INTERFACE DftiComputeFoward

/* C prototype */
long DftiComputeForward( DFTI_DESCRIPTOR_HANDLE );
                        void *,
                        ... );

```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリーに格納されるものとしてデータが扱われることを前提としている（詳細は「[ストライド](#)」、[3]を参照のこと）。関数は最初の成分の開始アドレスを期待する。そのため、Fortran では擬寸法宣言を使用する。

ディスクリプターには、引数の個数と存在すべきタイプを正確に決めるために必要な十分な情報が含まれている。実際の実装では、この情報を使って入力の変数を確認する必要がある。

ComputeBackward

逆方向 DFT を計算する。

構文

Fortran:

```
Status = DftiComputeBackward( Desc_Handle, X_inout )
Status = DftiComputeBackward( Desc_Handle, X_in, X_out )
```

C:

```
status = DftiComputeBackward( desc_handle, x_inout );
status = DftiComputeBackward( desc_handle, x_in, x_out );
```

説明

実際の DFT 計算はディスクリプターの生成と遂行を完了した後に実行する。
DftiComputeBackward 関数は逆方向 DFT を計算する。

これは、因子 $e^{i2\pi/n}$ を使用した変換である。構成の自由度が高いため、入力データを複数の方法で表すことができ、また出力結果を複数の方法で配置できる。そのため、入力パラメーターの個数とタイプは多様である。このような多様性は Fortran 95 の一般関数機能によって実現される。データと結果パラメーターは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言される。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
//Fortran interface.
INTERFACE DftiComputeBackward
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
// One argument single precision complex
FUNCTION some_actual_function_5_C( Desc_Handle, X )
    INTEGER :: some_actual_function_5_C
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    COMPLEX, INTENT(INOUT) :: X(*)
END FUNCTION some_actual_function_5_C
// One argument double precision complex
FUNCTION some_actual_function_5_Z( Desc_Handle, X )
    INTEGER :: some_actual_function_5_Z
    TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
```

```

        COMPLEX (Kind((0D0,0D0))), INTENT(INOUT) :: X(*)
    END FUNCTION some_actual_function_5_Z
    // One argument single precision real
    FUNCTION some_actual_function_5_R( Desc_Handle, X )
        INTEGER :: some_actual_function_5_R
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
        REAL, INTENT(INOUT) :: X(*)
    END FUNCTION some_actual_function_5_R
    // One argument double precision real
    ...
// Two argument single precision complex
    ...
    ...
    FUNCTION some_actual_function_5_CC( Desc_Handle, X_In, Y_Out )
        INTEGER :: some_actual_function_5_CC
        TYPE(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
        COMPLEX, INTENT(IN) :: X_In(*)
        COMPLEX, INTENT(OUT) :: Y_Out(*)
    END FUNCTION some_actual_function_5_CC
END INTERFACE DftiComputeBackward

/* C prototype */
long DftiComputeBackward( DFTI_DESCRIPTOR_HANDLE );
                        void *,
                        ... );

```

DFT インターフェイスはその実装で、規則的な「ストライド」パターンで線形にメモリーに格納されるものとしてデータが扱われることを前提としている（詳細は「[ストライド](#)」、[3]を参照のこと）。関数は最初の成分の開始アドレスを期待する。そのため、Fortran では擬寸法宣言を使用する。

ディスクリプターには、引数の個数と存在すべきタイプを正確に決めるために必要な十分な情報が含まれている。実際の実装では、この情報を使って入力のパラメータを確認する必要がある。

ディスクリプター構成

このカテゴリは、特定の1つの構成パラメーターに適切な値を設定する値設定関数 `DftiSetValue` と、特定の1つの構成パラメーター値を読み取る値取得関数 `DftiGetValue` の2つで構成される。すべての構成パラメーターは読み取り可能であるが、一部はユーザーが設定できない。構成パラメーターには、バージョン番号、動的な情報など、実装に固有な固定情報が含まれているが、固定情報は関数の実行時の実装によって求められる。詳細は「[構成設定](#)」を参照のこと。

SetValue

特定の1つの構成パラメーターを指定された構成値で設定する。

構文

Fortran:

```
Status = DftiSetValue( Desc_Handle, &
                      Config_Param, &
                      Config_Val )
```

C:

```
status = DftiSetValue( desc_handle );
                  config_param,
                  config_val );
```

説明

この関数は特定の1つの構成パラメーターを指定された構成値で設定する。構成パラメーターは表 11-3 に示される名前付き定数のうちの1つであり、構成値は対応する適切なタイプ(名前付き定数またはネイティブタイプ)である。設定の詳細は「[構成設定](#)」を参照のこと。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiSetValue
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_6_INTVAL( Desc_Handle, Config_Param, INTVAL )
    INTEGER :: some_actual_function_6_INTVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(IN) :: INTVAL
END FUNCTION some_actual_function_6_INTVAL

FUNCTION some_actual_function_6_SGLVAL( Desc_Handle, Config_Param, SGLVAL )
    INTEGER :: some_actual_function_6_SGLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
```

```

    REAL, INTENT(IN) :: SGLVAL
END FUNCTION some_actual_function_6_SGLVAL

FUNCTION some_actual_function_6_DBLVAL( Desc_Handle, Config_Param, DBLVAL )
    INTEGER :: some_actual_function_6_DBLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL (KIND(OD0)), INTENT(IN) :: DBLVAL
END FUNCTION some_actual_function_6_DBLVAL

FUNCTION some_actual_function_6_INTVEC( Desc_Handle, Config_Param, INTVEC )
    INTEGER :: some_actual_function_6_INTVEC
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(IN) :: INTVEC(*)
END FUNCTION some_actual_function_6_INTVEC

FUNCTION some_actual_function_6_CHARS( Desc_Handle, Config_Param, CHARS )
    INTEGER :: some_actual_function_6_CHARS
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    CHARACTER(*), INTENT(IN) :: CHARS
END FUNCTION some_actual_function_6_CHARS
END INTERFACE DftiSetValue

/* C prototype */
long DftiSetValue( DFTI_DESCRIPTOR_HANDLE );
                  DFTI_CONFIG_PARAM ,
                  ... );

```

GetValue

特定の1つの構成パラメーターの構成値を取得する。

構文

Fortran:

```

Status = DftiGetValue( Desc_Handle, &
                      Config_Param, &
                      Config_Val )

```

C:

```
status = DftiGetValue( desc_handle );
    config_param,
    &config_val );
```

説明

この関数は特定の1つの構成パラメーターを取得する。構成パラメーターは表 11-3 または表 11-4 に示される名前付き定数の中の1つであり、構成値は対応する適切なタイプ (名前付き定数またはネイティブタイプ) である。

この関数は正常に終了すると DFTI_NO_ERROR を返す。戻り値ステータスの詳細は「[ステータス確認関数](#)」を参照のこと。

インターフェイスとプロトタイプ

```
! Fortran interface
INTERFACE DftiGetValue
//Note that the body provided here is to illustrate the different
//argument list and types of dummy arguments. The interface
//does not guarantee what the actual function names are.
//Users can only rely on the function name following the
//keyword INTERFACE
FUNCTION some_actual_function_7_INTVAL( Desc_Handle, Config_Param, INTVAL )
    INTEGER :: some_actual_function_7_INTVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(OUT) :: INTVAL
END FUNCTION DFTI_GET_VALUE_INTVAL

FUNCTION some_actual_function_7_SGLVAL( Desc_Handle, Config_Param, SGLVAL )
    INTEGER :: some_actual_function_7_SGLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL, INTENT(OUT) :: SGLVAL
END FUNCTION some_actual_function_7_SGLVAL

FUNCTION some_actual_function_7_DBLVAL( Desc_Handle, Config_Param, DBLVAL )
    INTEGER :: some_actual_function_7_DBLVAL
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    REAL (KIND(0D0)), INTENT(OUT) :: DBLVAL
END FUNCTION some_actual_function_7_DBLVAL

FUNCTION some_actual_function_7_INTVEC( Desc_Handle, Config_Param, INTVEC )
    INTEGER :: some_actual_function_7_INTVEC
```

```

    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, INTENT(OUT) :: INTVEC(*)
END FUNCTION some_actual_function_7_INTVEC

FUNCTION some_actual_function_7_INTPNT( Desc_Handle, Config_Param, INTPNT )
    INTEGER :: some_actual_function_7_INTPNT
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    INTEGER, DIMENSION(*), POINTER :: INTPNT
END FUNCTION some_actual_function_7_INTPNT

FUNCTION some_actual_function_7_CHARS( Desc_Handle, Config_Param, CHARS )
    INTEGER :: some_actual_function_7_CHARS
    Type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle
    INTEGER, INTENT(IN) :: Config_Param
    CHARACTER(*), INTENT(OUT):: CHARS
END FUNCTION some_actual_function_7_CHARS
END INTERFACE DftiGetValue

/* C prototype */
long DftiGetValue( DFTI_DESCRIPTOR_HANDLE );
                  DFTI_CONFIG_PARAM ,
                  ... );

```

構成設定

各構成パラメーターは MKL_DFTI モジュール内で名前付き定数によって区別される。C では、これら名前付き定数は列挙型 DFTI_CONFIG_PARAM を持つ。ユーザーによって設定可能な構成パラメーターのリストを [表 11-3](#) に示す。読み取り専用構成パラメーターのリストを [表 11-4](#) に示す。なお、すべてのパラメーターは読み取り可能である。これらのパラメーターのほとんどは自明であるが、残りについては関連した関数に詳細を記載している。

表 11-3 **設定可能な構成パラメーター**

名前付き定数	値のタイプ	意味
<i>最も共通的な構成で、デフォルトはなく、明示的な設定が必要</i>		
DFTI_PRECISION	名前付き定数	計算精度
DFTI_FORWARD_DOMAIN	名前付き定数	順方向変換の領域
DFTI_DIMENSION	整数スカラー	変換の次元
DFTI_LENGTHS	整数スカラー / 配列	各次元の長さ
<i>複数の変換とデータ表現を含む共通的な構成</i>		
DFTI_NUMBER_OF_TRANSFORMS	整数スカラー	変換の回数

表 11-3 設定可能な構成パラメーター (続き)

名前付き定数	値のタイプ	意味
DFTI_FORWARD_SCALE	浮動小数点スカラー	順方向変換の倍率因子
DFTI_BACKWARD_SCALE	浮動小数点スカラー	逆方向変換の倍率因子
DFTI_PLACEMENT	名前付き定数	計算結果の配置
DFTI_COMPLEX_STORAGE	名前付き定数	格納方法、複素数領域データ
DFTI_REAL_STORAGE	名前付き定数	格納方法、実数領域データ
DFTI_CONJUGATE_EVEN_STORAGE	名前付き定数	格納方法、共役偶領域データ
DFTI_DESCRIPTOR_NAME	文字列	DFTI_MAX_NAME_LENGTH より短いこと
DFTI_PACKED_FORMAT	名前付き定数	圧縮形式、実数領域データ
DFTI_NUMBER_OF_USER_THREADS	整数スカラー	DFT 計算で同じディスクリプターを使用するユーザースレッド数
<i>データのストライドに関する構成</i>		
DFTI_INPUT_DISTANCE	整数スカラー	複素変換、最初の成分の距離
DFTI_OUTPUT_DISTANCE	整数スカラー	複素変換、最初の成分の距離
DFTI_INPUT_STRIDES	整数配列	入力データのストライド情報
DFTI_OUTPUT_STRIDES	整数配列	出力データのストライド情報
<i>その他の構成</i>		
DFTI_ORDERING	名前付き定数	データ順のスクランブル
DFTI_TRANSPOSE	名前付き定数	次元のスクランブル

表 11-4 読み取り専用構成パラメーター

名前付き定数	値のタイプ	意味
DFTI_COMMIT_STATUS	名前付き定数	ディスクリプターが遂行されているかを示す
DFTI_VERSION	文字列	インテル MKL のバージョン番号

構成パラメーターにはさまざまな値に設定される。これらの値の一部は、整数値 (例えば、同時変換要求数) や単精度値 (例えば、順方向変換に適用したい倍率因子) のようなネイティブデータ型で規定される。

その他の構成値は性質において個別であり (例えば、順方向変換の領域)、またそのため、DFTI モジュールでは名前付き定数として与えられる。C では、これらの名前付き定数は列挙型 DFTI_CONFIG_VALUE を持つ。この種の構成値で使用される名前付き定数の全リストを [表 11-5](#) に示す。

表 11-5 名前付き定数構成値

名前付き定数	意味
DFTI_SINGLE	単精度
DFTI_DOUBLE	倍精度
DFTI_COMPLEX	複素数領域
DFTI_REAL	実数領域
DFTI_INPLACE	出力は入力を上書き

表 11-5 名前付き定数構成値 (続き)

名前付き定数	意味
DFTI_NOT_INPLACE	出力は入力を上書きしない
DFTI_COMPLEX_COMPLEX	格納方法 (「格納体系」 を参照)
DFTI_REAL_REAL	格納方法 (「格納体系」 を参照)
DFTI_COMPLEX_REAL	格納方法 (「格納体系」 を参照)
DFTI_REAL_COMPLEX	格納方法 (「格納体系」 を参照)
DFTI_COMMITTED	ディスクリプターの遂行ステータス
DFTI_UNCOMMITTED	ディスクリプターの遂行ステータス
DFTI_ORDERED	順方向領域および逆方向領域で順序どおりのデータ
DFTI_BACKWARD_SCRAMBLED	逆方向領域 (順方向変換による) でスクランブルされたデータ
DFTI_NONE	転置なしを指定
DFTI_CCS_FORMAT	圧縮形式、実数データ (「圧縮形式」 を参照)
DFTI_PACK_FORMAT	圧縮形式、実数データ (「圧縮形式」 を参照)
DFTI_PERM_FORMAT	圧縮形式、実数データ (「圧縮形式」 を参照)
DFTI_CCE_FORMAT	圧縮形式、実数データ (「圧縮形式」 を参照)
DFTI_VERSION_LENGTH	ライブラリー・バージョン長の文字数
DFTI_MAX_NAME_LENGTH	ディスクリプター名の最大長
DFTI_MAX_MESSAGE_LENGTH	ステータスメッセージの最大長

性質において個別なこれら構成パラメーターで設定可能な値を [表 11-6](#) に示す。

表 11-6 個別の構成パラメーターの設定

名前付き定数	設定可能な値
DFTI_PRECISION	DFTI_SINGLE、または DFTI_DOUBLE (デフォルト値なし)
DFTI_FORWARD_DOMAIN	DFTI_COMPLEX、または DFTI_REAL
DFTI_PLACEMENT	DFTI_INPLACE (デフォルト) または DFTI_NOT_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX (デフォルト)
DFTI_REAL_STORAGE	DFTI_REAL_REAL (デフォルト)、または DFTI_REAL_COMPLEX
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_COMPLEX、または DFTI_COMPLEX_REAL (デフォルト)
DFTI_PACKED_FORMAT	DFTI_CCS_FORMAT (デフォルト)、または、 DFTI_PACK_FORMAT、または DFTI_PERM_FORMAT、または DFTI_CCE_FORMAT

[表 11-7](#) に設定可能な構成パラメーターのデフォルト値のリストを示す。

表 11-7 設定可能パラメーターのデフォルト構成値

名前付き定数	デフォルト値
DFTI_NUMBER_OF_TRANSFORMS	1
DFTI_NUMBER_OF_USER_THREADS	1
DFTI_FORWARD_SCALE	1.0
DFTI_BACKWARD_SCALE	1.0
DFTI_PLACEMENT	DFTI_INPLACE
DFTI_COMPLEX_STORAGE	DFTI_COMPLEX_COMPLEX
DFTI_REAL_STORAGE	DFTI_REAL_REAL
DFTI_CONJUGATE_EVEN_STORAGE	DFTI_COMPLEX_REAL
DFTI_PACKED_FORMAT	DFTI_CCS_FORMAT
DFTI_DESCRIPTOR_NAME	名前なし、長さゼロの文字列
DFTI_INPUT_DISTANCE	0
DFTI_OUTPUT_DISTANCE	0
DFTI_INPUT_STRIDES	次元、長さ、格納に従って厳密に圧縮されている
DFTI_OUTPUT_STRIDES	同上。詳細は「 ストライド 」を参照のこと。
DFTI_ORDERING	DFTI_ORDERED
DFTI_TRANSPOSE	DFTI_NONE

変換精度

構成パラメーター DFTI_PRECISION は変換の実行精度が浮動小数点であることを示す。設定が DFTI_SINGLE の場合は単精度が有効で、設定が DFTI_DOUBLE の場合は倍精度が有効となる。データがこの精度で示されることを意味する。すなわち計算はこの精度で実行され、結果はこの精度で与えられる。これはデフォルト値を持たない設定可能な4つの構成パラメーターのうちの1つである。ユーザーの明示的な設定が必要で、ディスクリプター生成関数 [DftiCreateDescriptor](#) の呼び出しが最も便利である。

順方向変換

離散フーリエ変換の一般の形は次のとおりである。

$$Z_{k_1, k_2, \dots, k_d} = \sigma \times \sum_{j_d=0}^{n_d-1} \cdots \sum_{j_2=0}^{n_2-1} \sum_{j_1=0}^{n_1-1} w_{j_1, j_2, \dots, j_d} \exp \left(\delta i 2\pi \sum_{l=1}^d j_l k_l / n_l \right)$$

$k_l = 0, \pm 1, \pm 2, \dots$ 、ここで σ は任意実数値の倍率因子、また $\delta = \pm 1$ である。順方向変換は $\sigma = 1$ および $\delta = -1$ として定義される。最も一般的な状況では、順方向変換の領域、すなわち入力(周期的)シーケンス $\{w_{j_1, j_2, \dots, j_d}\}$

が属するセットは、複素数シーケンス、実数シーケンス、複素共役偶シーケンスのいずれ

れかのセットである。構成パラメーター `DFTI_FORWARD_DOMAIN` は順方向変換の領域を示す。この黙示性によって、DFT の数学的な特性から逆方向変換の領域が指定される点に注意する。詳細は表 11-8 を参照のこと。

表 11-8 順方向領域と逆方向領域の対応

順方向領域		黙示的な逆方向領域
複素数	(<code>DFTI_COMPLEX</code>)	複素数
実数	(<code>DFTI_REAL</code>)	共役偶

実領域での変換において、一部のソフトウェア・パッケージは単一の「実数から複素数」変換のみを提供している。これは順方向変換での共役偶領域が本質的に省略されている。順方向領域構成パラメーター `DFTI_FORWARD_DOMAIN` は、デフォルトを持たない 4 つの構成パラメーターのうちの 2 つ目である。

変換の次元と長さ

変換の次元は、Fortran では Integer データ型、C では long データ型の整数スカラーで表される正の整数値である。1 次元変換において、変換の長さは、Fortran では Integer データ型、C では long データ型の整数スカラーで表される正の整数値によって指定される。多次元 (≥ 2) 変換では、各次元の長さは整数配列 (Fortran では Integer データ型、C では long データ型) で与えられる。`DFTI_DIMENSION` と `DFTI_LENGTHS` は、デフォルトを持たない 4 つの構成パラメーターのうちの残りの 2 つである。

これまでに述べたように、これら 4 つのパラメーターにはデフォルト値はない。これらはディスクリプター生成関数で設定するのが最も妥当である。これらの値は、`DftiSetValue` 関数では設定できず、ディスクリプター生成関数でしか設定できない。

変換回数

状況によっては、ユーザーは同じ次元、同じ長さの DFT 変換を複数回実行したい場合がある。最も一般的な状況として、同じ長さを持つ複数の 1 次元データの変換が考えられる。このパラメーターのデフォルト値は 1 で、Fortran では Integer データ型、C では long データ型によって、正の整数として設定する。データセットは共通成分を持たない。回数値が 1 より大きい場合は距離パラメーターを必須とする。

倍率

順方向変換と逆方向変換は、それぞれの倍率因子 σ のデフォルト値が 1 という点で互いに関連している。ユーザーは 2 つの構成パラメーター `DFTI_FORWARD_SCALE` と `DFTI_BACKWARD_SCALE` によって、どちらか、または両方の設定が可能である。例えば、長さ n の 1 次元の変換で、順方向変換の倍率はデフォルトの 1 を使用し、一方、逆方向変換の倍率を $1/n$ に設定すれば、逆方向変換を順方向変換の逆にできる。

倍率因子構成パラメーターは、`DFTI_PRECISION` と同じ精度の実数浮動小数点データ型によって設定しなければならない。

結果の配置

デフォルトでは、計算関数は入力データを出力結果で上書きする。すなわち、構成パラメーター `DFTI_PLACEMENT` のデフォルト設定は `DFTI_INPLACE` である。ユーザーは、これを `DFTI_NOT_INPLACE` に変更可能である。データセットは共通成分を持たない。

圧縮形式

実数の順方向変換 (例えば、周波数領域) の結果は、**Pack**、**Perm**、**CCS**、または **CCE** の圧縮形式によって表現される。実数データの DFT 変換の対称性によってデータは圧縮可能になっている。

CCE 形式は順方向 DFT から得られる出力複素共役偶信号の前半の値を格納する。CCE 形式に格納される 1 次元信号は 1 つの複素数成分だけ長いことに注意する。多次元実数変換 $n_1 * n_2 * n_3 * \dots * n_k$ に対する CCE 形式の複素行列のサイズは、 $(n_1/2+1) * n_2 * n_3 * \dots * n_k$ (Fortran の場合) または $n_1 * n_2 * \dots * (n_k/2+1)$ (C の場合) である。

CCS 形式は CCE 形式に似ている。1 次元変換では CCE と同じ形式で、多次元実数変換ではやや異なる。CCS 形式では、DFT の出力標本は、1 次元 DFT の [表 11-9](#) および 2 次元の [表 11-10](#) に示されるように整列配置される。

Pack 形式は複素共役対称シーケンスをコンパクトに表現したものである。この形式の欠点は、実数 DFT アルゴリズムで使用されている自然な形式とは異なることである (この「自然」とは配列が複素数 DFT に対して自然であることを意味する)。Pack 形式では、DFT の出力標本は、1 次元 DFT の [表 11-9](#) および 2 次元の [表 11-11](#) に示されるように整列配置される。

Perm 形式は、偶数長の Pack 形式に対しては任意の置換であり、奇数長の Pack 形式に対しては同一である。Perm 形式では、DFT の出力標本は、1 次元 DFT の [表 11-9](#) および 2 次元の [表 11-12](#) に示されるように整列配置される。

表 11-9 Pack 形式出力標本

(n = s*2) の場合										
DFT Real	0	1	2	3	...	n-2	n-1	n	n+1	
CCS	R ₀	0	R ₁	I ₁	...	R _{n/2-1}	I _{n/2-1}	R _{n/2}	0	
Pack	R ₀	R ₁	I ₁	R ₂	...	I _{n/2-1}	R _{n/2}			
Perm	R ₀	R _{n/2}	R ₁	I ₁	...	R _{n/2-1}	I _{n/2-1}			

(n = s*2 + 1) の場合											
DFT Real	0	1	2	3	...	n-4	n-3	n-2	n-1	n	n+1
CCS	R ₀	0	R ₁	I ₁	...	I _{s-2}	R _{s-1}	I _{s-1}	R _s	I _s	
Pack	R ₀	R ₁	I ₁	R ₂	...	R _{s-1}	I _{s-1}	R _{s-1}	I _s		
Perm	R ₀	R ₁	I ₁	R ₂	...	R _{s-1}	I _{s-1}	R _{s-1}	I _s		

[表 11-9](#) は複素数データ成分に対して次の表記を使用する点に注意する。

$$R_j = \text{Re } z_j$$

$$I_j = \text{Im } z_j$$

[表 11-13](#) と [表 11-14](#) も参照のこと。

表 11-10 CCS 形式出力標本 (2 次元行列 (m+2) × (n+2))

(m = s*2) の場合								
$z(1,1)$	0	$\text{RE}z(1,2)$	$\text{IM}z(1,2)$...	$\text{RE}z(1,k)$	$\text{IM}z(1,k)$	$z(1,k+1)$	0
0	0	0	0	...	0	0	0	0

表 11-10 CCS 形式出力標本 (2次元行列 (m+2) × (n+2))

(m = s*2) の場合								
z(1,1)	0	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)	z(1,k+1)	0
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)	n/u	n/u
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)	n/u	n/u
...	n/u	n/u
REz(m/2,1)	REz(m/2,2)	REz(m/2,3)	REz(m/2,4)	...	REz(m/2,n-1)	REz(m/2,n)	n/u	n/u
IMz(m/2,1)	IMz(m/2,2)	IMz(m/2,3)	IMz(m/2,4)	...	IMz(m/2,n-1)	IMz(m/2,n)	n/u	n/u
z(m/2+1,1)	0	REz(m/2+1,2)	IMz(m/2+1,2)	...	REz(m/2+1,k)	IMz(m/2+1,k)	z(m/2+1,k+1)	0
0	0	0	0	...	0	0	n/u	n/u
(m = s*2+1) の場合								
z(1,1)	0	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)	z(1,k+1)	0
0	0	0	0	...	0	0	0	0
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)	n/u	n/u
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)	n/u	n/u
...	n/u	n/u
REz(s,1)	REz(s,2)	REz(s,3)	REz(s,4)	...	REz(s,n-1)	REz(s,n)	n/u	n/u
IMz(s,1)	IMz(s,2)	IMz(s,3)	IMz(s,4)	...	IMz(s,n-1)	IMz(s,n)	n/u	n/u

* n/u - 使用しない

表 11-10 の (n+2) 列は偶数 $n = k*2$ に使用され、 n 列は奇数 $n = k*2+1$ に使用される。後者の場合、最初の行は次のとおりである。

$$z(1,1) \quad 0 \quad \text{REz}(1,2) \quad \text{IMz}(1,2) \quad \dots \quad \text{REz}(1,k) \quad \text{IMz}(1,k)$$

m が偶数の場合、(m+1) 番目の行は次のとおりである。

$$z(m/2+1,1) \quad 0 \quad \text{REz}(m/2+1,2) \quad \text{IMz}(m/2+1,2) \quad \dots \quad \text{REz}(m/2+1,k) \quad \text{IMz}(m/2+1,k)$$

表 11-11 Pack 形式出力標本 (2次元行列 (m) × (n))

(m = s*2) の場合						
z(1,1)	REz(1,2)	IMz(1,2)	REz(1,3)	...	IMz(1,k)	z(1,k+1)
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)
...
REz(m/2,1)	REz(m/2,2)	REz(m/2,3)	REz(m/2,4)	...	REz(m/2,n-1)	REz(m/2,n)
IMz(m/2,1)	IMz(m/2,2)	IMz(m/2,3)	IMz(m/2,4)	...	IMz(m/2,n-1)	IMz(m/2,n)
z(m/2+1,1)	REz(m/2+1,2)	IMz(m/2+1,2)	REz(m/2+1,3)	...	IMz(m/2+1,k)	z(m/2+1,k+1)
(m = s*2+1) の場合						
z(1,1)	REz(1,2)	IMz(1,2)	REz(1,3)	...	IMz(1,k)	z(1,n/2+1)
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)
...

表 11-11 Pack 形式出力標本 (2 次元行列 (m) × (n))

(m = s*2) の場合						
z(1,1)	REz(1,2)	IMz(1,2)	REz(1,3)	...	IMz(1,k)	z(1,k+1)
REz(s,1)	REz(s,2)	REz(s,3)	REz(s,4)	...	REz(s,n-1)	REz(s,n)
IMz(s,1)	IMz(s,2)	IMz(s,3)	IMz(s,4)	...	IMz(s,n-1)	IMz(s,n)

表 11-12 Perm 形式出力標本 (2 次元行列 (m) × (n))

(m = s*2) の場合						
z(1,1)	z(1,k+1)	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)
z(m/2+1,1)	z(m/2+1,k+1)	REz(m/2+1,2)	IMz(m/2+1,2)	...	REz(m/2+1,k)	IMz(m/2+1,k)
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)
...
REz(m/2,1)	REz(m/2,2)	REz(m/2,3)	REz(m/2,4)	...	REz(m/2,n-1)	REz(m/2,n)
IMz(m/2,1)	IMz(m/2,2)	IMz(m/2,3)	IMz(m/2,4)	...	IMz(m/2,n-1)	IMz(m/2,n)
(m = s*2+1) の場合						
z(1,1)	z(1,k+1)	REz(1,2)	IMz(1,2)	...	REz(1,k)	IMz(1,k)
REz(2,1)	REz(2,2)	REz(2,3)	REz(2,4)	...	REz(2,n-1)	REz(2,n)
IMz(2,1)	IMz(2,2)	IMz(2,3)	IMz(2,4)	...	IMz(2,n-1)	IMz(2,n)
...
REz(s,1)	REz(s,2)	REz(s,3)	REz(s,4)	...	REz(s,n-1)	REz(s,n)
IMz(s,1)	IMz(s,2)	IMz(s,3)	IMz(s,4)	...	IMz(s,n-1)	IMz(s,n)

表 11-11 および 表 11-12 では、列数が偶数の場合は $n = k*2$ 、奇数の場合は $n = k*2+1$ で、最初の行は次のとおりである。

z(1,1) REz(1,2) IMz(1,2) ... REz(1,k) IMz(1,k)

m が偶数の場合、Pack 形式の最後の行と Perm 形式の 2 番目の行は次のとおりである。
z(m/2+1,1) REz(m/2+1,2) IMz(m/2+1,2) ... REz(m/2+1,k) IMz(m/2+1,k)

2 次元 DFT の表は Fortran インターフェイス規則を使用する。圧縮データの格納における C インターフェイスの詳細は、次の「格納体系」のセクションを参照のこと。
また、Fortran インターフェイス形式と C インターフェイス形式の例は、表 11-15 と表 11-16 を参照のこと。

格納体系

DFTI_COMPLEX、DFTI_REAL、DFTI_CONJUGATE_EVEN (これは順方向の場合だが逆方向演算も同様) の 3 種類の領域の計算に対して、4 つの格納体系のサブセット DFTI_COMPLEX_COMPLEX、DFTI_COMPLEX_REAL、DFTI_REAL_COMPLEX、および DFTI_REAL_REAL が提供される。ここでは格納体系を説明するための例を示す。この格納体系の定義の背景にある理論的根拠は文献 [3] を参照のこと。



注: データは Fortran 形式でのみ格納される。すなわち、実数部と虚数部は隣接して格納される。

複素数領域の格納体系：この設定は構成パラメーター `DFTI_COMPLEX_STORAGE` に登録される。設定可能な3種類の値は、`DFTI_COMPLEX_COMPLEX`、`DFTI_COMPLEX_REAL`、または `DFTI_REAL_REAL` である。1次元、長さ n の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n}, \quad w_j, z_k \in \mathbb{C}.$$

ストライドはデフォルト値(単位ストライド)を持ち、`DFTI_PLACEMENT` 設定はデフォルトのインプレース(入力上書き)であると仮定する。

DFTI_COMPLEX_COMPLEX 格納体系(デフォルト)。典型的な使用法は次のとおりである。

```
COMPLEX :: X(0:n-1)
...some other code...
Status = DftiComputeForward( Desc_Handle, X )
```

入力では、

$$X(j) = w_j, j = 0, 1, \dots, n-1$$

出力において、

$$X(k) = z_k, k = 0, 1, \dots, n-1$$

実数領域と共役偶領域の格納体系：これらの領域に対する格納体系の設定は構成パラメーター `DFTI_REAL_STORAGE` と `DFTI_CONJUGATE_EVEN_STORAGE` に登録される。順方向の実数領域は逆方向の共役偶領域に一致することから、これらは一緒に扱われる。[1次元](#)、[2次元](#)、および [3次元](#) 実数を共役偶に変換する例を使用した。可能であればインプレース計算を仮定している(すなわち、入力データ型が出力データ型に一致する場合)。

1次元変換

1次元、長さ n の形式の変換を考える。

$$z_k = \sum_{j=0}^{n-1} w_j e^{-i2\pi jk/n}, \quad w_j \in \mathbb{R}, z_k \in \mathbb{C}.$$

これは対称である。

n が偶数の場合： $z(n/2+i) = \text{conjg}(z(n/2-i))$ 、 $1 \leq i \leq n/2-1$ 、またさらに、 $z(0)$ と $z(n/2)$ は実数値。

n が奇数の場合： $z(m+i) = \text{conjg}(z(m-i+1))$ 、 $1 \leq i \leq m$ 、またさらに、 $z(0)$ は実数値。

$$m = \text{floor}(n/2)$$

表 11-13 順方向変換での複素数 - 複素数 DFT および実数 - 複素数 DFT の格納効果の比較

N=8								
入力ベクトル			出力ベクトル					
複素数 DFT		実数 DFT	複素数 DFT		実数 DFT			
複素数データ		実数データ	複素数データ		実数データ			
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)	
w4	0.000000	w4	z4	0.000000	Re(z2)	Im(z2)	Re(z2)	
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)	
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)	
w7	0.000000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)	
					z4			
					0.000000			

N=7								
入力ベクトル			出力ベクトル					
複素数 DFT		実数 DFT	複素数 DFT		実数 DFT			
複素数データ		実数データ	複素数データ		実数データ			
実数部	虚数部		実数部	虚数部	CCS	Pack	Perm	
w0	0.000000	w0	z0	0.000000	z0	z0	z0	
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)	
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)	
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)	
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)	
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)	
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)	
					Im(z3)			

表 11-14 逆方向変換での複素数 - 複素数 DFT および複素数 - 実数 DFT の格納効果の比較

出力ベクトル			入力ベクトル				
複素数 DFT			複素数 DFT				
複素数データ			複素数データ				
実数部	虚数部	実数 DFT 実数データ	実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	z4
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Re(z1)
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Im(z1)
w4	0.000000	w4	z4		Re(z2)	Im(z2)	Re(z2)
w5	0.000000	w5	Re(z3)	-Im(z3)	Im(z2)	Re(z3)	Im(z2)
w6	0.000000	w6	Re(z2)	-Im(z2)	Re(z3)	Im(z3)	Re(z3)
w7	0.000000	w7	Re(z1)	-Im(z1)	Im(z3)	z4	Im(z3)
					z4		
					0.000000		

出力ベクトル			入力ベクトル				
複素数 DFT			複素数 DFT		実数 DFT		
複素数データ			複素数データ		実数データ		
実数部	虚数部	実数 DFT 実数データ	実数部	虚数部	CCS	Pack	Perm
w0	0.000000	w0	z0	0.000000	z0	z0	z0
w1	0.000000	w1	Re(z1)	Im(z1)	0.000000	Re(z1)	Re(z1)
w2	0.000000	w2	Re(z2)	Im(z2)	Re(z1)	Im(z1)	Im(z1)
w3	0.000000	w3	Re(z3)	Im(z3)	Im(z1)	Re(z2)	Re(z2)
w4	0.000000	w4	Re(z3)	-Im(z3)	Re(z2)	Im(z2)	Im(z2)
w5	0.000000	w5	Re(z2)	-Im(z2)	Im(z2)	Re(z3)	Re(z3)
w6	0.000000	w6	Re(z1)	-Im(z1)	Re(z3)	Im(z3)	Im(z3)
					Im(z3)		

ストライドはデフォルト値 (単位ストライド) であると仮定する。

この複素共役対称ベクトルは、圧縮形式に依存して、大きさ $m+1$ の複素数配列、または大きさ $2m+2$ か $2m$ の実数配列に格納できる。

2 次元変換

実数から複素数へのルーチンは、いずれも次の式に従って、2次元実数行列の順方向の DFT を計算する。

$$z_{i,j} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} t_{k,l} * w_m^{-i*k} * w_n^{-j*l}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1$$

$t_{k,l} = \text{cmplx}(r_{k,l}, 0)$ 、ここで $r_{k,l}$ は実数入力行列で、 $0 \leq k \leq m-1, 0 \leq l \leq n-1$ である。数値演算の結果 $z_{i,j}, 0 \leq i \leq m-1, 0 \leq j \leq n-1$ は、サイズ (m,n) の複素行列である。各列は、次のような複素共役対称ベクトルになる。

m が偶数の場合：

$0 \leq j \leq n-1$ の場合は、

$$z(m/2+i, j) = \text{conjg}(z(m/2-i, j)), \quad 1 \leq i \leq m/2-1$$

また、 $z(0, j)$ と $z(m/2, j)$ の場合は、 $j = 0$ と $j = n/2$ は実数値になる。

m が奇数の場合：

$0 \leq j \leq n-1$ の場合は、

$$z(s+i, j) = \text{conjg}(z(s-i, j)), \quad 1 \leq i \leq s-1$$

ここで $s = \text{floor}(m/2)$

また、 $z(0, j)$ の場合は $j = 0$ と $j = n/2$ は実数値になる。

この数値演算の結果は、実数 2 次元配列に次のように格納される。

$(m+2, n+2)$ (CCS 形式の場合)、

(m, n) (Pack または Perm 形式の場合)、

$(2*(m/1+1), n)$ (CCE 形式、Fortran インターフェイスの場合) または

$((m, 2*(n/2+1))$ (CCE 形式、C インターフェイスの場合)

または、複素数 2 次元配列に次のように格納される。

$(m/2+1, n)$ (CCE 形式、Fortran インターフェイスの場合) または

$(m, n/2+1)$ (CCE 形式、C インターフェイスの場合)

多次元配列データの配置は Fortran と C では異なるため (「[ストライド](#)」を参照)、計算結果を格納する出力配列は複素共役対称列 (Fortran の場合) または複素共役対称行 (C の場合) を含む。

次の表は、 6×4 の実数行列に対して実数から複素数への順方向 2 次元 DFT を行った場合の、Pack 形式の出力データのレイアウト例を示す。同じレイアウトが、対応する複素数から実数への逆方向 DFT の入力データに使用される点に注意する。

表 11-15 **6×4 の行列の Fortran インターフェイス・データ・レイアウト**

$z(1,1)$	Re $z(1,2)$	Im $z(1,2)$	$z(1,3)$
Re $z(2,1)$	Re $z(2,2)$	Re $z(2,3)$	Re $z(2,4)$
Im $z(2,1)$	Im $z(2,2)$	Im $z(2,3)$	Im $z(2,4)$
Re $z(3,1)$	Re $z(3,2)$	Re $z(3,3)$	Re $z(3,4)$
Im $z(3,1)$	Im $z(3,2)$	Im $z(3,3)$	Im $z(3,4)$
$z(4,1)$	Re $z(4,2)$	Im $z(4,2)$	$z(4,3)$

上の例で、ストライド配列は (0, 1, 6) をとる。

表 11-16 **6 × 4 の行列の C インターフェイス・データ・レイアウト**

z(1,1)	Re z(1,2)	Im z(1,2)	z(1,3)
Re z(2,1)	Re z(2,2)	Im z(2,2)	Re z(2,3)
Im z(2,1)	Re z(3,2)	Im z(3,2)	Im z(2,3)
Re z(3,1)	Re z(4,2)	Im z(4,2)	Re z(3,3)
Im z(3,1)	Re z(5,2)	Im z(5,2)	Im z(3,3)
z(4,1)	Re z(6,2)	Im z(6,2)	z(4,3)

2 つ目の例で、ストライド配列は (0, 4, 1) をとる。

「[圧縮形式](#)」も参照のこと。

3 次元変換

実数から複素数へのルーチンは、いずれも次の式に従って、3 次元実数行列の順方向の DFT を計算する。

$$z_{i,j,q} = \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} \sum_{s=0}^{k-1} t_{p,l,s} * w_m^{-i*p} * w_n^{-j*l} * w_k^{-q*s}, \quad 0 \leq i \leq m-1, \quad 0 \leq j \leq n-1,$$

$$0 \leq s \leq k-1$$

$t_{p,l,s} = \text{cplx}(r_{p,l,s}, 0)$ 、ここで $r_{p,l,s}$ は実数入力行列で、 $0 \leq k \leq m-1$ 、 $0 \leq l \leq n-1$ 、 $0 \leq s \leq k-1$ である。数値演算の結果 $z_{i,j,q}$ 、 $0 \leq i \leq m-1$ 、 $0 \leq j \leq n-1$ 、 $0 \leq q \leq k-1$ は、次のようなサイズ (m, n, k) の複素共役対称行列または複素共役偶行列である。

$z_{m1,n1,k1} = \text{conjg}(z_{m-m1,n-n1,k-k1})$ では、各次元は周期的である。

この数値演算の結果は、実数 3 次元配列に次のように格納される。

$(m/2+1, n, k)$ (CCE 形式、Fortran インターフェイスの場合) または

$(m, n, k/2+1)$ (CCE 形式、C インターフェイスの場合)

多次元配列データの配置は Fortran と C では異なるため (「[ストライド](#)」を参照)、計算結果を格納する出力配列は複素共役対称列 (Fortran の場合) または複素共役対称行 (C の場合) を含む。



注: 3D REAL DFT - CCE 形式の圧縮形式は 1 つである。インプレースおよびアウトオブプレース REAL DFT では、ストライド・パラメーターと距離パラメーターは REAL 単位 (実数データの場合) または COMPLEX 単位 (複素数データの場合) である。このため、入力/出力データの成分は、インプレース FFT の入力/出力配列の異なる成分に配置することができる。

1. 実数領域では **DFTI_REAL_REAL**、共役偶領域では **DFTI_COMPLEX_REAL** (デフォルト)。1 次元および 2 次元 REAL DFT に使用される。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:2*m+1)
...some other code...
...assuming inplace...
Status = DftiComputeForward( Desc_Handle, X )
```

入力では、

$$X(j) = w^j, j = 0, 1, \dots, n-1$$

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「[圧縮形式](#)」を参照)。

CCS 形式: $x(2*k) = \text{Re}(z_k)$, $x(2*k+1) = \text{Im}(z_k)$, $k = 0, 1, \dots, m$

Pack 形式: 偶数 n : $x(0) = \text{Re}(z_0)$, $x(2*k-1) = \text{Re}(z_k)$, $x(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m-1$, and $x(n-1) = \text{Re}(z_m)$

奇数 n : $x(0) = \text{Re}(z_0)$, $x(2*k-1) = \text{Re}(z_k)$, $x(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m$

Perm 形式: 偶数 n : $x(0) = \text{Re}(z_0)$, $x(1) = \text{Re}(z_m)$, $x(2*k) = \text{Re}(z_k)$, $x(2*k+1) = \text{Im}(z_k)$, $k = 1, \dots, m-1$

奇数 n : $x(0) = \text{Re}(z_0)$, $x(2*k-1) = \text{Re}(z_k)$, $x(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m$

2. 実数領域では **DFTI_REAL_REAL**、共役偶領域では **DFTI_COMPLEX_REAL** (デフォルト)。1次元および2次元 REAL DFT に使用される。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
REAL :: X(0:n-1)
REAL :: Y(0:2*m+1)
...some other code...
...assuming out-of-place...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力では、

$$X(j) = w^j, j = 0, 1, \dots, n-1$$

出力において、

出力データは、Pack、Perm、または CCS のどれか 1 つの形式で格納される (「[圧縮形式](#)」を参照)。

CCS 形式: $Y(2*k) = \text{Re}(z_k)$, $Y(2*k+1) = \text{Im}(z_k)$, $k = 0, 1, \dots, m$

Pack 形式: 偶数 n : $Y(0) = \text{Re}(z_0)$, $Y(2*k-1) = \text{Re}(z_k)$, $Y(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m-1$, and $Y(n-1) = \text{Re}(z_m)$

奇数 n : $Y(0) = \text{Re}(z_0)$, $Y(2*k-1) = \text{Re}(z_k)$, $Y(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m$

Perm 形式: 偶数 n : $Y(0) = \text{Re}(z_0)$, $Y(1) = \text{Re}(z_m)$, $Y(2*k) = \text{Re}(z_k)$, $Y(2*k+1) = \text{Im}(z_k)$, $k = 1, \dots, m-1$,

奇数 n : $Y(0) = \text{Re}(z_0)$, $Y(2*k-1) = \text{Re}(z_k)$, $Y(2*k) = \text{Im}(z_k)$, $k = 1, \dots, m$

出力配列のストライドがデフォルト値 (単位ストライド) に設定されていない場合、1つの複素数成分の実数部と虚数部はこのストライドで配置される。

例えば、

CCS 形式 : $Y(2*k*s) = \text{Re}(z_k)$, $Y((2*k+1)*s) = \text{Im}(z_k)$, $k = 0, 1, \dots, m$, $s = \text{stride}$.

3. 実数領域では **DFTI_REAL_REAL**、共役偶領域では **DFTI_COMPLEX_COMPLEX**。1次元、2次元、および3次元 REAL DFT に使用される。CCE 形式はデフォルトで設定される。典型的な使用法は次のとおりである。

- アウトオブプレース変換 :

```
// m = floor( n/2 )

REAL :: X(0:n-1)
COMPLEX :: Y(0:m)
...some other code...
Status = DftiComputeForward( Desc_Handle, X, Y )
```

入力では、

$X(j) = w_j$, $j = 0, 1, \dots, n-1$.

出力において、

$Y(k) = z_k$, $k = 0, 1, \dots, m$.

- インプレース変換 :

```
// m = floor( n/2 )

REAL :: X(0:m*2)
...some other code...
Status = DftiComputeForward( Desc_Handle, X )
```

入力では、

$X(j) = w_j$, $j = 0, 1, \dots, n-1$.

出力において、

$X(2*k) = \text{Re}(z_k)$, $X(2*k+1) = \text{Im}(z_k)$, $k = 0, 1, \dots, m$.

4. 実数領域では **DFTI_REAL_COMPLEX**、共役偶領域では **DFTI_COMPLEX_COMPLEX**。現在のバージョンでは使用しない。11-3 ページを参照のこと。典型的な使用法は次のとおりである。

```
// m = floor( n/2 )
COMPLEX :: X(0:m)
...some other code...
```

```
...inplace transform...
Status = DftiComputeForward( Desc_Handle, X )
```

入力では、

$$x(j) = wj, j = 0, 1, \dots, n-1$$

すなわち、 $x(j)$ の虚数部はゼロである。

出力において、

$$Y(k) = zk, k = 0, 1, \dots, m.$$

ここで、 m は $\text{floor}(n/2)$ である。

ユーザースレッド数

次に示す技法によりカスタム・アプリケーションを並列化することができる。

1. アプリケーションでスレッドを作成しないが、インテル MKL の DFT モジュール内で並列モードを指定する。詳細は「インテル MKL テクニカル・ユーザー・ノート」を参照のこと。
2. アプリケーションでスレッドを作成して、ディスクリプターの初期化、DFT 計算、ディスクリプターの割り当て解除を含む、DFT 実装のすべての段階を各スレッドに実行させる。この場合、それぞれのディスクリプターは対応するスレッド内でのみ使用される。
3. DFT ディスクリプターを初期化した後でスレッドを作成する。これは、スレッディングが並列 DFT 計算でのみ使用され、並列領域から戻った後でディスクリプターが解放されることを意味する。この場合、各スレッドは同じディスクリプターを使用する。

上記の 1 と 2 の技法では、特定のディスクリプターはそれぞれシングルスレッドでのみ使用されるため、パラメーター `DFTI_NUMBER_OF_USER_THREADS` を 1 (デフォルト値) に設定する。

3 の技法では、複数のスレッドが同じディスクリプターを使用するため、`DftiSetValue()` 関数を使用して `DFTI_NUMBER_OF_USER_THREADS` を実際の DFT 計算スレッド数に設定しなければならない。ディスクリプターには各スレッドの個々のデータが含まれるため、この設定を行わないとプログラムが正しく実行されない。



警告：

1. プログラムの並列化とインテル MKL の内部スレッディングの使用を同時に行うことは、性能に悪影響を与えるため推奨されない。上記の 3 の技法では、DFT 計算は自動的にシングル・スレッディング・モードで開始される点に注意する。
2. `DftiCommitDescriptor()` 関数によって DFT の初期化が実行された後に、スレッド数を変更してはならない。

付録 C の [例 C-24](#)、[例 C-25](#)、および [例 C-26](#) を参照のこと。

入力と出力との距離

インテル MKL の DFT インターフェイスは複数回の変換計算を許可している。そのため、複数セットのデータのデータ分布をユーザーが指定できる必要がある。これは、連続するデータセットの最初のデータ成分同士の距離で実現される。このパラメーターは変換回数が 1 より大きい場合は必須である。パラメーターは、Fortran では Integer データ型、C では long データ型の値である。データセットは何ら共通成分は持っていない。次の例で仕様を説明する。x(0:31, 1)、x(0:31, 2)、x(0:31, 3) に格納された 3 つの長さ 32 の複素数シーケンスの順方向 DFT を計算する場合を考える。結果は配列 Y(0:63, 3) の位置 Y(0:31, k)、k=1, 2, 3 に格納されるとする。入力距離は 32 となり、出力距離は 64 となる。計算関数のデータと結果パラメーターは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。以下にコードの一部を示す。

```
Complex :: X_2D(0:31,3), Y_2D(0:63, 3)
Complex :: X(96), Y(192)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
.....
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 32)
Status = DftiSetValue(Desc_Handle, DFTI_NUMBER_OF_TRANSFORMS, 3)
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_DISTANCE, 32)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_DISTANCE, 64)
Status = DftiSetValue(Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X, Y)
Status = DftiFreeDescriptor(Desc_Handle)
```

ストライド

DFT インターフェイスは、データセットの複数回変換のサポートに加えて、各データセット内のデータに対して単位ストライド以外のストライド分布をサポートしている。パラメーターは、Fortran では Integer データ型、C では long データ型の値の配列である。シーケンス x_j 、 $0 \leq j < 32$ において長さ 32 の DFT が計算される次のような状況を考える。これら値の実際の位置は、配列 x(1:68) の x(5)、x(7)、...、x(67) である。DFT インターフェイスによって実現されるストライドは、データ配列の最初の成分からの変位 $L0$ (この場合 4) と、続く成分の一定の距離 $L1$ (この場合 2) で構成される。よって Fortran 配列 x は次のようになる。

$$x_j = x(1 + L0 + L1 * j) = x(5 + L1 * j)$$

ストライドベクトル (4,2) は長さ 2、階数 1 の整数配列によって与えられる。

```
COMPLEX :: X(68)
INTEGER :: Stride(2)
.....
Status = DftiCreateDescriptor(Desc_Handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 32)
```

```
Stride = (/ 4, 2 /)
Status = DftiSetValue(Desc_Handle, DFTI_INPUT_STRIDES, Stride)
Status = DftiSetValue(Desc_Handle, DFTI_OUTPUT_STRIDES, Stride)
Status = DftiCommitDescriptor(Desc_Handle)
Status = DftiComputeForward(Desc_Handle, X)
Status = DftiFreeDescriptor(Desc_Handle)
```

一般に d 次元変換ではストライドは長さ $d+1$ の整数ベクトル $(L_0, L_1, L_2, \dots, L_d)$ で与えられ、それぞれの意味は次のとおりである。

- L_0 = 最初の配列成分からの変位
 - L_1 = 最初の次元の連続するデータ成分間の距離
 - L_2 = 2 番目の次元の連続するデータ成分間の距離
 - ... = ...
 - L_d = d 番目の次元の連続するデータ成分間の距離
- d 次元のデータシーケンス、

$$x_{j_1, j_2, \dots, j_d}, \quad 0 \leq j_i < J_i, \quad 1 \leq i \leq d$$

は、階数 1 の配列 x に以下のマッピングによって格納される。

$$x_{j_1, j_2, \dots, j_d} = x(\text{first index} + L_0 + j_1 L_1 + j_2 L_2 + \dots + j_d L_d)$$

複数回の変換では、値 L_0 は最初のデータシーケンスに適用され、 $L_j, j=1, 2, \dots, d$ はすべてのデータシーケンスに適用される。

単一の 1 次元シーケンスの場合、 L_1 は単純に通常のストライドとなる。一般的な多次元の状態でのストライドのデフォルト設定は、シーケンスが配列内に厳密に分布している場合に相当する。

$$L_1 = 1, L_2 = J_1, L_3 = J_1 J_2, \dots, L_d = \prod_{i=1}^{d-1} J_i$$

入力データと出力データのそれぞれは関連するストライドを有する。デフォルトは、データが言語に自然な方法でメモリーに隣接して格納されるように設定されている。上記の構成パラメーターの使用例については[例 C-23](#)を参照のこと。

順序指定

数々の FFT アルゴリズムには、処理時間を要する明示的な置換段階が適用されていることはよく知られている [4]。ただし、スクランブルされた入力データに DFT を適用するか、あるいはスクランブルした DFT 結果出力を許せば、上記段階を回避できる。畳み込みとパワー・スペクトラム計算のようなアプリケーションでは、結果またはデータの順序は重要ではないため、性能向上が得られるのであればスクランブル順の許可は魅力的である。インテル MKL では次の 3 つのオプションが利用可能である。

1. DFTI_ORDERED: 順方向変換データは順序どおり、逆方向変換データは順序どおり (デフォルトオプション)。
2. DFTI_BACKWARD_SCRAMBLED: 順方向変換データは順序どおり、逆方向変換データはスクランブル。

表 11-17 に、この構成設定の効果を示す。

表 11-17 スクランブル順の変換

	DftiComputeForward	DftiComputeBackward
DFTI_ORDERING	入力 → 出力	入力 → 出力
DFTI_ORDERED	順序どおり → 順序どおり	順序どおり → 順序どおり
DFTI_BACKWARD_SCRAMBLED	順序どおり → スクランブル	スクランブル → 順序どおり

後半の 2 つのオプションは「可能ならスクランブル順を許す」意味である点に注意する。データのスクランブルを許可しても性能向上が得られない状況もあり、そのため実装によっては示唆を無視する場合がある。厳密には、些細であるが、普通の順序もスクランブルされた順序の 1 つである。

転置

高次変換の結果を転置の形式で表現可能にするオプションである。デフォルト設定は DFTI_NONE だが、DFTI_ALLOW に設定可能である。スクランブル順と同様に、高次変換では、結果を転置形式で与えるようにすると性能が得られる場合がある。DFT インターフェイスは、性能向上が可能な場合に対応して、転置形式で結果を出力するオプションを備えている。一般的なストライド仕様は転置表現に自然に適合することから、このオプションを利用すると、出力のストライドがユーザーによって元々指定されたストライドと異なることがある。2 次元結果

$$Y_{j_1, j_2}, 0 \leq j_i < n_i,$$

が見込まれる例を考えてみる。はじめにユーザーは、汎用ストライド $L_1=1$ および $L_2=n_1$ を用いて結果を (フラットな) 配列 y に割り当てるように指定した。転置オプションを使用すると実際の計算結果はストライド $L_1=n_2$ と $L_2=1$ で y に返される場合がある。これらのストライドは対応する問い合わせ関数で取得できる。3 次元以上では、転置は次元の任意な置換を意味する点に注意する。

クラスター DFT 関数

このセクションでは、インテル MKL に実装されているクラスター離散フーリエ変換 (DFT) 関数 (インテル MKL Linux* 版および Windows* 版で利用可能) について説明する。

インテル MKL 9.0 以降では、2 つのバージョンのクラスター DFT インターフェイスが提供されている。

- インテル MKL 9.0 よりも古いバージョンで実装されたインターフェイスと互換性がない新しいバージョン。
- 下位互換性があるインテル MKL 8.1 から導入されたインターフェイス。

このセクションでは、新しいバージョンのクラスター DFT インターフェイスについて説明する。古いバージョンについてのドキュメントは、次の Web サイトを参照のこと。
<http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/219843.htm>

クラスター DFT 関数ライブラリーは、クラスター (すなわち、ネットワークにより相互に連結されたコンピューターのグループ) 上で離散フーリエ変換を実行するための関数である。クラスター上の各コンピューター (ノード) は個別のメモリーとプロセッサーを持ち、ノード間のデータ交換はネットワークにより提供される。

各クラスターノードで、並行して 1 つ以上のプロセスを実行することができる。クラスター DFT 関数ライブラリーは、MPI (Message Passing Interface) を使用して、クラスター上のノード間の通信を行う。利用可能な MPI 実装 (例: MPICH、インテル[®] MPI ほか) の数が与えられると、特定の MPI 実装に依存しないように、クラスター DFT は BLACS と呼ばれる線形代数用のメッセージ・パッシング・ライブラリーを介して、MPI と動作する。

インテル MKL のクラスター離散フーリエ変換関数ライブラリーは、1 次元、2 次元、多次元 (最大 7 次元) のルーチン、およびすべての変換関数で、Fortran インターフェイスと C インターフェイスの両方を提供する。

クラスター DFT を使用してアプリケーションを開発するには、MPI プログラミングの基礎知識とスキルが必要である。

インテル MKL クラスター・エディションの DFT 関数のインターフェイスは、この章ですでに説明した通常の MKL の [DFT 関数](#) に対応するインターフェイスとよく似ている。詳細は該当するセクションを参照のこと。

インテル MKL に実装されているすべてのクラスター DFT 関数を [表 11-18](#) に示す。

表 11-18 インテル MKL のクラスター DFT 関数

関数名	演算
ディスクリプター操作関数	
DftiCreateDescriptorDM	ディスクリプター・データ構造にメモリーを割り当て、デフォルト構成設定で具体化する。
DftiCommitDescriptorDM	実際の DFT 計算を実行可能にするすべての初期化を実行する。
DftiFreeDescriptorDM	ディスクリプターに割り当てられていたメモリーを解放する。
DFT 計算関数	
DftiComputeForwardDM	順方向 DFT を計算する。
DftiComputeBackwardDM	逆方向 DFT を計算する。
ディスクリプター構成関数	
DftiSetValueDM	特定の 1 つの構成パラメーターを指定された構成値で設定する。
DftiGetValueDM	特定の 1 つの構成パラメーターの構成値を取得する。

クラスター DFT の計算

このセクションで後述するクラスター DFT 関数は Fortran インターフェイスと C インターフェイスで実装される。Fortran とは Fortran 95 を意味する。

MPI を使用するプログラム (MPI プログラム) で呼び出される [DftiComputeForwardDM](#) 関数および [DftiComputeBackwardDM](#) 関数によってクラスター DFT 計算が実行される。MPI プログラムを開始後、プロセスが作成される。MPI は、階数によって各プロセスを識別する。プロセスは互いに独立しており、MPI による通信を行う。MPI プログラムで呼び出される関数は、すべてのプロセスで起動される。各プロセスはその階数により処理を決定する。クラスター DFT 変換の入力/出力データは、複素数値のシーケンスである。クラスター DFT 計算関数は、入力データのローカル部分 (特定のプロセスで演算が行われるデータの一部) の演算および出力データのローカル部分の生成を行う。各プロセスは計算の一部を行う。これらのプロセスは、並列で実行し、MPI 通信により、DFT 計算全体を行う。

インテル MKL クラスター DFT 関数を使用する DFT 計算は、通常、以下に示す手順によって成立する。

1. `MPI_Init` (C/C++ の場合) または `MPI_INIT` (Fortran の場合) を呼び出して MPI を初期化する (DFT 関数および MPI 関数を呼び出す前に行わなければならない)。
2. [DftiCreateDescriptorDM](#) を呼び出して、ディスクリプターにメモリーを割り当てる。
3. [DftiSetValueDM](#) への呼び出しにより、構成パラメーターの値を指定する。
4. [DftiGetValueDM](#) への呼び出しにより、ローカルデータ配列を作成するのに必要な構成パラメーターの値を取得する。
5. [DftiCommitDescriptorDM](#) への呼び出しにより、DFT 計算を実行可能にする初期化を実行する。

6. 入力/出力データのローカル部分の配列を作成し、入力データのローカル部分に値を設定する (詳細は、「[プロセス間でのデータの分配](#)」を参照)。
7. [DftiComputeForwardDM](#) または [DftiComputeBackwardDM](#) を呼び出して変換を計算する。
8. MPI 関数を使用してローカル出力データをグローバル配列に収集するか、またはデータを使用する。
9. [DftiFreeDescriptorDM](#) への呼び出しにより、ディスクリプターに割り当てられているメモリーを解放する。
10. `MPI_Finalize` (C/C++ の場合) または `MPI_FINALIZE` (Fortran の場合) を呼び出し、MPI 通信を終了する (クラスター DFT 関数および MPI 関数への最終呼び出し終了後に行わなければならない)。

付録 C の「[クラスター DFT 関数の例](#)」にあるクラスター DFT インターフェイス関数を使用するコード例で、クラスター DFT 計算方法を示す。

プロセス間でのデータの分配

インテル MKL クラスター DFT は、多次元の入力/出力配列 (行列) をすべて 1 次元配列 (ベクトル) に格納する。配列は、行優先順 (C/C++ の場合) および列優先順 (Fortran の場合) で格納される。例えば、サイズ (m,n) の 2 次元行列 A は、次のようにサイズ $m*n$ のベクトル B に格納される。

- C/C++ の場合: $B[i*n+j]=A[i][j]$ ($i=0, \dots, m-1, j=0, \dots, n-1$)
- Fortran の場合: $B(j*m+i)=A(i, j)$ ($i=1, \dots, m, j=1, \dots, n$)



注: FFT 次元の順序は、プログラミング言語の配列の次元の順序と同じである。例えば、長さ (m,n,l) の 3 次元 FFT は、配列 `Ar[m][n][l]` (C/C++ の場合) または `AR(m,n,l)` (Fortran の場合) 上で計算することができる。

クラスター DFT 計算に関与するすべての MPI プロセスは、データのそれぞれの部分の演算を行う。これらのローカル配列は、高速フーリエ変換が適用される仮想グローバル配列を構成する。適切にローカル配列を割り当て (必要な場合)、初期データを設定して、結果データを実際のグローバル配列に収集するか、または結果データを処理する。この操作を行うには、仮想グローバル配列がローカル配列からどのように構成されるのかを理解する必要がある。

多次元変換

多次元変換では、クラスター DFT はインデックスが最も緩やかに変化する次元でデータを分割するため、分割されたデータ部分にはこのインデックスの複数の連続する値とすべての成分が含まれる。C では第 1 次元、Fortran では最後の次元である。C ではグローバル配列が 2 次元の場合、各プロセスに複数の連続する行を与える。“行” という用語は、配列の次元およびプログラミング言語にかかわらず使用される。ローカル配列は、プロセスの階数によって決定された順序で、仮想グローバル配列に割り当てられたメモリーに連続して格納される。例えば、プロセスが 2 つの場合、サイズ $(11,15,12)$ の行列の 3 次元変換の計算では、それぞれサイズ $(6,15,12)$ および $(5,15,12)$ のローカル配列が格納される。

p を MPI プロセスの数、計算する変換の行列のサイズを (m,n,l) とする。C では、MPI プロセスは、サイズ (m_q,n,l) ($m_q=m, q=0, \dots, p-1$) のローカルデータ配列を処理する。ローカル入力配列には実際のグローバル入力配列の適切な部分が含まれる。ローカル出力配列には実際のグローバル出力配列の適切な部分が含まれる。次のクラスター DFT インターフェイスの構成パラメーターから、ローカル配列に含まれるグローバル配列の行を特定できる：CDFT_LOCAL_NX、CDFT_LOCAL_START_X、および CDFT_LOCAL_SIZE。パラメーターの値を取得するには、[DftiGetValueDM](#) 関数を使用する。

- CDFT_LOCAL_NX は、現在のプロセスに割り当てられるグローバル配列の行数を示す。
- CDFT_LOCAL_START_X は、ローカル入力 / 出力配列の最初の行に対応するグローバル入力 / 出力配列の行を示す。A をグローバル配列、L を適切なローカル配列とすると次のようになる。
 - $L[i][j][k]=A[i+cdft_local_start_x][j][k]$ 、
ここで $i=0, \dots, m_q-1, j=0, \dots, n-1, k=0, \dots, l-1$ (C/C++ の場合)
 - $L(i,j,k)=A(i,j,k+cdft_local_start_x-1)$ 、
ここで $i=1, \dots, m_q, j=1, \dots, n, k=1, \dots, l$ (Fortran の場合)

付録 C の [例 C-29](#) は、2 次元 FFT 計算においてクラスター DFT がプロセス間でデータを分配する方法を示す。

1 次元変換

1 次元変換では、初期データおよび結果データのプロセス間の分配は異なり、変換後に特定のプロセスで格納される成分の数も変更前とは異なる。各ローカル配列は、適切なグローバル配列の連続する成分のセグメントを格納する。そのようなセグメントは、成分の数と最初の配列成分のシフトによって決定される。そのため、特定のプロセスが受け取るグローバル入力 / 出力配列のセグメントを指定するには、4 つの構成パラメーターが必要になる。CDFT_LOCAL_NX および CDFT_LOCAL_START_X に加え、CDFT_LOCAL_OUT_NX および CDFT_LOCAL_OUT_START_X の値も使用される (これらのパラメーターの値は、[DftiGetValueDM](#) を使用して取得することができる)。4 つの構成パラメーターの意味は、[表 11-19](#) に示すように、変換の種類に依存する。

表 11-19 1 次元変換のデータ分配構成パラメーター

パラメーターの意味	順方向変換	逆方向変換
入力配列の成分の数	CDFT_LOCAL_NX	CDFT_LOCAL_OUT_NX
入力配列の成分のシフト	CDFT_LOCAL_START_X	CDFT_LOCAL_OUT_START_X
出力配列の成分の数	CDFT_LOCAL_OUT_NX	CDFT_LOCAL_NX
出力配列の成分のシフト	CDFT_LOCAL_OUT_START_X	CDFT_LOCAL_START_X

ローカルデータのメモリーサイズ

クラスター DFT では、正確なサブ配列のサイズよりもやや多めにローカルデータにメモリーを割り当てなければならないことがあるため、ローカル配列に必要なメモリーサイズは、単に CDFT_LOCAL_NX (CDFT_LOCAL_OUT_NX) から計算することはできない。構成パラメーター CDFT_LOCAL_SIZE は、データ成分のローカル入力 / 出力配列のサイズを指定する。現在のクラスター DFT インターフェイスの実装では、データ成分は実数部と虚数部で構成される複素数値である。ローカル入力 / 出力配列のサイズは、それぞれ

れ `CDFT_LOCAL_SIZE*size_of_element` 以上でなければならない。インプレース変換でユーザー定義のワークスペースを使用する場合 (詳細は、[表 11-20](#) を参照)、サイズは同じでなければならない。

付録 C の [例 C-30](#) は、ユーザー定義のワークスペースを使用する 1 次元 FFT 計算においてクラスター DFT がプロセス間でデータを分配する方法を示す。

利用可能な補助関数

1 つの MPI プロセス上にグローバル入力配列がある場合、または 1 つのプロセス上でグローバル出力配列を収集する場合、`MKL_CDFT_ScatterData` 関数および `MKL_CDFT_GatherData` 関数を使用して、プロセス間でデータを分配または収集することができる。これらの関数は、インテル MKL クラスタ・エディション製品に付属のファイルで定義されている。このファイルは、インテル MKL インストール・ディレクトリーの次のサブディレクトリーにある：`examples/cdft/examples_support.c` (C/C++ の場合) `examples/cdff/examples_support.f90` (Fortran の場合)

変換の長さに対する制限

プロセス間でデータを分配するのにクラスター DFT が使用するアルゴリズムでは、DFT 計算に使われる MPI プロセスの数に対して変換の長さが制限される。

- 多次元変換では、最初の 2 つの次元 (C/C++ の場合) または最後の 2 つの次元 (Fortran の場合) の長さは、MPI プロセスの数以上でなければならない。
- 1 次元変換の長さは、それぞれ MPI プロセスの数以上である 2 つの整数の積でなければならない。

この制限が順守されない場合、`CDFT_SPREAD_ERROR` エラーが発生する (詳細は、「[エラーコード](#)」を参照)。この制限を順守するために、変換の長さや MPI プロセスの数 (MPI プログラムの開始時に指定される) を変更することができる。MPI-2 を使用すると、MPI プログラムの実行中にプロセスの数を変更できる。



注: 変換の長さ L が整数の 2 乗 ($L=N^2$) の場合、1 次元 FFT の最高レベルの性能を実現できる。

クラスター DFT インターフェイス



注: インテル MKL 9.0 に実装されているクラスター DFT インターフェイスは、以前のバージョンとは互換性がない。詳細は、「[インテル MKL リリースノート](#)」を参照のこと。

クラスター DFT 関数を使用するには、Fortran では "use" 構文を使用して `MKL_CDFT` モジュールにアクセスする必要がある。C/C++ では "include" 構文を使用してヘッダーファイル `mk1_cdft.h` にアクセスする必要がある。

Fortran インターフェイスでは、派生型 `DFTI_DESCRIPTOR_DM`、さまざまな構成パラメーターの名称とそれらの取り得る値を表す名前付き定数、Fortran 95 の汎用機能性を介したさまざまなオーバーロード関数が利用できる。

C インターフェイスでは、構造型 `DFTI_DESCRIPTOR_DM_HANDLE` や、関数によっては異なる個数の入力引数を受け付ける種々の関数が利用できる。

並列プロセス間の MPI 通信を提供するには、コード内で次の `include` 構文を使用する必要がある。

- Fortran:

```
INCLUDE 'mpif.h'
```

(一部の MPI バージョンでは、'mpif90.h' ヘッダーが代わりに使用される)。

- C/C++:

```
#include "mpi.h"
```

インテル MKL では、クラスター DFT 関数を 3 種類の大きなカテゴリに分けている。

1. **ディスクリプター操作**。このカテゴリは 3 つの関数で構成される。1 つ目の [DftiCreateDescriptorDM](#) は、DFT ディスクリプターを生成しストレージを動的に割り当てる関数である。2 つ目の [DftiCommitDescriptorDM](#) は、ディスクリプターを設定に基づいて「遂行」させる関数である。3 つ目の [DftiFreeDescriptorDM](#) は、ディスクリプター情報に割り当てられているすべてのメモリーを解放する関数である。
2. **DFT 計算**。このカテゴリは 2 つの関数で構成される。1 つ目の [DftiComputeForwardDM](#) は順方向 DFT 計算を実行する関数で、2 つ目の [DftiComputeBackwardDM](#) は逆方向 DFT 計算を実行する関数である。
3. **ディスクリプター構成**。このカテゴリは 2 つの関数で構成される。1 つ目の関数 [DftiSetValueDM](#) は、特定の 1 つの構成値を、多くの構成パラメーターの 1 つに設定する。2 つ目の関数 [DftiGetValueDM](#) は、これら構成パラメーターの任意の 1 つの現在値を取得する (すべてが読み取り可能)。読み取り可能なパラメーター数は多いが、扱えるのは 1 回の関数呼び出しにつき 1 つである。

ディスクリプター操作

このカテゴリは、ディスクリプターの生成、ディスクリプターの遂行、ディスクリプターの解放の 3 つの関数で構成される。

CreateDescriptorDM

ディスクリプター・データ構造にメモリーを割り当て、デフォルト構成設定で具体化する。

構文

Fortran:

```
Status = DftiCreateDescriptorDM(comm, handle, v1, v2, dim, size)
```

```
Status = DftiCreateDescriptorDM(comm, handle, v1, v2, dim, sizes)
```

C/C++:

```
status = DftiCreateDescriptorDM(comm, &handle, v1, v2, dim, size);
status = DftiCreateDescriptorDM(comm, &handle, v1, v2, dim, sizes);
```

入力パラメーター

<i>comm</i>	MPI コミュニケーター。MPI_COMM_WORLD など。
<i>v1</i>	精度。
<i>v2</i>	順方向領域のタイプ。現在のバージョンでは DFTI_COMPLEX でなければならぬ。
<i>dim</i>	変換の次元。
<i>size</i>	1次元の場合の変換の長さ。
<i>sizes</i>	多次元の場合の変換の長さ。

出力パラメーター

<i>handle</i>	変換ハンドルのポインター。この関数は正常に終了すると、作成されたハンドルへのポインターが変数に格納される。
---------------	---

説明

この関数は特定の MPI プロセスにおいてディスクリプター・データ構造にメモリーを割り当て、変換対象の精度、領域、次元、長さをそれぞれのデフォルト構成設定を使って具体化する。領域は順方向変換での領域として解釈される。出力結果として生成されたディスクリプターのポインターが得られる。この関数は、DFT を計算する従来型の多くのソフトウェア・パッケージやライブラリーに見られる「初期化」ルーチン [DftiCommitDescriptorDM](#) とはやや異なっている。値設定関数 [DftiSetValueDM](#) を使用してデフォルト構成設定を変更できることから、この関数は回転因子計算のような計算量の多い計算は実行しない。

精度は、インターフェイスで構成値に与えられる名前付き定数によって指定される。精度の選択肢は DFTI_SINGLE か DFTI_DOUBLE である。入力データ、出力データ、および計算の精度に対応する。設定が DFTI_SINGLE の場合は単精度浮動小数点データ型が有効で、設定が DFTI_DOUBLE の場合は倍精度浮動小数点データ型が有効となる。

次元は変換の次元を表す単純な正の整数である。C/C++ のコンテキストでは、1次元変換の長さは long 型の整数値で、多次元変換の長さは long 型の整数配列である。Fortran のコンテキストでは、長さは整数値または整数配列である。

戻り値

この関数は正常に終了すると DFTI_NO_ERROR を返す。この場合、作成されたハンドルへのポインターは *handle* に格納される。関数が異常終了すると、別のエラークラス定数を返す(定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

```
! Fortran Interface
INTERFACE DftiCreateDescriptorDM
    INTEGER(4) FUNCTION DftiCreateDescriptorDMn(C,H,P1,P2,D,L)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) C,P1,P2,D,L(*)
    END FUNCTION
    INTEGER(4) FUNCTION DftiCreateDescriptorDMl(C,H,P1,P2,D,L)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) C,P1,P2,D,L
    END FUNCTION
END INTERFACE

/* C/C++ prototype */
long DftiCreateDescriptorDM(MPI_Comm,DFTI_DESCRIPTOR_DM_HANDLE*,enum
DFTI_CONFIG_VALUE,enum DFTI_CONFIG_VALUE,long,...);
```

CommitDescriptorDM

実際の DFT 計算を実行可能にするすべての初期化を実行する。

構文

Fortran:

```
Status = DftiCommitDescriptorDM(handle)
```

C/C++:

```
status = DftiCommitDescriptorDM(handle);
```

入力パラメーター

handle DftiCreateDescriptorDM から取得される有効なハンドル。

説明

DFT インターフェイスでは、特定の MPI プロセスでディスクリプターを使って DFT 計算を実行する前に、生成されたディスクリプターを遂行する関数を呼び出す必要がある。DftiCommitDescriptorDM 関数は実際の DFT 計算を実行可能にするすべての初期化を実行する。近代的な実装では、高度に効率化された計算方法を検索するために、さまざまな長さの入力を用いた因子分解を行う調査が含まれる場合がある。

遂行されたディスクリプターの構成パラメーターを、値設定関数(「[ディスクリプター構成](#)」を参照)を介して変更した場合は、計算関数を呼び出す前にディスクリプターの再遂行が必要となる。通常は、この遂行関数の呼び出しのすぐ後に計算関数の呼び出しを続ける(「[DFT 計算](#)」を参照)。

戻り値

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。関数が異常終了すると、別のエラークラス定数を返す(定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

```
! Fortran Interface
INTERFACE DftiCommitDescriptorDM
    INTEGER(4) FUNCTION DftiCommitDescriptorDM(handle);
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: handle
END FUNCTION
END INTERFACE

/* C/C++ prototype */
long DftiCommitDescriptorDM(DFTI_DESCRIPTOR_DM_HANDLE handle);
```

FreeDescriptorDM

ディスクリプターに割り当てられていたメモリーを解放する。

構文

Fortran:

```
Status = DftiFreeDescriptorDM(handle)
```

C/C++:

```
status = DftiFreeDescriptorDM(&handle);
```

入力パラメーター

`handle` DftiCreateDescriptorDM から取得される有効なハンドル。

出力パラメーター

`handle` ディスクリプター・ハンドル。ハンドルに割り当てられていたメモリーは出力時に解放される。

説明

この関数は、特定の MPI プロセスのディスクリプターに割り当てられているすべてのメモリーを解放する。DftiFreeDescriptorDM 関数を呼び出してディスクリプター・ハンドルを削除する。DftiFreeDescriptorDM が正常に実行されると、ディスクリプター・ハンドルは無効になる。

戻り値

この関数は正常に終了すると `DFTI_NO_ERROR` を返す。関数が異常終了すると、別のエラークラス定数を返す(定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

```
! Fortran Interface
INTERFACE DftiFreeDescriptorDM
    INTEGER(4) FUNCTION DftiFreeDescriptorDM(handle)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: handle
    END FUNCTION
END INTERFACE
/* C/C++ prototype */
long DftiFreeDescriptorDM(DFTI_DESCRIPTOR_DM_HANDLE *handle);
```

DFT 計算

このカテゴリは、順方向変換の計算、逆方向変換の計算の2つの関数で構成される。

ComputeForwardDM

順方向離散フーリエ変換を計算する。

構文

Fortran:

```
Status = DftiComputeForwardDM(handle, in_X, out_X)
Status = DftiComputeForwardDM(handle, in_out_X)
```

C/C++:

```
status = DftiComputeForwardDM(handle, in_X, out_X);
status = DftiComputeForwardDM(handle, in_out_X);
```

入力パラメーター

handle 有効なディスクリプター・ハンドル。

in_X, in_out_X 入力データのローカル部分。複素数配列。配列の割り当ておよび初期化については、「[プロセス間でのデータの分配](#)」のセクションを参照のこと。

出力パラメーター

out_X, in_out_X 出力データのローカル部分。複素数配列。配列の割り当てについては、「[プロセス間でのデータの分配](#)」のセクションを参照のこと。

説明

実際の DFT 計算はディスクリプターの生成と遂行を完了した後に実行する。DftiComputeForwardDM 関数は、特定の MPI プロセスで実行される順方向 DFT 計算の一部を実行する。

Forward DFT は、因子 $e^{-i2\pi/n}$ を使用した変換である。DftiComputeForward 関数を呼び出して計算される。この関数は DftiComputeForward 関数と多くの共通点があるため、詳細については [DftiComputeForward](#) の説明を参照のこと。

有効なディスクリプター・ハンドルは [DftiCreateDescriptorDM](#) によって作成され、[DftiCommitDescriptorDM](#) によって遂行される。ディスクリプター・ハンドルによってこの関数に渡される構成パラメーターのリストを [表 11-21](#) に示す。

入力/出力データのローカル部分は、特定のプロセスによって格納される適切な複素数値のシーケンス(それぞれの複素数値は、実数部と虚数部の2つの実数で構成される)である。詳細は、「[プロセス間でのデータの分配](#)」のセクションを参照。

入力データと出力データの精度の選択は、変換精度と同じである。設定が DFTI_SINGLE の場合は単精度浮動小数点データ型が有効で、設定が DFTI_DOUBLE の場合は倍精度浮動小数点データ型が有効となる。

構成パラメーター DFTI_PLACEMENT は、インプレースで計算を行うべきかどうかを関数に知らせる。このパラメーターの値が DFTI_INPLACE (デフォルト) の場合は、2つのパラメーターでこの関数を呼び出す。そうでない場合は、3つのパラメーターで呼び出す。DFTI_PLACEMENT = DFTI_INPLACE と3つのパラメーターが使用されると、3つ目のパラメーターは無視される。



注意: アウトオブプレース変換でも、入力データ in_X のローカル配列が変更されることがある。データを保存するためには、DftiComputeForwardDM を呼び出す前にコピーを作成する。

インプレース変換では、DftiComputeForwardDM はローカル入力/出力配列に必要なサイズと同じサイズのワークバッファーを動的に割り当て、その後割り当てを解除する。



注: 冗長なメモリーの割り当てを回避するために、構成パラメーター CDFT_WORKSPACE によって同じサイズの独自のワークスペースを指定することができる。

戻り値

この関数は正常に終了すると DFTI_NO_ERROR を返す。関数が異常終了すると、別のエラークラス定数を返す(定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

! Fortran Interface

```
INTERFACE DftiComputeForwardDM
  INTEGER(4) FUNCTION DftiComputeForwardDM(h, in_X, out_X)
    TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
    COMPLEX(8), DIMENSION(*) :: in_x, out_X
  END FUNCTION DftiComputeForwardDM
```

```

    INTEGER(4) FUNCTION DftiComputeForwardDMi(h, in_out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(8), DIMENSION(*) :: in_out_X
    END FUNCTION DftiComputeForwardDMi
    INTEGER(4) FUNCTION DftiComputeForwardDMs(h, in_X, out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(4), DIMENSION(*) :: in_X, out_X
    END FUNCTION DftiComputeForwardDMs
    INTEGER(4) FUNCTION DftiComputeForwardDMis(h, in_out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(4), DIMENSION(*) :: in_out_X
    END FUNCTION DftiComputeForwardDMis
END INTERFACE

/* C/C++ prototype */
long DftiComputeForwardDM(DFTI_DESCRIPTOR_DM_HANDLE handle, void *in_X,...);

```

ComputeBackwardDM

逆方向離散フーリエ変換を計算する。

構文

Fortran:

```

Status = DftiComputeBackwardDM(handle, in_X, out_X)
Status = DftiComputeBackwardDM(handle, in_out_X)

```

C/C++:

```

status = DftiComputeBackwardDM(handle, in_X, out_X);
status = DftiComputeBackwardDM(handle, in_out_X);

```

入力パラメーター

handle 有効なディスクリプター・ハンドル。

in_X, *in_out_X* 入力データのローカル部分。複素数配列。配列の割り当ておよび初期化については、「[プロセス間でのデータの分配](#)」のセクションを参照のこと。

出力パラメーター

out_X, *in_out_X* 出力データのローカル部分。複素数配列。配列の割り当てについては、「[プロセス間でのデータの分配](#)」のセクションを参照のこと。

説明

実際の DFT 計算はディスクリプターの生成と遂行を完了した後に実行する。
DftiComputeBackwardDM 関数は、特定の MPI プロセスで実行される逆方向 DFT 計算の一部を実行する。

Backward DFT は、因子 $e^{i2\pi/n}$ を使用した変換である。DftiComputeBackward 関数を呼び出して計算される。この関数は DftiComputeBackward 関数と多くの共通点があるため、詳細については [DftiComputeBackward](#) の説明を参照のこと。

有効なディスクリプター・ハンドルは [DftiCreateDescriptorDM](#) によって作成され、[DftiCommitDescriptorDM](#) によって遂行される。ディスクリプター・ハンドルによってこの関数に渡される構成パラメーターのリストを [表 11-21](#) に示す。

入力 / 出力データのローカル部分は、特定のプロセスによって格納される適切な複素数値のシーケンス (それぞれの複素数値は、実数部と虚数部の 2 つの実数で構成される) である。詳細は、「[プロセス間でのデータの分配](#)」のセクションを参照。

入力データと出力データの精度の選択は、変換精度と同じである。設定が DFTI_SINGLE の場合は単精度浮動小数点データ型が有効で、設定が DFTI_DOUBLE の場合は倍精度浮動小数点データ型が有効となる。

構成パラメーター DFTI_PLACEMENT は、インプレースで計算を行うべきかどうかを関数に知らせる。このパラメーターの値が DFTI_INPLACE (デフォルト) の場合は、2 つのパラメーターでこの関数を呼び出す。そうでない場合は、3 つのパラメーターで呼び出す。DFTI_PLACEMENT=DFTI_INPLACE と 3 つのパラメーターが使用されると、3 つ目のパラメーターは無視される。



注意: アウトオブプレース変換でも、入力データ in_x のローカル配列が変更されることがある。データを保存するためには、DftiComputeForwardDM を呼び出す前にコピーを作成する。

インプレース変換では、DftiComputeBackwardDM はローカル入力 / 出力配列に必要なサイズと同じサイズのワークバッファーを動的に割り当て、その後割り当てを解除する。



注: 冗長なメモリーの割り当てを回避するために、構成パラメーター CDFT_WORKSPACE によって同じサイズ以上の独自のワークスペースを指定することができる。

戻り値

この関数は正常に終了すると DFTI_NO_ERROR を返す。関数が異常終了すると、別のエラーコードを返す (定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

! Fortran Interface

```

INTERFACE DftiComputeBackwardDM
    INTEGER(4) FUNCTION DftiComputeBackwardDM(h, in_X, out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(8), DIMENSION(*) :: in_x, out_X
    END FUNCTION DftiComputeBackwardDM
    INTEGER(4) FUNCTION DftiComputeBackwardDMi(h, in_out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(8), DIMENSION(*) :: in_out_X
    END FUNCTION DftiComputeBackwardDMi
    INTEGER(4) FUNCTION DftiComputeBackwardDMs(h, in_X, out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(4), DIMENSION(*) :: in_x, out_X
    END FUNCTION DftiComputeBackwardDMs
    INTEGER(4) FUNCTION DftiComputeBackwardDMis(h, in_out_X)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        COMPLEX(4), DIMENSION(*) :: in_out_X
    END FUNCTION DftiComputeBackwardDMis
END INTERFACE

/* C/C++ prototype */
long DftiComputeBackwardDM(DFTI_DESCRIPTOR_DM_HANDLE handle, void *in_X,...);

```

ディスクリプター構成

このカテゴリは、特定の1つの構成パラメーターに適切な値を設定する値設定関数 [DftiSetValueDM](#) と、特定の1つの構成パラメーター値を読み取る値取得関数 [DftiGetValueDM](#) の2つで構成される。

クラスター DFT 関数によって使用されるいくつかの構成パラメーターは、従来の DFT インターフェイスからのものである (詳細は、「[DFT 関数](#)」セクションの「[構成設定](#)」を参照)。

その他のパラメーターはクラスター DFT 独自のものである。これらの構成パラメーターの整数値は、long 型 (C/C++ の場合) および INTEGER(4) 型 (Fortran の場合) である。浮動小数点スカラーである構成パラメーターは、float 型または double 型 (C/C++ の場合) および REAL(4) 型または REAL(8) 型 (Fortran の場合) である。名前付き定数によって値を識別される構成パラメーターは、enum 型 (C/C++ の場合) および INTEGER 型 (Fortran の場合) である。これらは、mkl_cdft.h ヘッダーファイル (C/C++ の場合) および MKL_CDFT モジュール (Fortran の場合) で定義されている。

CDFT 固有の構成パラメーターのプリフィックスは CDFT である。

SetValueDM

特定の1つの構成パラメーターを指定された構成値で設定する。

構文

Fortran:

```
Status = DftiSetValueDM(handle, param, value)
```

C/C++:

```
status = DftiSetValueDM(handle, param, value);
```

入力パラメーター

<i>handle</i>	有効なディスクリプター・ハンドル。
<i>param</i>	ディスクリプター・ハンドルに設定されるパラメーター名。利用できる名前の一覧は 表 11-20 を参照のこと。
<i>value</i>	パラメーターの値。

説明

この関数は特定の1つの構成パラメーターを指定された構成値で設定する。構成パラメーターは、以下の表でデフォルト値(存在する場合)とともに示される名前付き定数の中の1つである。設定およびその他の詳細については、「[構成設定](#)」を参照のこと。特に、名前付き定数タイプの構成パラメーターに指定可能な値は[表 11-6](#)を参照のこと。

表 11-20 設定可能な構成パラメーター

名前付き定数	パラメーターのタイプ	説明	デフォルト値
DFTI_FORWARD_SCALE	浮動小数点スカラー	順方向変換の倍率因子。	1.0
DFTI_BACKWARD_SCALE	浮動小数点スカラー	逆方向変換の倍率因子。	1.0
DFTI_PLACEMENT	名前付き定数	計算結果の配置。	DFTI_INPLACE
CDFT_WORKSPACE	適切な型の配列	補助バッファ。ユーザー定義のワークスペース。インプレース計算時のメモリの保存を有効にする。	NULL (ワークスペースの動的な割り当て)

戻り値

この関数は正常に終了すると DFTI_NO_ERROR を返す。関数が異常終了すると、別のエラーコードを返す(定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

```
! Fortran Interface
INTERFACE DftiSetValueDM
    INTEGER(4) FUNCTION DftiSetValueDM(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p, v
    END FUNCTION
    INTEGER(4) FUNCTION DftiSetValueDMd(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        REAL(8) :: v
    END FUNCTION
    INTEGER(4) FUNCTION DftiSetValueDMs(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        REAL(4) :: v
    END FUNCTION
    INTEGER(4) FUNCTION DftiSetValueDMsw(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        COMPLEX(4) :: v(*)
    END FUNCTION
    INTEGER(4) FUNCTION DftiSetValueDMdw(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        COMPLEX(8) :: v(*)
    END FUNCTION
END INTERFACE
/* C/C++ prototype */
long DftiSetValueDM(DFTI_DESCRIPTOR_DM_HANDLE handle, int param,...);
```

GetValueDM

特定の1つの構成パラメーターの構成値を取得する。

構文

Fortran:

```
Status = DftiGetValueDM(handle, param, value)
```

C/C++:

```
status = DftiGetValueDM(handle, param, &value);
```

入力パラメーター

- handle* 有効なディスクリプター・ハンドル。
- param* ディスクリプター・ハンドルから取得されるパラメーター名。利用できる名前の一覧は[表 11-21](#) を参照のこと。

出力パラメーター

- value* パラメーターの値。

説明

この関数は特定の 1 つの構成パラメーターを取得する。構成パラメーターは以下の表に示される名前付き定数のうちの 1 つである。構成値は対応する適切なタイプ (名前付き定数またはネイティブタイプ) である。名前付き定数に指定可能な値は [表 11-6](#) を参照のこと。

表 11-21 取得可能な構成パラメーター

名前付き定数	パラメーターのタイプ	説明
DFTI_PRECISION	名前付き定数	計算精度、入力データおよび出力データ。
DFTI_DIMENSION	整数スカラー	変換の次元。
DFTI_LENGTHS	整数値の配列	変換の長さの配列。成分の数は変換の次元に等しい。
DFTI_FORWARD_SCALE	浮動小数点スカラー	順方向変換の倍率因子。
DFTI_BACKWARD_SCALE	浮動小数点スカラー	逆方向変換の倍率因子。
DFTI_PLACEMENT	名前付き定数	計算結果の配置。
DFTI_COMMIT_STATUS	名前付き定数	ディスクリプターが遂行されているかを示す。
DFTI_FORWARD_DOMAIN	名前付き定数	変換の順方向領域 (現在のクラスター DFT インターフェイスの実装では、常に DFTI_COMPLEX)。
CDFT_MPI_COMM	MPI コミュニケーターのタイプ	変換に使用される MPI コミュニケーター。
CDFT_LOCAL_SIZE	整数スカラー	データ成分の入力配列、出力配列、バッファ配列に必要なサイズ。
CDFT_LOCAL_X_START	整数スカラー	ローカル配列の最初の行 / 成分に対応するグローバル配列の行 / 成分の番号。詳細は、「 プロセス間でのデータの分配 」を参照。
CDFT_LOCAL_NX	整数スカラー	ローカル配列に格納されるグローバル配列の行 / 成分の数。詳細は、「 プロセス間でのデータの分配 」を参照。
CDFT_LOCAL_OUT_X_START	整数スカラー	1 次元の場合に、入力 / 出力ローカル配列の最初の成分に対応する適切なグローバル配列の成分の番号。詳細は「 プロセス間でのデータの分配 」を参照のこと。

表 11-21 取得可能な構成パラメーター (続き)

名前付き定数	パラメーターのタイプ	説明
CDFT_LOCAL_OUT_NX	整数スカラー	1次元の場合に、入力 / 出力ローカル配列に格納される適切なグローバル配列の成分の数。詳細は「 プロセス間でのデータの分配 」を参照のこと。

戻り値

この関数は正常に終了すると DFTI_NO_ERROR を返す。関数が異常終了すると、別のエラーコードを返す (定数のリストは、「[エラーコード](#)」のセクションを参照)。

インターフェイスとプロトタイプ

! Fortran Interface

```

INTERFACE DftiGetValueDM
    INTEGER(4) FUNCTION DftiGetValueDM(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p, v
    END FUNCTION
    INTEGER(4) FUNCTION DftiGetValueDMar(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p, v(*)
    END FUNCTION
    INTEGER(4) FUNCTION DftiGetValueDMd(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        REAL(8) :: v
    END FUNCTION
    INTEGER(4) FUNCTION DftiGetValueDMs(h, p, v)
        TYPE(DFTI_DESCRIPTOR_DM), POINTER :: h
        INTEGER(4) :: p
        REAL(4) :: v
    END FUNCTION
END INTERFACE
/* C/C++ prototype */
long DftiGetValueDM(DFTI_DESCRIPTOR_DM_HANDLE handle, int param,...);

```

エラーコード

すべてのクラスター DFT 関数は演算のステータスを表す整数値を返す。これらの値は名前付き定数によって識別される。実行時にエラーが発生しなかった場合、各関数は DFTI_NO_ERROR を返す。そうでない場合、関数はエラーコードを生成する。DFT エラーコードに加え、CDFT 固有のエラーコードがある。CDFT 固有の名前付き定数のプリフィックスは "CDFT" である。[表 11-22](#) は、クラスター DFT 関数によって返されるエラーコードをまとめたものである。

表 11-22 クラスタ DFT 関数によって返されるエラーコード

名前付き定数	意味
DFTI_NO_ERROR	エラーなし。
DFTI_MEMORY_ERROR	通常、メモリー割り当てに関係する。
DFTI_INVALID_CONFIGURATION	構成パラメーターの 1 つ以上の設定が無効。
DFTI_INCONSISTENT_CONFIGURATION	構成パラメーターまたは入力パラメーターが不整合。
DFTI_NUMBER_OF_THREADS_ERROR	計算関数の OMP スレッド数が、(遂行関数の) 初期化段階の OMP スレッド数と等しくない。
DFTI_MULTITHREADED_ERROR	通常、OMP ルーチンのエラー戻り値に関係する。
DFTI_BAD_DESCRIPTOR	ディスクリプターを計算に使用できない。
DFTI_UNIMPLEMENTED	正当な設定が未実装、実装依存。
DFTI_MKL_INTERNAL_ERROR	ライブラリー内部エラー。
DFTI_1D_LENGTH_EXCEEDS_INT32	1 つの次元の長さが $2^{32} - 1$ (4 バイト) を超えている。
CDFT_SPREAD_ERROR	データを分配できない (詳細は「 プロセス間でのデータの分配 」を参照)。
CDFT_MPI_ERROR	MPI エラー。MPI の呼び出し時に発生する。

区間線形ソルバー

12

この章では以下の目的に使用することのできるインテル® MKL ルーチンについて説明する。

- 連立区間線形方程式 $Ax = b$ を解く。 $A = (a_{ij})$ は区間行列、 $b = (b_i)$ は右辺の区間ベクトルである。
- 区間行列の特性を調べる。

区間線形方程式の主要な概念についての詳細は 付録 A の「[線形ソルバーの基礎](#)」を参照のこと。

以下のルーチンの説明は、課題に応じて次のグループに分けて行う。

「[区間方程式を高速に解くためのルーチン](#)」

「[区間方程式の精密な解を求めるためのルーチン](#)」

「[区間行列逆転用のルーチン](#)」

「[区間行列の特性確認用のルーチン](#)」

「[補助およびユーティリティー・ルーチン](#)」

表 12-1 は、区間線形方程式を解くためのインテル MKL ルーチンの全リストである。

表 12-1 インテル MKL 区間線形ソルバールーチン

ルーチン名	説明
?trtrrs	三角連立区間線形方程式を後方置換プロシージャによって解く。
?qegas	連立区間線形方程式を区間ガウス法によって解く。
?qehss	連立区間線形方程式を区間 Housholder 法によって解く。
?qekws	連立区間線形方程式を Krawczyk 反復法によって解く。
?qegss	連立区間線形方程式を区間 Gauss-Seidel 反復法によって解く。
?qehbs	連立区間線形方程式を Hansen-Bliek-Rohn プロシージャによって解く。
?qepbs	連立区間線形方程式をパラメーター分割法によって解く。
?qepss	連立区間線形方程式を解分割法によって解く。
?trtri	三角区間行列の逆区間行列を計算する。
?qeszi	逆区間行列を Schulz 区間反復プロシージャによって計算する。
?qerbr	区間行列の正則性を Ris-Beeck と Rex-Rohn 判定基準によってテストする。
?qesvr	区間行列の正則性 / 特異性を Rump と Rex-Rohn 特異値判定基準によってテストする。

表 12-1 インテル MKL 区間線形ソルバールーチン (続き)

ルーチン名	説明
?gemip	区間線形方程式の中間逆転プリコンディショニングを実行する。

ルーチン命名規則

以下に紹介するルーチンでは、LAPACK に類似する命名規則が使用されている。具体的には、すべてのルーチン名は `xxyyzzz` の構造である。冒頭の文字 `xx` は、データ型を示す。

<code>si</code>	単精度実数区間
<code>di</code>	倍精度実数区間
<code>cr</code>	単精度複素数区間 (矩形領域)
<code>zr</code>	倍精度複素数区間 (矩形領域)
<code>cc</code>	単精度複素数区間 (円領域)
<code>zc</code>	倍精度複素数区間 (円領域)

3 番目と 4 番目の文字 `yy` は、行列のタイプを示す。

<code>ge</code>	一般行列
<code>tr</code>	三角行列

最後の 3 文字 `zzz` は、実行される処理 (下記を参照) を示す。

<code>trs</code>	三角区間線形方程式の後方置換ソルバー
<code>gas</code>	区間線形方程式の区間ガウスソルバー
<code>has</code>	区間線形方程式の区間 Householder 法ソルバー
<code>kws</code>	区間線形方程式の Krawczyk 反復法ソルバー
<code>gss</code>	区間線形方程式の区間 Gauss-Seidel 反復法ソルバー
<code>hbs</code>	区間線形方程式の Hansen-Bliek-Rohn ソルバー
<code>pps</code>	区間線形方程式のパラメーター分割法に基づくソルバー
<code>pss</code>	区間線形方程式の解分割法に基づくソルバー
<code>tri</code>	後方置換に基づく三角区間行列の逆行列計算
<code>szi</code>	Schulz 反復法による一般区間行列の逆行列計算
<code>rbr</code>	Ris-Beeck 判定基準による区間行列の正則性 / 特異性のテスト
<code>svr</code>	Rump と Rex-Rohn 特異値判定基準による区間行列の正則性 / 特異性のテスト
<code>mip</code>	区間線形方程式の中間逆転プリコンディショニング

グループ名にある疑問符は、データ型を示す各種のキャラクター・コード (`si`、`di`、`cr`、`zr`、`cc`、`zc`) に対応している。例えば、`?trtri` は、ルーチン `sitrtri`、`ditrtri`、`ctrtri`、`zrtrtri`、`cctrtri`、`zctrtri` のグループ名を示す。

区間方程式を高速に解くためのルーチン

?trtrs

三角連立区間線形方程式を後方置換プロシージャによって解く。

構文

```
call sitrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call ditrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call crtrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call zrtrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call cctrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
call zctrtrs(uplo, trans, diag, n, nrhs, a, lda, b, ldb, info)
```

説明

?trtrs ルーチンは、行列 \mathbf{B} に格納された複数の右辺を使用して、三角行列 \mathbf{A} を持つ以下の連立区間線形方程式を \mathbf{X} について解く。

$\mathbf{AX} = \mathbf{B}$ ($trans = 'N'$ の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ ($trans = 'T'$ の場合)

$\mathbf{A}^H \mathbf{X} = \mathbf{B}$ ($trans = 'C'$ の場合。ただし、複素行列のみ)

このルーチンは、後方置換アルゴリズムを実装し、行列 \mathbf{A} の単純な構造により、区間線形方程式に対する解集合の最適なエンクロージャを生成する。

入力パラメーター

<i>uplo</i>	CHARACTER(1)。'U'、'L'、'u'、または 'l' のいずれかでなければならない。 \mathbf{A} が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' または 'u' の場合、 \mathbf{A} は上三角行列である。 <i>uplo</i> = 'L' または 'l' の場合、 \mathbf{A} は下三角行列である。
<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 <i>trans</i> = 'N' または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を \mathbf{X} について解く。 <i>trans</i> = 'T' または 't' の場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。 <i>trans</i> = 'C' または 'c' の場合、 $\mathbf{A}^H \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>diag</i>	CHARACTER(1)。'N'、'U'、'n'、または 'u' のいずれかでなければならない。 <i>diag</i> = 'N' または 'n' の場合、 \mathbf{A} は単位三角行列ではない。 <i>diag</i> = 'U' または 'u' の場合、 \mathbf{A} は単位三角行列である。 \mathbf{A} の対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。
<i>n</i>	INTEGER。 \mathbf{A} の次数。 \mathbf{B} の行の数 ($n \geq 0$)。

<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	S_INTERVAL (<i>sitrtrs</i> の場合) D_INTERVAL (<i>ditrtrs</i> の場合) CR_INTERVAL (<i>crtrtrs</i> の場合) ZR_INTERVAL (<i>zrtrtrs</i> の場合) CC_INTERVAL (<i>cctrtrs</i> の場合) ZC_INTERVAL (<i>zctrtrs</i> の場合) 配列: <i>a</i> (<i>lda, *</i>)、 <i>b</i> (<i>ldb, *</i>)。 配列 <i>a</i> には、行列 A を格納する。 配列 <i>b</i> には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 X によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

?gegas

連立区間線形方程式を
区間ガウス法によって解く。

構文

```
call sigegas(trans, n, nrhs, a, lda, b, ldb, info)
call digegas(trans, n, nrhs, a, lda, b, ldb, info)
call crgegas(trans, n, nrhs, a, lda, b, ldb, info)
call zrgegas(trans, n, nrhs, a, lda, b, ldb, info)
call ccgegas(trans, n, nrhs, a, lda, b, ldb, info)
call zcgegas(trans, n, nrhs, a, lda, b, ldb, info)
```

説明

?gegas ルーチンは、区間ガウス法を使用して、以下の連立区間線形方程式に対する解集合のエンクロージャを計算する。

$\mathbf{AX} = \mathbf{B}$ (*trans* = 'N' の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ ($trans = 'T'$ の場合)

$\mathbf{A}^H \mathbf{X} = \mathbf{B}$ ($trans = 'C'$ の場合。ただし、複素行列のみ)

入力パラメーター

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'T'$ または 't' の場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'C'$ または 'c' の場合、 $\mathbf{A}^H \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	S_INTERVAL (sigegas の場合) D_INTERVAL (digegas の場合) CR_INTERVAL (crgegas の場合) ZR_INTERVAL (zrgegas の場合) CC_INTERVAL (ccgegas の場合) ZC_INTERVAL (zcgegas の場合) 配列: $a(lda, *)$ 、 $b(ldb, *)$ 。 配列 a には、行列 \mathbf{A} を格納する。 配列 b には、行列 \mathbf{B} を格納する。この行列の列は、連立方程式の右辺である。 a の第 2 次元は $\max(1, n)$ 以上でなければならない。 b の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 \mathbf{X} によって上書きされる。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info > 0$ の場合、実行は正常に終了しなかったことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正だったことを示す。

例 12-1 区間ガウス法の Fortran 90 コード

次の Fortran コードの部分は、digegas ルーチンを使用して、区間ガウス法で、連立区間線形方程式に対する解集合のエンクロージャを計算する。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix} \mathbf{X} = \begin{pmatrix} [0, 120] \\ [60, 240] \end{pmatrix}$$

```

.....
USE INTERVAL_ARITHMETIC
.....
TYPE(D_INTERVAL)    :: A(2,2), B(2)
INTEGER              :: N, INFO
CHARACTER(1)        :: TRANS = 'n'
.....
N = 2
A(1,1) = DINTERVAL(2.,3.); A(1,2) = DINTERVAL(0.,1.)
A(2,1) = DINTERVAL(1.,2.); A(2,2) = DINTERVAL(2.,3.)
B(1,1) = DINTERVAL(0.,120.); B(2,1) = DINTERVAL(60.,240.)
.....
CALL DIGEGAS( TRANS, N, 1, A, 2, B, 2, INFO )

```

行列 **A** と右辺ベクトル **B** の成分への倍精度区間の割り当ては、INTERVAL_ARITHMETIC モジュールで供給される DINTERVAL 関数によって実行される点に注意すること。

?gehss

連立区間線形方程式を
Householder 法によって解く

構文

```

call sigehss(trans, n, nrhs, a, lda, b, ldb, info)
call digehss(trans, n, nrhs, a, lda, b, ldb, info)

```

説明

?gegas ルーチンは、区間 Householder 法を使用して、以下の連立区間線形方程式に対する解集合のエンクロージャを計算する。

$\mathbf{AX} = \mathbf{B}$ (trans = 'N' の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ (trans = 'T' または 'C' の場合)

入力パラメーター

trans CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。
連立方程式の形式を指定する。
trans = 'N' または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を **X** について解く。
trans = 'T'、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を **X** について解く。

<i>n</i>	INTEGER。 A の次数。 B の行の数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	REAL (sigehsss の場合)。 DOUBLE PRECISION (digehtss の場合)。 配列 : <i>a</i> (<i>lda</i> , *), <i>b</i> (<i>ldb</i> , *)。 配列 <i>a</i> には、行列 A を格納する。 配列 <i>b</i> には、行列 B を格納する。この行列の列は、連立方程式の右辺である。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 X のエンクロージャによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

?gekws

連立区間線形方程式を
Krawczyk 反復法によって解く。

構文

```
call sigekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call digekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call crgekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call zrgekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call ccgekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
call zcgekws(trans, n, nrhs, a, lda, b, ldb, epsilon, info)
```

説明

?gekws ルーチンは、Krawczyk 区間反復法を使用して、以下の連立区間線形方程式に対する解集合のエンクロージャを計算する。

$\mathbf{AX} = \mathbf{B}$ (*trans* = 'N' の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ (*trans* = 'T' の場合)

$\mathbf{A}^H \mathbf{X} = \mathbf{B}$ (*trans* = 'C' の場合。ただし、複素行列のみ)

入力パラメーター

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または'c'のいずれかでなければならない。 連立方程式の形式を指定する。 <i>trans</i> = 'N' または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を \mathbf{X} について解く。 <i>trans</i> = 'T' または 't' の場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。 <i>trans</i> = 'C' または 'c' の場合、 $\mathbf{A}^H \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>n</i>	INTEGER。 \mathbf{A} の次数。 \mathbf{B} の行の数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。 右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	S_INTERVAL (<i>sigekws</i> の場合) D_INTERVAL (<i>digekws</i> の場合) CR_INTERVAL (<i>crgekws</i> の場合) ZR_INTERVAL (<i>zrgekws</i> の場合) CC_INTERVAL (<i>ccgekws</i> の場合) ZC_INTERVAL (<i>zcgekws</i> の場合) 配列: <i>a</i> (<i>lda</i> , *), <i>b</i> (<i>ldb</i> , *)。 配列 <i>a</i> には、行列 \mathbf{A} を格納する。 配列 <i>b</i> には、行列 \mathbf{B} を格納する。この行列の列は、連立方程式の右辺である。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>epsilon</i>	REAL (<i>sigekws</i> , <i>crgekws</i> , <i>ccgekws</i> の場合) DOUBLE PRECISION (<i>digekws</i> , <i>zrgekws</i> , <i>zcgekws</i> の場合) 反復の終了に対する許容値。

出力パラメーター

<i>b</i>	解の行列 \mathbf{X} のエンクロージャによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

Krawczyk 区間反復法では、すでに中点逆転プリコンディショニングが組み込まれているため、さらに [?gemip](#) ルーチンを適用する必要はなく、全体の効率向上にもつながらない。

?gegss

連立区間線形方程式を
区間 Gauss-Seidel 反復法によって解く。

構文

```
call sigegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call digegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call crgegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call zrgegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call ccgegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
call zcgegss(trans, n, nrhs, a, lda, b, ldb, encl, epsilon, nits, info)
```

説明

?gegss ルーチンは、区間 Gauss-Seidel 反復法を使用して、以下の連立区間線形方程式に対する解集合の部分のエンクロージャを計算する。

$\mathbf{AX} = \mathbf{B}$ ($trans = 'N'$ の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ ($trans = 'T'$ の場合)

$\mathbf{A}^H \mathbf{X} = \mathbf{B}$ ($trans = 'C'$ の場合。ただし、複素行列のみ)

このルーチンを使用したコードの例は、本書付録 C の「[区間連立線形方程式ソルバーのコード例](#)」を参照のこと。

入力パラメーター

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または 'n' の場合、 $\mathbf{AX} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'T'$ または 't' の場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'C'$ または 'c' の場合、 $\mathbf{A}^H \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	S_INTERVAL (sigegss の場合) D_INTERVAL (digegss の場合) CR_INTERVAL (crgegss の場合) ZR_INTERVAL (zrgegss の場合) CC_INTERVAL (ccgegss の場合) ZC_INTERVAL (zcgegss の場合) 配列 : $a(lda, *)$ 、 $b(ldb, *)$ 。 配列 a には、行列 \mathbf{A} を格納する。 配列 b には、行列 \mathbf{B} を格納する。この行列の列は、連立方程式の右辺である。

<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>encl</i>	S_INTERVAL (<i>sigegss</i> の場合) D_INTERVAL (<i>digegss</i> の場合) CR_INTERVAL (<i>crgegss</i> の場合) ZR_INTERVAL (<i>zrgegss</i> の場合) CC_INTERVAL (<i>ccgegss</i> の場合) ZC_INTERVAL (<i>zcgegss</i> の場合) 配列: <i>encl</i> (<i>ldb</i> , *)。 配列 <i>encl</i> は、このルーチンが推定する解集合の部分境界を境界付ける区間箱を定義する。
<i>epsilon</i>	REAL (<i>sigegss</i> 、 <i>crgegss</i> 、 <i>ccgegss</i> の場合) DOUBLE PRECISION (<i>digegss</i> 、 <i>zrgegss</i> 、 <i>zcgegss</i> の場合) 反復の終了に対する許容値。
<i>nits</i>	INTEGER。割り当てた Gauss-Seidel 反復法の回数 ($nits \geq 0$)。

出力パラメーター

<i>b</i>	解の行列 <i>X</i> のエンクロージャによって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、行列の対角成分 <i>a</i> (<i>i</i> , <i>i</i>) はゼロを含む。ルーチンの実行は失敗ではないが、行列の行や列を交換し、その主対角からゼロを含む成分を除外することが推奨される。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーター値が不正だったことを示す。

アプリケーション・ノート

区間 Gauss-Seidel 反復法は、区間線形方程式の局所ソルバーである。すなわち、この処理は \mathbf{R}^n 空間において与えられた区間箱で境界付けられた解集合の部分のエンクロージャを計算することを目的とするものである。

?gegss によって区間線形方程式に対する全体の解の集合のエンクロージャを計算する場合は、その最初の (自然の) エンクロージャを *encl* 引数で指定しなければならない。

?gehbs

連立区間線形方程式を
Hansen-Blik-Rohn プロシージャによって解く。

構文

```
call sigehbs(trans, n, a, lda, b, ldb, info)
call digehbs(trans, n, a, lda, b, ldb, info)
```

説明

?gehbs ルーチンは、区間 Hansen-Blick-Rohn 法を使用して、以下の連立区間線形方程式に対する解集合のエンクロージャを計算する。

$\mathbf{A}\mathbf{X} = \mathbf{B}$ ($trans = 'N'$ の場合)

$\mathbf{A}^T\mathbf{X} = \mathbf{B}$ ($trans = 'T'$ または $'C'$ の場合)

このルーチンを使用したコードの例は、本書付録 C の「[区間連立線形方程式ソルバーのコード例](#)」を参照のこと。

入力パラメーター

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または 'c' のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または $'n'$ の場合、 $\mathbf{A}\mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T\mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
<i>a, b</i>	REAL (sigehbs の場合)。 DOUBLE PRECISION (digehbs の場合)。 配列: $a(lda, *)$ 、 $b ldb, *$ 。 配列 a には、行列 A を格納する。 配列 b には、行列 B を格納する。この行列の列は、連立方程式の右辺である。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>b</i>	解の行列 \mathbf{X} によって上書きされる。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info > 0$ の場合、実行は正常に終了しなかったことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

A の中点行列が対角行列に近くない場合には、区間線形方程式を修正し、より良い結果を得るために、[?qemip](#) ルーチンによる中点逆転プリコンディショニングが必要と考えられる。

区間方程式の精密な解を求めるためのルーチン

?gepps

連立区間線形方程式を
パラメーター分割法によって解く。

構文

```
call siggepps(trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits,
             info)
call digepps(trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits,
             info)
```

説明

?gepps ルーチンは、パラメーター分割法 (PPS) を使用して、以下の連立区間線形方程式に対する解集合のいくつかの (またはすべての) 精密な外部の成分に関する推定を計算する。

$\mathbf{AX} = \mathbf{B}$ ($trans = 'N'$ の場合)

$\mathbf{A}^T \mathbf{X} = \mathbf{B}$ ($trans = 'T'$ または $'C'$ の場合)

入力パラメーター

<i>trans</i>	CHARACTER(1)。'N'、'T'、'C'、'n'、't'、または'c'のいずれかでなければならない。 連立方程式の形式を指定する。 $trans = 'N'$ または $'n'$ の場合、 $\mathbf{AX} = \mathbf{B}$ を \mathbf{X} について解く。 $trans = 'T'$ 、'C'、't'、'c' のいずれかの場合、 $\mathbf{A}^T \mathbf{X} = \mathbf{B}$ を \mathbf{X} について解く。
<i>n</i>	INTEGER。A の次数。B の行の数 ($n \geq 0$)。
<i>a, b</i>	REAL (siggepps の場合)。 DOUBLE PRECISION (digepps の場合)。 配列: <i>a</i> (<i>lda</i> , *), <i>b</i> (<i>ldb</i>)。 配列 <i>a</i> には、行列 A を格納する。 配列 <i>b</i> には、行列 B を格納する。この行列の列は、連立方程式の右辺である。
<i>lda</i>	INTEGER。a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>cmpt</i>	INTEGER。推定する解集合の成分の数。
<i>mode</i>	CHARACTER(1)。「L」または「U」(または対応する小文字)でなければならない。 パラメーター <i>cmpt</i> で指定された座標に沿う解集合の推定方法を示す。 $mode = 'L'$ または $'l'$ の場合、ルーチンは <i>cmpt</i> 番目の座標で

の解集合の推定下端を計算する。
 $mode = 'U'$ または $'u'$ の場合、ルーチン $cmpt$ 番目の座標での解集合の推定上端を計算する。

$epsilon$ REAL ($sigepps$ の場合)。
 DOUBLE PRECISION ($digepps$ の場合)。
 推定の指示精度。

$nits$ INTEGER。割り当てた PPS アルゴリズムの反復回数 ($nits \geq 0$)。

出力パラメーター

$epsilon$ REAL ($sigepps$ の場合)。
 DOUBLE PRECISION ($digepps$ の場合)。
 $cmpt$ 番目の座標軸に沿う解集合の推定値。 $mode = 'L'$ の場合、 $estm$ は解集合の推定下端を表す。 $mode = 'U'$ の場合、 $estm$ は解集合の推定上端を表す。

$epsilon$ 推定の実際の精度。

$nits$ 実際に実行した PPS アルゴリズムの反復回数。

$info$ INTEGER。 $info = 0$ の場合、正常に終了したことを示す。
 $info = i > 0$ の場合、実行は正常に終了しなかったことを示す。
 $info = -i$ の場合、 i 番目のパラメーターの値が不正だったことを示す。

?gepss

連立区間線形方程式を解分割法によって解く。

構文

```
call sigepps(trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits, info)
call digepss(trans, n, a, lda, b, ldb, cmpt, mode, estm, epsilon, nits, info)
```

説明

?gepss ルーチンは、解分割法 (PSS) を使用して、以下の連立区間線形方程式に対する解集合の精密な外部の成分に関する推定を指示座標に沿って計算する。

$AX = B$ ($trans = 'N'$ の場合)

$A^T X = B$ ($trans = 'T'$ または $'C'$ の場合)

入力パラメーター

$trans$ CHARACTER(1)。 $'N'$ 、 $'T'$ 、 $'C'$ 、 $'n'$ 、 $'t'$ 、または $'c'$ のいずれかでなければならない。
 連立方程式の形式を指定する。

$trans = 'N'$ または $'n'$ の場合、 $\mathbf{Ax} = \mathbf{B}$ を \mathbf{x} について解く。
 $trans = 'T'$ 、 $'C'$ 、 $'t'$ 、 $'c'$ のいずれかの場合、 $\mathbf{A}^T \mathbf{x} = \mathbf{B}$ を \mathbf{x} について解く。

<i>n</i>	INTEGER。 \mathbf{A} の次数。 \mathbf{B} の行の数 ($n \geq 0$)。
<i>a, b</i>	REAL (sigepss の場合) DOUBLE PRECISION (digepps の場合) 配列: $a(lda, *)$, $b ldb$ 。 配列 a には、行列 \mathbf{A} を格納する。 配列 b には、解を求める連立方程式の右辺のベクトル \mathbf{B} を格納する。
<i>lda</i>	INTEGER。 a の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 b の第 1 次元。 $ldb \geq \max(1, n)$ 。
<i>cmpt</i>	INTEGER。 推定する解集合の成分の数。
<i>mode</i>	CHARACTER(1)。 $'L'$ または $'U'$ (または対応する小文字) でなければならない。 パラメーター <i>cmpt</i> で指定された座標に沿う解集合の推定方法を示す。 $mode = 'L'$ または $'l'$ の場合、ルーチンは <i>cmpt</i> 番目の座標での解集合の推定下端を計算する。 $mode = 'U'$ または $'u'$ の場合、ルーチン <i>cmpt</i> 番目の座標での解集合の推定上端を計算する。
<i>epsilon</i>	REAL (sigepss の場合) DOUBLE PRECISION (digepps の場合) 解集合に対する推定の指示精度。
<i>nits</i>	INTEGER。 割り当てた PSS アルゴリズムの反復回数 ($nits \geq 0$)。

出力パラメーター

<i>estm</i>	REAL (sigepss の場合) DOUBLE PRECISION (digepps の場合) <i>cmpt</i> 番目の座標軸に沿う解集合の推定値。 $mode = 'L'$ の場合、 <i>estm</i> は解集合の推定下端を表す。 $mode = 'U'$ の場合、 <i>estm</i> は解集合の推定上端を表す。
<i>epsilon</i>	推定 <i>estm</i> の実際の精度。
<i>nits</i>	実際に実行した PPS アルゴリズムの反復回数。
<i>info</i>	INTEGER。 $info = 0$ の場合、正常に終了したことを示す。 $info = i > 0$ の場合、実行は正常に終了しなかったことを示す。 $info = -i$ の場合、 i 番目のパラメーターの値が不正だったことを示す。ルーチンに対応するメッセージを出力する。

アプリケーション・ノート

?gepps ルーチンおよび ?gepps ルーチンは、区間線形方程式のパラメーター集合または解集合の適合分割を使用して、方程式の解集合の最適な推定を計算する 2 つの相互に二元的なアルゴリズムを実装する。どの方法を使用するかは、区間パラメーターの数や解集合の形状など、問題の解の特性に基づくべきである。

例えば、区間パラメーターの数が少ない場合は PPS 法を使用し、解集合の形状が単純な区間方程式では PSS 法を使用するとよい。

指示した精度内に収まっていることが保証されるエンクロージャを求めることと同様に、連立区間線形方程式に対する解集合の最適な（あるいは精密な）エンクロージャを計算することは、NP 困難な問題として知られている。したがって、?gepps ルーチンおよび ?geps ルーチンを効率的に使用して目的の結果を得るには、パラメーター *epsilon* と *nits* を適切に選択することが極めて重要である。

この事情を念頭に置き、全体のソリューション・プロセスを対話式に構成することが推奨される。すなわち、?gepps または ?geps の呼び出しを、適度な *nits* と粗い *epsilon* で始め、?gepps または ?geps が実行を完了するまで、*nits* を増加、*epsilon* を減少して繰り返す。

例 12-2 パラメーター分割法 (PPS) の Fortran 90 コード

第 1 座標軸に沿って、(例えば、 $1.E-4$ 以内の精度で) 連立区間線形方程式に対する解集合の精密な推定下限を計算するサンプル問題を考える。

$$\begin{pmatrix} 3.5 & [0,2] & [0,2] \\ [0,2] & 3.5 & [0,2] \\ [0,2] & [0,2] & 3.5 \end{pmatrix} x = \begin{pmatrix} [-1,1] \\ [-1,1] \\ [-1,1] \end{pmatrix}$$

この問題は、パラメーター分割法 (PPS 法) を実装し、*siggepps* ルーチンを使用する以下の Fortran コードで解くことができる。

```

.....
USE INTERVAL_ARITHMETIC
.....
INTEGER, PARAMETER :: LDA = 3, LDB = 3
INTEGER              :: NITS, CMPT, INFO, I, J
CHARACTER(1)        :: MODE = 'L'
REAL(4)             :: EPS, ESTM
TYPE(S_INTERVAL)    :: A(3,3), B(3)
.....
DO I = 1, 3
  DO J = 1, 3
    IF( I/=J ) THEN
      A(I,J) = SINTERVAL(0.,2.)
    ELSE
      A(I,J) = SINTERVAL(3.5)
    END IF
    B(I) = SINTERVAL(-1.,1.)
  END DO
END DO

```

```

CMPT = 1
NITS = 100
EPS= 1.E-4
.....
CALL SIGEPPS( 'n', 3, A, LDA, B, LDB, CMPT, MODE, ESTM, EPS, NITS, INFO )

```

アルゴリズムが確実に完了するように、パラメーター NITS の値は 100 (回) とした。上記の例では、この値で十分である。行列 **A** と右辺ベクトル **B** の成分への単精度区間の割り当ては、INTERVAL_ARITHMETIC モジュールで供給される SINTERVAL 関数によって実行される点に注意すること。

区間行列逆転用のルーチン

?trtri

三角区間行列の
逆区間行列を計算する。

構文

```

call sitrtri(uplo, diag, n, a, lda, info)
call ditrtri(uplo, diag, n, a, lda, info)
call crtrtri(uplo, diag, n, a, lda, info)
call zrtrtri(uplo, diag, n, a, lda, info)
call cctrtri(uplo, diag, n, a, lda, info)
call zctrtri(uplo, diag, n, a, lda, info)

```

説明

?trtri ルーチンは、区間三角行列 **A** の逆行列 \mathbf{A}^{-1} の区間エンクロージャを計算する。

このルーチンは、後方置換アルゴリズムを実装し、元の行列の単純な構造により、逆区間行列の最適なエンクロージャを生成する。

入力パラメーター

<i>uplo</i>	CHARACTER(1)。'U'、'L'、'u'、または 'l' のいずれかでなければならない。 A が上三角行列か、下三角行列かを指定する。 <i>uplo</i> = 'U' または 'u' の場合、 A は上三角行列である。 <i>uplo</i> = 'L' または 'l' の場合、 A は下三角行列である。
<i>diag</i>	CHARACTER(1)。'N'、'U'、'n'、または 'u' のいずれかでなければならない。 <i>diag</i> = 'N' または 'n' の場合、 A は単位三角行列ではない。 <i>diag</i> = 'U' または 'u' の場合、 A は単位三角行列である。 A の対角成分は 1 とみなされ、配列 <i>a</i> 内で参照されない。

n INTEGER。行列 **A** の次数 ($n \geq 0$)。

a S_INTERVAL (*sitrtri* の場合)
D_INTERVAL (*ditrtri* の場合)
CR_INTERVAL (*crtrtri* の場合)
ZR_INTERVAL (*zrtrtri* の場合)
CC_INTERVAL (*cctrtri* の場合)
ZC_INTERVAL (*zctrtri* の場合)

配列、次元は (*lda*, *)。
行列 **A** を格納する。
a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

lda INTEGER。 *a* の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

a 逆行列 \mathbf{A}^{-1} のエンクロージャとなる $n \times n$ の区間行列によって上書きされる。

info INTEGER。
info = 0 の場合、実行は正常に終了したことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正だったことを示す。
info = *i* の場合、**A** の *i* 番目の対角成分がゼロで、**A** が特異になるため、行列の反転を完了できなかったことを示す。

?geszi

逆区間行列を
Schulz 反復法によって計算する。

構文

```
call sigeszi(n, a, lda, info)
call digeszi(n, a, lda, info)
```

説明

一般の区間正方行列 **A** に対して、?geszi ルーチンは、Schulz 反復法により逆区間行列 \mathbf{A}^{-1} のエンクロージャを計算する。このルーチンを使用したコードの例は、本書付録 C の「[区間連立線形方程式ソルバーのコード例](#)」を参照のこと。

入力パラメーター

n INTEGER。行列 **A** の次数 ($n \geq 0$)。

a REAL (*sigeszi* の場合)。
DOUBLE PRECISION (*digeszi* の場合)。
配列、次元は (*lda*, *)。
行列 **A** を格納する。
a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

lda INTEGER。 *a* の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

a 逆区間行列 A^{-1} のエンクロージャによって上書きされる。

info INTEGER。 *info* = 0 の場合、正常に終了したことを示す。
info = *i* > 0 の場合、実行は正常に終了しなかったことを示す。
info = -*i* の場合、*i* 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

?geszi ルーチンで使用されている Schulz 反復は区間行列 A が「広すぎる」ことがない条件でのみ収束する。逆に、Schulz 反復が発散するとき、結果は [-*infty*, +*infty*] 要素の区間行列に等しいとされる。ここで、*infty* は、コンピューター上で対応する数値表現での最大値である。

区間行列の特性確認用のルーチン

?gerbr

区間行列の正則性を *Ris-Beeck* と *Rex-Rohn* 判定基準によってテストする。

構文

```
call sigerbr(n, a, lda, sr, reg, info)
call digerbr(n, a, lda, sr, reg, info)
```

説明

?gerbr ルーチンは、*Ris-Beeck* のスペクトル判定基準と *Rex-Rohn* テストを組み合わせ使用して、一般区間正方行列 A が正則か特異かを確認する。

入力パラメーター

n INTEGER。 行列 A の次数 ($n \geq 0$)。

a REAL (*sigerbr* の場合)。
 DOUBLE PRECISION (*digerbr* の場合)。
 配列、次元は (*lda*, *)。
 行列 A を格納する。
a の第 2 次元は、 $\max(1, n)$ 以上でなければならない。

lda INTEGER。 *a* の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>sr</i>	REAL (sigerbr の場合)。 DOUBLE PRECISION (digerbr の場合)。 行列 $(\text{mid } \mathbf{A})^{-1} \cdot \text{rad } \mathbf{A}$ のスペクトル半径の推定上限。 これは行列 \mathbf{A} についての付加情報であるが、 \mathbf{A} のいわゆる強固な正則性にとって非常に重要なものである。
<i>reg</i>	INTEGER。特異性テストの結果を示す。 <i>reg</i> > 0 の場合、 \mathbf{A} は正則である。 <i>reg</i> < 0 の場合、 \mathbf{A} は特異である。 <i>reg</i> = 0 の場合、結果は不定である。すなわち、テストは行列 \mathbf{A} が正則か特異かを検出できるほど感度が高くなかった。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

?gerbr ルーチンに実装されているテストは大まかなものであり、境界上の場合には行列 \mathbf{A} をさらに調査することが推奨される。しかし、このルーチンは (*sr* の値を 1 と比較することで) その区間行列が強固に正則かどうかを判定するのに役立つ。

?gesvr

区間行列の正則性 / 特異性を *Rump* と *Rex-Rohn* 特異値判定基準によってテストする。

構文

```
call sigesvr(n, a, lda, msr, rsr, reg, info)
call digesvr(n, a, lda, msr, rsr, reg, info)
```

説明

?gesvr ルーチンは、*Rump* と *Rex-Rohn* 特異値判定基準を使用して、一般区間正方行列 \mathbf{A} が正則か特異かを確認する。

入力パラメーター

<i>n</i>	INTEGER。行列 \mathbf{A} の次数 ($n \geq 0$)。
<i>a</i>	REAL (sigesvr の場合)。 DOUBLE PRECISION (digesvr の場合)。 配列、次元は (<i>lda</i> , *)。 行列 \mathbf{A} を格納する。 <i>a</i> の第 2 次元は、 $\max(1, n)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。

出力パラメーター

<i>msr, rsr</i>	S_INTERVAL (sigesvr の場合)。 D_INTERVAL (digesvr の場合)。 行列 A に関するその他の情報。 これらの区間はそれぞれ中点行列と半径行列の特異スペクトルの範囲を示す。
<i>reg</i>	INTEGER。特異性テストの結果を示す。 <i>reg</i> > 0 の場合、 A は正則である。 <i>reg</i> < 0 の場合、 A は特異である。 <i>reg</i> = 0 の場合、結果は不定である。すなわち、テストは行列 A が正則か特異かを検出できるほど感度が高くなかった。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> = <i>i</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

?gesvr ルーチンは、行列が正則か特異かに関する十分条件のみをテストするものである。これは、いくつかの境界上では、行列が正則か特異かを検出できるほどテストの感度が高くないことを意味する。この場合、ルーチンは *reg* = 0 を返す。

例 12-3 特異値判定基準による区間行列の正則性をテストする Fortran 90 のコード

区間行列

$$\begin{pmatrix} [2,4] & [-1,2] \\ [-2,1] & [2,4] \end{pmatrix}$$

の正則性を特異値判定基準によってテストするには、以下の Fortran 90 コードの断片が参考になる。

```

.....
USE INTERVAL_ARITHMETIC
.....
INTEGER, PARAMETER :: LDA = 2, N = 2
TYPE(D_INTERVAL)   :: A(LDA,N), MSR, RSR
INTEGER             :: REG, INFO
.....
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,4.)
.....
CALL DIGESVR( N, A, LDA, MSR, RSR, REG, INFO )
.....

```

実軸上の区間 MSR と RSR の相互関係は、正則性にどの程度の余裕があるか (MSR > RSR の場合)、あるいは行列が正則なものからどの程度かけ離れているか (MSR < RSR の場合) の基準として、ある程度まで使用できる。

補助およびユーティリティー・ルーチン

?gemip

区間線形方程式の midpoint 逆転
 プリコンディショニングを実行する。

構文

```

call sigemip(n, nrhs, a, lda, b, ldb, info)
call digemip(n, nrhs, a, lda, b, ldb, info)
call crgemip(n, nrhs, a, lda, b, ldb, info)
call zrgemip(n, nrhs, a, lda, b, ldb, info)
call ccgemip(n, nrhs, a, lda, b, ldb, info)
call zcgemip(n, nrhs, a, lda, b, ldb, info)

```

説明

?gemip ルーチンは、区間線形方程式 $\mathbf{Ax} = \mathbf{B}$ の中点逆転プリコンディショニングを実行する。この処理は、 \mathbf{A} と \mathbf{B} の両行列に、(丸め付き) 区間演算の中点逆行列 $(\text{mid } \mathbf{A})^{-1}$ を乗算して行われる。

入力パラメーター

<i>n</i>	INTEGER。行列 \mathbf{A} の次数。
<i>nrhs</i>	INTEGER。右辺の数 ($nrhs \geq 0$)。
<i>a, b</i>	S_INTERVAL (sigemip の場合) D_INTERVAL (digemip の場合) CR_INTERVAL (crgemip の場合) ZR_INTERVAL (zrgemip の場合) CC_INTERVAL (ccgemip の場合) ZC_INTERVAL (zcgemip の場合) 配列: <i>a</i> (<i>lda</i> , *), <i>b</i> (<i>ldb</i> , *)。 配列 <i>a</i> には、行列 \mathbf{A} を格納する。 配列 <i>b</i> には、行列 \mathbf{B} が格納される。この行列の列は、連立方程式の右辺である。 <i>a</i> の第 2 次元は $\max(1, n)$ 以上でなければならない。 <i>b</i> の第 2 次元は $\max(1, nrhs)$ 以上でなければならない。
<i>lda</i>	INTEGER。 <i>a</i> の第 1 次元。 $lda \geq \max(1, n)$ 。
<i>ldb</i>	INTEGER。 <i>b</i> の第 1 次元。 $ldb \geq \max(1, n)$ 。

出力パラメーター

<i>a</i>	プリコンディショニング済の行列 \mathbf{A} によって上書きされる。
<i>b</i>	プリコンディショニング済の行列 \mathbf{B} によって上書きされる。
<i>info</i>	INTEGER。 <i>info</i> = 0 の場合、正常に終了したことを示す。 <i>info</i> > 0 の場合、実行は正常に終了しなかったことを示す。 <i>info</i> = - <i>i</i> の場合、 <i>i</i> 番目のパラメーターの値が不正だったことを示す。

アプリケーション・ノート

プリコンディショニングによって、区間線形方程式を解くアルゴリズムの適用範囲が拡張したり、そのアルゴリズムで生成される結果の質が向上する可能性がある。

特に、区間ガウス法、区間 Householder 法、および区間 Gauss-Seidel 反復法を、対角の比重が大きくない「広い」行列を持つ区間線形方程式に適用する場合は、より良い結果を得るため、その前にプリコンディショニングをしておくべきである。

Hansen-Blick-Rohn プロシージャについては、方程式の中点行列が対角からかけ離れている場合、中点逆転プリコンディショニングを実施することが推奨される。

例 12-4 区間線形方程式のプリコンディショニング用の Fortran 90 コード

以下の Fortran コードは、区間線形方程式

$$\begin{pmatrix} [2, 4] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix} x = \begin{pmatrix} [0, 2] & [-2, 2] \\ [0, 2] & [-2, 2] \end{pmatrix}$$

に対してプリコンディショニングを実施した上で区間ガウス法で解く方法を示している。

```

.....
USE INTERVAL_ARITHMETIC
.....
TYPE(D_INTERVAL)      :: A(2,2), B(2,2)
INTEGER                :: N = 2, NRHS = 2, LDA = 2, LDB = 2, INFO
CHARACTER(1)          :: TRANS = 'n'
.....
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,4.)
B(1,1) = DINTERVAL(0.,2.); B(2,1) = DINTERVAL(0.,2.)
B(1,2) = DINTERVAL(-2.,2.); B(2,2) = DINTERVAL(-2.,2.)
.....
CALL DIGEMIP( N, NRHS, A, LDA, B, LDB, INFO )
CALL DIGEGAS( TRANS, N, NRHS, A, LDA, B, LDB, INFO )

```

このルーチンを使用したほかのコード例は、本書付録 C の「[区間連立線形方程式ソルバーのコード例](#)」を参照のこと。

偏微分方程式のサポート

13

インテル[®] マス・カーネル・ライブラリー (インテル[®] MKL) に、偏微分方程式 (PDE) を解くツールが追加された。三角変換インターフェイス・ルーチン ([「三角変換ルーチン」](#)を参照) およびポアソン・ライブラリー ([「ポアソン・ライブラリー・ルーチン」](#)を参照) である。

ポアソン・ライブラリーは、単純なヘルムホルツ、ポアソン、およびラプラス問題を素早く解くように設計されている。ソルバーの基本となる三角変換インターフェイスは、インテル MKL DFT インターフェイス ([「フーリエ変換関数」](#)を参照) に基づき、インテル[®] プロセッサ向けに最適化されている。

三角変換ルーチンを直接使用することは、ポアソン・ライブラリーが提供するソルバーに似た独自のソルバーをすでに実装しているユーザーに役立つ。ポアソン・ライブラリーに合わせてオリジナルコードを修正することは困難であるため、三角変換インターフェイスに実装されている高速な正弦、余弦、およびスタaggered 余弦変換を使用して、ソルバーのパフォーマンスを向上させることを推奨する。

インターフェイスの記述には C の表記が使用されているが、三角変換とポアソン・ライブラリー・ルーチンはどちらも、C と Fortran-90 から呼び出すことができる。Fortran-90 からの呼び出し方法については、[「Fortran-90 からの PDE サポートルーチンの呼び出し」](#) セクションを参照のこと。

三角変換ルーチン

第 11 章で説明している離散フーリエ変換 (DFT) インターフェイスに加えて、インテル MKL は、TT インターフェイスと呼ばれる実離散三角変換インターフェイスをサポートしている。インターフェイスは、正弦、余弦、およびスタaggered 余弦変換を計算するために使用するルーチンのグループ (TT ルーチン) を実装している。TT インターフェイスは、柔軟性が高い。ルーチンのパラメーターを手動でチューニングして特定のニーズを満たすようにルーチンを調整することもできるし、デフォルト・パラメーター値でルーチンを呼び出すこともできる。TT インターフェイスの現在のインテル MKL 実装は、偏微分方程式を解くために使用でき、高速ポアソンおよび類似ソルバー用の有用なルーチンを含んでいる。

ここでは、C の表記を使用してインテル MKL TT インターフェイスを説明する。Fortran ユーザーは、この章の [「Fortran-90 からの PDE サポートルーチンの呼び出し」](#) セクションを参照のこと。

以下のセクションでは、TT インターフェイスに現在実装され、この章で説明されている三角関数の一覧を示す。

実装されている変換

TT ルーチンは、以下の変換を計算できる。

- 順方向正弦変換

$$F(k) = \frac{2}{n} \sum_{i=1}^{n-1} f(i) \sin \frac{ki\pi}{n}, k = 1, \dots, n-1$$

- 逆方向正弦変換

$$f(i) = \sum_{k=1}^{n-1} F(k) \sin \frac{ki\pi}{n}, i = 1, \dots, n-1$$

- 順方向余弦変換

$$F(k) = \frac{1}{n} \left[f(0) + \cos \frac{k\pi}{n} f(n) \right] + \frac{2}{n} \sum_{i=1}^{n-1} f(i) \cos \frac{ki\pi}{n}, k = 0, \dots, n$$

- 逆方向余弦変換

$$f(i) = \frac{1}{2} \left[F(0) + \cos \frac{i\pi}{n} F(n) \right] + \sum_{k=1}^{n-1} F(k) \cos \frac{ki\pi}{n}, i = 0, \dots, n$$

- 順方向スタガード余弦変換

$$F(k) = \frac{1}{n} f(0) + \frac{2}{n} \sum_{i=1}^{n-1} f(i) \cos \frac{(2k+1)i\pi}{2n}, k = 0, \dots, n-1$$

- 逆方向スタガード余弦変換

$$f(i) = \frac{1}{2} F(0) + \sum_{k=1}^{n-1} F(k) \cos \frac{(2k+1)i\pi}{2n}, i = 0, \dots, n-1.$$



注: 変換のサイズ n は偶数でなければならない。三角変換の現在の実装では、奇数サイズの変換をサポートしていない。

TT ルーチン起動の順序

TT インターフェイスを使用した変換の計算は、概念的に4つのステップに分けられる。各ステップは、専用ルーチンによって実行される。[表 13-1](#) に、ルーチンの名前、各ルーチンの目的と用途を簡単に説明する。

ほとんどの TT ルーチンには、単精度データ用と倍精度データ用のバージョンがある。そのようなルーチンの名前は、"s" および "d" でそれぞれ始まる。ルーチン名のワイルドカード "?" は、これらのシンボルのいずれかを表す。

表 13-1 TT インターフェイス・ルーチン

ルーチン	説明
? init trig transform	三角変換の基本的なデータ構造を初期化する。
? commit trig transform	ユーザー定義データの一貫性や正確性を検証し、インテル MKL DFT インターフェイス ¹ で使用されるデータ構造を作成する。
? forward trig transform ? backward trig transform	適切な式を使用して指定された型の順方向 / 逆方向三角変換を計算する (「 実装されている変換 」を参照)。
free trig transform	DFT インターフェイス ¹ の呼び出しで使用するデータ構造に割り当てられているメモリーをクリーンする。

1. TT ルーチンは、より高いパフォーマンスが得られるように、インテル MKL DFT インターフェイスを呼び出す。

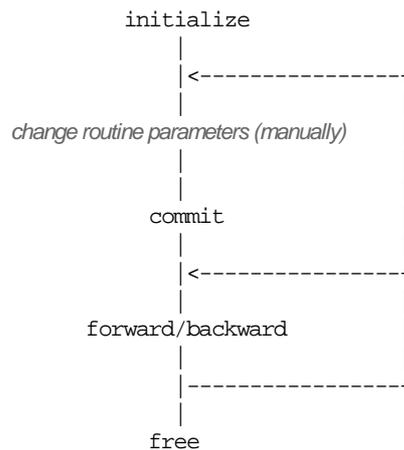
特定の入力ベクトルの変換されたベクトルを求めるために、インテル MKL TT インターフェイス・ルーチンは通常、[表 13-1](#) にリストされている順序で起動される。



注: TT ルーチンを起動する順序は変更できるが、次の図にリストされているルーチン呼び出しの順序に従うことを強く推奨する。

[図 13-1](#) のダイアグラムは、典型的なケースにおける TT インターフェイス・ルーチンの一般的な起動順を示す (ルーチン名のプリフィックスおよびサフィックスは省略されている)。

図 13-1 TT インターフェイス・ルーチン起動の一般的な順序



倍精度計算で TT ルーチンを使用する一般的なスキームを次に示す。ルーチン名の最初の文字を変更することで、同様のスキームを単精度計算にも適用できる。

```

...
    d_init_trig_transform(&n, &tt_type, ipar, dpar, &ir);
/* Change parameters in ipar if necessary. */
/* Note that the result of the Transform will be in f ! If you want to
preserve the data stored in f, save them before this place in your code */
    d_commit_trig_transform(f, &handle, ipar, dpar, &ir);
    d_forward_trig_transform(f, &handle, ipar, dpar, &ir);
    d_backward_trig_transform(f, &handle, ipar, dpar, &ir);
    free_trig_transform(&handle, ipar, &ir);
/* here the user may clean the memory used by f, dpar, ipar */
...

```

TT インターフェイス・ルーチンを使用して 1 次元のヘルムホルツ問題を解く Fortran-90 と C コードの例が付録 C の「[三角変換のコード例](#)」セクションに示されている。

インターフェイスの説明

本書のすべての型は、標準 C 型の INT、FLOAT、および DOUBLE である。Fortran-90 ユーザーは、それぞれ、INTEGER、REAL、および DOUBLE PRECISION Fortran 型を使用してルーチンを呼び出すことができる（付録 C の「[三角変換のコード例](#)」セクションの例を参照）。

ルーチンオプション

すべての TT ルーチンにはパラメーターがあり、さまざまなオプションをルーチンに渡す。これらのパラメーターは、配列 *ipar*、*dpar* および *spar* である。これらのパラメーターの値は、非常に注意して指定する必要がある（「[共通パラメーター](#)」を参照）。これらの値は、必要に応じて計算中に変更できる。



注：計算に失敗したり、間違った結果を取得しないように、正確で一貫性のあるパラメーターをルーチンに渡さなければならない。

ユーザーデータ配列

TT ルーチンは、ユーザーデータ配列を入力として受け取る。例えば、順方向三角変換を計算するには、ユーザー配列を *d_forward_trig_transform* ルーチンに渡す。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL TT ルーチンはユーザー入力配列のコピーを作成しない。



注：入力データ配列のコピーが必要な場合、各自で保存する必要がある。

TT ルーチン

このセクションでは、TT ルーチンの構文、パラメーターおよび戻り値を詳細に説明する。同じルーチンの倍精度バージョンと単精度バージョンと一緒に説明する。

TT ルーチンは、ルーチンの性能を強化するインテル MKL DFT インターフェイスを呼び出す(第 11 章の「[DFT 関数](#)」セクションを参照)。

?_init_trig_transform

三角変換の基本的なデータ構造を初期化する。

構文

```
void d_init_trig_transform(int *n, int *tt_type, int ipar[], double
    dpar[], int *stat);
void s_init_trig_transform(int *n, int *tt_type, int ipar[], float
    spar[], int *stat);
```

入力パラメーター

n int*. 問題のサイズを格納する。偶数の正の整数でなければならない。他の TT ルーチンが使用する変換のデータベクトルのサイズは $n+1$ でなければならないことに注意する。

tt_type int*. 計算する変換の型を格納する。名前付き定数のセットで定義される。TT インターフェイスの現在の実装では、次の定数を利用できる。MKL_SINE_TRANSFORM、MKL_COSINE_TRANSFORM および MKL_STAGGERED_COSINE_TRANSFORM。

出力パラメーター

ipar int。サイズ 128 の配列。三角変換計算に必要な整数データを格納する。

dpar double。サイズ $3n/2+1$ の配列。三角変換計算に必要な倍精度データを格納する。

spar float。サイズ $3n/2+1$ の配列。三角変換計算に必要な単精度データを格納する。

stat int*. ルーチンの終了ステータスを格納する。*ipar*[6] にも書き込まれる。他の TT ルーチンに進むにはステータスは 0 でなければならない。

説明

ルーチンは、適切な精度の三角変換用に基本的なデータ構造を初期化する。`?_init_trig_transform` への呼び出しの後、続いて起動されるすべての TT ルーチンは、`?_init_trig_transform` で返された *ipar* および *dpar* (*spar*) 配列パラメーターの値を使用する。ルーチンは *ipar* 配列全体を初期化する。*dpar* または *spar* 配列で、`?_init_trig_transform` は変換の型に依存しない要素を初期化する。*ipar*、*dpar* および *spar* 配列の詳細な説明は、「[共通パラメーター](#)」セクションを参照のこと。

コードで初期化ルーチンの呼び出しをスキップすることができる。詳細は、「[パラメーター修正に関する警告](#)」を参照。

戻り値

`stat = 0` ルーチンは、作業を通常通り終了した。一般に、計算を続行するには、ルーチンはこの `stat` 値で終了する必要がある。

`stat = -99999` ルーチンは、作業に失敗した。

?_commit_trig_transform

ユーザーのデータの一貫性や正確性を検証し、三角変換を実行するために必要な特定のデータ構造を初期化する。

構文

```
void d_commit_trig_transform (double f[], DFTI_DESCRIPTOR_HANDLE
    *handle, int ipar[], double dpar[], int *stat);
void s_commit_trig_transform (float f[], DFTI_DESCRIPTOR_HANDLE *handle,
    int ipar[], float spar[], int *stat);
```

入力パラメーター

`f` `double` (`d_commit_trig_transform` の場合)、`float` (`s_commit_trig_transform` の場合)。サイズ $n+1$ の配列。ここで n は問題のサイズ。変換されるデータベクトルを格納する。

`ipar` `int`。サイズ 128 の配列。三角変換計算に必要な整数データを格納する。

`dpar` `double`。サイズ $3n/2+1$ の配列。三角変換計算に必要な倍精度データを格納する。ほとんどの配列要素はルーチンによって初期化される。

`spar` `float`。サイズ $3n/2+1$ の配列。三角変換計算に必要な単精度データを格納する。ほとんどの配列要素はルーチンによって初期化される。

出力パラメーター

`handle` `DFTI_DESCRIPTOR_HANDLE*`。インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「[DFT 関数](#)」セクションを参照)。

`ipar` 三角変換計算に必要な整数データを格納する。出力時に、`ipar[6]` は `stat` 値で更新される。

`dpar` 三角変換計算に必要な倍精度データを格納する。出力時に、配列全体が初期化される。

`spar` 三角変換計算に必要な単精度データを格納する。出力時に、配列全体が初期化される。

stat int*. ルーチンの終了ステータスを格納する。*ipar*[6]にも書き込まれる。

説明

?_commit_trig_transform ルーチンは、変換ルーチン [?_forward_trig_transform](#) および [?_backward_trig_transform](#) に渡されるパラメーターの一貫性と正確性を検証する。ルーチンは、次のデータ構造も初期化する。

handle, *dpar* (*d_commit_trig_transform* の場合) および *spar* (*s_commit_trig_transform* の場合)。

?_commit_trig_transform ルーチンは、[?_init_trig_transform](#) ルーチンで定義され、*ipar* 配列とともに ?_commit_trig_transform に渡された変換の型に応じて *dpar* または *spar* の要素のみを初期化する。配列パラメーターのサイズを決定する問題のサイズ *n* は、*ipar* 配列とともにルーチンに渡され、以前に呼び出した

?_init_trig_transform ルーチンでも定義される。*ipar*, *dpar* および *spar* 配列の詳細な説明は、「[共通パラメーター](#)」セクションを参照のこと。このルーチンは、パラメーターの正確性と一貫性について基本的な検証のみを行う。TT ルーチンのパラメーターを修正する場合は、「[パラメーター修正に関する警告](#)」セクションを参照のこと。?_init_trig_transform とは異なり、コードでこのルーチンの呼び出しをスキップすることはできない。

戻り値

<i>stat</i> =11	ルーチンは、警告を出力し、正確性と一貫性のためにパラメーターに変更を加えた。パラメーターが正しいことがわかっているならば、 <i>ipar</i> [6]=0 を割り当てて計算を続行してよい。
<i>stat</i> =10	ルーチンは、正確性と一貫性のためにパラメーターに変更を加えた。パラメーターが正しいことがわかっているならば、 <i>ipar</i> [6]=0 を割り当てて計算を続行してよい。
<i>stat</i> =1	ルーチンは、警告を出力した。パラメーターが正しいことがわかっているならば、 <i>ipar</i> [6]=0 を割り当てて計算を続行してよい。
<i>stat</i> =0	ルーチンは、作業を通常通り終了した。
<i>stat</i> =-100	ルーチンは、次のいずれかの理由により停止した。 <ul style="list-style-type: none"> • ユーザーデータでエラーが発生した。 • 修正の結果、<i>ipar</i>, <i>dpar</i>, または <i>spar</i> パラメーターのデータが正しくなくなった。または不整合になった。
<i>stat</i> =-1000	ルーチンは、DFT インターフェイス・エラーにより停止した。
<i>stat</i> =-10000	ルーチンは、初期化に失敗した。またはパラメーター <i>ipar</i> [0] が誤って変更されたために停止した。



注: 正の値の *stat* は通常、入力データのマイナーな問題を示している。この場合、三角変換計算は続行できるが、最初に問題を調査して *stat*=0 にすることを強く推奨する。

?_forward_trig_transform

パラメーターによって指定された型の順方向三角変換を計算する。

構文

```
void d_forward_trig_transform(double f[], DFTI_DESCRIPTOR_HANDLE
    *handle, int ipar[], double dpar[], int *stat);
void s_forward_trig_transform(float f[], DFTI_DESCRIPTOR_HANDLE *handle,
    int ipar[], float spar[], int *stat);
```

入力パラメーター

<i>f</i>	double (d_forward_trig_transform の場合)、float (s_forward_trig_transform の場合)。サイズ $n+1$ の配列。ここで n は問題のサイズ。変換されるデータベクトルを格納する。
<i>handle</i>	DFTI_DESCRIPTOR_HANDLE*。インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「 DFT 関数 」セクションを参照)。
<i>ipar</i>	int。サイズ 128 の配列。三角変換計算に必要な整数データを格納する。
<i>dpar</i>	double。サイズ $3n/2+1$ の配列。三角変換計算に必要な倍精度データを格納する。
<i>spar</i>	float。サイズ $3n/2+1$ の配列。三角変換計算に必要な単精度データを格納する。

出力パラメーター

<i>f</i>	変換されたベクトルを格納する。
<i>ipar</i>	三角変換計算に必要な整数データを格納する。出力時に、 <i>ipar</i> [6] は <i>stat</i> 値で更新される。
<i>stat</i>	int*。ルーチンの終了ステータスを格納する。 <i>ipar</i> [6] にも書き込まれる。

説明

ルーチンは、[?_init_trig_transform](#) ルーチンで定義され、[?_forward_trig_transform](#) に *ipar* 配列とともに渡された型の順方向三角変換を計算する。配列パラメーターのサイズを決定する問題のサイズ n は、*ipar* 配列とともにルーチンに渡され、以前に呼び出した [?_init_trig_transform](#) ルーチンでも定義される。計算を容易にする他のデータは、[?_commit_trig_transform](#) によって作成され、*dpar* または *spar* で提供される。*ipar*、*dpar* および *spar* 配列の詳細な説明は、「[共通パラメーター](#)」セクションを参照のこと。ルーチンには、[?_commit_trig_transform](#) ルーチンを呼び出す *commit* ステップがある。変換は、「[実装されている変換](#)」セクションの式に従って計算される。ルーチンは、入力ベクトル *f* を変換されたベクトルと置換する。



注: 変換するデータベクトル f のコピーが必要な場合は、`?_forward_trig_transform` ルーチンを呼び出す前にコピーを作成する必要がある。

戻り値

<code>stat=0</code>	ルーチンは、作業を通常通り終了した。
<code>stat=-100</code>	ルーチンは、次のいずれかの理由により停止した。 <ul style="list-style-type: none"> • ユーザーデータでエラーが発生した。 • 修正の結果、<code>ipar</code>、<code>dpar</code>、または <code>spar</code> パラメーターのデータが正しくなくなった。または不整合になった。
<code>stat=-1000</code>	ルーチンは、DFT インターフェイス・エラーにより停止した。
<code>stat=-10000</code>	ルーチンは、 <code>commit</code> ステップに失敗した。またはパラメーター <code>ipar[0]</code> が誤って変更されたために停止した。

?_backward_trig_transform

パラメーターによって指定された型の逆方向三角変換を計算する。

構文

```
void d_backward_trig_transform(double f[], DFTI_DESCRIPTOR_HANDLE
    *handle, int ipar[], double dpar[], int *stat);
void s_backward_trig_transform(float f[], DFTI_DESCRIPTOR_HANDLE
    *handle, int ipar[], float spar[], int *stat);
```

入力パラメーター

<code>f</code>	<code>double</code> (<code>d_backward_trig_transform</code> の場合)、 <code>float</code> (<code>s_backward_trig_transform</code> の場合)。 サイズ $n+1$ の配列。ここで n は問題のサイズ。変換されるデータベクトルを格納する。
<code>handle</code>	<code>DFTI_DESCRIPTOR_HANDLE*</code> 。インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「 DFT 関数 」セクションを参照)。
<code>ipar</code>	<code>int</code> 。サイズ 128 の配列。三角変換計算に必要な整数データを格納する。
<code>dpar</code>	<code>double</code> 。サイズ $3n/2+1$ の配列。三角変換計算に必要な倍精度データを格納する。
<code>spar</code>	<code>float</code> 。サイズ $3n/2+1$ の配列。三角変換計算に必要な単精度データを格納する。

出力パラメーター

<i>f</i>	変換されたベクトルを格納する。
<i>ipar</i>	三角変換計算に必要な整数データを格納する。出力時に、 <i>ipar</i> [6] は <i>stat</i> 値で更新される。
<i>stat</i>	int*。ルーチンの終了ステータスを格納する。 <i>ipar</i> [6] にも書き込まれる。

説明

ルーチンは、[?_init_trig_transform](#) ルーチンで定義され、[?_backward_trig_transform](#) に *ipar* 配列とともに渡された型の逆方向三角変換を計算する。配列パラメーターのサイズを決定する問題のサイズ *n* は、*ipar* 配列とともにルーチンに渡され、以前に呼び出した [?_init_trig_transform](#) ルーチンでも定義される。計算を容易にする他のデータは、[?_commit_trig_transform](#) によって作成され、*dpar* または *spar* で提供される。*ipar*、*dpar* および *spar* 配列の詳細な説明は、「[共通パラメーター](#)」セクションを参照のこと。ルーチンには、[?_commit_trig_transform](#) ルーチンを呼び出す *commit* ステップがあり、変換は、「[実装されている変換](#)」セクションの式に従って計算される。ルーチンは、入力ベクトル *f* を変換されたベクトルと置換する。



注: 変換するデータベクトル *f* のコピーが必要な場合は、[?_backward_trig_transform](#) ルーチンを呼び出す前にコピーを作成する必要がある。

戻り値

<i>stat</i> =0	ルーチンは、作業を通常通り終了した。
<i>stat</i> =-100	ルーチンは、次のいずれかの理由により停止した。 <ul style="list-style-type: none"> • ユーザーデータでエラーが発生した。 • 修正の結果、<i>ipar</i>、<i>dpar</i>、または <i>spar</i> パラメーターのデータが正しくなくなった。または不整合になった。
<i>stat</i> =-1000	ルーチンは、DFT インターフェイス・エラーにより停止した。
<i>stat</i> =-10000	ルーチンは、 <i>commit</i> ステップに失敗した。またはパラメーター <i>ipar</i> [0] が誤って変更されたために停止した。

free_trig_transform

DFT インターフェイスで使用するデータ構造に割り当てられていたメモリーをクリーンする。

構文

```
void free_trig_transform(DFTI_DESCRIPTOR_HANDLE *handle, int ipar[],
                        int *stat);
```

入力パラメーター

ipar int。サイズ 128 の配列。三角変換計算に必要な整数データを格納する (詳細は、「[共通パラメーター](#)」を参照)。

handle DFTI_DESCRIPTOR_HANDLE*。インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「[DFT 関数](#)」セクションを参照)。

出力パラメーター

handle 構造に割り当てられていたメモリーは出力時にリリースされる。

ipar 三角変換計算に必要な整数データを格納する。出力時に、*ipar*[6] は *stat* 値で更新される。

stat int*。ルーチンの終了ステータスを格納する。*ipar*[6] にも書き込まれる。

説明

ルーチンは、インテル MKL DFT 関数に必要な *handle* 構造で使用されたメモリーをクリーンする。他のパラメーター用に割り当てられていたメモリーをリリースする場合は、コード中にメモリーのクリーンを含める必要がある。

戻り値

stat=0 ルーチンは、作業を通常通り終了した。

stat=-1000 ルーチンは、DFT インターフェイス・エラーにより停止した。

stat=-99999 ルーチンは、作業に失敗した。

共通パラメーター

このセクションでは、TT ルーチンのオプションを格納する配列パラメーター *ipar*、*dpar*、および *spar* について説明する。



注: 初期値は、[?_init_trig_transform](#) および [?_commit_trig_transform](#) ルーチンによって配列パラメーターに割り当てられる。

ipar int。サイズ 128 の配列。三角変換計算に必要な整数データを格納する。要素は、[表 13-2](#) で説明する。

表 13-2 ipar 配列の要素

インデックス	説明
0	解く問題のサイズを格納する。 ? init trig transform ルーチンは $ipar[0]=n$ に設定し、続いて呼び出されるすべての TT ルーチンは、変換のサイズとして $ipar[0]$ を使用する。TT インターフェイスの現在の実装では、偶数サイズの変換のみをサポートしている。
1	エラー・メッセージ・オプションを格納する。 <ul style="list-style-type: none"> $ipar[1]=-1$ は、ルーチンが呼び出されたフォルダーの <code>MKL_Trig_Transforms_log.txt</code> ファイルにすべてのエラーメッセージが出力されることを示す。ファイルが存在しない場合、ルーチンはファイルを作成する。ファイルの作成に失敗した場合、ルーチンは、標準出力デバイスにファイルが作成できないという情報を出力する。 $ipar[1]=0$ は、エラーメッセージが出力されないことを示す。 $ipar[1]=1$ は、デフォルト値である。これは、あらかじめ接続されているデフォルトの出力デバイス (通常、スクリーン) にすべてのエラーメッセージが出力されることを示す。 エラーが発生した場合、TT ルーチンは $ipar[1]$ の設定に関係なく、非ゼロの値を <i>stat</i> に割り当てる。
2	警告メッセージオプションを格納する。 <ul style="list-style-type: none"> $ipar[2]=-1$ は、ルーチンが呼び出されたフォルダーの <code>MKL_Trig_Transforms_log.txt</code> ファイルにすべての警告メッセージが出力されることを示す。ファイルが存在しない場合、ルーチンはファイルを作成する。ファイルの作成に失敗した場合、ルーチンは、標準出力デバイスにファイルが作成できないという情報を出力する。 $ipar[2]=0$ は、警告メッセージが出力されないことを示す。 $ipar[2]=1$ は、デフォルト値である。これは、あらかじめ接続されているデフォルトの出力デバイス (通常、スクリーン) にすべての警告メッセージが出力されることを示す。 警告が出力された場合、 <i>stat</i> パラメーターは $ipar[2]$ 設定に関係なく、非ゼロの値となる。
3 ~ 4	この値は将来のために予約されている。
5	変換の型を格納する。 ? init trig transform ルーチンは $ipar[5]=tt_type$ に設定し、続いて呼び出されるすべての TT ルーチンは、変換の型として $ipar[5]$ を使用する。
6	最後に終了した TT ルーチンから返された <i>stat</i> 値を格納する。以前の TT ルーチンへの呼び出しが $stat=0$ で終了したかどうかを確認するために使用される。
7	? commit trig transform ルーチンにデータ構造 <i>dpar</i> (<i>spar</i>) および <i>handle</i> を初期化するかどうかを知らせる。 $ipar[7]=0$ は、ルーチンが初期化をスキップし、パラメーターの正確性と一貫性の検証のみを行うことを示す。それ以外の場合、ルーチンはデータ構造を初期化する。デフォルト値は 1。 データ構造 <i>dpar</i> 、 <i>spar</i> および <i>handle</i> を初期化しないで入力データの正確性と一貫性を検証すると、異なるデータベクトルに同じ変換を繰り返して使用することによるパフォーマンスの低下を防ぐことができる。 $ipar[7]$ による利点は、 <i>dpar</i> または <i>spar</i> 配列に正しい許容値を格納した場合にのみ得られることに注意する。それ以外の場合、このパラメーターは利用しないこと。

表 13-2 ipar 配列の要素 (続き)

インデックス	説明
8	TT ルーチンのメッセージ・スタイル・オプションを格納する。 <i>ipar[8]=0</i> の場合、TT ルーチンは Fortran スタイルの表記ですべてのエラーおよび警告メッセージを出力する。それ以外の場合、TT ルーチンは C スタイルの表記でメッセージを出力する。デフォルト値は 1。これらの表記を選択する場合、デフォルトで、配列の要素が C では 0 から、Fortran では 1 から番号付けされることに注意する。例えば、C スタイルのメッセージの一部が "parameter ipar[0]=3 should be an even integer" の場合、対応する Fortran スタイルのメッセージは "parameter ipar(1)=3 should be an even integer" になる。ユーザーは、 <i>ipar[8]</i> を使用することで、より便利なスタイルでメッセージを表示できる。
9	ポアソン・ライブラリーの OpenMP 環境で TT ルーチンを実行する OpenMP のスレッド数を指定する。デフォルト値は 1。この値を変更しないことを強く推奨する。「 パラメーター修正に関する警告 」も参照のこと。
10 ~ 127	この値は将来のために予約されている。



注: 現在、*ipar* 配列は、コードで `int ipar[10]` として宣言できる。しかし、より多くの *ipar* 値を使用するインテル MKL TT インターフェイスの将来のバージョンとの互換性のため、*ipar* を `int ipar[128]` として宣言することを強く推奨する。

配列 *dpar* および *spar* は類似しており、データ精度のみ異なる。

<i>dpar</i>	double。サイズ $3n/2+1$ の配列。TT 計算を行うために倍精度ルーチンで必要なデータを格納する。この配列は、 <code>d_init_trig_transform</code> および <code>d_commit_trig_transform</code> ルーチンで初期化される。
<i>spar</i>	float。サイズ $3n/2+1$ の配列。TT 計算を行うために単精度ルーチンで必要なデータを格納する。この配列は、 <code>s_init_trig_transform</code> および <code>s_commit_trig_transform</code> ルーチンで初期化される。

dpar および *spar* には、それぞれの位置に同様の要素がある。これらの要素は、[表 13-3](#) で説明する。

表 13-3 dpar および spar 配列の要素

インデックス	説明
0	この要素は、適切な <code>?_commit_trig_transform</code> ルーチンで使用される最初の絶対許容値を格納する。スタガード余弦または正弦変換の場合、 $f[n]$ は 0.0 でなければならない。正弦変換の場合、 $f[0]$ は 0.0 でなければならない。 <code>?_commit_trig_transform</code> ルーチンは、ルーチンの精度に応じて、これらのパラメーターの絶対値が $dpar[0]*n$ または $spar[0]*n$ 以下かどうかを確認する。 <i>dpar[0]</i> または <i>spar[0]</i> に大きな数を設定することで、許容値の確認結果で警告の表示を抑制することができる。
1	この要素は将来のために予約されている。

表 13-3 dpar および spar 配列の要素 (続き)

インデックス	説明
2 ~ 3n/2	<p>要素には、三角関数のタビュレート値が格納される。要素の内容は、?_commit_trig_transform ルーチンで設定された変換の型 <i>tt_type</i> に依存する。</p> <ul style="list-style-type: none"> • <i>tt_type</i>=MKL_SINE_TRANSFORM の場合、配列の 3 番目 (インデックス 2) の要素から $n/2$ 個の連続した配列要素に、タビュレート正弦値を含む $n/2$ 個の要素が格納される。配列の残りの部分はこの変換では使用されない。 • <i>tt_type</i>=MKL_COSINE_TRANSFORM の場合、配列の 3 番目 (インデックス 2) の要素から n 個の連続した配列要素に、タビュレート余弦値を含む n 個の要素が格納される。配列の残りの部分はこの変換では使用されない。 • <i>tt_type</i>=MKL_STAGGERED_COSINE_TRANSFORM の場合、配列の 3 番目 (インデックス 2) の要素から $3n/2-2$ 個の連続した配列要素に、タビュレート正弦値と余弦値を含む $3n/2-2$ の要素が格納される。配列の残りの部分はこの変換では使用されない。



注：変換の型に応じて配列サイズを定義できる。

パラメーター修正に関する警告

TT インターフェイスは柔軟であるため、[?_init_trig_transform](#) ルーチンの呼び出しをスキップしてユーザーのコード中で明示的に基本的なデータ構造を初期化することができる。初期化の後、*ipar*、*dpar* および *spar* 配列の内容も修正する必要がある。この際、正確で一貫したデータを配列で使用する必要がある。配列を誤って変更すると、エラーが発生したり、間違った計算が行われる。[?_commit_trig_transform](#) ルーチンを呼び出すことにより、パラメーターの正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、変換の結果が正しいことを保証するものではなく、エラーや間違った結果になる可能性を減らすだけであることに注意する。



注：正確で一貫したパラメーターを TT ルーチンに提供するには、TT インターフェイスを何度も利用し、*ipar*、*spar* および *dpar* 配列に格納されている要素と、これらの要素の値の依存性についてよく理解している必要がある。

しかし、経験を積んだユーザーでも、パラメーターの修正後、TT ルーチンを使用した変換の計算に失敗することがある。



警告: 三角変換の計算を正しく行う唯一の方法は、一般的なルーチン起動の順序に従い、パラメーターのデフォルトセットを変更しないことである。このため、特別に必要な場合を除いて、*ipar*、*dpar* および *spar* 配列を修正しないことを推奨する。

実装の詳細

インテル MKL TT インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル MKL TT の言語固有のヘッダーファイルが提供されている。現在、以下のヘッダーファイルが利用できる。

- `mk1_trig_transforms.h` は C プログラム用で、`mk1_dfti.h` とともに使用する。
- `mk1_trig_transforms.f90` は Fortran-90 プログラム用で、`mk1_dfti.f90` とともに使用する。



注: 上記のヘッダーファイルのいずれかをインクルードしないと、インテル MKL TT ソフトウェアを使用することはできない。

C 固有のヘッダーファイル

C 固有のヘッダーファイルは、以下の関数プロトタイプを定義する。

```
void d_init_trig_transform(int *, int *, int *, double *, int *);
void d_commit_trig_transform(double *, DFTI_DESCRIPTOR_HANDLE *, int *,
double *, int *);
void d_forward_trig_transform(double *, DFTI_DESCRIPTOR_HANDLE *, int *,
double *, int *);
void d_backward_trig_transform(double *, DFTI_DESCRIPTOR_HANDLE *, int
*, double *, int *);

void s_init_trig_transform(int *, int *, int *, float *, int *);
void s_commit_trig_transform(float *, DFTI_DESCRIPTOR_HANDLE *, int *,
float *, int *);
void s_forward_trig_transform(float *, DFTI_DESCRIPTOR_HANDLE *, int *,
float *, int *);
void s_backward_trig_transform(float *, DFTI_DESCRIPTOR_HANDLE *, int *,
float *, int *);

void free_trig_transform(DFTI_DESCRIPTOR_HANDLE *, int *, int *);
```

Fortran 固有のヘッダーファイル

Fortran-90 固有のヘッダーファイルは、以下の関数プロトタイプを定義する。

```
SUBROUTINE D_INIT_TRIG_TRANSFORM(n, tt_type, ipar, dpar, stat)
```

```
INTEGER, INTENT(IN) :: n, tt_type
INTEGER, INTENT(INOUT) :: ipar(*)
REAL(8), INTENT(INOUT) :: dpar(*)
INTEGER, INTENT(OUT) :: stat
END SUBROUTINE D_INIT_TRIG_TRANSFORM
```

```
SUBROUTINE D_COMMIT_TRIG_TRANSFORM(f, handle, ipar, dpar, stat)
  REAL(8), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(8), INTENT(INOUT) :: dpar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE D_COMMIT_TRIG_TRANSFORM

SUBROUTINE D_FORWARD_TRIG_TRANSFORM(f, handle, ipar, dpar, stat)
  REAL(8), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(8), INTENT(INOUT) :: dpar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE D_FORWARD_TRIG_TRANSFORM

SUBROUTINE D_BACKWARD_TRIG_TRANSFORM(f, handle, ipar, dpar, stat)
  REAL(8), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(8), INTENT(INOUT) :: dpar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE D_BACKWARD_TRIG_TRANSFORM

SUBROUTINE S_INIT_TRIG_TRANSFORM(n, tt_type, ipar, spar, stat)
  INTEGER, INTENT(IN) :: n, tt_type
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(4), INTENT(INOUT) :: spar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE S_INIT_TRIG_TRANSFORM

SUBROUTINE S_COMMIT_TRIG_TRANSFORM(f, handle, ipar, spar, stat)
  REAL(4), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(4), INTENT(INOUT) :: spar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE S_COMMIT_TRIG_TRANSFORM
```

```
SUBROUTINE S_FORWARD_TRIG_TRANSFORM(f, handle, ipar, spar, stat)
  REAL(4), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(4), INTENT(INOUT) :: spar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE S_FORWARD_TRIG_TRANSFORM
```

```
SUBROUTINE S_BACKWARD_TRIG_TRANSFORM(f, handle, ipar, spar, stat)
  REAL(4), INTENT(INOUT) :: f(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(INOUT) :: ipar(*)
  REAL(4), INTENT(INOUT) :: spar(*)
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE S_BACKWARD_TRIG_TRANSFORM
```

```
SUBROUTINE FREE_TRIG_TRANSFORM(handle, ipar, stat)
  INTEGER, INTENT(INOUT) :: ipar(*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: handle
  INTEGER, INTENT(OUT) :: stat
END SUBROUTINE FREE_TRIG_TRANSFORM
```

TT ルーチンの Fortran-90 固有の使用法は、すべてのインテル MKL PDE サポートツールと同様であり、この章の「[Fortran-90 からの PDE サポートルーチンの呼び出し](#)」セクションで説明されている。

ポアソン・ライブラリー・ルーチン

実離散三角変換 (TT) インターフェイス (「[三角変換ルーチン](#)」を参照)に加えて、インテル MKL は、ポアソン・ライブラリー・インターフェイス (PL インターフェイス) をサポートしている。インターフェイスは、離散フーリエ変換を使用して特別な種類のラプラス、ポアソン、およびヘルムホルツ問題の解を計算するために使用するルーチンのグループ (PL ルーチン) を実装している。ラプラスおよびポアソン問題は、より一般的なヘルムホルツ問題の特別なケースである。解かれる問題は、「[実装されているポアソン・ライブラリー](#)」サブセクションでより詳細に定義されている。PL インターフェイスは、柔軟性が高い。ルーチンのパラメーターを手動でチューニングして特定のニーズを満たすようにルーチンを調整することもできるし、デフォルト・パラメーター値でルーチンを呼び出すこともできる。インターフェイスは、専用のパラメーターを設定して、C または Fortran 表記用にエラーと警告メッセージのスタイルを調整することができる。ユーザーが自然な方法でコードの情報を読むことができるため、デバッグが容易になる。現在のインテル MKL で実装されている PL インターフェイスには、デカルト座標系で高速ラプラス、ポアソンおよびヘルムホルツ・ソルバーを実装するルーチンのみ含まれている。

ここでは、C の表記を使用してインテル MKL PL インターフェイスを説明する。Fortran の使用法の詳細は、この章の「[Fortran-90 からの PDE サポートルーチンの呼び出し](#)」セクションを参照のこと。



注: Fortran ユーザーは、Fortran では配列インデックスがそれぞれ 1 異なることに注意する。

実装されているポアソン・ライブラリー

PL ルーチンは、特定の 2 次元および 3 次元問題の近似解を求めることができる。

2 次元問題

表記の規則

PL インターフェイスの記述では、矩形領域 $a_x < x < b_x$ 、 $a_y < y < b_y$ の境界に以下の表記を使用する。

$$bd_a_x = \{x=a_x, a_y \leq y \leq b_y\}, \quad bd_b_x = \{x=b_x, a_y \leq y \leq b_y\}$$

$$bd_a_y = \{a_x \leq x \leq b_x, y=a_y\}, \quad bd_b_y = \{a_x \leq x \leq b_x, y=b_y\}$$

ワイルドカード "+" は、シンボル a_x 、 b_x 、 a_y 、 b_y のいずれかを表す。つまり、 bd_+ は上記境界のいずれかを表す。

2 次元 (2D) ヘルムホルツ問題

2D ヘルムホルツ問題は、次のヘルムホルツ方程式の近似解を求める。

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + qu = f(x, y), \quad q = const \geq 0$$

矩形領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ で各境界 bd_+ 上で以下の境界条件のいずれかを持つ矩形で求める。

- ディリクレ境界条件

$$u(x, y) = G(x, y)$$

- ノイマン境界条件

$$\frac{\partial u}{\partial n}(x, y) = g(x, y)$$

ここで、

$n = -x$ (bd_{a_x} の場合)、 $n = x$ (bd_{b_x} の場合)、

$n = -y$ (bd_{a_y} の場合)、 $n = y$ (bd_{b_y} の場合)。

2次元 (2D) ポアソン問題

ポアソン問題は、ヘルムホルツ問題の特別なケース ($q=0$) である。2D ポアソン問題は、次のポアソン方程式の近似解を求める。

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

矩形領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ で各境界 bd_+ 上でディリクレ境界条件またはノイマン境界条件を持つ矩形で求める。

2次元 (2D) ラプラス問題

ラプラス問題は、ヘルムホルツ問題の特別なケース ($q=0$ および $f(x, y)=0$) である。

2D ラプラス問題は、次のラプラス方程式の近似解を求める。

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

矩形領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ で各境界 bd_+ 上でディリクレ境界条件またはノイマン境界条件を持つ矩形で求める。

2D 問題の近似

任意の 2D 問題の近似解を求めるため、一様メッシュが矩形領域に構築される。

$$\{x_i = a_x + ih_x, y_j = a_y + jh_y\}, i = 0, \dots, n_x, j = 0, \dots, n_y, h_x = \frac{b_x - a_x}{n_x}, h_y = \frac{b_y - a_y}{n_y}$$

ポアソン・ライブラリーは、このメッシュ上で標準の 5 点差分近似を使用して解の近似を計算する。メッシュ点 (x_i, y_j) の右辺 $f(x, y)$ の値と矩形領域の境界上にあるメッシュ点の適切な境界関数 $G(x, y)$ または $g(x, y)$ の値が分かれば、近似解の値はこれらのメッシュ点で計算される。



注: x 方向のメッシュ区間の値 n_x は偶数でなければならない。ポアソン・ライブラリーの現在の実装では、奇数の区間のメッシュはサポートしていない。

3 次元問題

表記の規則

PL インターフェイスの記述では、平行六面体領域 $a_x < x < b_x$ 、 $a_y < y < b_y$ 、 $a_z < z < b_z$ の境界に以下の表記を使用する。

$$bd_a_x = \{x=a_x, a_y \leq y \leq b_y, a_z \leq z \leq b_z\}, bd_b_x = \{x=b_x, a_y \leq y \leq b_y, a_z \leq z \leq b_z\}$$

$$bd_a_y = \{a_x \leq x \leq b_x, y=a_y, a_z \leq z \leq b_z\}, bd_b_y = \{a_x \leq x \leq b_x, y=b_y, a_z \leq z \leq b_z\}$$

$$bd_a_z = \{a_x \leq x \leq b_x, a_y \leq y \leq b_y, z=a_z\}, bd_b_z = \{a_x \leq x \leq b_x, a_y \leq y \leq b_y, z=b_z\}$$

ワイルドカード "+" は、シンボル a_x 、 b_x 、 a_y 、 b_y 、 a_z 、 b_z のいずれかを表す。つまり、 bd_+ は上記境界のいずれかを表す。

3 次元 (3D) ヘルムホルツ問題

3D ヘルムホルツ問題は、次のヘルムホルツ方程式の近似解を求める。

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} + qu = f(x, y, z), \quad q = \text{const} \geq 0$$

平行六面体領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ 、 $a_z < z < b_z$ で各境界 bd_+ 上で以下の境界条件のいずれかを持つ平行六面体で求める。

- ディリクレ境界条件
 $u(x, y, z) = G(x, y, z)$
- ノイマン境界条件

$$\frac{\partial u}{\partial n}(x, y, z) = g(x, y, z)$$

ここで、
 $n = -x$ (bd_a_x の場合)、 $n = x$ (bd_b_x の場合)、
 $n = -y$ (bd_a_y の場合)、 $n = y$ (bd_b_y の場合)、
 $n = -z$ (bd_a_z の場合)、 $n = z$ (bd_b_z の場合)。

3 次元 (3D) ポアソン問題

ポアソン問題は、ヘルムホルツ問題の特別なケース ($q=0$) である。3D ポアソン問題は、次のポアソン方程式の近似解を求める。

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} - \frac{\partial^2 u}{\partial z^2} = f(x, y, z)$$

平行六面体領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ 、 $a_z < z < b_z$ で各境界 bd_+ 上でディリクレ境界条件またはノイマン境界条件を持つ平行六面体で求める。

3次元 (3D) ラプラス問題

ラプラス問題は、ヘルムホルツ問題の特別なケース ($q=0$ および $f(x, y, z)=0$) である。3D ラプラス問題は、次のラプラス方程式の近似解を求める。

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0$$

平行六面体領域が $a_x < x < b_x$ 、 $a_y < y < b_y$ 、 $a_z < z < b_z$ で各境界 bd_+ 上でディリクレ境界条件またはノイマン境界条件を持つ平行六面体で求める。

3D 問題の近似

任意の 3D 問題の近似解を求めるため、一様メッシュが平行六面体領域に構築される。

$$\{x_i = a_x + ih_x, y_j = a_y + jh_y, z_k = a_z + kh_z\}$$

ここで、

$$i = 0, \dots, n_x, j = 0, \dots, n_y, k = 0, \dots, n_z$$

$$h_x = \frac{b_x - a_x}{n_x}, h_y = \frac{b_y - a_y}{n_y}, h_z = \frac{b_z - a_z}{n_z}$$

ポアソン・ライブラリーは、このメッシュ上で標準の 7 点差分近似を使用して解の近似を計算する。メッシュ点 (x_i, y_j, z_k) の右辺 $f(x, y, z)$ の値と平行六面体領域の境界上にあるメッシュ点の適切な境界関数 $G(x, y, z)$ または $g(x, y, z)$ の値が分かれば、近似解の値はこれらのメッシュ点で計算される。



注: x および y 方向のメッシュ区間の値 n_x と n_y は偶数でなければならない。ポアソン・ライブラリーの現在の実装では、奇数の区間のメッシュはサポートしていない。

PL ルーチン起動の順序



注: この後の説明では、高速ポアソンソルバーと高速ラプラスソルバーは高速ヘルムホルツ・ソルバーの特別なケースであるため、ヘルムホルツ問題の解のプロセスを常に考慮する ([「実装されているポアソン・ライブラリー」](#)を参照)。

PL インターフェイスを使用したヘルムホルツ問題の解の計算は、専用ルーチンによって実行される 4 つのステップに分けられる。[表 13-4](#) に、ルーチンの名前と、各ルーチンの目的の簡単な説明を示す。

ほとんどの PL ルーチンには、単精度データ用と倍精度データ用のバージョンがある。そのようなルーチンの名前は、"s" および "d" でそれぞれ始まる。ルーチン名のワイルドカード "?" は、これらのシンボルのいずれかを表す。各 PL ルーチンには、2D バージョンと 3D バージョンがある。それらの名前には、最後にそれぞれ "2D" および "3D" がつく。

表 13-4 PL インターフェイス・ルーチン

ルーチン	説明
? init Helmholtz 2D/? init Helmholtz 3D	高速 2D/3D ヘルムホルツ・ソルバーのポアソン・ライブラリーの基本的なデータ構造を初期化する。
? commit Helmholtz 2D/? commit Helmholtz 3D	ユーザーのデータの一貫性と正確性を検証し、インテル MKL DFT インターフェイスで使用するデータ構造とソルバーに必要な他のデータ構造を作成して初期化する。
? Helmholtz 2D/? Helmholtz 3D	パラメーターによって指定された 2D/3D ヘルムホルツ問題 (「実装されているポアソン・ライブラリー」 を参照) の近似解を計算する。
free Helmholtz 2D/free Helmholtz 3D	DFT インターフェイスの呼び出しで使用するデータ構造に割り当てられていたメモリーをクリーンする。

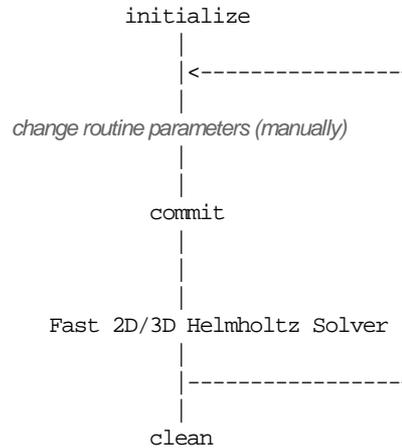
ヘルムホルツ問題の近似解特定を求めるために、インテル MKL PL インターフェイス・ルーチンは通常、[表 13-4](#) にリストされている順序に起動される。



注: PL ルーチンを起動する順序は変更できるが、次の図にリストされているルーチン呼び出しの順序に従うことを強く推奨する。

[図 13-2](#) のダイアグラムは、典型的なケースの PL ルーチンの一般的な起動順を示す。

図 13-2 PL インターフェイス・ルーチン起動の一般的な順序



3D の倍精度計算で PL ルーチンを使用する一般的なスキームを下記に示す。ルーチン名の最初の文字を変更することで、同様のスキームを単精度計算にも適用できる。2D の場合の一般的なスキームは、以下のルーチン・パラメーターのセットとは異なり、ルーチン名の最後が "3D" の代わりに "2D" になる。

```

...
d_init_Helmholtz_3D(&ax, &bx, &ay, &by, &az, &bz, &nx, &ny, &nz, BCtype,
ipar, dpar, &stat);
/* change parameters in ipar if necessary. */
/* note that the result of the Fast Helmholtz Solver will be in f! If you want to
keep the data stored in f, save it before the function call below */
d_commit_Helmholtz_3D(f, bd_ax, bd_bx, bd_ay, bd_by, bd_az, bd_bz,
&xhandle, &yhandle, ipar, dpar, &stat)
d_Helmholtz_3D(f, bd_ax, bd_bx, bd_ay, bd_by, bd_az, bd_bz, &xhandle,
&yhandle, ipar, dpar, &stat);
free_Helmholtz_3D (&xhandle, &yhandle, ipar, &stat);
/* here the user may clean the memory used by f, dpar, ipar */
...
  
```

PL ルーチンを使用してヘルムホルツ問題を解く Fortran-90 と C コードの例が付録 C の「[ポアソン・ライブラリーのコード例](#)」セクションに示されている。

インターフェイスの説明

本書のすべての型は、標準 C 型の INT、FLOAT、および DOUBLE である。Fortran-90 ユーザーは、それぞれ、INTEGER、REAL、および DOUBLE PRECISION Fortran 型を使用してルーチンを呼び出すことができる（付録 C の「[ポアソン・ライブラリーのコード例](#)」セクションの例を参照）。

ルーチンオプション

すべての PL ルーチンにはパラメーターがあり、さまざまなオプションをルーチンに渡す。これらのパラメーターは、配列 *ipar*、*dpar* および *spar* である。これらのパラメーターの値は、非常に注意して指定する必要がある（「[共通パラメーター](#)」を参照）。これらの値は、必要に応じて計算中に変更できる。



注：計算に失敗したり、間違った結果を取得しないように、正確で一貫性のあるパラメーターをルーチンに渡さなければならない。

ユーザーデータ配列

PL ルーチンは、ユーザーデータ配列を入力として受け取る。例えば、3D ヘルムホルツ問題の近似解を計算するには、ユーザー配列を `d_Helmholtz_3D` ルーチンに渡す。格納要件を最小限に抑えて、全体のランタイム効率を向上させるため、インテル MKL PL ルーチンはユーザー入力配列のコピーを作成しない。



注：入力データ配列のコピーが必要な場合、各自で保存する必要がある。

PL ルーチン

このセクションでは、PL ルーチンの構文、パラメーターおよび戻り値を詳細に説明する。同じルーチンの倍精度バージョンと単精度バージョン、2D バージョンと 3D バージョンと一緒に説明する。



注：ルーチン・パラメーターの一部は、3D 高速ヘルムホルツ・ソルバーでのみ使用される。

PL ルーチンは、ルーチンの性能を強化するインテル MKL DFT インターフェイスを呼び出す（第 11 章の「[DFT 関数](#)」セクションを参照）。

?_init_Helmholtz_2D/?_init_Helmholtz_3D

高速 2D/3D ヘルムホルツ・ソルバーの基本的なデータ構造を初期化する。

構文

```
void d_init_Helmholtz_2D(double* ax, double* bx, double* ay, double* by,
    int* nx, int* ny, char* BCtype, double* q, int* ipar, double* dpar,
    int* stat);

void s_init_Helmholtz_2D(float* ax, float* bx, float* ay, float* by, int*
    nx, int* ny, char* BCtype, float* q, int* ipar, float* spar, int*
    stat);

void d_init_Helmholtz_3D(double* ax, double* bx, double* ay, double* by,
    double* az, double* bz, int* nx, int* ny, int* nz, char* BCtype,
    double* q, int* ipar, double* dpar, int* stat);

void s_init_Helmholtz_3D(float* ax, float* bx, float* ay, float* by,
    float* az, float* bz, int* nx, int* ny, int* nz, char* BCtype, float*
    q, int* ipar, float* spar, int* stat);
```

入力パラメーター

<i>ax</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) x 軸に沿った領域の左端境界の座標。
<i>bx</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) x 軸に沿った領域の右端境界の座標。
<i>ay</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) y 軸に沿った領域の左端境界の座標。
<i>by</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) y 軸に沿った領域の右端境界の座標。
<i>az</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) z 軸に沿った領域の左端境界の座標。このパラメーターは、?_init_Helmholtz_3D ルーチンでのみ必要である。
<i>bz</i>	double*(d_init_Helmholtz_2D/d_init_Helmholtz_3D の場合)、float*(s_init_Helmholtz_2D/s_init_Helmholtz_3D の場合) z 軸に沿った領域の右端境界の座標。このパラメーターは、?_init_Helmholtz_3D ルーチンでのみ必要である。
<i>nx</i>	int*。x 軸に沿ったメッシュ区間の数。偶数でなければならない。
<i>ny</i>	int*。y 軸に沿ったメッシュ区間の数。3D の場合は偶数でなければならない。

<code>nz</code>	<code>int*</code> 。z 軸に沿ったメッシュ区間の数。このパラメーターは、 <code>?_init_Helmholtz_3D</code> ルーチンでのみ必要である。
<code>BCtype</code>	<code>char*</code> 。各境界の境界条件の型を格納する。 <code>?_init_Helmholtz_2D</code> では 4 文字、 <code>?_init_Helmholtz_3D</code> では 6 文字を格納しなければならない。文字はそれぞれ、'N' (ノイマン境界条件) または 'D' (ディリクレ境界条件) になる。境界の境界条件の型は、次の順序で指定しなければならない。 <code>bd_ax</code> 、 <code>bd_bx</code> 、 <code>bd_ay</code> 、 <code>bd_by</code> 、 <code>bd_az</code> 、 <code>bd_bz</code> 。 最後の 2 つの境界の境界条件の型は、3D の場合にのみ指定する。
<code>q</code>	<code>double*</code> (<code>d_init_Helmholtz_2D/d_init_Helmholtz_3D</code> の場合)、 <code>float*</code> (<code>s_init_Helmholtz_2D/s_init_Helmholtz_3D</code> の場合) ヘルムホルツ係数を格納する。ポアソン問題またはラプラス問題を解くには、 <code>q</code> の値を 0 に設定する必要があることに注意する。

出力パラメーター

<code>ipar</code>	<code>int</code> 。サイズ 128 の配列。高速ヘルムホルツ・ソルバーで使用する整数データを格納する (詳細は、「 共通パラメーター 」を参照)。
<code>dpar</code>	<code>double</code> 。サイズ $5 \times nx/2 + 7$ (2D の場合) または $5 \times (nx + ny)/2 + 9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーで使用する倍精度データを格納する (詳細は、「 共通パラメーター 」を参照)。
<code>spar</code>	<code>float</code> 。サイズ $5 \times nx/2 + 7$ (2D の場合) または $5 \times (nx + ny)/2 + 9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーで使用する単精度データを格納する (詳細は、「 共通パラメーター 」を参照)。
<code>stat</code>	<code>int*</code> 。ルーチンの終了ステータスを格納する。 <code>ipar[0]</code> にも書き込まれる。他の PL ルーチンに進むにはステータスは 0 でなければならない。

説明

`?_init_Helmholtz_2D/?_init_Helmholtz_3D` ルーチンは、適切な精度のポアソン・ライブラリー計算用の基本的なデータ構造を初期化するために呼び出される。`?_init_Helmholtz_2D/?_init_Helmholtz_3D` ルーチンへの呼び出しの後に起動されるすべてのルーチンは、ルーチンによって返された `ipar`、`dpar` および `spar` 配列パラメーターの値を使用する。配列パラメーターの詳細は、「[共通パラメーター](#)」セクションに記載されている。



警告: 2D/3D ルーチンで初期化および作成されたデータ構造は、PL ルーチンの 2D/3D ルーチンでは使用できない。

コードでこのルーチンの呼び出しをスキップすることができる。しかし、その前に「[パラメーター修正に関する警告](#)」を参照すること。

戻り値

<code>stat = 0</code>	ルーチンは、作業を正常に終了した。一般に、計算を続行するには、ルーチンはこの <code>stat</code> 値で終了する必要がある。
-----------------------	---

stat=99999

ルーチンは、致命的なエラーのために作業に失敗した。

?_commit_Helmholtz_2D/?_commit_Helmholtz_3D

ユーザーのデータの一貫性や正確性を検証し、2D/3D ヘルムホルツ問題を解くために必要な特定のデータ構造を初期化する。

構文

```
void d_commit_Helmholtz_2D(double* f, double* bd_ax, double* bd_bx,
    double* bd_ay, double* bd_by, DFTI_DESCRIPTOR* xhandle, int* ipar,
    double* dpar, int* stat);

void s_commit_Helmholtz_2D(float* f, float* bd_ax, float* bd_bx, float*
    bd_ay, float* bd_by, DFTI_DESCRIPTOR* xhandle, int* ipar, float* spar,
    int* stat);

void d_commit_Helmholtz_3D(double* f, double* bd_ax, double* bd_bx,
    double* bd_ay, double* bd_by, double* bd_az, double* bd_bz,
    DFTI_DESCRIPTOR* xhandle, DFTI_DESCRIPTOR* yhandle, int* ipar, double*
    dpar, int* stat);

void s_commit_Helmholtz_3D(float* f, float* bd_ax, float* bd_bx, float*
    bd_ay, float* bd_by, float* bd_az, float* bd_bz, DFTI_DESCRIPTOR*
    xhandle, DFTI_DESCRIPTOR* yhandle, int* ipar, float* spar, int*
    stat);
```

入力パラメーター

f double* (d_commit_Helmholtz_2D/d_commit_Helmholtz_3D の場合)、float* (s_commit_Helmholtz_2D/s_commit_Helmholtz_3D の場合)。単一ベクトルに格納された問題の右辺を格納する。2D の場合、ベクトルのサイズは $(nx+1)*(ny+1)$ 。この場合、メッシュ点 (i, j) の右辺の値は $f[i+j*(nx+1)]$ に格納される。3D の場合、ベクトルのサイズは $(nx+1)*(ny+1)*(nz+1)$ 。この場合、メッシュ点 (i, j, k) の右辺の値は $f[i+j*(nx+1)+k*(nx+1)*(ny+1)]$ に格納される。ラプラス問題を解くには、配列 *f* のすべての要素を 0 に設定する必要があることに注意する。配列 *f* は、ルーチンによって変更される場合があることに注意する。配列を保存する場合は、別のメモリーの場所にこのベクトルを保存する。

ipar int。サイズ 128 の配列。高速ヘルムホルツ・ソルバーで使用する整数データを格納する (詳細は、「[共通パラメーター](#)」を参照)。

dpar double。サイズ $5*nx/2+7$ (2D の場合) または $5*(nx+ny)/2+9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーで使用する倍精度データを格納する (詳細は、「[共通パラメーター](#)」を参照)。

<i>spar</i>	float。サイズ $5 * n_x / 2 + 7$ (2D の場合) または $5 * (n_x + n_y) / 2 + 9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーで使用する単精度データを格納する (詳細は、「 共通パラメーター 」を参照)。
<i>bd_ax</i>	double* (d_commit_Helmholtz_2D/d_commit_Helmholtz_3D の場合)、 float* (s_commit_Helmholtz_2D/s_commit_Helmholtz_3D の場合)。 x 軸に沿った領域の左端にある境界の境界条件の値を格納する。 ?_commit_Helmholtz_2D の場合、配列のサイズは $n_y + 1$ 。 ディリクレ境界条件の場合 (BCtype[0] の値が 'D')、関数 $G(ax, y_j)$, $j=0, \dots, n_y$ の値を格納する。ノイマン境界条件の場合 (BCtype[0] の値が 'N')、関数 $g(ax, y_j)$, $j=0, \dots, n_y$ の値を格納する。インデックス j に対応する値は、 $bd_ax[j]$ に格納される。 ?_commit_Helmholtz_3D の場合、配列のサイズは $(n_y + 1) * (n_z + 1)$ 。 ディリクレ境界条件の場合 (BCtype[0] の値が 'D')、関数 $G(ax, y_j, z_k)$, $j=0, \dots, n_y, k=0, \dots, n_z$ の値を格納する。ノイマン境界条件の場合 (BCtype[0] の値が 'N')、関数 $g(ax, y_j, z_k)$, $j=0, \dots, n_y, k=0, \dots, n_z$ の値を格納する。値は配列に格納される。インデックス (j, k) に対応する値は、 $bd_ax[j+k*(n_y+1)]$ に格納される。
<i>bd_bx</i>	double* (d_commit_Helmholtz_2D/d_commit_Helmholtz_3D の場合)、 float* (s_commit_Helmholtz_2D/s_commit_Helmholtz_3D の場合)。 x 軸に沿った領域の右端にある境界の境界条件の値を格納する。 ?_commit_Helmholtz_2D の場合、配列のサイズは $n_y + 1$ 。 ディリクレ境界条件の場合 (BCtype[1] の値が 'D')、関数 $G(bx, y_j)$, $j=0, \dots, n_y$ の値を格納する。ノイマン境界条件の場合 (BCtype[1] の値が 'N')、関数 $g(bx, y_j)$, $j=0, \dots, n_y$ の値を格納する。インデックス j に対応する値は、 $bd_bx[j]$ に格納される。 ?_commit_Helmholtz_3D の場合、配列のサイズは $(n_y + 1) * (n_z + 1)$ 。 ディリクレ境界条件の場合 (BCtype[1] の値が 'D')、関数 $G(bx, y_j, z_k)$, $j=0, \dots, n_y, k=0, \dots, n_z$ の値を格納する。ノイマン境界条件の場合 (BCtype[1] の値が 'N')、関数 $g(bx, y_j, z_k)$, $j=0, \dots, n_y, k=0, \dots, n_z$ の値を格納する。値は配列に格納される。インデックス (j, k) に対応する値は、 $bd_bx[j+k*(n_y+1)]$ に格納される。
<i>bd_ay</i>	double* (d_commit_Helmholtz_2D/d_commit_Helmholtz_3D の場合)、 float* (s_commit_Helmholtz_2D/s_commit_Helmholtz_3D の場合)。 y 軸に沿った領域の左端にある境界の境界条件の値を格納する。 ?_commit_Helmholtz_2D の場合、配列のサイズは $n_x + 1$ 。 ディリクレ境界条件の場合 (BCtype[2] の値が 'D')、関数 $G(x_i, ay)$, $i=0, \dots, n_x$ の値を格納する。ノイマン境界条件の場合 (BCtype[2] の値が 'N')、関数 $g(x_i, ay)$, $i=0, \dots, n_x$ の値を格納する。インデックス i に対応する値は、 $bd_ay[i]$ に格納される。

?_commit_Helmholtz_3D の場合、配列のサイズは $(nx+1)*(nz+1)$ 。ディリクレ境界条件の場合 (BCtype[2] の値が 'D')、関数 $G(x_i, ay, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合 (BCtype[2] の値が 'N')、関数 $g(x_i, ay, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。値は配列に格納される。インデックス (i, k) に対応する値は、 $bd_ay[i+k*(nx+1)]$ に格納される。

bd_by double* (d_commit_Helmholtz_2D/d_commit_Helmholtz_3D の場合)、
float* (s_commit_Helmholtz_2D/s_commit_Helmholtz_3D の場合)。
y 軸に沿った領域の右端にある境界の境界条件の値を格納する。

?_commit_Helmholtz_2D の場合、配列のサイズは $nx+1$ 。ディリクレ境界条件の場合 (BCtype[3] の値が 'D')、関数 $G(x_i, by)$, $i=0, \dots, nx$ の値を格納する。ノイマン境界条件の場合 (BCtype[3] の値が 'N')、関数 $g(x_i, by)$, $i=0, \dots, nx$ の値を格納する。インデックス i に対応する値は、 $bd_by[i]$ に格納される。

?_commit_Helmholtz_3D の場合、配列のサイズは $(nx+1)*(nz+1)$ 。ディリクレ境界条件の場合 (BCtype[3] の値が 'D')、関数 $G(x_i, by, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合 (BCtype[3] の値が 'N')、関数 $g(x_i, by, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。値は配列に格納される。インデックス (i, k) に対応する値は、 $bd_by[i+k*(nx+1)]$ に格納される。

bd_az double* (d_commit_Helmholtz_3D の場合)、
float* (s_commit_Helmholtz_3D の場合)。
このパラメーターは、?_commit_Helmholtz_3D ルーチンでのみ必要である。z 軸に沿った領域の左端にある境界の境界条件の値を格納する。

配列のサイズは $(nx+1)*(ny+1)$ 。ディリクレ境界条件の場合 (BCtype[4] の値が 'D')、関数 $G(x_i, y_j, az)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。ディリクレ境界条件の場合 (BCtype[4] の値が 'N')、関数 $g(x_i, y_j, az)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。値は配列に格納される。インデックス (i, j) に対応する値は、 $bd_az[i+j*(nx+1)]$ に格納される。

bd_bz double* (d_commit_Helmholtz_3D の場合)、
float* (s_commit_Helmholtz_3D の場合)。
このパラメーターは、?_commit_Helmholtz_3D ルーチンでのみ必要である。z 軸に沿った領域の右端にある境界の境界条件の値を格納する。

配列のサイズは $(nx+1)*(ny+1)$ 。ディリクレ境界条件の場合 (BCtype[5] の値が 'D')、関数 $G(x_i, y_j, bz)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。ディリクレ境界条件の場合 (BCtype[5] の値が 'N')、関数 $g(x_i, y_j, bz)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。値は配列に格納される。インデックス (i, j) に対応する値は、 $bd_bz[i+j*(nx+1)]$ に格納される。

出力パラメーター

<i>f</i>	問題の右辺のベクトル。出力時に変更される。
<i>ipar</i>	高速ヘルムホルツ・ソルバーで使用する整数データを格納する。「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>dpar</i>	高速ヘルムホルツ・ソルバーで使用する倍精度データを格納する。「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>spar</i>	高速ヘルムホルツ・ソルバーで使用する単精度データを格納する。「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>xhandle</i> , <i>yhandle</i>	DESCRIPTOR_HANDLE*。インテル MKL DFT インターフェイスで使用されるデータ構造(詳細は、第 11 章の「 DFT 関数 」セクションを参照)。yhandle は、?_commit_Helmholtz_3D でのみ使用される。
<i>stat</i>	int*。ルーチンの終了ステータスを格納する。 <i>ipar</i> [0] にも書き込まれる。他の PL ルーチンに進むにはステータスは 0 でなければならない。

説明

?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンは、ソルバールーチン [?_Helmholtz_2D/?_Helmholtz_3D](#) に渡されるパラメーターの一貫性と正確性を検証する。また、データ構造 *xhandle*, *yhandle* と、ルーチンの精度に応じて配列 *ipar* および *dpar*/*spar* を初期化する。?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンが初期化する配列要素と格納される値については、「[共通パラメーター](#)」を参照のこと。このルーチンは、正確性と一貫性について基本的な検証のみを行う。PL ルーチンのパラメーターを修正する場合は、「[パラメーター修正に関する警告](#)」セクションを参照のこと。[?_init_Helmholtz_2D/?_init_Helmholtz_3D](#) とは異なり、コードでこれらのルーチンの呼び出しをスキップすることはできない。*ax*, *bx*, *ay*, *by*, *az*, *bz*, *nx*, *ny*, *nz* および *Bctype* の値は、各ルーチンに *ipar* 配列とともに渡され、適切な [?_init_Helmholtz_2D/?_init_Helmholtz_3D](#) ルーチンへの以前の呼び出しで定義される。

戻り値

<i>stat</i> =1	ルーチンは、エラーなしで終了したが、警告を出力した。
<i>stat</i> =0	ルーチンは、作業を正常に終了した。
<i>stat</i> =-100	ルーチンは、ユーザーのデータにエラーが見つかった。または <i>dpar</i> , <i>spar</i> または <i>ipar</i> 配列のデータが誤って変更されたために停止した。
<i>stat</i> =-1000	ルーチンは、インテル MKL DFT または TT インターフェイス・エラーにより停止した。
<i>stat</i> =-10000	ルーチンは、初期化に失敗した。またはパラメーター <i>ipar</i> [0] が誤って変更されたために停止した。
<i>stat</i> =-99999	ルーチンは、致命的なエラーのために作業に失敗した。

?_Helmholtz_2D/?_Helmholtz_3D

パラメーターによって指定された 2D/3D ヘルムホルツ問題の解を計算する。

構文

```
void d_Helmholtz_2D(double* f, double* bd_ax, double* bd_bx, double*
    bd_ay, double* bd_by, DFTI_DESCRIPTOR* xhandle, int* ipar, double*
    dpar, int* stat);

void s_Helmholtz_2D(float* f, float* bd_ax, float* bd_bx, float* bd_ay,
    float* bd_by, DFTI_DESCRIPTOR* xhandle, int* ipar, float* spar, int*
    stat);

void d_Helmholtz_3D(double* f, double* bd_ax, double* bd_bx, double*
    bd_ay, double* bd_by, double* bd_az, double* bd_bz, DFTI_DESCRIPTOR*
    xhandle, DFTI_DESCRIPTOR* yhandle, int* ipar, double* dpar, int*
    stat);

void s_Helmholtz_3D(float* f, float* bd_ax, float* bd_bx, float* bd_ay,
    float* bd_by, float* bd_az, float* bd_bz, DFTI_DESCRIPTOR* xhandle,
    DFTI_DESCRIPTOR* yhandle, int* ipar, float* spar, int* stat);
```

入力パラメーター

<i>f</i>	double*(d_Helmholtz_2D/d_Helmholtz_3D の場合)、float*(s_Helmholtz_2D/s_Helmholtz_3D の場合)。単一ベクトルに格納され、適切な ?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンで修正された問題の右辺を格納する。オリジナルの右辺ベクトルに置換すると、間違った解が得られることに注意する。2D の場合、ベクトルのサイズは $(nx+1)*(ny+1)$ 。この場合、メッシュ点 (i, j) の右辺の値は $f[i+j*(nx+1)]$ に格納される。3D の場合、ベクトルのサイズは $(nx+1)*(ny+1)*(nz+1)$ 。この場合、メッシュ点 (i, j, k) の右辺の値は $f[i+j*(nx+1)+k*(nx+1)*(ny+1)]$ に格納される。
<i>xhandle</i> , <i>yhandle</i>	DESCRIPTOR_HANDLE*。インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「 DFT 関数 」セクションを参照)。yhandle は、?_Helmholtz_3D でのみ使用される。
<i>ipar</i>	int。サイズ 128 の配列。高速ヘルムホルツ・ソルバーに必要な整数データを格納する (詳細は、「 共通パラメーター 」を参照)。
<i>dpar</i>	double。サイズ $5*nx/2+7$ (2D の場合) または $5*(nx+ny)/2+9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーに必要な倍精度データを格納する (詳細は、「 共通パラメーター 」を参照)。
<i>spar</i>	float。サイズ $5*nx/2+7$ (2D の場合) または $5*(nx+ny)/2+9$ (3D の場合) の配列。高速ヘルムホルツ・ソルバーに必要な単精度データを格納する (詳細は、「 共通パラメーター 」を参照)。

bd_ax `double*` (`d_Helmholtz_2D/d_Helmholtz_3D` の場合),
 `float*` (`s_Helmholtz_2D/s_Helmholtz_3D` の場合).
 x 軸に沿った領域の左端にある境界の境界条件の値を格納する。
 `?_Helmholtz_2D` の場合、配列のサイズは $ny+1$ 。ディリクレ境
 界条件の場合 (`BCtype[0]` の値が 'D')、関数 $G(ax, y_j), j=0, \dots, ny$ の
 値を格納する。ノイマン境界条件の場合 (`BCtype[0]` の値が 'N')、
 関数 $g(ax, y_j), j=0, \dots, ny$ の値を格納する。インデックス j に対応
 する値は、`bd_ax[j]` に格納される。
 `?_Helmholtz_3D` の場合、配列のサイズは $(ny+1)*(nz+1)$ 。ディ
 リクレ境界条件の場合 (`BCtype[0]` の値が 'D')、関数 $G(ax, y_j, z_k),$
 $j=0, \dots, ny, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合
 (`BCtype[0]` の値が 'N')、関数 $g(ax, y_j, z_k), j=0, \dots, ny, k=0, \dots, nz$ の
 値を格納する。値は配列に格納される。インデックス (j, k) に対
 応する値は、`bd_ax[j+k*(ny+1)]` に格納される。

bd_bx `double*` (`d_Helmholtz_2D/d_Helmholtz_3D` の場合),
 `float*` (`s_Helmholtz_2D/s_Helmholtz_3D` の場合).
 x 軸に沿った領域の右端にある境界の境界条件の値を格納する。
 `?_Helmholtz_2D` の場合、配列のサイズは $ny+1$ 。ディリクレ境
 界条件の場合 (`BCtype[1]` の値が 'D')、関数 $G(bx, y_j), j=0, \dots, ny$ の
 値を格納する。ノイマン境界条件の場合 (`BCtype[1]` の値が 'N')、
 関数 $g(bx, y_j), j=0, \dots, ny$ の値を格納する。インデックス j に対応
 する値は、`bd_bx[j]` に格納される。
 `?_Helmholtz_3D` の場合、配列のサイズは $(ny+1)*(nz+1)$ 。ディ
 リクレ境界条件の場合 (`BCtype[1]` の値が 'D')、関数 $G(bx, y_j, z_k),$
 $j=0, \dots, ny, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合
 (`BCtype[1]` の値が 'N')、関数 $g(bx, y_j, z_k), j=0, \dots, ny, k=0, \dots, nz$ の
 値を格納する。値は配列に格納される。インデックス (j, k) に対
 応する値は、`bd_bx[j+k*(ny+1)]` に格納される。

bd_ay `double*` (`d_Helmholtz_2D/d_Helmholtz_3D` の場合),
 `float*` (`s_Helmholtz_2D/s_Helmholtz_3D` の場合).
 y 軸に沿った領域の左端にある境界の境界条件の値を格納する。
 `?_Helmholtz_2D` の場合、配列のサイズは $nx+1$ 。ディリクレ境
 界条件の場合 (`BCtype[2]` の値が 'D')、関数 $G(x_i, ay), i=0, \dots, nx$ の
 値を格納する。ノイマン境界条件の場合 (`BCtype[2]` の値が 'N')、
 関数 $g(x_i, ay), i=0, \dots, nx$ の値を格納する。インデックス i に対応
 する値は、`bd_ay[i]` に格納される。
 `?_Helmholtz_3D` の場合、配列のサイズは $(nx+1)*(nz+1)$ 。ディ
 リクレ境界条件の場合 (`BCtype[2]` の値が 'D')、関数 $G(x_i, ay, z_k),$
 $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合
 (`BCtype[2]` の値が 'N')、関数 $g(x_i, ay, z_k), i=0, \dots, nx, k=0, \dots, nz$ の
 値を格納する。値は配列に格納される。インデックス (i, k) に対
 応する値は、`bd_ay[i+k*(nx+1)]` に格納される。

bd_by `double*` (`d_Helmholtz_2D/d_Helmholtz_3D` の場合),
 `float*` (`s_Helmholtz_2D/s_Helmholtz_3D` の場合).
 y 軸に沿った領域の右端にある境界の境界条件の値を格納する。

?_Helmholtz_2D の場合、配列のサイズは $nx+1$ 。ディリクレ境界条件の場合 (BCtype[3] の値が 'D')、関数 $G(x_i, by)$, $i=0, \dots, nx$ の値を格納する。ノイマン境界条件の場合 (BCtype[3] の値が 'N')、関数 $g(x_i, by)$, $i=0, \dots, nx$ の値を格納する。インデックス i に対応する値は、 $bd_by[i]$ に格納される。

?_Helmholtz_3D の場合、配列のサイズは $(nx+1)*(nz+1)$ 。ディリクレ境界条件の場合 (BCtype[3] の値が 'D')、関数 $G(x_i, by, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。ノイマン境界条件の場合 (BCtype[3] の値が 'N')、関数 $g(x_i, by, z_k)$, $i=0, \dots, nx, k=0, \dots, nz$ の値を格納する。値は配列に格納される。インデックス (i, k) に対応する値は、 $bd_by[i+k*(nx+1)]$ に格納される。

bd_az

double* (d_Helmholtz_3D の場合)、
float* (s_Helmholtz_3D の場合)。

このパラメーターは、?_Helmholtz_3D ルーチンでのみ必要である。z 軸に沿った領域の左端にある境界の境界条件の値を格納する。

配列のサイズは $(nx+1)*(ny+1)$ 。ディリクレ境界条件の場合 (BCtype[4] の値が 'D')、関数 $G(x_i, y_j, az)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。ディリクレ境界条件の場合 (BCtype[4] の値が 'N')、関数 $g(x_i, y_j, az)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。値は配列に格納される。インデックス (i, j) に対応する値は、 $bd_az[i+j*(nx+1)]$ に格納される。

bd_bz

double* (d_Helmholtz_3D の場合)、
float* (s_Helmholtz_3D の場合)。

このパラメーターは、?_Helmholtz_3D ルーチンでのみ必要である。z 軸に沿った領域の右端にある境界の境界条件の値を格納する。

配列のサイズは $(nx+1)*(ny+1)$ 。ディリクレ境界条件の場合 (BCtype[5] の値が 'D')、関数 $G(x_i, y_j, bz)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。ディリクレ境界条件の場合 (BCtype[5] の値が 'N')、関数 $g(x_i, y_j, bz)$, $i=0, \dots, nx, j=0, \dots, ny$ の値を格納する。値は配列に格納される。インデックス (i, j) に対応する値は、 $bd_bz[i+j*(nx+1)]$ に格納される。



注: 間違った計算が行われることを防ぐため、?_commit_Helmholtz_2D/ ?_commit_Helmholtz_3D ルーチンへの呼び出しおよびそれに続く適切な ?_Helmholtz_2D/?_Helmholtz_3D ルーチンへの呼び出しの間は、配列 *bd_ax*、*bd_bx*、*bd_ay*、*bd_by*、*bd_az*、*bd_bz* を変更すべきでない。

出力パラメーター

f

入力時に問題の右辺が格納されたのと同じ方法で格納された問題の近似解を格納する。

xhandle、*yhandle*

インテル MKL DFT インターフェイスで使用されるデータ構造。

<i>ipar</i>	高速ヘルムホルツ・ソルバーに必要な整数データを格納する。 「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>dpar</i>	高速ヘルムホルツ・ソルバーに必要な倍精度データを格納する。 「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>spar</i>	高速ヘルムホルツ・ソルバーに必要な単精度データを格納する。 「 共通パラメーター 」セクションで説明されているように、出力時に修正される。
<i>stat</i>	int*。ルーチンの終了ステータスを格納する。 <i>ipar</i> [0]にも書き込まれる。他の PL ルーチンに進むにはステータスは 0 でなければならない。

説明

ルーチンは、対応する初期化および `commit` ルーチンへの以前の呼び出しで定義されたヘルムホルツ問題の近似解を計算する。解は、「[実装されているポアソン・ライブラリー](#)」セクションの式に従って計算される。問題の右辺のベクトルを最初に保持する *f* パラメーターは、同じ方法で格納された解で置換される。*ax*, *bx*, *ay*, *by*, *az*, *bz*, *nx*, *ny*, *nz* および *Bctype* の値は、各ルーチンに *ipar* 配列とともに渡され、適切な [? init Helmholtz 2D/? init Helmholtz 3D](#) ルーチンへの以前の呼び出しで定義される。

戻り値

<i>stat</i> =1	ルーチンは、エラーなしで終了したが、警告を出力した。
<i>stat</i> =0	ルーチンは、作業を正常に終了した。
<i>stat</i> =-2	ルーチンは、ゼロ除算が発生したために停止した。ゼロ除算は、 <i>dpar</i> または <i>spar</i> 配列のデータが誤って変更された場合に発生する。
<i>stat</i> =-3	ルーチンは、計算を完了するためのメモリーが不足していたために停止した。
<i>stat</i> =-100	ルーチンは、ユーザーのデータにエラーが見つかった。または <i>dpar</i> 、 <i>spar</i> または <i>ipar</i> 配列のデータが誤って変更されたために停止した。
<i>stat</i> =-1000	ルーチンは、インテル MKL DFT または TT インターフェイス・エラーにより停止した。
<i>stat</i> =-10000	ルーチンは、初期化に失敗した。またはパラメーター <i>ipar</i> [0] が誤って変更されたために停止した。
<i>stat</i> =-99999	ルーチンは、致命的なエラーのために作業に失敗した。

free_Helmholtz_2D/free_Helmholtz_3D

DFT インターフェイスで使用するデータ構造に割り当てられていたメモリーをクリーンする。

構文

```
void free_Helmholtz_2D(DFTI_DESCRIPTOR_HANDLE* xhandle, int* ipar, int*
    stat);

void free_Helmholtz_3D(DFTI_DESCRIPTOR_HANDLE* xhandle,
    DFTI_DESCRIPTOR_HANDLE* yhandle, int* ipar, int* stat);
```

入力パラメーター

xhandle、*yhandle* DESCRIPTOR_HANDLE*. インテル MKL DFT インターフェイスで使用されるデータ構造 (詳細は、第 11 章の「[DFT 関数](#)」セクションを参照)。 *yhandle* は、`free_Helmholtz_3D` でのみ使用される。

ipar int。 サイズ 128 の配列。 高速ヘルムホルツ・ソルバーに必要な整数データを格納する (詳細は、「[共通パラメーター](#)」を参照)。

出力パラメーター

xhandle、*yhandle* インテル MKL DFT インターフェイスで使用されるデータ構造。 構造に割り当てられていたメモリーは出力時にリリースされる。

ipar 高速ヘルムホルツ・ソルバーに必要な整数データを格納する。 ルーチン呼び出しのステータスは、*ipar*[0] にも書き込まれる。

stat int*。 ルーチンの終了ステータスを格納する。 *ipar*[0] にも書き込まれる。

説明

ルーチンは、*xhandle* および *yhandle* 構造で使用し、インテル MKL DFT 関数の呼び出しに必要なメモリーをクリーンする。他のパラメーター用に割り当てられていたメモリーをリリースする場合は、コード中にメモリーのクリーンを含める必要がある。

stat=0 ルーチンは、作業を正常に終了した。

stat=-1000 ルーチンは、インテル MKL DFT または TT インターフェイス・エラーにより停止した。

stat=-99999 ルーチンは、致命的なエラーのために作業に失敗した。

共通パラメーター

このセクションでは、PL ルーチンのオプションを格納する配列パラメーター *ipar*、*dpar* および *spar* について説明する。



注: 初期値は、[? init Helmholtz 2D/? init Helmholtz 3D](#) および [? commit Helmholtz 2D/? commit Helmholtz 3D](#) ルーチンによって配列パラメーターに割り当てられる。

ipar int。サイズ 128 の配列。高速ヘルムホルツ・ソルバーに必要な整数データを格納する。要素は、[表 13-5](#) で説明する。

表 13-5 ipar 配列の要素

インデックス	説明
0	<p>最後に呼び出した PL ルーチンのステータス値を格納する。一般に、高速ヘルムホルツ・ソルバーに進むにはステータスは 0 でなければならない。要素には事前に定義された値はない。この要素は、計算の commit ステップ (図 13-2 を参照) をどのように行うべきか ? commit Helmholtz 2D/? commit Helmholtz 3D ルーチンに通知するためにも使用できる。次の 10 進表記で <i>ipar</i>[0] の非ゼロの値 $abc = 100a + 10b + c$</p> <p>(ここで、a、b、および c は 0 または 9) は、commit ステップの一部を省略すべきであることを示す。c=9 の場合、ルーチンはパラメーターの検証とデータ構造の初期化を省略する。b=9 の場合、ルーチンはノイマン境界条件 (境界値の 0.5 での乗算と境界関数 g の組み込み) またはディリクレ境界条件 (境界値の 0 への設定と境界関数 G の組み込み) に対する右辺ベクトル <i>f</i> の調整を省略する。この場合、ルーチンは特定の境界関数に対する右辺ベクトル <i>f</i> の調整も省略する。a=9 の場合、ルーチンは右辺ベクトル <i>f</i> の正規化を省略する (2D の場合、hy^2 での乗算。ここで、<i>hy</i> は y 方向のメッシュサイズ (詳細は、「実装されているポアソン・ライブラリー」を参照)。3D の場合、hz^2 での乗算。ここで、<i>hz</i> は z 方向のメッシュサイズ)。必要に応じ、<i>ipar</i>[0] でルーチンを調整することで、右辺のみ異なる複数のヘルムホルツ問題を効率良く解くことができる。この方法を利用する場合、commit プロセスの誤りは、間違った結果やプログラムの失敗の原因となることに注意する (「パラメーター修正に関する警告」も参照)。</p>
1	<p>エラー・メッセージ・オプションを格納する。</p> <ul style="list-style-type: none"> <i>ipar</i>[1]=-1 は、ルーチンが呼び出されたフォルダーの <i>MKL_Poisson_Library_log.txt</i> ファイルに、すべてのエラーメッセージが出力されることを示す。ファイルが存在しない場合、ルーチンはファイルを作成する。ファイルの作成に失敗した場合、ルーチンは、標準出力デバイスにファイルが作成できないという情報を入力する。 <i>ipar</i>[1]=0 は、エラーメッセージが出力されないことを示す。 <i>ipar</i>[1]=1 は、デフォルト値である。これは、あらかじめ接続されているデフォルトの出力デバイス (通常、スクリーン) にすべてのエラーメッセージが出力されることを示す。 <p>警告が出力された場合、<i>stat</i> パラメーターは <i>ipar</i>[1] 設定に関係なく、非ゼロの値となる。</p>

表 13-5 ipar 配列の要素 (続き)

インデックス	説明	
2	警告メッセージオプションを格納する。 <ul style="list-style-type: none"> $ipar[2]=-1$ は、ルーチンが呼び出されたフォルダーの <i>MKL_Poisson_Library_log.txt</i> ファイルに、すべてのエラーメッセージが出力されることを示す。ファイルが存在しない場合、ルーチンはファイルを作成する。ファイルの作成に失敗した場合、ルーチンは、標準出力デバイスにファイルが作成できないという情報を出力する。 $ipar[2]=0$ は、警告メッセージが出力されないことを示す。 $ipar[2]=1$ は、デフォルト値である。これは、あらかじめ接続されているデフォルトの出力デバイス (通常、スクリーン) にすべての警告メッセージが出力されることを示す。 警告が出力された場合、 <i>stat</i> パラメーターは $ipar[2]$ 設定に関係なく、非ゼロの値となる。	
3	<i>BCtype</i> パラメーターを格納する境界条件の組み合わせの数を格納する。 <ul style="list-style-type: none"> 2D の場合、 0 は 'DDDD' に対応し、 1 は 'DDDN' に対応し、 ... 15 は 'NNNN' に対応する。 3D の場合、 0 は 'DDDDDD' に対応し、 1 は 'DDDDDN' に対応し、 ... 63 は 'NNNNNN' に対応する。 	
4	$BCtype[0]='N'$ の場合は 1、 $BCtype[0]='D'$ の場合は 0、それ以外の場合は -1。	
5	$BCtype[1]='N'$ の場合は 1、 $BCtype[1]='D'$ の場合は 0、それ以外の場合は -1。	
6	$BCtype[2]='N'$ の場合は 1、 $BCtype[2]='D'$ の場合は 0、それ以外の場合は -1。	
7	$BCtype[3]='N'$ の場合は 1、 $BCtype[3]='D'$ の場合は 0、それ以外の場合は -1。	
8	$BCtype[4]='N'$ の場合は 1、 $BCtype[4]='D'$ の場合は 0、それ以外の場合は -1。 このパラメーターは 3D の場合にのみ使用される。	
9	$BCtype[5]='N'$ の場合は 1、 $BCtype[5]='D'$ の場合は 0、それ以外の場合は -1。 このパラメーターは 3D の場合にのみ使用される。	
10	x 軸に沿った区間の数、 nx の値。	
11	y 軸に沿った区間の数、 ny の値。	
12	z 軸に沿った区間の数、 nz の値。このパラメーターは 3D の場合にのみ使用される。	
13	6. <i>dpar/spar</i> 配列の内部分割を指定する。	
14	$ipar[13]+ipar[10]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。 <i>ipar</i> の続く値は、問題の次元に応じて異なる。	
	2D の場合	3D の場合
15	未使用。	$ipar[14]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。
16	未使用。	$ipar[14]+ipar[11]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。
17	$ipar[14]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。	$ipar[16]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。

表 13-5 ipar 配列の要素 (続き)

インデックス	説明
18	$ipar[14]+3*ipar[10]/2+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。
19	未使用。 $ipar[18]+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。
20	未使用。 $ipar[18]+3*ipar[11]/2+1$ 。 <i>dpar/spar</i> 配列の内部分割を指定する。
<i>ipar</i> の続く 2 つの値は、問題の次元に応じて異なる。	
21	メッセージ・スタイル・オプションを格納する。 <i>ipar</i> [21]=0 の場合、PL ルーチンは Fortran スタイルの表記ですべてのエラーおよび警告メッセージを出力する。 <i>ipar</i> [21]=1 の場合、PL ルーチンは C スタイルの表記ですべてのエラーおよび警告メッセージを出力する。デフォルト値は 1。
22	マルチスレッド環境で計算に使用されるスレッドの数を格納する。デフォルト値は 1。
23 ~ 39	ポアソン・ライブラリーの現在の実装では未使用。
40 ~ 59	ソルバーが使用する最初の三角変換の <i>ipar</i> 配列に格納されている最初の 20 の要素を格納する。(詳細は、「 三角変換ルーチン 」の章の「 共通パラメーター 」を参照)。
60 ~ 79	3D ソルバーが使用する 2 番目の三角変換の <i>ipar</i> 配列に格納されている最初の 20 の要素を格納する。(詳細は、「 三角変換ルーチン 」の章の「 共通パラメーター 」を参照)。



注: 現在、*ipar* 配列は、コードで `int ipar[80]` として宣言できる。しかし、より多くの *ipar* 値を使用するインテル MKL ポアソン・ライブラリーの将来のバージョンとの互換性のため、*ipar* を `int ipar[128]` として宣言することを強く推奨する。

配列 *dpar* および *spar* は類似しており、データ精度のみ異なる。

dpar double。サイズ $5*nx/2+7$ (2D の場合) または $5*(nx+ny)/2+9$ (3D の場合) の配列。倍精度高速ヘルムホルツ・ソルバー計算に必要なデータを格納する。この配列は、
`d_init_Helmholtz_2D/d_init_Helmholtz_3D` および
`d_commit_Helmholtz_2D/d_commit_Helmholtz_3D` ルーチンで初期化される。

spar double。サイズ $5*nx/2+7$ (2D の場合) または $5*(nx+ny)/2+9$ (3D の場合) の配列。単精度高速ヘルムホルツ・ソルバー計算に必要なデータを格納する。この配列は、
`s_init_Helmholtz_2D/s_init_Helmholtz_3D` および
`s_commit_Helmholtz_2D/s_commit_Helmholtz_3D` ルーチンで初期化される。

dpar および *spar* には、それぞれの位置に同様の要素がある。これらの要素は、[表 13-6](#) で説明する。

表 13-6 dpar および spar 配列の要素

インデックス	説明
0	?_init_Helmholtz_2D/?_init_Helmholtz_3D ルーチンへの呼び出しの直後の x 軸に沿った区間の長さ、または ?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の x 方向のメッシュサイズ h_x (詳細は、「 実装されているポアソン・ライブラリー 」を参照) を格納する。
1	?_init_Helmholtz_2D/?_init_Helmholtz_3D ルーチンへの呼び出しの直後の y 軸に沿った区間の長さ、または ?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の y 方向のメッシュサイズ h_y (詳細は、「 実装されているポアソン・ライブラリー 」を参照) を格納する。
2	?_init_Helmholtz_3D ルーチンへの呼び出しの直後の z 軸に沿った区間の長さ、または ?_commit_Helmholtz_3D ルーチンへの呼び出しの後の z 方向のメッシュサイズ h_z (詳細は、「 実装されているポアソン・ライブラリー 」を参照) を格納する。このパラメーターは 3D の場合にのみ使用される。
3	?_init_Helmholtz_2D/?_init_Helmholtz_3D ルーチンへの呼び出しの後の係数 q の値を格納する。
4	?_init_Helmholtz_2D/?_init_Helmholtz_3D ルーチンへの呼び出しの後の許容値パラメーターを格納する。この値は、純ノイマン境界条件 ($BCtype="NNNN"$ (2D の場合) または $BCtype="NNNNNN"$ (3D の場合)) でのみ使用される。係数 q がゼロの場合、問題のサイズは任意にできないため、これは特別なケースである。ポアソン・ライブラリーは、この許容値を使用する典型的な解が (丸め誤差まで) 存在するかどうかを確認する。どんな場合でも、ポアソン・ライブラリーは標準の解、つまりユークリッド残差ノルムが最小の解を計算する。しかし、 ?_Helmholtz_2D/?_Helmholtz_3D ルーチンは (丸め誤差まで) 典型的な意味での解は存在しないことをユーザーに通知する。このパラメーターのデフォルト値は、倍精度計算の場合は $1.0E-10$ で、単精度計算の場合は $1.0E-4$ である。ユーザーは警告が表示されないように許容値を増やしてもよい。
<code>ipar[13]-1</code> ~ <code>ipar[14]-1</code>	?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の x 軸に沿った 1D 問題のスペクトルを格納する。
<code>ipar[15]-1</code> ~ <code>ipar[16]-1</code>	?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の y 軸に沿った 1D 問題のスペクトルを格納する。これらのパラメーターは 3D の場合にのみ使用される。
<code>ipar[17]-1</code> ~ <code>ipar[18]-1</code>	?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の x 軸に沿ったメッシュ点の (スタッガード) 正弦/余弦の値を格納する。
<code>ipar[19]-1</code> ~ <code>ipar[20]-1</code>	?_commit_Helmholtz_2D/?_commit_Helmholtz_3D ルーチンへの呼び出しの後の y 軸に沿ったメッシュ点の (スタッガード) 正弦/余弦の値を格納する。これらのパラメーターは 3D の場合にのみ使用される。



注: 解く問題の型に応じて配列サイズを定義できる。

パラメーター修正に関する警告

PL インターフェイスは柔軟であるため、[? init Helmholtz 2D/? init Helmholtz 3D](#) ルーチンの呼び出しをスキップしてユーザーのコード中で明示的に基本的なデータ構造を初期化することができる。初期化の後、*ipar*、*dpar* および *spar* 配列の内容も修正する必要がある。この際、正確で一貫したデータを配列で使用する必要がある。配列を誤って変更すると、エラーが発生したり、間違った計算が行われる。[? commit Helmholtz 2D/? commit Helmholtz 3D](#) ルーチンを呼び出すことにより、パラメーターの正確性と一貫性の基本的な検証を行うことができる。ただし、この基本的な検証は、解が正しいことを保証するものではなく、エラーや間違った結果になる可能性を減らすだけであることに注意する。



注：PL ルーチンに正確で一貫したパラメーターを提供するには、PL インターフェイスを何度も利用し、解のプロセスに加えて、*ipar*、*spar* および *dpar* 配列に格納されている要素と、これらの要素の値の依存性についてよく理解している必要がある。

しかし、経験を積んだユーザーでも、高速ヘルムホルツ・ソルバーの使用に失敗することがある。



警告：ヘルムホルツ問題の正しい解を導く唯一の方法は、一般的なルーチン起動の順序に従い、パラメーターのデフォルトセットを変更しないことである。このため、特別に必要な場合を除いて、*ipar*、*dpar* および *spar* 配列を修正しないことを推奨する。

実装の詳細

インテル MKL PL インターフェイスの一部は、プラットフォーム固有でかつ言語固有である。プラットフォーム間の移植性の確保と、異なる言語でも簡単に使用できるように、インテル MKL PL の言語固有のヘッダーファイルが提供されている。現在、以下のヘッダーファイルが利用できる。

- `mkl_poisson.h` は C プログラム用で、`mkl_dfti.h` とともに使用する。
- `mkl_poisson.f90` は Fortran-90 プログラム用で、`mkl_dfti.f90` とともに使用する。



注：上記のヘッダーファイルのいずれかをインクルードしないと、インテル MKL PL ソフトウェアを使用することはできない。

インクルード・ファイルは、適切な言語用の関数プロトタイプを定義する。

C 固有のヘッダーファイル

C 固有のヘッダーファイルは、以下の関数プロトタイプを定義する。

```
void d_init_Helmholtz_2D(double*, double*, double*, double*, int*, int*,
char*, double*, int*, double*, int*);

void d_commit_Helmholtz_2D(double*, double*, double*, double*, double*,
DFTI_DESCRIPTOR_HANDLE*, int*, double*, int*);

void d_Helmholtz_2D(double*, double*, double*, double*, double*,
DFTI_DESCRIPTOR_HANDLE*, int*, double*, int*);

void s_init_Helmholtz_2D(float*, float*, float*, float*, int*, int*,
char*, float*, int*, float*, int*);

void s_commit_Helmholtz_2D(float*, float*, float*, float*, float*,
DFTI_DESCRIPTOR_HANDLE*, int*, float*, int*);

void s_Helmholtz_2D(float*, float*, float*, float*, float*,
DFTI_DESCRIPTOR_HANDLE*, int*, float*, int*);

void free_Helmholtz_2D(DFTI_DESCRIPTOR_HANDLE*, int*, int*);

void d_init_Helmholtz_3D(double*, double*, double*, double*, double*,
double*, int*, int*, int*, char*, double*, int*, double*, int*);

void d_commit_Helmholtz_3D(double*, double*, double*, double*, double*,
double*, double*, DFTI_DESCRIPTOR_HANDLE*, DFTI_DESCRIPTOR_HANDLE*,
int*, double*, int*);

void d_Helmholtz_3D(double*, double*, double*, double*, double*,
double*, double*, DFTI_DESCRIPTOR_HANDLE*, DFTI_DESCRIPTOR_HANDLE*,
int*, double*, int*);

void s_init_Helmholtz_3D(float*, float*, float*, float*, float*, float*,
int*, int*, int*, char*, float*, int*, float*, int*);

void s_commit_Helmholtz_3D(float*, float*, float*, float*, float*,
float*, float*, DFTI_DESCRIPTOR_HANDLE*, DFTI_DESCRIPTOR_HANDLE*, int*,
float*, int*);

void s_Helmholtz_3D(float*, float*, float*, float*, float*, float*,
float*, DFTI_DESCRIPTOR_HANDLE*, DFTI_DESCRIPTOR_HANDLE*, int*, float*,
int*);

void free_Helmholtz_3D(DFTI_DESCRIPTOR_HANDLE*, DFTI_DESCRIPTOR_HANDLE*,
int*, int*);
```

Fortran 固有のヘッダーファイル

Fortran-90 固有のヘッダーファイルは、以下の関数プロトタイプを定義する。

```
SUBROUTINE D_INIT_HELMHOLTZ_2D (AX, BX, AY, BY, NX, NY, BCTYPE, Q, IPAR,
DPAR, STAT)
```

```
USE MKL_DFTI
```

```
INTEGER NX, NY, STAT
```

```
INTEGER IPAR(*)
```

```
DOUBLE PRECISION AX, BX, AY, BY, Q
```

```
DOUBLE PRECISION DPAR(*)
```

```
CHARACTER(4) BCTYPE
```

```
END SUBROUTINE
```

```
SUBROUTINE D_COMMIT_HELMHOLTZ_2D (F, BD_AX, BD_BX, BD_AY, BD_BY, XHANDLE,
IPAR, DPAR, STAT)
```

```
USE MKL_DFTI
```

```
INTEGER STAT
```

```
INTEGER IPAR(*)
```

```
DOUBLE PRECISION F(IPAR(11)+1,*)
```

```
DOUBLE PRECISION DPAR(*)
```

```
DOUBLE PRECISION BD_AX(*), BD_BX(*), BD_AY(*), BD_BY(*)
```

```
TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE
```

```
END SUBROUTINE
```

```
SUBROUTINE D_HELMHOLTZ_2D (F, BD_AX, BD_BX, BD_AY, BD_BY, XHANDLE, IPAR,
DPAR, STAT)
```

```
USE MKL_DFTI
```

```
INTEGER STAT
```

```
INTEGER IPAR(*)
```

```
DOUBLE PRECISION F(IPAR(11)+1,*)
```

```
DOUBLE PRECISION DPAR(*)
```

```
DOUBLE PRECISION BD_AX(*), BD_BX(*), BD_AY(*), BD_BY(*)
```

```
TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE
```

```
END SUBROUTINE
```

```
SUBROUTINE S_INIT_HELMHOLTZ_2D (AX, BX, AY, BY, NX, NY, BCTYPE, Q, IPAR,
DPAR, STAT)
```

```
USE MKL_DFTI
```

```
INTEGER NX, NY, STAT
```

```
INTEGER IPAR(*)
```

```
REAL AX, BX, AY, BY, Q
```

```

    REAL DPAR(*)
    CHARACTER(4) BCTYPE
END SUBROUTINE
SUBROUTINE S_COMMIT_HELMHOLTZ_2D (F, BD_AX, BD_BX, BD_AY, BD_BY, XHANDLE,
IPAR, DPAR, STAT)
    USE MKL_DFTI

    INTEGER STAT
    INTEGER IPAR(*)
    REAL F(IPAR(11)+1,*)
    REAL DPAR(*)
    REAL BD_AX(*), BD_BX(*), BD_AY(*), BD_BY(*)
    TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE
END SUBROUTINE
SUBROUTINE S_HELMHOLTZ_2D (F, BD_AX, BD_BX, BD_AY, BD_BY, XHANDLE, IPAR,
DPAR, STAT)
    USE MKL_DFTI

    INTEGER STAT
    INTEGER IPAR(*)
    REAL F(IPAR(11)+1,*)
    REAL DPAR(*)
    REAL BD_AX(*), BD_BX(*), BD_AY(*), BD_BY(*)
    TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE
END SUBROUTINE

SUBROUTINE FREE_HELMHOLTZ_2D (XHANDLE, IPAR, STAT)
    USE MKL_DFTI

    INTEGER STAT
    INTEGER IPAR(*)
    TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE
END SUBROUTINE

SUBROUTINE D_INIT_HELMHOLTZ_3D (AX, BX, AY, BY, AZ, BZ, NX, NY, NZ,
BCTYPE, Q, IPAR, DPAR, STAT)
    USE MKL_DFTI

    INTEGER NX, NY, NZ, STAT
    INTEGER IPAR(*)
    DOUBLE PRECISION AX, BX, AY, BY, AZ, BZ, Q
    DOUBLE PRECISION DPAR(*)

```

```
CHARACTER(6) BCTYPE
END SUBROUTINE

SUBROUTINE D_COMMIT_HELMHOLTZ_3D (F, BD_AX, BD_BX, BD_AY, BD_BY, BD_AZ,
BD_BZ, XHANDLE, YHANDLE, IPAR, DPAR, STAT)
  USE MKL_DFTI

  INTEGER STAT
  INTEGER IPAR(*)
  DOUBLE PRECISION F(IPAR(11)+1,IPAR(12)+1,*)
  DOUBLE PRECISION DPAR(*)
  DOUBLE PRECISION BD_AX(IPAR(12)+1,*), BD_BX(IPAR(12)+1,*),
BD_AY(IPAR(11)+1,*)
  DOUBLE PRECISION BD_BY(IPAR(11)+1,*), BD_AZ(IPAR(11)+1,*),
BD_BZ(IPAR(11)+1,*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE, YHANDLE
END SUBROUTINE

SUBROUTINE D_HELMHOLTZ_3D (F, BD_AX, BD_BX, BD_AY, BD_BY, BD_AZ, BD_BZ,
XHANDLE, YHANDLE, IPAR, DPAR, STAT)
  USE MKL_DFTI

  INTEGER STAT
  INTEGER IPAR(*)
  DOUBLE PRECISION F(IPAR(11)+1,IPAR(12)+1,*)
  DOUBLE PRECISION DPAR(*)
  DOUBLE PRECISION BD_AX(IPAR(12)+1,*), BD_BX(IPAR(12)+1,*),
BD_AY(IPAR(11)+1,*)
  DOUBLE PRECISION BD_BY(IPAR(11)+1,*), BD_AZ(IPAR(11)+1,*),
BD_BZ(IPAR(11)+1,*)
  TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE, YHANDLE
END SUBROUTINE

SUBROUTINE S_INIT_HELMHOLTZ_3D (AX, BX, AY, BY, AZ, BZ, NX, NY, NZ,
BCTYPE, Q, IPAR, DPAR, STAT)
  USE MKL_DFTI

  INTEGER NX, NY, NZ, STAT
  INTEGER IPAR(*)
  REAL AX, BX, AY, BY, AZ, BZ, Q
  REAL DPAR(*)
  CHARACTER(6) BCTYPE
END SUBROUTINE

SUBROUTINE S_COMMIT_HELMHOLTZ_3D (F, BD_AX, BD_BX, BD_AY, BD_BY, BD_AZ,
BD_BZ, XHANDLE, YHANDLE, IPAR, DPAR, STAT)
```

```
USE MKL_DFTI

INTEGER STAT
INTEGER IPAR(*)
REAL F(IPAR(11)+1,IPAR(12)+1,*)
REAL DPAR(*)
REAL BD_AX(IPAR(12)+1,*), BD_BX(IPAR(12)+1,*), BD_AY(IPAR(11)+1,*)
REAL BD_BY(IPAR(11)+1,*), BD_AZ(IPAR(11)+1,*), BD_BZ(IPAR(11)+1,*)
TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE, YHANDLE
END SUBROUTINE

SUBROUTINE S_HELMHOLTZ_3D (F, BD_AX, BD_BX, BD_AY, BD_BY, BD_AZ, BD_BZ,
XHANDLE, YHANDLE, IPAR, DPAR, STAT)
USE MKL_DFTI

INTEGER STAT
INTEGER IPAR(*)
REAL F(IPAR(11)+1,IPAR(12)+1,*)
REAL DPAR(*)
REAL BD_AX(IPAR(12)+1,*), BD_BX(IPAR(12)+1,*), BD_AY(IPAR(11)+1,*)
REAL BD_BY(IPAR(11)+1,*), BD_AZ(IPAR(11)+1,*), BD_BZ(IPAR(11)+1,*)
TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE, YHANDLE
END SUBROUTINE

SUBROUTINE FREE_HELMHOLTZ_3D (XHANDLE, YHANDLE, IPAR, STAT)
USE MKL_DFTI

INTEGER STAT
INTEGER IPAR(*)
TYPE(DFTI_DESCRIPTOR), POINTER :: XHANDLE, YHANDLE
END SUBROUTINE
```

PL ルーチンの Fortran-90 固有の使用法は、すべてのインテル MKL PDE サポートツールと同様であり、この章の「[Fortran-90 からの PDE サポートルーチンの呼び出し](#)」セクションで説明されている。

Fortran-90 からの PDE サポートルーチンの呼び出し

すべてのインテル MKL TT および PL ルーチンの呼び出しインターフェイスは、C で簡単に使用できるように設計されている。ただし、使用しているプラットフォームの異言語呼び出し規則について知識がある場合、すべての TT および PL ルーチンは Fortran-90 から直接起動することもできる。



注：インテル MKL DFT インターフェイスの使用による制限のため、TT および PL インターフェイスはどちらも Fortran-77 から起動できない。

言語間の呼び出し規則には、言語の引数の渡し方、C から Fortran-90 へのデータ型のマップ方法、および C の外部名の修飾方法が含まれているが、限定的ではない。

移植性を高め、呼び出し規則に詳細に対処しなくても済むように、Fortran-90 ヘッダーファイル `mkl_trig_transforms.f90` (TT ルーチン) または `mkl_poisson.f90` (PL ルーチン) と `mkl_dfti.f90` を使用すると、マクロのセットが宣言され、言語間呼び出し規則を隠す型の定義が導入され、Fortran-90 で自然に見えるルーチンへのインターフェイスが提供される。

例えば、長さが `n` の倍精度ベクトルを受け取る `foo` というライブラリー・ルーチンがあるとすると、C の場合、このような関数には次のようにアクセスする。

```
int n;
double *x;
...
foo(x, &n);
```

Fortran-90 では、`foo` を起動する場合に、C の `int` 型と `double` 型 (単精度の場合は `float` 型) に対応する Fortran-90 のデータ型、C コンパイラーが使用する引数の渡し方、そして (該当する場合は) 外部シンボル `foo` の生成時に C コンパイラーにより実行される名前修飾などを知っておく必要がある。

しかし、Fortran-90 ヘッダーファイル `mkl_trig_transforms.f90` / `mkl_poisson.f90` および `mkl_dfti.f90` をインクルードすると、Fortran-90 プログラムでの `foo` の起動は次のようになる。

- TT インターフェイスの場合、


```
use mkl_dfti
use mkl_trig_transforms
INTEGER n
DOUBLE PRECISION, ALLOCATABLE :: x
...
CALL FOO(x,n)
```
- PL インターフェイスの場合、


```
use mkl_dfti
use mkl_poisson
INTEGER n
DOUBLE PRECISION, ALLOCATABLE :: x
...
```

```
CALL FOO(x,n)
```

上記の例で、ヘッダーファイル `mkl_trig_transforms.f90` / `mkl_poisson.f90` および `mkl_dfti.f90` は、サブルーチン `FOO` の定義を提供する。Fortran-90 で `PL` または `TT` ルーチンの使用を簡単にするため、名前の Fortran-90 定義を提供する一般的なアプローチがライブラリー全体で使用されている。PL または TT インターフェイスからの名前で C 固有の名前 `foo` が使用されている場合、Fortran-90 ヘッダーファイルは適切な Fortran-90 言語型定義 `FOO` を提供する。

Fortran-90 と C の主な違いは、引数の渡し方が異なることである。C プログラムは値渡しセマンティクスを使用するのに対し、Fortran-90 プログラムは参照渡しセマンティクスを使用する。Fortran-90 ヘッダーは、この違いが適切に処理されることを保証する。特に、上記の例では、ヘッダーファイル `mkl_trig_transforms.f90` / `mkl_poisson.f90` および `mkl_dfti.f90` は、適切な引数のアドレスを処理するマクロ `FOO` を定義することにより、この相違を隠蔽しようとしている。

線形ソルバーの基礎

A

科学工学分野の多くのアプリケーションでは、連立線形方程式を解く必要がある。この問題は通常、行列とベクトルによる式 $Ax = b$ 。ここで、 A は $m \times n$ の行列、 x は n 成分の列ベクトル、 b は m 成分の列ベクトルである。行列 A は通常、係数行列と呼ばれる。ベクトル x と b は、それぞれ、解ベクトルと右辺と呼ばれる。

スパース行列を使用した連立線形方程式の解法に関する基本的な概念は、この後の「[スパース連立線形方程式](#)」セクションで説明する。

行列 A の係数とベクトル b の右辺が確定できず、既知の区間に属している場合、方程式は [区間線形方程式](#) と呼ばれる。区間線形方程式の解法に使用される基本的な定義と概念は、この後の「[区間線形方程式](#)」セクションで説明する。

スパース連立線形方程式

現実の多くのアプリケーションでは、 A の多くの成分はゼロである。このような行列は、スパース行列（疎行列）と呼ばれる。逆に、ゼロの成分が少ない行列は、密行列と呼ばれる。スパース行列では、行列が疎であることを活用できれば、方程式 $Ax = b$ の解を計算することで、容量と計算時間の両方の面で効率を上げることができる。アルゴリズムが正確さを犠牲にすることなく、疎であることを活用できれば、アルゴリズムはより良くなる。

一般に、連立線形方程式の解を求めるソフトウェアは、ソルバーと呼ばれる。特に、スパース線形方程式を解くために設計されたソルバーは、スパースソルバーと呼ばれる。ソルバーは通常、反復ソルバーと直接ソルバーの2つのグループに分類される。

反復ソルバー 解の初期近似で開始し、近似と真の解との差を推定する。その差をもとに、反復ソルバーは初期近似より真の解により近い新しい近似を計算する。この処理過程は、近似と真の解との差が十分小さくなるまで繰り返される。反復ソルバーの主な欠点は、収束率が行列 A の値に大きく依存することである。したがって、解を得るまでの所要時間を予測することができない。事実、悪条件の行列では、反復処理は全く解に収束しない。しかし、好条件の行列では、反復ソルバーは非常に速く解に収束する可能性がある。したがって、適切なアプリケーションでは、反復ソルバーは非常に有効である。

直接ソルバー 行列 A を2つの三角行列の積に分解し、その上で順方向と逆方向から三角行列を解く。

この方法では、行列の大きさをもとに、連立線形方程式を解くために要する時間を予測できる。実際、スパース行列では、配列 A の非ゼロ成分の数をもとに計算時間を予測できる。

行列の基礎事項

行列は、実数または複素数の長方形の配列である。行列は大文字で示され、その成分は同じ文字の小文字で行と列の添字を付けて示される。したがって、行列 A の i 行 j 列の成分の値は $a(i, j)$ と示される。

例えば、3 行 4 列の行列 A は、次のように記述される。

$$A = \begin{bmatrix} a(1, 1) & a(1, 2) & a(1, 3) & a(1, 4) \\ a(2, 1) & a(2, 2) & a(2, 3) & a(2, 4) \\ a(3, 1) & a(3, 2) & a(3, 3) & a(3, 4) \end{bmatrix}$$

上の表記では、配列のインデックスは、0 から始まる C プログラム言語表記ではなく、1 から始まる標準 Fortran プログラム言語表記であると仮定している。

すべての成分が実数の行列は、実行列と呼ばれる。1 つでも複素数を含む行列は、複素行列と呼ばれる。

実 / 複素行列 A でプロパティが $a(i, j) = a(j, i)$ のものは、対称行列と呼ばれる。

実 / 複素行列 A でプロパティが $a(i, j) = \text{conj}(a(j, i))$ のものは、エルミート行列と呼ばれる。対称 / エルミート行列を扱うプログラムでは、格納されていない成分の値は格納された値から素早く再構成できるため、行列の半分の値を格納するだけでよい。

行数と列数が同じ行列は、正方行列と呼ばれる。正方行列で行インデックスと列インデックスが同じ成分は、行列の対角成分、または単に行列の対角と呼ばれる。

行列 A の転置とは、配列の成分を対角で「反転」させて得られる行列である。すなわち、成分 $a(i, j)$ と $a(j, i)$ を交換する。複素行列で、対角で成分を反転してから成分の複素共役をとった場合、結果の行列は元の行列のエルミート転置または共役転置と呼ばれる。行列 A の転置およびエルミート転置は、それぞれ、 A^T および A^H で示される。

列ベクトル (または単にベクトル) は $n \times 1$ の行列であり、行ベクトルは $1 \times n$ の行列である。

すべて非ゼロのベクトル x でベクトル - 行列の積 $x^T A x$ がゼロより大きい場合、実 / 複素行列 A は正定値である。正定値ではない行列は不定値と呼ばれる。

上 (下) 三角行列は、対角の下 (上) のすべての成分がゼロである正方行列である。単位三角行列は、対角がすべて 1 である上あるいは下三角行列である。

任意の行列 A で、行列積の結果 PA が A の行の交換を除いて A と同一である場合、行列 P は、置換行列と呼ばれる。正方行列の場合、 PA が A の行の置換行列であれば、 AP^T は A の同じ列の置換行列である。さらに P の逆行列は P^T となる。

スペースを節約するため、置換行列は通常、行列としてではなく、置換ベクトルと呼ばれる線形配列として格納される。特に、置換行列が行列の i 番目の行を j 番目の行にマップする場合、置換ベクトルの i 番目の成分は j となる。

対角上にもみ非ゼロ成分を持つ行列は、対角行列と呼ばれる。置換行列の場合と同様に、それは行列としてではなく、ベクトルとして格納される。

直接法

直接法を用いるソルバーでは、方程式 $Ax = b$ の解を求める基本的な技法は、最初に A を三角行列に因子分解することである。すなわち、 $A = LU$ となるような下三角行列 L と上三角行列 U を求めることである。そのような因子分解（通常 LU 分解あるいは LU 因子分解と呼ばれる）を用いることで、元の問題の解は次のように記述できる。

$$\begin{aligned} Ax &= b \\ \Rightarrow LUx &= b \\ \Rightarrow (Ux) &= b \end{aligned}$$

この結果、次に示す2ステップのプロセスで元の方程式の解を求めることになる。

1. 方程式 $Ly = b$ を解く。
2. 方程式 $Ux = y$ を解く。

方程式 $Ly = b$ と $Ux = y$ を解くことはそれぞれ、順方向で解く、逆方向で解く、と呼ばれる。

対称行列 A が正定値の場合、 A は LL^T として因子分解できる。ここで L は下三角行列である。同様に、エルミート行列 A が正定値の場合、 $A = LL^H$ として因子分解できる。対称/エルミート行列では、この形式の因子分解は、コレスキー因子分解と呼ばれる。

コレスキー因子分解で、 LU 分解の行列 U は、 L^T または L^H である。したがって、 L と A の半分のみを格納すればよく、 U の計算も省けるため、ソルバーの効率を上げることができる。そのため、正定値方程式の解としてアプリケーションを表現できるユーザーは、一般的な場合に比べて、大幅な効率向上が可能である。

対称（またはエルミート）であるが正定値ではない行列についても、まだかなりの効率向上の余地がある。 A が対称であるが正定値ではない場合、 A は $A = LDL^T$ として因子分解できる。ここで、 D は対角行列、 L は下単位三角行列である。同様に、 A がエルミートの場合、 $A = LDL^H$ として因子分解できる。どちらの場合も、 L 、 D 、と A の半分のみを格納すればよく、 U を計算する必要はない。ただし、逆方向に解く場合は、 $L^T x = y$ ではなく、 $L^T x = D^{-1}y$ を解く必要がある。

スパース行列のフィルインとリオーダーリング

スパース方程式を解くことに関連した2つの重要な概念は、フィルインとリオーダーリングである。次の例で、これらの概念を説明する。

連立線形方程式 $Ax = b$ を考える。ここで、 A は対称正定値のスパース行列であり、次のように定義される。

$$A = \begin{bmatrix} 9 & \frac{3}{2} & 6 & \frac{3}{4} & 3 \\ \frac{3}{2} & \frac{1}{2} & * & * & * \\ 6 & * & 12 & * & * \\ \frac{3}{4} & * & * & \frac{5}{8} & * \\ 3 & * & * & * & 16 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

星印(*)はゼロを表し、 A が疎であることを強調している。 A のコレスキー因子分解は $A=LL^T$ とする。ここで、 L は次のようになる。

$$L = \begin{bmatrix} 3 & * & * & * & * \\ \frac{1}{2} & \frac{1}{2} & * & * & * \\ 2 & -2 & 2 & * & * \\ \frac{1}{4} & \frac{1}{-4} & \frac{1}{-2} & \frac{1}{2} & * \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix}$$

行列 A が相対的に疎であるにもかかわらず、下三角行列 L は対角より下にゼロがないことに注意する。 L を計算してから、順方向および逆方向に解く段階で使用すると、 A が密である場合と同じ計算量となる。

A がゼロの場所に L が非ゼロの成分を持つことは、フィルインと呼ばれる。計算量の面からは、ソルバーが A の非ゼロ構造を活用して L の計算でフィルインを減らせれば、より効率が向上する。この結果、ソルバーは L の非ゼロの箇所のみ計算すればよいことになる。この目的に沿って、 A の行と列の置換を考える。「[行列の基礎事項](#)」セクションで説明したように、 A の行の置換は、置換行列 P で表される。行の置換の結果は、 P と A の積となる。上記の例で、 A の最初の行と5番目行を交換し、 A の最初の列と5番目の列を交換し、結果が行列 B になるとする。数学的には、 A の行と列を置換して B を得る処理を $B=PAP^T$ として表現できる。 A の行と列を置換すると、 B は次のようになる。

$$B = \begin{bmatrix} 16 & * & * & * & 3 \\ * & \frac{1}{2} & * & * & \frac{3}{2} \\ * & * & 12 & * & 6 \\ * & * & * & \frac{5}{8} & \frac{3}{4} \\ 3 & \frac{3}{2} & 6 & \frac{3}{4} & 9 \end{bmatrix}$$

B は A の行と列を入れ替えただけなので、 A と B の非ゼロ成分の数は同じである。しかし、コレスキー因子分解 $B=LL^T$ を求めると、次のようになる。

$$L = \begin{bmatrix} 4 & * & * & * & * \\ * & \frac{1}{\sqrt{2}} & * & * & * \\ * & * & 2(\sqrt{3}) & * & * \\ * & * & * & \frac{\sqrt{10}}{4} & * \\ \frac{3}{4} & \frac{3}{\sqrt{2}} & \sqrt{3} & \frac{3}{\sqrt{10}} & \frac{\sqrt{3}}{4} \end{bmatrix}$$

B を元にしたフィルインは、 A を元にしたフィルインよりも少ない。したがって、 B の因子分解に必要な容量と計算時間は A の因子分解よりも少なくなる。このことから、効率のよいスパースソルバーは、行列 A の置換 P を求めて、 $B = PAP^T$ の因子分解でのフィルインを最小にする必要があることが分かる。そして B の因子分解を元の方程式を解くのに使用する。

上記の例は対称正定値行列とコレスキー分解をもとにしたが、同様の手法は一般の LU 分解にも有効である。具体的には、 P を置換行列として、 $B = PAP^T$ であり、 B は $B = LU$ と因子分解できる。すると、

$$\begin{aligned} Ax &= b \\ \Rightarrow PA(P^{-1}P)x &= Pb \\ \Rightarrow PA(P^T P)x &= Pb \\ \Rightarrow (PAP^T)(Px) &= Pb \\ \Rightarrow B(Px) &= Pb \\ \Rightarrow LU(Px) &= Pb \end{aligned}$$

したがって、 B に対する LU 分解が得られれば、元の方程式を次の 3 ステップのプロセスで解くことができる。

1. $Ly = Pb$ を解く。
2. $Uz = y$ を解く。
3. $x = P^T z$ とする。

この 3 ステップのプロセスを上例に適用する場合、最初に、連立線形方程式 $Ly = Pb$ を順方向に解く必要がある。

$$\begin{bmatrix} 4 & * & * & * & * \\ * & \frac{1}{\sqrt{2}} & * & * & * \\ * & * & 2(\sqrt{3}) & * & * \\ * & * & * & \frac{\sqrt{10}}{4} & * \\ \frac{3}{4} & \frac{3}{\sqrt{2}} & \sqrt{3} & \frac{3}{\sqrt{10}} & \frac{\sqrt{3}}{4} \end{bmatrix} * \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \\ y5 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}$$

結果は、 $y^T = \frac{5}{4}, 2\sqrt{2}, \frac{\sqrt{3}}{2}, \frac{16}{\sqrt{10}}, \frac{-979\sqrt{3}}{12}$ になる。

2 番目のステップでは、 $Uz = y$ を逆方向から解く。または、コレスキー因子分解を用いて、 $L^T z = y$ を解く。

$$\begin{bmatrix} 4 & * & * & * & \frac{3}{4} \\ * & \frac{1}{\sqrt{2}} & * & * & \frac{3}{\sqrt{2}} \\ * & * & 2(\sqrt{3}) & * & \sqrt{3} \\ * & * & * & \frac{\sqrt{10}}{4} & \frac{3}{\sqrt{10}} \\ * & * & * & * & \frac{\sqrt{3}}{\sqrt{5}} \\ * & * & * & * & \frac{\sqrt{5}}{4} \end{bmatrix} * \begin{bmatrix} z1 \\ z2 \\ z3 \\ z4 \\ z5 \end{bmatrix} = \begin{bmatrix} \frac{5}{4} \\ 2(\sqrt{2}) \\ \frac{\sqrt{3}}{2} \\ \frac{16}{\sqrt{10}} \\ -979 \frac{\sqrt{3}}{\sqrt{5}} \\ \frac{123}{12} \end{bmatrix}$$

結果は、 $z = \frac{123}{2}, 983, \frac{1961}{12}, 398, \frac{-979}{3}$ になる。

最後のステップでは、 $x = P^T z$ を計算する。結果は、 $x^T = \frac{-979}{3}, 983, \frac{1961}{12}, 398, \frac{123}{2}$ になる。

スパース行列の格納形式

これまで説明したように、スパース行列の非ゼロ成分のみを格納すると効率が向上する。これは、疎の程度が大きい、すなわち、非ゼロ成分が非常に少ないと仮定しているからである。逆に、ゼロ成分が非常に少ない場合、スパース行列用の処理は、単純に行列を密として処理する、すなわちゼロも非ゼロもすべての値を計算に用いる場合よりも遅くなってしまう。

スパース行列用に使用する一般的な格納方式はいくつかあるが、ほとんどの方式が同じ基本的な技法を用いている。つまり、行列のすべての非ゼロ成分を線形配列に圧縮し、補助配列を使用して、元の行列における非ゼロの位置を格納する。

PARDISO* ソルバーの格納形式

スパース行列 A の非ゼロ成分の線形配列への圧縮は、各列 (列主体形式) または各行 (行主体形式) を順番に調べて、見つかった非ゼロ成分を順番に線形配列に書き込むことで行われる。

対称行列を格納するときは、行列の上半分の三角のみ (上三角形式) または行列の下半分の三角のみ (下三角形式) を格納すればよい。

インテル® マス・カーネル・ライブラリー (インテル® MKL) の直接法スパースソルバーは、行主体の上三角格納形式を採用している。すなわち、行列は行ごとに圧縮され、対称行列は行列の上三角のみが格納される。

スパース行列用の PARDISO ソフトウェアで利用できるインテル MKL の格納形式は、*values*、*columns*、および *rowIndex* 配列と呼ばれる 3 つの配列からなる。次の表は、スパース行列 A の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

values A の非ゼロ成分を含む実数または複素数配列。 A の非ゼロの値は、上で説明した行主体の上三角格納マッピングを使用して *values* 配列にマップされる。

columns 整数配列 *columns* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の列の番号を格納する。

rowIndex 整数配列 *rowIndex* の *j* 番目の成分は、*A* の行 *j* の最初の非ゼロ成分を含む *values* 配列へのインデックスを格納する。

values と *columns* 配列の長さは *A* の非ゼロの個数に等しい。

rowIndex 配列は行で最初の非ゼロの位置を示し、さらに非ゼロが続けて格納されるため、*i* 番目の行の非ゼロの数を *rowIndex*(*i*) と *rowIndex*(*i*+1) の差として計算できる。

この関係を *A* の最後の行まで保持するため、*rowIndex* の最後に 1 つの成分 (ダミー成分) を追加し、*A* の非ゼロの数に 1 を加えた値を設定する。これにより、*rowIndex* 配列の合計の長さは、*A* の行数より 1 大きくなる。



注: インテル MKL のスパース格納方式では、配列のインデックスは、0 から始まる C プログラム言語表記ではなく、1 から始まる標準 Fortran プログラム言語表記を使用している。

上記を念頭において、以前のセクションの例で使用した、次の対称行列を格納することを考える。

$$A = \begin{bmatrix} 9 & \frac{3}{2} & 6 & \frac{3}{4} & 3 \\ * & \frac{1}{2} & * & * & * \\ * & * & \frac{1}{2} & * & * \\ * & * & * & \frac{5}{8} & * \\ * & * & * & * & 16 \end{bmatrix}$$

この場合、*A* には 9 個の非ゼロ成分があるので、*values* と *columns* 配列の長さは 9 である。また、行列 *A* には 5 つの行があるので *rowIndex* 配列の長さは 6 である。この例では、各配列の実際の値は次のようになる。

表 A-1 対称行列の例に対する格納配列

<i>values</i>	=	(9 3/2 6 3/4 3 1/2 1/2 5/8 16)
<i>columns</i>	=	(1 2 3 4 5 2 3 4 5)
<i>rowIndex</i>	=	(1 6 7 8 9 10)

非対称または非エルミート配列では、すべての非ゼロを格納する必要がある。次のように定義される非対称行列 B を考える。

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

B には 13 個の非ゼロがあり、次のように B を格納する。

表 A-2 非対称行列の例に対する格納配列

```
values    = (1  -1  -3  -2  5  4  6  4  -4  2  7  8  -5)
columns   = (1  2  4  1  2  3  4  5  1  3  4  2  5)
rowIndex  = (1  4  6  9  12  14)
```

対称構造の連立方程式とは、非ゼロのパターンが対称のものである。すなわち、 $a(i,j)$ が非ゼロの場合に $a(j,i)$ も非ゼロの場合にのみ、行列は対称構造である。ソルバー・ソフトウェアの観点からすると、行列の非ゼロ成分は、*values* 配列に格納されるものすべてである。これはつまり、*values* 配列に注意深くゼロを追加することにより、任意の非対称行列を対称構造の行列に変換できることを意味する。例えば、行列 B に次のような非ゼロ成分が含まれているとする。

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & 0 \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & 0 & * & -5 \end{bmatrix}$$

ここで、 B は 15 の非ゼロ成分を含む対称構造であると考えられる。

表 A-3 対称構造の行列の例に対する格納配列

```
values    = (1  -1  -3  -2  5  0  4  6  4  -4  2  7  8  0  -5)
columns   = (1  2  4  1  2  5  3  4  5  1  3  4  2  3  5)
rowIndex  = (1  4  7  10  13  16)
```

格納形式の制約 : スパースソルバー用の格納形式は、2 つの重要な制約に従わなければならない。

1) 行の非ゼロの値は、行で非ゼロが存在する順に (左から右に) *values* 配列に配置しなければならない。2) 対称または対称構造の行列で、対角成分を *values* 配列から省略してはならない。

2 の制約は、対角上にゼロを含む対称または対称構造の行列を扱う場合、ゼロ対角成分を *values* 配列で (省略しないで) 明示的に表現しなければならないことを意味する。

スパース BLAS レベル 2-3 のスパース格納形式

このセクションでは、インテル MKL の現在のバージョンでサポートしているスパース BLAS レベル 2 および 3 のスパースデータ構造の詳細について説明する。

CSR 形式

スパース行列用のインテル MKL の圧縮スパース行 (CSR) 形式は、*values*、*columns*、*pointerB*、および *pointerE* 配列と呼ばれる 4 つの配列からなる。次の表は、スパース行列 *A* の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

values *A* の非ゼロ成分を含む実数または複素数配列。*A* の非ゼロの値は、上で説明した行主体格納マッピングを使用して *values* 配列にマップされる。

columns 整数配列 *columns* の *i* 番目の成分は、*values*(*i*) の値を含む *A* の列の番号を格納する。

pointerB この整数配列の *j* 番目の成分は、*A* の行 *j* の最初の非ゼロ成分を含む *values* 配列へのインデックスを格納する。このインデックスは *pointerB*(*j*) - *pointerB*(1)+1 に等しいことに注意。

pointerE この整数配列は、*A* の行 *j* の最後の非ゼロ成分を含む *values* 配列へのインデックス *pointerE*(*j*)-*pointerB*(1) のような行インデックスを格納する。

values および *columns* 配列の長さは *A* の非ゼロの個数に等しい。*pointerB* と *pointerE* の配列の長さは *A* の行数に等しい。

前に定義した行列 *B*

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

は、CSR 形式では次のように表される。

表 A-4 例の行列に対する格納配列 (CSR 形式)

<i>values</i>	=	(1 -1 -3 -2 5 4 6 4 -4 2 7 8 -5)
<i>columns</i>	=	(1 2 4 1 2 3 4 5 1 3 4 2 5)
<i>pointerB</i>	=	(1 4 6 9 12)
<i>pointerE</i>	=	(4 6 9 12 14)

この格納形式は、NIST スパース BLAS ライブラリー [Rem05] で使用されている。

PARDISO ソフトウェアで利用できる、上で記述した格納形式 (「[PARDISO* ソルバーの格納形式](#)」参照) は、CSR 形式を変形したものである。PARDISO 形式では、すべての非ゼロ成分が連続して格納されなければならないという制約があった。すなわち、*J* 行の非ゼロ成分の集合は、*J-1* 行の非ゼロ成分の集合の直後にななければならない。

CSR 形式にはそのような制約はない。これは、例えば、同時に異なる部分行列を処理する必要があるときに有用である。この場合、必要なそれぞれの部分行列に対して *pointerB* と *pointerE* の配列を定義して、それらの配列を 1 つの *values* 配列へのポインターとすればよい。

表 A-2 の `rowIndex` 配列と 表 A-4 の `pointerB` および `pointerE` 配列を比較すると、
 $pointerB(i) = rowIndex(i)$ ($i=1,..5$ の場合)
 $pointerE(i) = rowIndex(i+1)$ ($i=1,..5$ の場合)
 となっていることが分かる。

このことから、`values`、`columns`、`pointerB` および `pointerE` を入力パラメーターとして持つルーチンは PARDISO に採用された形式で格納されたスパース行列に対しても適用できる可能性がある。例えば、次のインターフェイスを持つルーチン

```
Subroutine name_routine(... , values, columns, pointerB, pointerE, ...)
は、引数 values, columns, rowIndex を使用して次のように呼び出すことができる。
call name_routine(... , values, columns, rowIndex, rowindex(2), ...).
```



注： インテル MKL のスパース BLAS レベル 2 ライブラリーでは、CSR 形式の両方の変形に対するルーチンを用意している。

CSC 形式

圧縮スパース列形式 (CSC、*Harwell-Boeing* スパース行列形式と呼ばれることもある) は CSR に似た形式であるが、行の代わりに列が使用される。つまり、CSC 形式は転置行列に対する CSR 形式に等しい。

CSR 形式の場合と同様に、インテル MKL のスパース BLAS レベル 2 ライブラリーでは、CSC 形式の両方の型に対するルーチンを用意している。

PARDISO ソフトウェアで利用できる、この形式の変形は、`values`、`rows`、および `colIndex` 配列と呼ばれる 3 つの配列からなる。次の表で、これらの配列を説明する。

<code>values</code>	A の非ゼロ成分を含む実数または複素数配列。 A の非ゼロの値は、上で説明した列主体格納マッピングを使用して <code>values</code> 配列にマップされる。
<code>rows</code>	整数配列 <code>rows</code> の i 番目の成分は、 <code>values(i)</code> の値を含む A の行の番号を格納する。
<code>colIndex</code>	整数配列 <code>colIndex</code> の j 番目の成分は、 A の列 j の最初の非ゼロ成分を含む <code>values</code> 配列へのインデックスを格納する。

`values` と `rows` 配列の長さは A の非ゼロの個数に等しい。

例えば、次のスパース行列 B

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -2 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -4 & * & 2 & 7 & * \\ * & 8 & * & * & -5 \end{bmatrix}$$

は、PARDISO 用の CSC 形式では次のように表される。

表 A-5 例の行列に対する格納配列 (Harwell-Boeing 形式)

```
values = (1 -2 -4 -1 5 8 4 2 -3 6 7 4 -5)
rows   = (1 2 4 1 2 5 3 4 1 3 4 3 5)
colIndex = (1 4 7 9 12 14)
```

座標形式

座標形式は、スパース行列の表現として最も柔軟で簡潔な形式である。非ゼロ成分のみが対象とされ、各非ゼロ成分の座標が明示的に示される。多くの商用ライブラリーは、座標形式でスパース行列用の行列・ベクトル乗算をサポートしている。

インテル MKL の座標形式は、*values*、*rows*、および *column* 配列と呼ばれる 3 つの配列と、*A* の非ゼロ成分の数を示すパラメーター *nnz* からなる。3 つの配列の次元はすべて *nnz* である。次の表は、スパース行列 *A* の非ゼロ成分の値、行、および列の位置の点から配列を説明している。

<i>values</i>	任意の順序で <i>A</i> の非ゼロ成分を含む実数または複素数配列。
<i>rows</i>	整数配列 <i>rows</i> の <i>i</i> 番目の成分は、 <i>values</i> (<i>i</i>) の値を含む <i>A</i> の行の番号を格納する。
<i>columns</i>	整数配列 <i>columns</i> の <i>i</i> 番目の成分は、 <i>values</i> (<i>i</i>) の値を含む <i>A</i> の列の番号を格納する。

例えば、次のスパース行列 *C*

$$C = \begin{bmatrix} 1 & -1 & -3 & 0 & 0 \\ -2 & 5 & 0 & 0 & 0 \\ 0 & 0 & 4 & 6 & 4 \\ -4 & 0 & 2 & 7 & 0 \\ 0 & 8 & 0 & 0 & -5 \end{bmatrix}$$

は、座標形式では次のように表される。

表 A-6 例の行列に対する格納配列 (座標形式)

```
values = (1 -1 -3 -2 5 4 6 4 -4 2 7 8 -5)
rows   = (1 1 1 2 2 3 3 3 4 4 4 5 5)
columns = (1 2 3 1 2 3 4 5 1 3 4 2 5)
```

対角格納方式

行列 *A* に対角が含まれている場合、その構造を使用して非ゼロ成分の位置に必要な情報を削減できる。この格納方式は、行列が有限成分あるいは差分離散化から発生するようなアプリケーションにおいて特に有用である。インテル MKL の対角格納方式は、*values* と *distance* 配列と呼ばれる 2 つの配列、空でない対角の数を示すパラメーター *ndiag*、および呼び出す (サブ) プログラムで宣言されるリーディング・ディメンジョン *lval* からなる。次の表で、配列 *values* と *distance* を説明する。

<i>values</i>	実数または複素数の 2 次元配列。次元は <i>lval</i> と <i>ndiag</i> 。A の非ゼロ対角を含む。この格納形式の重要な点は、 <i>values</i> の各成分が元の行列の行に対応する行を保持していることである。このために、A の下三角部分にある対角は上から、上三角にある対角は下からパディングされる。 <i>distance(i)</i> の値は、対角 <i>i</i> でパディングされる成分の数である。
<i>distance</i>	整数配列。次元は <i>ndiag</i> 。整数配列 <i>distance</i> の <i>i</i> 番目の成分は、 <i>i</i> 番目の対角と主対角との間の距離を格納する。この距離は、対角が主対角の上にあるときは正、主対角の下にあるときは負である。主対角の距離はゼロである。

上で説明したスパース行列 *C* は、対角格納方式では次のように格納される。

distance = (-3 -1 0 1 2)

values =
$$\begin{bmatrix} * & * & 1 & -1 & -3 \\ * & -2 & 5 & 0 & 0 \\ * & 0 & 4 & 6 & 4 \\ -4 & 2 & 7 & 0 & * \\ 8 & 0 & -5 & * & * \end{bmatrix}$$

ここでアスタリスクはパディングされた成分を示す。

行列が対称、エルミート、またはスキュー対称の場合、上三角または下三角が格納されることは明らかである。

スパース対角表現をインテル MKL のスパース行列 - 行列 (または行列 - ベクトル) 乗算ルーチンで使用する場合、対角は任意の順序で格納できる。しかし、スパース対角表現をインテル MKL のスパース三角ソルバールーチンで使用する場合、*distance* 配列の全成分は昇順に格納されていなければならない。

スカイライン格納方式

スカイライン格納方式は、直接法スパースソルバーにおいて重要である。また、ピボット演算を必要としないコレスキー分解や LU 分解にも適している。

インテル MKL で利用できるスカイライン格納方式は、三角行列または行列の三角部分のみを格納できる。この変形は、*values* と *pointers* 配列と呼ばれる 2 つの配列からなる。次の表で、これらの配列を説明する。

<i>values</i>	スカラー配列。行列が下三角の場合、最初の非ゼロ成分から対角成分までの (対角成分を含む) A の各行の成分を格納する。行列が上三角の場合、最初の非ゼロ成分から対角成分までの (対角成分を含む) A の各列の成分を格納する。途中のゼロ成分も含まれる。
<i>pointers</i>	整数配列。次元は <i>m+1</i> 。ここで、 <i>m</i> は下三角では行 (上三角では列) の数である。 <i>pointers(i) - pointers(1)+1</i> は、行 (列) <i>i</i> の最初の非ゼロ成分の <i>values</i> における位置を指す。 <i>pointers(m+1)</i> の値は、値 <i>nmz+pointers(1)</i> に設定される。ここで、 <i>nmz</i> は配列 <i>values</i> の成分の数である。

インテル MKL のスパース BLAS は、スカイライン格納形式で処理するルーチンで一般行列をサポートしていない点に注意。

例えば、上で説明した行列 C の上三角は次のように格納される。

```
values = (1 -2 5 4 -4 0 2 7 8 0 0 -5)
pointers = (1 2 4 5 9 13)
```

また、行列 C の上三角は次のように格納される。

```
values = (1 -1 5 -3 0 4 6 7 4 0 -5)
pointers = (1 2 4 7 9 12)
```

この格納形式は、NIST スペース BLAS ライブラリー [\[Rem05\]](#) で使用されている。

区間線形方程式

区間

区間とは、実数軸 \mathbf{R} の密で連続した部分集合である。したがって、それは2つの数値すなわちその下端と上端(それぞれ、左端と右端と呼ばれる場合もある)で完全に定義でき、 $[a, b]$ で区間 $\{x \in \mathbf{R} | a \leq x \leq b\}$ を表記できる。すべての実数区間の集合は \mathbf{IR} で表記する。数学的表記法では、区間の上端および下端を取り出すには次のように表記する。

$$\inf[a, b] = a, \quad \sup[a, b] = b.$$

以下の説明で、区間と区間オブジェクトは太字で表記し、下線と上線で区間 $\mathbf{x} = [\underline{x}, \bar{x}]$ の下端と上端を示す。

各区間は次の中点、

$$\text{mid } \mathbf{a} = \frac{1}{2}(\bar{\mathbf{a}} + \underline{\mathbf{a}}),$$

および半径、

$$\text{rad } \mathbf{a} = \frac{1}{2}(\bar{\mathbf{a}} - \underline{\mathbf{a}})$$

で一意的に決定できる。後者は幅 $\text{wid } \mathbf{a} = (\bar{\mathbf{a}} - \underline{\mathbf{a}})$ と等価である。 $[a, a]$ の形で下端と上端が等しいもの、すなわち、幅0の区間は、退化した区間、点の区間、または薄い区間と呼ばれ、通常の実数と一致する。したがって $\mathbf{R} \subset \mathbf{IR}$ と言える。逆に、幅が0でない区間は、厚い区間と呼ばれる。

区間は集合なので、例えば、包含、共通集合などの集合論での相互の関係、操作が適用できる。特に点 $t \in \mathbf{R}$ は、 $\mathbf{a} \leq t \leq \bar{\mathbf{a}}$ の場合、区間 \mathbf{a} の成分である ($t \in \mathbf{a}$ と書く)。さらに、包含は次のように定義できる。 $\underline{\mathbf{a}} \geq \underline{\mathbf{b}}$ かつ $\bar{\mathbf{a}} \leq \bar{\mathbf{b}}$ の場合に限り $\mathbf{a} \subseteq \mathbf{b}$ である。

区間と区間オブジェクト(ベクトル、行列、その他)は、ある変数の変動可能な下端と上端のみが既知であるとき、いわゆる境界のある不確定性とあいまい度を表現するのに便利なツールである。この意味で、区間は、定量的不確定性を記述する確率論やファジー論のアプローチと並んでその代換案を提供するものである。

加算、減算、乗算、および除算といった算術的な操作も、次の基本原理に従って区間に拡張できる。

$$\mathbf{a} * \mathbf{b} := \{a * b | a \in \mathbf{a}, b \in \mathbf{b}\}, \quad * \in \{+, -, \cdot, / \}, \quad (1)$$

これはいわゆる古典区間算術を定義することを可能にする。空の区間 $[\emptyset]$ がしばしばコンピューター区間算術構造に導入されることに注意。

実数軸 \mathbf{R} と同様に、複素数 \mathbf{C} においても区間を使用して、複素数の不確定性を表現することができる。複素数は、 $z = a + ib$ 形式で表されるオブジェクトである。ここで、 $a, b \in \mathbf{R}$ 、 i は虚数単位の記号で $i^2 = -1$ である。

区間が $\mathbf{a}, \mathbf{b} \in \mathbf{IR}$ である場合、複素平面上の集合

$$\{z = a + ib \in \mathbf{C} | a \in \mathbf{a}, b \in \mathbf{b}\}$$

は、複素数 $\mathbf{a} + i\mathbf{b}$ の矩形領域と呼ばれる。

また、別の一般的な複素区間として、複素平面 \mathbf{C} 上の円領域(円板領域とも呼ぶ)がある。円領域 $\langle \mathbf{c}, r \rangle$ ($\mathbf{c} \in \mathbf{C}$, $r \in \mathbf{R}$ 、および $r \geq 0$) は、次のように定義される複素平面上の集合である。

$$\{z \in \mathbf{C} \mid |z - c| \leq r\}$$

すべての矩形領域の集合は $\mathbf{IC}_{\text{rect}}$ と表記され、すべての円領域の集合は $\mathbf{IC}_{\text{circ}}$ と表記される、複素数演算の形式が指定されていない場合、または複素数演算の形式がコンテキストから明らかな場合、複素区間に対して一般表記 \mathbf{IC} が使用される。複素区間が $\mathbf{u}, \mathbf{v} \in \mathbf{IC}$ の場合、算術演算は (1) を弱めるプロパティに従って定義される。

$$\mathbf{u} * \mathbf{v} \supseteq \{u * v \mid u \in \mathbf{u}, v \in \mathbf{v}\}, * \in \{+, -, \cdot, /\}$$
 (2)

一般的に、(2) において包含する代わりに厳密に同等とすることは不可能だが、(2) によって導入される複素区間演算は、アプリケーションで非常に役立ち、複素区間分析を行うことができる。

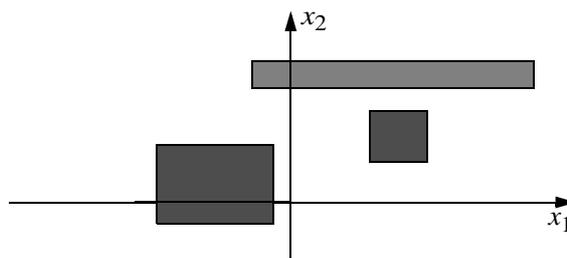
区間ベクトルと行列

区間ベクトルとは垂直 (列ベクトル) あるいは水平 (行ベクトル) に並べた順序付きの区間の組である。そこで $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ が区間とすると、

$$\mathbf{a} = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{a}_n \end{pmatrix} \text{ は列ベクトルであり、}$$

$\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ は行ベクトルである。

すべての区間 n 次ベクトルの集合は、 \mathbf{IR}^n と表記される。



区間ベクトルは、側面が座標軸と並行であるという幾何学的なイメージ (すなわち、区間 \mathbf{R}^n の長方形の箱) で連想できる。この理由から、区間ベクトルはしばしばボックスと呼ばれる。

区間行列は、区間からなる長方形の表である。

$$A := \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix},$$

あるいは $A = (a_{ij})$ 。区間ベクトルは、大きさが $n \times 1$ (列ベクトル) または $1 \times n$ (行ベクトル) のいずれかの区間行列と同一である。区間 $m \times n$ 行列の集合は $\mathbf{IR}^{m \times n}$ と表記される。区間ベクトルと行列の間の算術演算は、(1) 式を一般化する関係に基づいて導入できる ([Alefheld83], [Neumaier90] を参照)。

区間正方行列 $A \in \mathbf{IR}^{n \times n}$ は、すべての点行列 $A \in A$ が正則 (非特異) すなわち、非ゼロの決定要素を持つ場合に限り、*正則* (非特異) と呼ばれる。そうでない場合、区間行列 $A \in \mathbf{IR}^{n \times n}$ は *特異* と呼ばれ、少なくとも 1 つの特異な点行列を含むことを意味する。

一般に、区間行列が正則か特異かを見分けることは、この問題を妥当な時間内に解く比較的単純 (多項式の複雑度) なアルゴリズムがないため、NP 困難な問題とされている。

実用的な必要性からは、広い範囲の区間行列に対して正則性を調べるための、いくつかの実行可能で十分な基準を持つことが重要である。インテル MKL には、区間行列の正則性 / 特異性をテストするために **Ris-Beeck** スペクトル判定、**Rump** 特異値判定、および **Rohn-Rex** 特異値判定を実装したルーチンが用意されている。

ときどき、区間行列について関連するプロパティ (*強固な正則性*) を調べる必要が生じる。強固な正則性には、区間行列とその中点逆行列との積が正則であることが求められる。?gerbr ルーチンは、その出力パラメーター *sr* の値を判定して強固な正則性を調べることができる。

区間線形方程式

次の形式の線形代数方程式

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m, \end{cases} \quad (2)$$

あるいは、簡潔に、

$$Ax = b$$

(ここで A は $m \times n$ 次元の行列、 b は m 次のベクトルである) を解くことは、科学工学分野における主要問題の 1 つである。 a_{ij} と b_i が確定できず、それぞれ、区間 a_{ij} と b_i に属している場合、方程式は *区間線形方程式* と呼ばれ、次のように記述される。

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2, \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ \quad \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \quad \cdot \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n, \end{array} \right. \quad (3)$$

ここで、 a_{ij} と b_i は区間である。簡潔な形で記述すると次のようになる。

$$Ax = b \quad (4)$$

ここで、区間行列 $A = (a_{ij})$ および右辺ベクトル $b = (b_i)$ である。

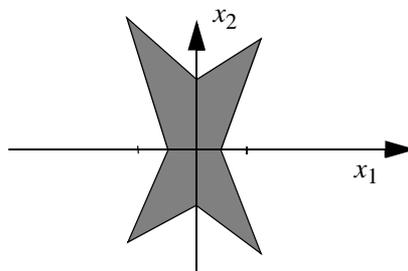
区間線形方程式 (3)–(4) は、同じ形の点線形方程式

$Ax = b$ の集合とも考えられる。ここで、パラメーター a_{ij} と b_i は $a_{ij} \in \mathbf{a}_{ij}$ と $b_i \in \mathbf{b}_i$ を満たす。

a_{ij} と b_i が区間 \mathbf{a}_{ij} と \mathbf{b}_i 内で変化するとき、 $A = (a_{ij})$ および $b = (b_i)$ の対応する点方程式 $Ax = b$ の解は \mathbf{R}^n 空間での集合を形成する。すなわち、

$$\Xi(A, b) := \{x \in \mathbf{R}^n \mid (\exists A \in \mathbf{A})(\exists b \in \mathbf{b})(Ax = b)\}. \quad (5)$$

(5) の集合は、 $A \in \mathbf{A}$ および $b \in \mathbf{b}$ のすべての点方程式 $Ax = b$ の解から形成されており、区間線形方程式 (3)–(4) の解集合と呼ばれる。通常、個々の a_{ij} と b_i ($1 \leq i, j \leq n$) の解集合は \mathbf{R}^n における多面体であり、ときには下の図のように星形をしている。



注： 区間線形方程式には、その他のさまざまな解集合が存在するため、上に記述された集合 $\Xi(A, b)$ は、しばしば、*結合解集合*と呼ばれる ([Shary02] を参照)。

解集合の複雑さは次元 n のべき乗で増加するため、その正確な記述は次元 n が大きくなると現実的には不可能である。また、ほとんどの場合、そのような正確な記述は実際には必要ない。通常は、単に解集合の推定を計算するだけでよい。実際に最も一般的に使用されるのは、次に述べる外からの (超集合による) 区間推定問題である。

次の区間線形方程式について、 $Ax = b$
解集合 $\Xi(A, b)$ の区間エンクロージャを求めよ。 (6)

しばしば、次のように問題 (6) の成分ごとの形が考えられる。

$$\begin{aligned} & \text{次の区間線形方程式について、} \mathbf{Ax} = \mathbf{b} \\ & \text{下から } \min\{x_v | x \in \Xi(\mathbf{A}, \mathbf{b})\} \text{ の推定を求めよ。} \\ & \text{上から } \max\{x_v | x \in \Xi(\mathbf{A}, \mathbf{b})\} \text{ の推定を求めよ (} v = 1, 2, \dots, n \text{)。} \end{aligned} \quad (7)$$

特に、インテル MKL の `?gepps` ルーチンは、この問題設定で処理される。

問題 (6)–(7) は、現代の区間解析において最もポピュラーな問題の 1 つである。この問題に関する多数の文献がある (例えば、[Alefeld83]、[Kearfott96]、[Neumaier90])。

つまり、区間線形方程式を解くことは、ここでは区間線形方程式 (3)–(4) の解集合に対する外からの区間推定を計算することと理解されている。方程式の行列 \mathbf{A} は通常、正方で非特異であると仮定される。

古典計算線形代数とは異なり、区間線形方程式は一般に、計算が非常に困難であることがわかっている。(6) の解の最適 (最小) 区間エンクロージャを計算すること、または等価的に (7) の解集合の正確な推定を計算することは、方程式の区間の幅や行列 \mathbf{A} の非ゼロ成分の構造に制限がない場合は、NP 困難な問題である ([Kreinovich97] を参照)。さらに、解への要求を緩和し、精度が前もって定義した絶対的 / 相対的な精度内になるように解集合の推定を計算したとしても、問題はやはり NP 困難なままである。

実際的な見地から NP 困難が意味するところは、一般的な問題では、高い確率で、問題のサイズに関して多項式時間で解くことができないことである。

この理由により、インテル MKL では、区間線形方程式用の数値計算アルゴリズムは、結果の精度が保証されているかどうかによって 2 つのクラスに分かれる。「高速」アルゴリズムは、高速に妥当な時間で解集合のエンクロージャを計算するが、精度の保証はない。「最適」あるいは「正確」アルゴリズムは、実行の完了に時間がかかるが、得られた結果は精度の要求を満たすものである。

インテル MKL には、両方の種類のアルゴリズムを実装した区間ソルバールーチンが用意されている。例えば、区間 Gauss 法、区間 Householder 法、Hansen-Blik-Rohn 法、および Krawczyk 反復のような高速手法は、それぞれ、`?gegas`、`?gehss`、`?gehbs`、および `?gekws` ルーチンとして実装されている。パラメーター分割法 (PPS 法) は、正確な手法の例として `?gepps` ルーチンで実装されている。`?trtrs` ルーチンは、非常に特殊な行列構造のため、両方のカテゴリーに含まれる。

前処理

区間線形方程式 (4) の前処理は、システムの特性を改善するために行列 \mathbf{A} と右辺ベクトル \mathbf{b} とを点行列で掛け合わせる。方程式 $\mathbf{Ax} = \mathbf{b}$ は次の方程式で置き換えられる。

$$(\mathbf{CA})x = \mathbf{Cb}$$

ここで \mathbf{C} はある正方点行列である。前処理は、古典計算線形代数において広く使用されており、多くの区間ソルバー・アルゴリズム (例えば、区間 Gauss 法、区間 Gauss-Seidel 法など) も使用の前に適切な前処理を必要とする。

区間線形方程式用に広く使用されている前処理の 1 つは、中点行列の逆行列による前処理で、しばしば *中点逆転前処理* と呼ばれる。インテル MKL では、中点逆転前処理は `?gemip` ルーチンで実装されている。

区間行列の逆転

区間正方行列 \mathbf{A} に対して、 \mathbf{A} に含まれる全点行列の逆行列の集合のエンクロージャは、*逆区間行列* \mathbf{A}^{-1} と呼ばれる。すなわち、

$$\mathbf{A}^{-1} \supseteq \{A^{-1} \mid A \in \mathbf{A}\}。$$

古典線形代数では、正方非特異行列 A を持つ線形代数方程式 $Ax = b$ の解は逆行列 A^{-1} と右辺ベクトルの積として表現される。すなわち、 $x = A^{-1}b$ 。

区間解析でも、同様の積 $\mathbf{A}^{-1}\mathbf{b}$ で区間線形方程式 $\Xi(\mathbf{A}, \mathbf{b})$ の解集合 $\mathbf{Ax} = \mathbf{b}$ へのエンクロージャが求められる。しかし、この方法は通常かなりの過大推定を引き起こすため推奨されない。解集合の外からの推定に特化したプロシージャの使用が好ましい。

それでもなお、逆区間行列の密なエンクロージャを計算することは、方程式の感度解析などには欠かせないものである。

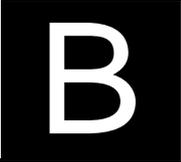
逆区間行列の計算は、次の区間行列方程式の解集合のエンクロージャを見つけることとして実行される。

$\mathbf{AY} = \mathbf{I}$ 、ここで \mathbf{I} は単位行列で、

区間線形方程式のいずれかの手法を n 回 (行列 \mathbf{Y} の各列ごとに) 適用して実行される。

Schulz 法 ([\[Herzberger94\]](#) を参照) のような、逆区間行列を見出すための直接反復プロシージャもある。これは、インテル MKL に `?geszi` ルーチンとして含まれている。

ルーチンおよび関数の 引数



BLAS ルーチンの主要な引数は、ベクトルと行列である。一方、VML 関数はベクトル引数のみで動作する。
この後の各セクションでは、これらの引数それぞれについて、例を挙げながら説明する。

BLAS のベクトル引数

ベクトル引数は、1次元の配列で渡される。配列の次元(長さ)とベクトルの増分は、整数の変数として渡される。長さによって、ベクトル内の成分の数が決まる。増分(ストライドとも呼ばれる)によって、ベクトルの成分間の間隔と、ベクトルが渡される配列内の成分の順序が決まる。

長さが n で増分が $incx$ のベクトルは、その値が次のように定義される1次元の配列 x で渡される。

$x(1), x(1+|incx|), \dots, x(1+(n-1)*|incx|)$

$incx$ が正の場合は、配列 x 内の成分は昇順に格納される。 $incx$ が負の場合は、配列 x 内の成分は降順に格納され、また最初の成分は $x(1+(n-1)*|incx|)$ として定義される。 $incx$ がゼロの場合は、ベクトルの全成分は同じ値 $x(1)$ をとる。ベクトルを格納する1次元配列の次元は、常に、

$idimx = 1 + (n-1)*|incx|$ 以上でなければならない。

例 B-1 1次元の実数配列

$x(1:7)$ が、次の1次元の実数配列とする。

$x = (1.0, 3.0, 5.0, 7.0, 9.0, 11.0, 13.0)$

$incx = 2$ かつ $n = 3$ の場合は、このベクトル引数の成分を始めから終わりまで並べると、 $(1.0, 5.0, 9.0)$ になる。

$incx = -2$ かつ $n = 4$ の場合は、このベクトル成分を始めから終わりまで並べると、 $(13.0, 9.0, 5.0, 1.0)$ になる。

$incx = 0$ かつ $n = 4$ の場合は、このベクトル成分を始めから終わりまで並べると、 $(1.0, 1.0, 1.0, 1.0)$ になる。

B

行列の 1 次元下部構造 (行、列、対角成分など) は、開始アドレスと増分を指定してベクトル引数として渡せる。Fortran では、 m 行 n 列の行列は、列主体の順序に基づいて格納される。つまり、同じ列内の成分間の増分は 1、同じ行内の成分間の増分は m 、同じ対角成分上の成分間の増分は $m + 1$ になる。

例 B-2 2次元の実行列

a が、REAL A(5, 4) として宣言された 5×4 の実行列とする。

a の 3 番目の列を 2.0 でスケールするには、BLAS ルーチン sscal を次の呼び出しシーケンスで使用する。

```
call sscal (5, 2.0, a(1,3), 1).
```

2 番目の行をスケールするには、次の構文を使用する。

```
call sscal (4, 2.0, a(2,1), 5).
```

A の主対角成分を 2.0 でスケールするには、次の構文を使用する。

```
call sscal (5, 2.0, a(1,1), 6).
```



注: デフォルトのベクトル引数は 1 とみなされる。

VML のベクトル引数

VML 数学関数のベクトル引数は、単位ベクトル増分を使用する 1 次元配列に渡される。つまり、長さ n のベクトルは、 $a[0], a[1], \dots, a[n-1]$ として値が定義される配列 a に連続して渡される (C インターフェイスの場合)。
他の増分を使用する配列、またはより複雑なインデックスを使用する配列を収容するため、VML では、`pack/unpack` 関数を補助的に用意している。`pack/unpack` 関数は、配列の成分を収集して連続するベクトルを生成し、計算の完了後、それらを分散するものである。

一般に、ベクトル成分が 1 次元配列 a に次のように格納されており、

$$a[m0], a[m1], \dots, a[mn-1]$$

配列 y に次のように再グループ化する必要がある場合、

$$y[k0], y[k1], \dots, y[kn-1],$$

VML `pack/unpack` 関数では、以下のいずれかのインデックス方式を使用できる。

正増分インデックス

$$kj = \text{incy} * j, \quad mj = \text{inca} * j, \quad j = 0, \dots, n-1$$

制約: $\text{incy} > 0, \text{inca} > 0$

例えば、 $\text{incy} = 1$ を設定すると、配列の成分が収集されて連続するベクトルが生成される。

この方式は、負とゼロの増分が許可されない点を除けば、BLAS で使用される方式と同じである。

インデックス・ベクトル・インデックス

$$kj = \text{iy}[j], \quad mj = \text{ia}[j], \quad j = 0, \dots, n-1,$$

ia と iy は長さ n の配列であり、それぞれ入力配列 a 用のインデックス・ベクトルと出力配列 y 用のインデックス・ベクトルを格納する。

マスク・ベクトル・インデックス

インデックス kj, mj は、以下の条件を満たす。

$$my[kj] \neq 0, \quad ma[mj] \neq 0, \quad j = 0, \dots, n-1,$$

ma と my は、それぞれ入力配列 a 用のマスクベクトル、出力配列 y 用のマスクベクトルを格納する配列である。

行列引数

インテル[®] マス・カーネル・ライブラリー・ルーチンの行列引数は、以下に挙げる格納形式を使用して、1 次元または 2 次元の配列に格納できる。

- 通常のフル格納 (2 次元配列に格納)
- エルミート行列、対称行列、または三角行列用の圧縮格納 (1 次元配列に格納)
- 帯行列用の帯格納 (2 次元配列に格納)

フル格納は、次のような明確な形式である。行列 A は、2次元配列 a に格納される。このとき、行列の成分 a_{ij} は、配列の成分 $a(i, j)$ に格納される。

行列が、**三角行列**(引数 $uplo$ に指定した値に従って、上三角行列または下三角行列) の場合は、関連する三角成分だけが格納される。配列の残りの成分は、設定する必要はない。

対称行列またはエルミート行列を処理するルーチンを使用すると、行列の上三角または下三角のいずれかを、配列の対応する成分に格納できる。

$uplo = 'U'$ の場合は、 $i \leq j$ に対して、 a_{ij} が $a(i, j)$ に格納される。
 a の他の成分は、設定する必要はない。

$uplo = 'L'$ の場合は、 $j \leq i$ に対して、 a_{ij} が $a(i, j)$ に格納される。
 a の他の成分は、設定する必要はない。

圧縮格納を使用すると、対称行列、エルミート行列、または三角行列をよりコンパクトに格納できる。関連する三角部分(この場合も、引数 $uplo$ で指定)は、列ごとに1次元配列 ap にパックされる。

$uplo = 'U'$ の場合は、 $i \leq j$ に対して、 a_{ij} が $ap(i+j(j-1)/2)$ に格納される。

$uplo = 'L'$ の場合は、 $j \leq i$ に対して、 a_{ij} が $ap(i+(2*n-j)*(j-1)/2)$ に格納される。

LAPACK ルーチンの説明では、圧縮形式の行列を保持する配列は、名前の末尾に p が付いている。

帯格納では、 k_l 個の非ゼロの劣対角成分と k_u 個の優対角成分を持つ m 行 n 列の帯行列は、 (k_l+k_u+1) 行 n 列の2次元配列 ab にコンパクトに格納される。行列の各列は配列の対応する列に、行列の各対角成分は配列の各行にそれぞれ格納される。したがって、

a_{ij} は、 $\max(1, j-k_u) \leq i \leq \min(n, j+k_l)$ に対して、 $ab(k_u+1+i-j, j)$ に格納される。

帯格納形式は、 k_l と k_u が行列のサイズ n よりはるかに小さい場合にのみ使用する。ルーチンは、 k_l と k_u がどのような値であっても正常に動作するが、行列が実際に帯になっていない場合は、帯格納を使用すると効率が低下する。

帯格納方式の例を以下に示す

($m = n = 6$, $k_l = 2$, $k_u = 1$ の場合)。

* でマークされた配列の成分はルーチンで使用されない。

帯行列 A	A の帯格納
$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{bmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{matrix}$

一般帯行列を LU 因子分解用に入力として与える場合は、行の交換の結果として埋め込みによって生成される優対角成分をさらに k_l 個だけ格納するためのスペースがなくてはならない。つまり、行列は上記の形式で格納されるが、 $k_l + k_u$ 個の優対角成分を持つことにならない。したがって、

a_{ij} は、 $\max(1, j - k_u) \leq i \leq \min(n, j + k_l)$ に対して、 $ab(k_l + k_u + 1 + i - j, j)$ に格納される。

LU 因子分解の帯格納方式の例を以下に示す
($m = n = 6, k_l = 2, k_u = 1$ の場合)。

帯行列 A	A の帯格納
$a_{11} \ a_{12} \ 0 \ 0 \ 0 \ 0$	* * * + + +
$a_{21} \ a_{22} \ a_{23} \ 0 \ 0 \ 0$	* * + + + +
$a_{31} \ a_{32} \ a_{33} \ a_{34} \ 0 \ 0$	* $a_{12} \ a_{23} \ a_{34} \ a_{45} \ a_{56}$
$0 \ a_{42} \ a_{43} \ a_{44} \ a_{45} \ 0$	$a_{11} \ a_{22} \ a_{33} \ a_{44} \ a_{55} \ a_{66}$
$0 \ 0 \ a_{53} \ a_{54} \ a_{55} \ a_{56}$	$a_{21} \ a_{32} \ a_{43} \ a_{54} \ a_{65} \ *$
$0 \ 0 \ 0 \ a_{64} \ a_{65} \ a_{66}$	$a_{31} \ a_{42} \ a_{53} \ a_{64} \ * \ *$

* でマークされた配列の成分はルーチンで使用されない。+ でマークされた成分は入力時に設定する必要はないが、結果を格納するために LU 因子分解ルーチンで必要となる。入力配列は、終了時に LU 因子分解の各成分で次のように上書きされる。

*	*	*	u_{14}	u_{25}	u_{36}
*	*	u_{13}	u_{24}	u_{35}	u_{46}
*	u_{12}	u_{23}	u_{34}	u_{45}	u_{56}
u_{11}	u_{22}	u_{33}	u_{44}	u_{55}	u_{66}
m_{21}	m_{32}	m_{43}	m_{54}	m_{65}	*
m_{31}	m_{42}	m_{53}	m_{64}	*	*

ここで u_{ij} は上三角行列 U の各成分であり、 m_{ij} は因子分解で使用される乗数である。

三角帯行列は、上三角行列の場合は $k_l = 0$ 、下三角行列の場合は $k_u = 0$ として同じ形式で格納される。 k 個の劣対角成分または優対角成分を持つ対称帯行列またはエルミート帯行列に対しては、引数 $uplo$ で指定したとおりに、上三角行列または下三角行列だけを格納しなければならない。

$uplo = 'U'$ の場合は、 $\max(1, j - k) \leq i \leq j$ に対して、 a_{ij} は $ab(k + 1 + i - j, j)$ に格納される。
 $uplo = 'L'$ の場合は、 $j \leq i \leq \min(n, j + k)$ に対して、 a_{ij} は $ab(1 + i - j, j)$ に格納される。

LAPACK ルーチンの説明では、帯格納で行列を保持する配列は、名前の末尾に *b* が付いている。

Fortran では、列を主体とする順序での格納方法が想定されている。つまり、同じ列の成分が続けて格納されることになる。

2次元配列の引数には、リーディング・ディメンジョン (同じ行内の成分間の格納位置の数を指定する)、行数、列数の3つの特性が関連付けられるのが普通である。フル格納の行列に対しては、配列のリーディング・ディメンジョンは、行列内の行数と同じ大きさ以上でなければならない。

文字転置パラメーターは、行列引数を通常形式と転置形式のどちらで使用するか、あるいは、複素行列の場合は行列の共役転置を使用するかどうかを指示するために頻繁に使用される。

これら3つのケースに対する転置パラメーターの値は、次のようになる。

'N' または 'n' 通常形式 (共役や転置は使用しない)

'T' または 't' 転置

'C' または 'c' 共役転置

例 B-3 2次元複素配列

A (1:5, 1:4) は、次の行列で表される2次元の複素配列とする。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) & (1.3, 0.13) & (1.4, 0.14) \\ (2.1, 0.21) & (2.2, 0.22) & (2.3, 0.23) & (2.4, 0.24) \\ (3.1, 0.31) & (3.2, 0.32) & (3.3, 0.33) & (3.4, 0.34) \\ (4.1, 0.41) & (4.2, 0.42) & (4.3, 0.43) & (4.4, 0.44) \\ (5.1, 0.51) & (5.2, 0.52) & (5.3, 0.53) & (5.4, 0.54) \end{bmatrix}$$

transa が転置パラメーター、*m* が行数、*n* が列数、*lda* がリーディング・ディメンジョンとする。このとき、

transa = 'N'、*m* = 4、*n* = 2、*lda* = 5 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (1.2, 0.12) \\ (2.1, 0.21) & (2.2, 0.22) \\ (3.1, 0.31) & (3.2, 0.32) \\ (4.1, 0.41) & (4.2, 0.42) \end{bmatrix}$$

transa = 'T'、*m* = 4、*n* = 2、*lda* = 5 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (2.1, 0.21) & (3.1, 0.31) & (4.1, 0.41) \\ (1.2, 0.12) & (2.2, 0.22) & (3.2, 0.32) & (4.2, 0.42) \end{bmatrix}$$

transa = 'C'、*m* = 4、*n* = 2、*lda* = 5 の場合は、
行列引数は次のようになる。

$$\begin{bmatrix} (1.1, -0.11) & (2.1, -0.21) & (3.1, -0.31) & (4.1, -0.41) \\ (1.2, -0.12) & (2.2, -0.22) & (3.2, -0.32) & (4.2, -0.42) \end{bmatrix}$$

リーディング・ディメンジョンの値として、2次元配列の宣言で指定した行数とは異なる値を使用する場合は、特に注意が必要である。例えば、上記の配列 *A* を COMPLEX A (5, 4) として宣言するものとする。

このとき、*transa* = 'N'、*m* = 3、*n* = 4、*lda* = 4 の場合は、行列引数は次のようになる。

$$\begin{bmatrix} (1.1, 0.11) & (5.1, 0.51) & (4.2, 0.42) & (3.3, 0.33) \\ (2.1, 0.21) & (1.2, 0.12) & (5.2, 0.52) & (4.3, 0.43) \\ (3.1, 0.31) & (2.2, 0.22) & (1.3, 0.13) & (5.3, 0.53) \end{bmatrix}$$

コード例

C

本付録では、インテル[®] MKL のルーチンおよび関数を使用するコード例を示す。
このコード例には、Fortran と C の両方の例が用意されている。

以下のセクションが含まれる。

- [BLAS のコード例](#)
- [PARDISO のコード例](#)
- [直接法スパースソルバーのコード例](#)
- [反復法スパースソルバーのコード例](#)
- [フーリエ変換関数のコード例](#)
- [区間連立線形方程式ソルバーのコード例](#)
- [PDE サポートのコード例](#)

関数パラメーターと演算に関する詳細は、本マニュアルのそれぞれの章を参照のこと。

BLAS のコード例

例 C-1 BLAS レベル 1 の関数の使用法

次の例は、BLAS レベル 1 の関数 `sdot` の呼び出し方法を表す。この関数は、2 つの単精度実ベクトル x と y のスカラー積を計算するベクトル-ベクトル演算を実行する。

パラメーター

<code>n</code>	ベクトル x と y の次数を指定する。
<code>incx</code>	x の成分に対する増分を指定する。
<code>incy</code>	y の成分に対する増分を指定する。

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot

n = 5
incx = 2
incy = 1

do i = 1, 10
  x(i) = 2.0e0
  y(i) = 1.0e0
end do
```

```
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

このプログラムを実行すると、次の行が出力される。

```
SDOT = 10.000
```

例 C-2 BLAS レベル 1 のルーチンの使用法

次の例は、BLAS レベル 1 のルーチン `scopy` の呼び出し方法を表す。このルーチンは、2つの単精度実ベクトル x を y にコピーするベクトル-ベクトル演算を実行する。

パラメーター

<code>n</code>	ベクトル x と y の次数を指定する。
<code>incx</code>	x の成分に対する増分を指定する。
<code>incy</code>	y の成分に対する増分を指定する。

```
program copy_main
real x(10), y(10)
integer n, incx, incy, i
n = 3
incx = 3
incy = 1
do i = 1, 10
  x(i) = i
end do
call scopy (n, x, incx, y, incy)
print*, 'Y = ', (y(i), i = 1, n)
end
```

このプログラムを実行すると、次の行が出力される。

```
Y = 1.00000 4.00000 7.00000
```

例 C-3 BLAS レベル 2 のルーチンの使用法

次の例は、BLAS レベル 2 のルーチン `sger` の呼び出し方法を表す。このルーチンは、次に示す行列-ベクトル演算を実行する。

```
a := alpha*x*y' + a
```

パラメーター

<code>alpha</code>	スカラー <code>alpha</code> を指定する。
<code>x</code>	m 個の成分を持つベクトル。

y n 個の成分を持つベクトル。

a $m \times n$ の行列。

```
program ger_main
real a(5,3), x(10), y(10), alpha
integer m, n, incx, incy, i, j, lda
m = 2
n = 3
lda = 5
incx = 2
incy = 1
alpha = 0.5
do i = 1, 10
  x(i) = 1.0
  y(i) = 1.0
end do
do i = 1, m
  do j = 1, n
    a(i,j) = j
  end do
end do
call sger (m, n, alpha, x, incx, y, incy, a, lda)
print*, 'Matrix A: \'
do i = 1, m
  print*, (a(i,j), j = 1, n)
end do
end
```

このプログラムを実行すると、次に示す行列 a が出力される。

行列 A:

1.50000 2.50000 3.50000

1.50000 2.50000 3.50000

例 C-4 BLAS レベル 3 のルーチンの使用法

次の例は、BLAS レベル 3 のルーチン `ssymm` の呼び出し方法を表す。このルーチンは、次に示す行列 - 行列演算を実行する。

```
c := alpha*a*b' + beta*c
```

パラメーター

$alpha$ スカラー $alpha$ を指定する。

$beta$ スカラー $beta$ を指定する。

a 対称行列。

b $m \times n$ の行列。

c $m \times n$ の行列。

```
program symm_main
real a(3,3), b(3,2), c(3,3), alpha, beta
integer m, n, lda, ldb, ldc, i, j
character uplo, side
uplo = 'u'
side = 'l'
m = 3
n = 2
lda = 3
ldb = 3
ldc = 3
alpha = 0.5
beta = 2.0
do i = 1, m
  do j = 1, m
    a(i,j) = 1.0
  end do
end do
do i = 1, m
  do j = 1, n
    c(i,j) = 1.0
    b(i,j) = 2.0
  end do
end do
call ssymm (side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc)
print*, 'Matrix C: '
do i = 1, m
  print*, (c(i,j), j = 1, n)
end do
end
```

このプログラムを実行すると、次に示す行列 c が出力される。

行列 C:

```
5.00000 5.00000
5.00000 5.00000
5.00000 5.00000
```

例 C-5 BLAS レベル 1 の複素関数を C から呼び出す

次の例は、C プログラムから BLAS レベル 1 の複素関数 `zdotc()` を呼び出す方法を表す。この関数は、2 つの倍精度複素ベクトルのドット積を計算する。

この例では、複素数型のドット積が構造体 `c` に返される。

```
#define N 5
void main()
{

    int n, inca = 1, incb = 1, i;
    typedef struct{ double re; double im; } complex16;
    complex16 a[N], b[N], c;
    void zdotc();
    n = N;
    for( i = 0; i < n; i++ ){
        a[i].re = (double)i; a[i].im = (double)i * 2.0;
        b[i].re = (double)(n - i); b[i].im = (double)i * 2.0;
    }
    zdotc( &c, &n, a, &inca, b, &incb );
    printf( "The complex dot product is: ( %6.2f, %6.2f )\n", c.re, c.im );
}
```



注: C プログラムから BLAS を直接呼び出す代わりに、CBLAS インターフェイスの使用も可能である。これは、C から BLAS を呼び出す方法としてサポートされている。CBLAS の詳細については、[付録 D](#) を参照。CBLAS は、インテル[®] MKL に導入された BLAS (Basic Linear Algebra Subprograms) に対する C インターフェイスである。

PARDISO のコード例

このセクションでは、PARDISO 直説法ソルバーを使用してスパース行列の連立線形方程式を計算するコード例を示す。このソルバーの詳細は、本マニュアルの[第 8 章](#)を参照のこと。

スパース対称連立線形方程式の例

このセクションでは、PARDISO を使用して対称連立線形方程式の解を算出する 2 つの例 (Fortran および C) を示す。連立方程式 $Ax = b$ を解く。

$$A = \begin{bmatrix} 7.0 & 0.0 & 1.0 & 0.0 & 0.0 & 2.0 & 7.0 & 0.0 \\ 0.0 & -4.0 & 8.0 & 0.0 & 2.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 8.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 5.0 \\ 0.0 & 0.0 & 0.0 & 7.0 & 0.0 & 0.0 & 9.0 & 0.0 \\ 0.0 & 2.0 & 0.0 & 0.0 & 5.0 & 1.0 & 5.0 & 0.0 \\ 2.0 & 0.0 & 0.0 & 0.0 & 1.0 & -1.0 & 0.0 & 5.0 \\ 7.0 & 0.0 & 0.0 & 9.0 & 5.0 & 0.0 & 11.0 & 0.0 \\ 0.0 & 0.0 & 5.0 & 0.0 & 0.0 & 5.0 & 0.0 & 5.0 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

対称連立線形方程式の計算結果の例

ソルバーが正常に実行されると、 X の解は次のようになる。

```
Reordering completed ...
Number of nonzeros in factors = 30
Number of factorization MFLOPS = 0
Factorization completed ...
Solve completed ...
The solution of the system is
x(1) = -0.0418602013
x(2) = -0.00341312416
x(3) = 0.117250377
x(4) = -0.11263958
x(5) = 0.0241722445
x(6) = -0.10763334
x(7) = 0.198719673
x(8) = 0.190382964
```

例 C-6 対称連立線形方程式の例 (pardiso_sym.f)

```
C-----
C Example program to show the use of the "PARDISO" routine
C for symmetric linear systems
```

```
C-----
C This program can be downloaded from the following site:
C http://www.computational.unibas.ch/cs/scicomp
C
C (C) Olaf Schenk, Department of Computer Science,
C University of Basel, Switzerland.
C Email: olaf.schenk@unibas.ch
C
C-----

      PROGRAM pardiso_sym
      IMPLICIT NONE

C.. Internal solver memory pointer for 64-bit architectures
C.. INTEGER*8 pt(64)
C.. Internal solver memory pointer for 32-bit architectures
C.. INTEGER*4 pt(64)
C.. This is OK in both cases
      INTEGER*8 pt(64)
C.. All other variables
      INTEGER maxfct, mnum, mtype, phase, n, nrhs, error, msglvl
      INTEGER iparm(64)
      INTEGER ia(9)
      INTEGER ja(18)
      REAL*8 a(18)
      REAL*8 b(8)
      REAL*8 x(8)
      INTEGER i, idum
      REAL*8 waltimel, waltime2, ddum
C.. Fill all arrays containing matrix data.
      DATA n /8/, nrhs /1/, maxfct /1/, mnum /1/
      DATA ia /1,5,8,10,12,15,17,18,19/
      DATA ja
1 /1, 3, 6,7,
2 2,3, 5,
3 3, 8,
4 4, 7,
5 5,6,7,
6 6, 8,
7 7,
8 8/
      DATA a
1 /7.d0, 1.d0, 2.d0,7.d0,
```

```

2      -4.d0,8.d0,      2.d0,
3          1.d0,          5.d0,
4          7.d0,      9.d0,
5          5.d0,1.d0,5.d0,
6          -1.d0,      5.d0,
7          11.d0,
8          5.d0/

integer omp_get_max_threads
external omp_get_max_threads

C..
C.. Set up PARDISO control parameter
C..

do i = 1, 64
    iparm(i) = 0
end do

iparm(1) = 1 ! no solver default
iparm(2) = 2 ! fill-in reordering from METIS
iparm(3) = omp_get_max_threads() !numbers of processors, value of OMP_NUM_THREADS
iparm(4) = 0 ! no iterative-direct algorithm
iparm(5) = 0 ! no user fill-in reducing permutation
iparm(6) = 0 ! =0 solution on the first n compoments of x
iparm(7) = 16 ! default logical fortran unit number for output
iparm(8) = 9 ! numbers of iterative refinement steps
iparm(9) = 0 ! not in use
iparm(10) = 13 ! perturbe the pivot elements with 1E-13
iparm(11) = 1 ! use nonsymmetric permutation and scaling MPS
iparm(12) = 0 ! not in use
iparm(13) = 0 ! not in use
iparm(14) = 0 ! Output: number of perturbed pivots
iparm(15) = 0 ! not in use
iparm(16) = 0 ! not in use
iparm(17) = 0 ! not in use
iparm(18) = -1 ! Output: number of nonzeros in the factor LU
iparm(19) = -1 ! Output: Mflops for LU factorization
iparm(20) = 0 ! Output: Numbers of CG Iterations
error = 0 ! initialize error flag
msglvl = 0 ! don't print statistical information
mtype = -2 ! unsymmetric matrix symmetric, indefinite, no pivoting
C.. Initilize the internal solver memory pointer.This is only
C necessary for the FIRST call of the PARDISO solver.

do i = 1, 64

```

```

        pt(i) = 0
    end do
C.. Reordering and Symbolic Factorization, This step also allocates
C all memory that is necessary for the factorization
    phase = 11 ! only reordering and symbolic factorization
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    WRITE(*,*) 'Reordering completed ... '
    IF (error .NE. 0) THEN
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    END IF
    WRITE(*,*) 'Number of nonzeros in factors = ',iparm(18)
    WRITE(*,*) 'Number of factorization MFLOPS = ',iparm(19)
C.. Factorization.
    phase = 22 ! only factorization
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    WRITE(*,*) 'Factorization completed ... '
    IF (error .NE. 0) THEN
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    ENDIF
C.. Back substitution and iterative refinement
    iparm(8) = 2 ! max numbers of iterative refinement steps
    phase = 33 ! only factorization
    do i = 1, n
        b(i) = 1.d0
    end do
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, b, x, error)
    WRITE(*,*) 'Solve completed ... '
    WRITE(*,*) 'The solution of the system is '
    DO i = 1, n
        WRITE(*,*) ' x(',i,') = ', x(i)
    END DO
C.. Termination and release of memory
    phase = -1 ! release internal memory
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, ddum, idum, idum,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    END

```

例 C-7 対称連立線形方程式の例 (pardiso_sym.c)

```
/* ----- */
/* Example program to show the use of the "PARDISO" routine */
/* on symmetric linear systems */
/* ----- */
/* This program can be downloaded from the following site: */
/* http://www.computational.unibas.ch/cs/scicomp */
/* */
/* (C) Olaf Schenk, Department of Computer Science, */
/* University of Basel, Switzerland.*/
/* Email: olaf.schenk@unibas.ch */
/* ----- */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
extern int omp_get_max_threads();
/* PARDISO prototype.*/
extern int PARDISO
    (void *, int *, int *, int *, int *, int *,
     double *, int *, int *, int *, int *, int *,
     int *, double *, double *, int *);

int main( void ) {
    /* Matrix data.*/
    int n = 8;
    int ia[ 9] = { 1, 5, 8, 10, 12, 15, 17, 18, 19 };
    int ja[18] = { 1, 3, 6, 7,
                  2, 3, 5,
                  3, 8,
                  4, 7,
                  5, 6, 7,
                  6, 8,
                  7,
                  8 };
    double a[18] = { 7.0, 1.0, 2.0, 7.0,
                    -4.0, 8.0, 2.0,
                    1.0, 5.0,
                    7.0, 9.0,
                    5.0, 1.0, 5.0,
                    -1.0, 5.0,
                    11.0,
```

```

        5.0 };
int mtype = -2; /* Real symmetric matrix */
/* RHS and solution vectors.*/
double b[8], x[8];
int nrhs = 1; /* Number of right hand sides.*/
/* Internal solver memory pointer pt, */
/* 32-bit: int pt[64]; 64-bit: long int pt[64] */
/* or void *pt[64] should be OK on both architectures */
void *pt[64];
/* Pardiso control parameters.*/
int iparm[64];
int maxfct, mnum, phase, error, msglvl;
/* Auxiliary variables.*/
int i;
double ddum; /* Double dummy */
int idum; /* Integer dummy.*/
/* ----- */
/* ..Setup Pardiso control parameters.*/
/* ----- */
for (i = 0; i < 64; i++) {
    iparm[i] = 0;
}
iparm[0] = 1; /* No solver default */
iparm[1] = 2; /* Fill-in reordering from METIS */
/* Numbers of processors, value of OMP_NUM_THREADS */
iparm[2] = omp_get_max_threads();
iparm[3] = 0; /* No iterative-direct algorithm */
iparm[4] = 0; /* No user fill-in reducing permutation */
iparm[5] = 0; /* Write solution into x */
iparm[6] = 16; /* Default logical fortran unit number for output */
iparm[7] = 2; /* Max numbers of iterative refinement steps */
iparm[8] = 0; /* Not in use */
iparm[9] = 13; /* Perturb the pivot elements with 1E-13 */
iparm[10] = 1; /* Use nonsymmetric permutation and scaling MPS */
iparm[11] = 0; /* Not in use */
iparm[12] = 0; /* Not in use */
iparm[13] = 0; /* Output: Number of perturbed pivots */
iparm[14] = 0; /* Not in use */
iparm[15] = 0; /* Not in use */
iparm[16] = 0; /* Not in use */
iparm[17] = -1; /* Output: Number of nonzeros in the factor LU */

```

```
    iparm[18] = -1; /* Output: Mflops for LU factorization */
    iparm[19] = 0; /* Output: Numbers of CG Iterations */
    maxfct = 1; /* Maximum number of numerical factorizations.*/
    mnum = 1; /* Which factorization to use.*/
    msglvl = 0; /* Don't print statistical information in file */
    error = 0; /* Initialize error flag */

/* ----- */
/* ..Initialize the internal solver memory pointer.This is only */
/* necessary for the FIRST call of the PARDISO solver.*/
/* ----- */
    for (i = 0; i < 64; i++) {
        pt[i] = 0;
    }

/* ----- */
/* ..Reordering and Symbolic Factorization.This step also allocates */
/* all memory that is necessary for the factorization.*/
/* ----- */
    phase = 11;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during symbolic factorization: %d", error);
        exit(1);
    }
    printf("\nReordering completed ... ");
    printf("\nNumber of nonzeros in factors = %d", iparm[17]);
    printf("\nNumber of factorization MFLOPS = %d", iparm[18]);
/* ----- */
/* ..Numerical factorization.*/
/* ----- */
    phase = 22;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during numerical factorization: %d", error);
        exit(2);
    }
    printf("\nFactorization completed ... ");
/* ----- */
```

```
/* ..Back substitution and iterative refinement.*/
/* ----- */
    phase = 33;
    iparm[7] = 2; /* Max numbers of iterative refinement steps.*/
    /* Set right hand side to one.*/
    for (i = 0; i < n; i++) {
        b[i] = 1;
    }
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, b, x, &error);
    if (error != 0) {
        printf("\nERROR during solution: %d", error);
        exit(3);
    }
    printf("\nSolve completed ... ");
    printf("\nThe solution of the system is: ");
    for (i = 0; i < n; i++) {
        printf("\n x [%d] = % f", i, x[i] );
    }
    printf ("\n");
/* ----- */
/* ..Termination and release of memory.*/
/* ----- */
    phase = -1; /* Release internal memory.*/
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, &ddum, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    return 0;
}
```

スパース非対称連立線形方程式の例

このセクションでは、PARDISO を使用して非対称連立線形方程式の解を算出する 2 つの例 (Fortran および C) を示す。連立方程式 $Ax = b$ を解く。

$$A = \begin{bmatrix} 1.0 & -1.0 & 0.0 & -3.0 & 0.0 \\ -2.0 & 5.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 4.0 & 6.0 & 4.0 \\ -4.0 & 0.0 & 2.0 & 7.0 & 0.0 \\ 0.0 & 8.0 & 0.0 & 0.0 & -5.0 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

非対称連立線形方程式の計算結果の例

ソルバーが正常に実行されると、 X の解は次のようになる。

```
Reordering completed ...
Number of nonzeros in factors = 21
Number of factorization MFLOPS = 0
Factorization completed ...
Solve completed ...
The solution of the system is
x( 1) = -0.522321429
x( 2) = -0.00892857143
x( 3) = 1.22098214
x( 4) = -0.504464286
x( 5) = -0.214285714
```

例 C-8 非対称連立線形方程式の例 (pardiso_unsym.f)

```
*****
* Copyright(C) 2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : MKL DSS Fortran-77 example
*
```

```

*****
*
C-----
C Example program to show the use of the "PARDISO" routine
C for symmetric linear systems
C-----
C This program can be downloaded from the following site:
C http://www.computational.unibas.ch/cs/scicomp
C
C (C) Olaf Schenk, Department of Computer Science,
C University of Basel, Switzerland.
C Email: olaf.schenk@unibas.ch
C
C-----
      PROGRAM pardiso_unsym
      IMPLICIT NONE
C.. Internal solver memory pointer for 64-bit architectures
C.. INTEGER*8 pt(64)
C.. Internal solver memory pointer for 32-bit architectures
C.. INTEGER*4 pt(64)
C.. This is OK in both cases
      INTEGER*8 pt(64)
C.. All other variables
      INTEGER maxfct, mnum, mtype, phase, n, nrhs, error, msglvl
      INTEGER iparm(64)
      INTEGER ia(6)
      INTEGER ja(13)
      REAL*8 a(13)
      REAL*8 b(5)
      REAL*8 x(5)
      INTEGER i, idum
      REAL*8 waltimel, waltime2, ddum
C.. Fill all arrays containing matrix data.
      DATA n /5/, nrhs /1/, maxfct /1/, mnum /1/
      DATA ia /1,4,6,9,12,14/
      DATA ja
1 / 1, 2, 4,
2 1, 2,
3 3, 4, 5,
4 1, 3, 4,
5 2, 5/
      DATA a
1 /1.d0,-1.d0, -3.d0,
2 -2.d0, 5.d0,
3 4.d0, 6.d0, 4.d0,
4 -4.d0, 2.d0, 7.d0,
5 8.d0, -5.d0/
      integer omp_get_max_threads
      external omp_get_max_threads
C..
C.. Set up PARDISO control parameter
C..
      do i = 1, 64
         iparm(i) = 0
      end do

```

```

    iparm(1) = 1 ! no solver default
    iparm(2) = 2 ! fill-in reordering from METIS
    iparm(3) = omp_get_max_threads() ! numbers of processors, value of
OMP_NUM_THREADS
    iparm(4) = 0 ! no iterative-direct algorithm
    iparm(5) = 0 ! no user fill-in reducing permutation
    iparm(6) = 0 ! =0 solution on the first n components of x
    iparm(7) = 0 ! not in use
    iparm(8) = 9 ! numbers of iterative refinement steps
    iparm(9) = 0 ! not in use
    iparm(10) = 13 ! perturb the pivot elements with 1E-13
    iparm(11) = 1 ! use nonsymmetric permutation and scaling MPS
    iparm(12) = 0 ! not in use
    iparm(13) = 0 ! not in use
    iparm(14) = 0 ! Output: number of perturbed pivots
    iparm(15) = 0 ! not in use
    iparm(16) = 0 ! not in use
    iparm(17) = 0 ! not in use
    iparm(18) = -1 ! Output: number of nonzeros in the factor LU
    iparm(19) = -1 ! Output: Mflops for LU factorization
    iparm(20) = 0 ! Output: Numbers of CG Iterations
    error = 0 ! initialize error flag
    msglvl = 1 ! print statistical information
    mtype = 11 ! real unsymmetric
C.. Inititalize the internal solver memory pointer.This is only
C necessary for the FIRST call of the PARDISO solver.
    do i = 1, 64
        pt(i) = 0
    end do
C.. Reordering and Symbolic Factorization, This step also allocates
C all memory that is necessary for the factorization
    phase = 11 ! only reordering and symbolic factorization
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    WRITE(*,*) 'Reordering completed ... '
    IF (error .NE. 0) THEN
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    END IF
    WRITE(*,*) 'Number of nonzeros in factors = ',iparm(18)
    WRITE(*,*) 'Number of factorization MFLOPS = ',iparm(19)
C.. Factorization.
    phase = 22 ! only factorization
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    WRITE(*,*) 'Factorization completed ... '
    IF (error .NE. 0) THEN
        WRITE(*,*) 'The following ERROR was detected: ', error
        STOP
    ENDIF
C.. Back substitution and iterative refinement
    iparm(8) = 2 ! max numbers of iterative refinement steps
    phase = 33 ! only factorization
    do i = 1, n
        b(i) = 1.d0

```

```

    end do
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, a, ia, ja,
1 idum, nrhs, iparm, msglvl, b, x, error)
    WRITE(*,*) 'Solve completed ... '
    WRITE(*,*) 'The solution of the system is '
    DO i = 1, n
        WRITE(*,*) ' x(' ,i,') = ', x(i)
    END DO
C.. Termination and release of memory
    phase = -1 ! release internal memory
    CALL pardiso (pt, maxfct, mnum, mtype, phase, n, ddum, idum, idum,
1 idum, nrhs, iparm, msglvl, ddum, ddum, error)
    END

```

例 C-9 非対称連立線形方程式の例 (pardiso_unsym.c)

```

/*
*****
*
* Copyright(C) 2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : MKL DSS C example
*
*****
*/
/* ----- */
/* Example program to show the use of the "PARDISO" routine */
/* on symmetric linear systems */
/* ----- */
/* This program can be downloaded from the following site: */
/* http://www.computational.unibas.ch/cs/scicomp */
/* */
/* (C) Olaf Schenk, Department of Computer Science, */
/* University of Basel, Switzerland.*/
/* Email: olaf.schenk@unibas.ch */
/* ----- */
#include <stdio.h>
#include <stdlib.h>

```

```

#include <math.h>
extern int omp_get_max_threads();
/* PARDISO prototype.*/
#if defined(_WIN32) || defined(_WIN64)
#define pardiso_ PARDISO
#else
#define PARDISO pardiso_
#endif
extern int PARDISO
    (void *, int *, int *, int *, int *, int *,
     double *, int *, int *, int *, int *, int *,
     int *, double *, double *, int *);

int main( void ) {
    /* Matrix data.*/
    int n = 5;
    int ia[ 6] = { 1, 4, 6, 9, 12, 14 };
    int ja[13] = { 1, 2, 4,
                  1, 2,
                  3, 4, 5,
                  1, 3, 4,
                  2, 5 };
    double a[18] = { 1.0, -1.0, -3.0,
                    -2.0, 5.0,
                    4.0, 6.0, 4.0,
                    -4.0, 2.0, 7.0,
                    8.0, -5.0 };
    int mtype = 11; /* Real unsymmetric matrix */
    /* RHS and solution vectors.*/
    double b[5], x[5];
    int nrhs = 1; /* Number of right hand sides.*/
    /* Internal solver memory pointer pt, */
    /* 32-bit: int pt[64]; 64-bit: long int pt[64] */
    /* or void *pt[64] should be OK on both architectures */
    void *pt[64];
    /* Pardiso control parameters.*/
    int iparm[64];
    int maxfct, mnum, phase, error, msglvl;
    /* Auxiliary variables.*/
    int i;
    double ddum; /* Double dummy */
    int idum; /* Integer dummy.*/
    /* ----- */
    /* ..Setup Pardiso control parameters.*/
    /* ----- */
    for (i = 0; i < 64; i++) {
        iparm[i] = 0;
    }
    iparm[0] = 1; /* No solver default */
    iparm[1] = 2; /* Fill-in reordering from METIS */
    /* Numbers of processors, value of OMP_NUM_THREADS */
    iparm[2] = omp_get_max_threads();
    iparm[3] = 0; /* No iterative-direct algorithm */
    iparm[4] = 0; /* No user fill-in reducing permutation */
    iparm[5] = 0; /* Write solution into x */

```

```

    iparm[6] = 0; /* Not in use */
    iparm[7] = 2; /* Max numbers of iterative refinement steps */
    iparm[8] = 0; /* Not in use */
    iparm[9] = 13; /* Perturb the pivot elements with 1E-13 */
    iparm[10] = 1; /* Use nonsymmetric permutation and scaling MPS */
    iparm[11] = 0; /* Not in use */
    iparm[12] = 0; /* Not in use */
    iparm[13] = 0; /* Output: Number of perturbed pivots */
    iparm[14] = 0; /* Not in use */
    iparm[15] = 0; /* Not in use */
    iparm[16] = 0; /* Not in use */
    iparm[17] = -1; /* Output: Number of nonzeros in the factor LU */
    iparm[18] = -1; /* Output: Mflops for LU factorization */
    iparm[19] = 0; /* Output: Numbers of CG Iterations */
    maxfct = 1; /* Maximum number of numerical factorizations.*/
    mnum = 1; /* Which factorization to use.*/
    msglvl = 1; /* Print statistical information in file */
    error = 0; /* Initialize error flag */

/* ----- */
/* ..Initialize the internal solver memory pointer.This is only */
/* necessary for the FIRST call of the PARDISO solver.*/
/* ----- */
    for (i = 0; i < 64; i++) {
        pt[i] = 0;
    }

/* ----- */
/* ..Reordering and Symbolic Factorization.This step also allocates */
/* all memory that is necessary for the factorization.*/
/* ----- */
    phase = 11;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during symbolic factorization: %d", error);
        exit(1);
    }
    printf("\nReordering completed ... ");
    printf("\nNumber of nonzeros in factors = %d", iparm[17]);
    printf("\nNumber of factorization MFLOPS = %d", iparm[18]);

/* ----- */
/* ..Numerical factorization.*/
/* ----- */
    phase = 22;
    PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
             &n, a, ia, ja, &idum, &nrhs,
             iparm, &msglvl, &ddum, &ddum, &error);
    if (error != 0) {
        printf("\nERROR during numerical factorization: %d", error);
        exit(2);
    }
    printf("\nFactorization completed ... ");

/* ----- */
/* ..Back substitution and iterative refinement.*/
/* ----- */

```

```

phase = 33;
iparm[7] = 2; /* Max numbers of iterative refinement steps.*/
/* Set right hand side to one.*/
for (i = 0; i < n; i++) {
    b[i] = 1;
}
PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
        &n, a, ia, ja, &idum, &nrhs,
        iparm, &msglvl, b, x, &error);
if (error != 0) {
    printf("\nERROR during solution: %d", error);
    exit(3);
}
printf("\nSolve completed ... ");
printf("\nThe solution of the system is: ");
for (i = 0; i < n; i++) {
    printf("\n x [%d] = % f", i, x[i] );
}
printf ("\n");
/* ----- */
/* ..Termination and release of memory.*/
/* ----- */

phase = -1; /* Release internal memory.*/
PARDISO (pt, &maxfct, &mnum, &mtype, &phase,
        &n, &ddum, ia, ja, &idum, &nrhs,
        iparm, &msglvl, &ddum, &ddum, &error);
return 0;
}

```

直接法スパースソルバーのコード例

このセクションでは、Fortran 77、Fortran 90、およびCのコード例を示す。コードで使われるスパース・ソルバー・ルーチンの詳細は、本マニュアル第8章の「[直接法スパースソルバー \(DSS\) インターフェイス・ルーチン](#)」を参照のこと。
このコード例は、付録Aの「[直接法](#)」のセクションにある方程式 (スパース行列による対称正定値連立1次方程式 $Ax = b$) の解を算出する。

$$A = \begin{bmatrix} 9 & 1.5 & 6 & 0.75 & 3 \\ 1.5 & 0.5 & 0 & 0 & 0 \\ 6 & 0 & 12 & 0 & 0 \\ 0.75 & 0 & 0 & 0.625 & 0 \\ 3 & 0 & 0 & 0 & 16 \end{bmatrix} \quad \text{および} \quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

対称連立線形方程式の計算結果の例

ソルバーが正常に実行されると、行列式および解の配列は次のようになる。

```

pow of determinant is      0.000
base of determinant is    2.250
Determinant is            2.250
Solution Array:   -326.333   983.000   163.417   398.000   61.500

```

例 C-10 対称正定値連立方程式の例 (Fortran 77)

```

*****
*
* Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : Intel MKL DSS Fortran-77 example
*
*****
C-----
C Example program for solving symmetric positive definite system of
C equations.
C-----
PROGRAM solver_f77_test
IMPLICIT NONE
INCLUDE 'mkl_dss.f77'
C-----
C Define the array and rhs vectors
C-----
INTEGER nRows, nCols, nNonZeros, i, nRhs
PARAMETER (nRows = 5,
1 nCols = 5,
2 nNonZeros = 9,
3 nRhs = 1)
INTEGER rowIndex(nRows + 1), columns(nNonZeros)
DOUBLE PRECISION values(nNonZeros), rhs(nRows)
DATA rowIndex / 1, 6, 7, 8, 9, 10 /
DATA columns / 1, 2, 3, 4, 5, 2, 3, 4, 5 /
DATA values / 9, 1.5, 6, .75, 3, 0.5, 12, .625, 16 /
DATA rhs / 1, 2, 3, 4, 5 /
C-----
C Allocate storage for the solver handle and the solution vector
C-----
DOUBLE PRECISION solution(nRows)
INTEGER*8 handle
INTEGER error
CHARACTER*15 statIn
DOUBLE PRECISION statOut(5)
INTEGER bufLen

```

```

        PARAMETER(bufLen = 20)
        INTEGER buff(bufLen)
C-----
C Initialize the solver
C-----
        error = dss_create(handle, MKL_DSS_DEFAULTS)
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Define the non-zero structure of the matrix
C-----
        error = dss_define_structure( handle, MKL_DSS_SYMMETRIC,
& rowIndex, nRows, nCols, columns, nNonZeros )
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Reorder the matrix
C-----
        error = dss_reorder( handle, MKL_DSS_DEFAULTS, 0)
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Factor the matrix
C-----
        error = dss_factor_real( handle,
& MKL_DSS_DEFAULTS, VALUES)
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Get the solution vector
C-----
        error = dss_solve_real( handle, MKL_DSS_DEFAULTS,
& rhs, nRhs, solution)
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Print Determinant of the matrix
C-----
        statIn = 'determinant'
        call mkl_cvt_to_null_terminated_str(buff,bufLen,statIn)
        error = dss_statistics(handle, MKL_DSS_DEFAULTS,
& buff,statOut)
        WRITE(*,"(' pow of determinant is ', 5(F10.3))") statOut(1)
        WRITE(*,"(' base of determinant is ', 5(F10.3))") statOut(2)
        WRITE(*,"(' Determinant is ', 5(F10.3))")(10**statOut(1))*
& statOut(2)
C-----
C Deallocate solver storage
C-----
        error = dss_delete( handle, MKL_DSS_DEFAULTS )
        IF (error .NE. MKL_DSS_SUCCESS ) GOTO 999
C-----
C Print solution vector
C-----
        WRITE(*,900) (solution(i), i = 1, nCols)
        900 FORMAT(' Solution Array: ',5(F10.3))
        GOTO 1000
        999 WRITE(*,*) "Solver returned error code ", error
        1000 END

```

例 C-11 対称正定値連立方程式の例 (C)

```

/*
*****
* Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : Intel MKL DSS C example
*
*****
*/
/*

** Example program to solve symmetric positive definite system of equations.
*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "mkl_dss.h"
/*
** Define the array and rhs vectors
*/
#define NROWS 5
#define NCOLS 5
#define NNONZEROS 9
#define NRHS 1
static const int nRows = NROWS ;
static const int nCols = NCOLS ;
static const int nNonZeros = NNONZEROS ;
static const int nRhs = NRHS ;
static _INTEGER_t rowIndex[NROWS+1] = { 1, 6, 7, 8, 9, 10 };
static _INTEGER_t columns[NNONZEROS] = { 1, 2, 3, 4, 5, 2, 3, 4, 5 };
static _DOUBLE_PRECISION_t values[NNONZEROS] = { 9, 1.5, 6, .75, 3, 0.5, 12, .625, 16 };
static _DOUBLE_PRECISION_t rhs[NCOLS] = { 1, 2, 3, 4, 5 };

int main() {
    int i;
    /* Allocate storage for the solver handle and the right-hand side.*/
    _DOUBLE_PRECISION_t solValues[NROWS];
    _MKL_DSS_HANDLE_t handle;
    _INTEGER_t error;

```

```

    _CHARACTER_STR_t statIn[] = "determinant";
    _DOUBLE_PRECISION_t statOut[5];
    int opt = MKL_DSS_DEFAULTS;
    int sym = MKL_DSS_SYMMETRIC;
    int type = MKL_DSS_POSITIVE_DEFINITE;
/* ----- */
/* Initialize the solver */
/* ----- */
    error = dss_create(handle, opt );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Define the non-zero structure of the matrix */
/* ----- */
    error = dss_define_structure(
        handle, sym, rowIndex, nRows, nCols,
        columns, nNonZeros );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Reorder the matrix */
/* ----- */
    error = dss_reorder( handle, opt, 0);
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Factor the matrix */
/* ----- */
    error = dss_factor_real( handle, type, values );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Get the solution vector */
/* ----- */
    error = dss_solve_real( handle, opt, rhs, nRhs, solValues );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Get the determinant */
/*-----*/
    error = dss_statistics(handle, opt, statIn, statOut);
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/*-----*/
/* print determinant */
/*-----*/
    printf(" determinant power is %g \n", statOut[0]);
    printf(" determinant base is %g \n", statOut[1]);
    printf(" Determinant is %g \n", (pow(10.0,statOut[0]))*statOut[1]);
/* ----- */
/* Deallocate solver storage */
/* ----- */
    error = dss_delete( handle, opt );
    if ( error != MKL_DSS_SUCCESS ) goto printError;
/* ----- */
/* Print solution vector */
/* ----- */
    printf(" Solution array: ");
    for(i = 0; i< nCols; i++)
        printf(" %g", solValues[i] );
    printf("\n");

```

```

        exit(0);
printError:
        printf("Solver returned error code %d\n", error);
        exit(1);
}

```

例 C-12 対称正定値連立方程式の例 (Fortran 90)

```

!*****
*
!   Copyright(C) 2001-2004 Intel Corporation.All Rights Reserved.
!   The source code contained or described herein and all documents related to
!   the source code ("Material") are owned by Intel Corporation or its suppliers
!   or licensors.Title to the Material remains with Intel Corporation or its
!   suppliers and licensors.The Material contains trade secrets and proprietary
!   and confidential information of Intel or its suppliers and licensors.The
!   Material is protected by worldwide copyright and trade secret laws and
!   treaty provisions.No part of the Material may be used, copied, reproduced,
!   modified, published, uploaded, posted, transmitted, distributed or disclosed
!   in any way without Intel's prior express written permission.
!   No license under any patent, copyright, trade secret or other intellectual
!   property right is granted to or conferred upon you by disclosure or delivery
!   of the Materials, either expressly, by implication, inducement, estoppel or
!   otherwise.Any license under such intellectual property rights must be
!   express and approved by Intel in writing.
!
!*****
*
!   Content : Intel MKL DSS Fortran-90 example
!
!*****
*
!-----
!
! Example program for solving a symmetric positive definite system of
! equations.
!
!-----
INCLUDE 'mkl_dss.f90' ! Include the standard DSS "header file."
PROGRAM solver_f90_test
use mkl_dss
IMPLICIT NONE
INTEGER, PARAMETER :: dp = KIND(1.0D0)
INTEGER :: error
INTEGER :: i
INTEGER, PARAMETER :: bufLen = 20
! Define the data arrays and the solution and rhs vectors.
INTEGER, ALLOCATABLE :: columns( : )
INTEGER :: nCols
INTEGER :: nNonZeros
INTEGER :: nRhs
INTEGER :: nRows
REAL(KIND=DP), ALLOCATABLE :: rhs( : )
INTEGER, ALLOCATABLE :: rowIndex( : )
REAL(KIND=DP), ALLOCATABLE :: solution( : )

```

```

REAL(KIND=DP), ALLOCATABLE :: values( : )
TYPE(MKL_DSS_HANDLE) :: handle ! Allocate storage for the solver handle.
REAL(KIND=DP), ALLOCATABLE :: statOut( : )
CHARACTER*15 statIn
INTEGER perm(1)
INTEGER buff(bufLen)
EXTERNAL MKL_CVT_TO_NULL_TERMINATED_STR
! Set the problem to be solved.
nRows = 5
nCols = 5
nNonZeros = 9
nRhs = 1
perm(1) = 0
ALLOCATE( rowIndex( nRows + 1 ) )
rowIndex = (/ 1, 6, 7, 8, 9, 10 /)
ALLOCATE( columns( nNonZeros ) )
columns = (/ 1, 2, 3, 4, 5, 2, 3, 4, 5 /)
ALLOCATE( values( nNonZeros ) )
values = (/ 9.0_DP, 1.5_DP, 6.0_DP, 0.75_DP, 3.0_DP, 0.5_DP, 12.0_DP, &
& 0.625_DP, 16.0_DP /)
ALLOCATE( rhs( nRows ) )
rhs = (/ 1.0_DP, 2.0_DP, 3.0_DP, 4.0_DP, 5.0_DP /)
! Initialize the solver.
error = dss_create( handle, MKL_DSS_DEFAULTS )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
! Define the non-zero structure of the matrix.
error = dss_define_structure( handle, MKL_DSS_SYMMETRIC, rowIndex, nRows, &
& nCols, columns, nNonZeros )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
! Reorder the matrix.
error = dss_reorder( handle, MKL_DSS_DEFAULTS, perm )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
! Factor the matrix.
error = dss_factor_real( handle, MKL_DSS_DEFAULTS, values )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
! Allocate the solution vector and solve the problem.
ALLOCATE( solution( nRows ) )
error = dss_solve_real(handle, MKL_DSS_DEFAULTS, rhs, nRhs, solution )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
! Print Out the determinant of the matrix
ALLOCATE(statOut( 5 ) )
statIn = 'determinant'
call mkl_cvt_to_null_terminated_str(buff,bufLen,statIn);
error = dss_statistics(handle, MKL_DSS_DEFAULTS, buff, statOut )
IF (error /= MKL_DSS_SUCCESS) GOTO 999
WRITE(*, "('pow of determinant is '(5F10.3))") ( statOut(1) )
WRITE(*, "('base of determinant is '(5F10.3))") ( statOut(2) )
WRITE(*, "('Determinant is '(5F10.3))") ( (10**statOut(1))*statOut(2) )
! Deallocate solver storage and various local arrays.
error = dss_delete( handle, MKL_DSS_DEFAULTS )
IF (error /= MKL_DSS_SUCCESS ) GOTO 999
IF ( ALLOCATED( rowIndex ) ) DEALLOCATE( rowIndex )
IF ( ALLOCATED( columns ) ) DEALLOCATE( columns )
IF ( ALLOCATED( values ) ) DEALLOCATE( values )
IF ( ALLOCATED( rhs ) ) DEALLOCATE( rhs )

```

```
IF ( ALLOCATED( statOut ) ) DEALLOCATE( statOut )
! Print the solution vector, deallocate it and exit
WRITE(*, "('Solution Array: '(5F10.3))") ( solution(i), i = 1, nCols )
IF ( ALLOCATED( solution ) ) DEALLOCATE( solution )
GOTO 1000
! Print an error message and exit
999 WRITE(*,*) "Solver returned error code ", error
1000 CONTINUE
END PROGRAM solver_f90_test
```

反復法スパースソルバーのコード例

このセクションでは、Fortran 77 および C のコード例を示す。コードで使用されるリバース・コミュニケーション・インターフェイス (RCI ISS) に基づく反復法スパース・ソルバー・ルーチンの詳細は、本マニュアル [第 8 章](#) の「[リバース・コミュニケーション・インターフェイス \(RCI ISS\) に基づく反復法スパースソルバー](#)」を参照のこと。

RCI (前処理付き) 共役勾配ソルバーの使用例

対称正定値連立線形代数方程式の計算結果の例を示す。ソルバーが正常に実行されると、解の配列は次のようになる。

```
The system is successfully solved
The following solution obtained
  1.000    0.000    1.000    0.000
  1.000    0.000    1.000    0.000
Expected solution
  1.000    0.000    1.000    0.000
  1.000    0.000    1.000    0.000
Number of iterations:  8
```

例 C-13 対称正定値連立方程式の例 (Fortran 77)

```
*****
*
* Copyright(C) 2001-2005 Intel Corporation.All Rights Reserved.
* The source code contained or described herein and all documents related to
* the source code ("Material") are owned by Intel Corporation or its suppliers
* or licensors.Title to the Material remains with Intel Corporation or its
* suppliers and licensors.The Material contains trade secrets and proprietary
* and confidential information of Intel or its suppliers and licensors.The
* Material is protected by worldwide copyright and trade secret laws and
* treaty provisions.No part of the Material may be used, copied, reproduced,
* modified, published, uploaded, posted, transmitted, distributed or disclosed
* in any way without Intel's prior express written permission.
* No license under any patent, copyright, trade secret or other intellectual
* property right is granted to or conferred upon you by disclosure or delivery
* of the Materials, either expressly, by implication, inducement, estoppel or
* otherwise.Any license under such intellectual property rights must be
* express and approved by Intel in writing.
*
*****
*
* Content : Intel MKL RCI (P)CG Fortran-77 example
*
*****
C-----
C Example program for solving symmetric positive definite system of
C equations.
C-----
      PROGRAM rci_pcg_f77_test
      IMPLICIT NONE
C-----
C Define arrays for the upper triangle of the coefficient matrix and rhs vector
C Compressed sparse row storage is used for sparse representation
C-----
      INTEGER N, RCI_request, itercount, i
      PARAMETER (N=8)
      DOUBLE PRECISION  rhs(N), solution(N)
```

例 C-13 対称正定値連立方程式の例 (Fortran 77) (続き)

```

      INTEGER ia(9)
      INTEGER ja(18)
      DOUBLE PRECISION a(18)
C.. Fill all arrays containing matrix data.
      DATA ia /1,5,8,10,12,15,17,18,19/
      DATA ja
1 /1, 3, 6,7,
2 2, 3, 5,
3 3, 8,
4 4, 7,
5 5,6,7,
6 6, 8,
7 7,
8 8/
      DATA a
1 /7.D0, 1.D0, 2.D0, 7.D0,
2 -4.D0,8.D0, 2.D0,
3 1.D0, 5.D0,
4 7.D0, 9.D0,
5 5.D0, 1.D0, 5.D0,
6 -1.D0, 5.D0,
7 11.D0,
8 5.D0/
C-----
C Allocate storage for the solver ?par and the initial solution vector
C-----
      INTEGER length
      PARAMETER (length=128)
      DOUBLE PRECISION expected(N)
      DATA expected/1.D0, 0.D0, 1.D0, 0.D0, 1.D0, 0.D0, 1.D0, 0.D0/
      INTEGER ipar(length)
      DOUBLE PRECISION dpar(length),tmp(N,4)
C-----
C Initialize the right hand side through matrix-vector product
C-----
      CALL DCSRMV_SY('U', N, A, IA, JA, expected, rhs)

```

例 C-13 対称正定値連立方程式の例 (Fortran 77) (続き)

```
C-----
C Initialize the initial guess
C-----
      DO I=1, N
          solution(I)=1.D0
      ENDDO
C-----
C Initialize the solver
C-----
      CALL dcg_init(N,solution,rhs,RCI_request,ipar,dpar,tmp)
      IF (RCI_request .NE. 0 ) GOTO 999
C-----
C Set the desired parameters:
C LOGICAL parameters:
C do residual stopping test
C do not request for the user defined stopping test
C do Preconditioned Conjugate Gradient iterations
C DOUBLE PRECISION parameters
C set the relative tolerance to 1.0D-5 instead of default value 1.0D-6
C-----
      ipar(9)=1
      ipar(10)=0
      ipar(11)=1
      dpar(1)=1.D-5
C-----
C Check the correctness and consistency of the newly set parameters
C-----
      CALL dcg_check(N,solution,rhs,RCI_request,ipar,dpar,tmp)
      IF (RCI_request .NE. 0 ) GOTO 999
C-----
C Compute the solution by RCI PCG solver
C Reverse Communications starts here
C-----
1     CALL dcg(N,solution,rhs,RCI_request,ipar,dpar,tmp)
C-----
C If RCI_request=0, then the solution was found with the required precision
C-----
      IF (RCI_request .EQ. 0) THEN
          GOTO 700
C-----
C If RCI_request=1, then compute the vector A*tmp(:,1)
C and put the result in vector tmp(:,2)
C-----
```

例 C-13 対称正定値連立方程式の例 (Fortran 77) (続き)

```

      ELSE IF (RCI_request .EQ. 1) THEN
        CALL DCSRMV_SY('U', N, A, IA, JA, TMP, TMP(1, 2))
        GOTO 1
C-----
C If RCI_request=3, then compute vector preconditioner matrix on tmp(:,3)
C and put the result in vector tmp(:,4)
C-----
      ELSE IF (RCI_request .EQ. 3) THEN
        CALL DCOPY(N, TMP(1,3),1, TMP(1, 4), 1)
        GOTO 1
      ELSE
C-----
C If RCI_request=anything else, then dcg subroutine failed
C to compute the solution vector: solution(N)
C-----
        GOTO 999
      ENDIF
C-----
C Reverse Communication ends here
C Get the current iteration number
C-----
700  CALL dcg_get(N,solution,rhs,RCI_request,ipar,dpar,tmp,
      &            itercount)
C-----
C Print solution vector: solution(N) and number of iterations: itercount
C-----
      WRITE(*, *) ' The system is successfully solved '
      WRITE(*, *) ' The following solution obtained '
      WRITE(*,800)(solution(i),i =1,N)
      WRITE(*, *) ' Expected solution '
      WRITE(*,800)(expected(i),i =1,N)
800  FORMAT(4(F10.3))
      WRITE(*,900)(itercount)
900  FORMAT(' Number of iterations: ',1(I2))
      GOTO 1000
999  WRITE(*,*) 'Solver returned error code ', RCI_request
      STOP
1000 CONTINUE
      read *
      END

```

RCI (前処理付き) フレキシブル汎用最小残差ソルバーの使用例**Fortran の例**

非対称不定値連立方程式の計算結果の Fortran の例を示す。ソルバーが正常に実行されると、次の結果が出力される (丸め誤差は使用したコンピューター・システムに依存する)。

```

-----
The SIMPLEST example of usage of RCI FGMRES solver
to solve a non-symmetric indefinite non-degenerate
algebraic system of linear equations

```

```

-----
Some info about the current run of RCI FGMRES method:
As IPAR(8)=1, the automatic test for the maximal number of iterations
will be performed
+++
As IPAR(9)=1, the automatic residual test will be performed
+++
As IPAR(10)=0, the user-defined stopping test will not be requested,
thus, RCI_REQUEST will not take the value 2
+++
As IPAR(11)=0, the Preconditioned FGMRES iterations will not be
performed, thus, RCI_REQUEST will not take the value 3
+++
As IPAR(12)=1, the automatic test for the norm of the next generated
vector is not equal to zero up to rounding and computational errors will
be performed, thus, RCI_REQUEST will not take the value 4
+++
The system has been SUCCESSFULLY solved
The following solution has been obtained:
COMPUTED_SOLUTION(1)=-0.100E+01
COMPUTED_SOLUTION(2)= 0.100E+01
COMPUTED_SOLUTION(3)= 0.305E-15
COMPUTED_SOLUTION(4)= 0.100E+01
COMPUTED_SOLUTION(5)=-0.100E+01

The expected solution is:
EXPECTED_SOLUTION(1)=-0.100E+01
EXPECTED_SOLUTION(2)= 0.100E+01
EXPECTED_SOLUTION(3)= 0.000E+00
EXPECTED_SOLUTION(4)= 0.100E+01
EXPECTED_SOLUTION(5)=-0.100E+01

Number of iterations:          5

```

例 C-14 非対称不定値連立方程式の例 (Fortran)

```

C*****
C                                     INTEL CONFIDENTIAL
C Copyright(C) 2005 Intel Corporation.All Rights Reserved.
C The source code contained or described herein and all documents related to
C the source code ("Material") are owned by Intel Corporation or its suppliers
C or licensors.Title to the Material remains with Intel Corporation or its
C suppliers and licensors.The Material contains trade secrets and proprietary
C and confidential information of Intel or its suppliers and licensors.The
C Material is protected by worldwide copyright and trade secret laws and
C treaty provisions.No part of the Material may be used, copied, reproduced,
C modified, published, uploaded, posted, transmitted, distributed or disclosed
C in any way without Intel's prior express written permission.
C No license under any patent, copyright, trade secret or other intellectual
C property right is granted to or conferred upon you by disclosure or delivery
C of the Materials, either expressly, by implication, inducement, estoppel or
C otherwise.Any license under such intellectual property rights must be
C express and approved by Intel in writing.
C
C*****
C Content:
C Intel MKL RCI (P)FGMRES ((Preconditioned) Flexible Generalized Minimal
C                                     RESidual method) example
C*****

```

例 C-14 非対称不定値連立方程式の例 (Fortran) (続き)

```

C-----
C Example program for solving non-symmetric indefinite system of equations
C Simplest case: no preconditioning and no user-defined stopping tests
C-----

PROGRAM DFGMRES_NO_PRECON_F

INCLUDE "mkl_rci.fi"

INTEGER N
PARAMETER(N=5)
INTEGER SIZE
PARAMETER (SIZE=128)

C-----
C Define arrays for the upper triangle of the coefficient matrix
C Compressed sparse row storage is used for sparse representation
C-----
INTEGER IA(6)
DATA IA /1,3,6,9,12,14/
INTEGER JA(13)
DATA JA / 1, 3,
1 1, 2, 4,
2 2, 3, 5,
3 3, 4, 5,
4 4, 5 /
DOUBLE PRECISION A(13)
DATA A / 1.0, -1.0,
1 -1.0, 1.0, -1.0,
2 1.0,-2.0, 1.0,
3 -1.0, 2.0,-1.0,
4 -1.0,-3.0 /

C-----
C Allocate storage for the ?par parameters and the solution/rhs vectors
C-----
INTEGER IPAR(SIZE)
DOUBLE PRECISION DPAR(SIZE), TMP(N*(2*N+1)+(N*(N+9))/2+1)
DOUBLE PRECISION EXPECTED_SOLUTION(N)
DATA EXPECTED_SOLUTION /-1.0,1.0,0.0,1.0,-1.0/
DOUBLE PRECISION RHS(N)
DOUBLE PRECISION COMPUTED_SOLUTION(N)

C-----
C Some additional variables to use with the RCI (P)FGMRES solver
C-----
INTEGER ITERCOUNT
INTEGER RCI_REQUEST, I

PRINT *, '-----'
PRINT *, 'The SIMPLEST example of usage of RCI FGMRES solver'
PRINT *, 'to solve a non-symmetric indefinite non-degenerate'
PRINT *, ' algebraic system of linear equations'
PRINT *, '-----'

C-----
C Initialize variables and the right hand side through matrix-vector product
C-----
CALL MKL_DCSRGMV('N', N, A, IA, JA, EXPECTED_SOLUTION, RHS)

C-----
C Initialize the initial guess
C-----
DO I=1,N
COMPUTED_SOLUTION(I)=1.0
ENDDO

C-----
C Initialize the solver
C-----

```

例 C-14 非対称不定値連立方程式の例 (Fortran) (続き)

```

      CALL DFGMRES_INIT(N, COMPUTED_SOLUTION, RHS, RCI_REQUEST, IPAR,
1 DPAR, TMP)
      IF (RCI_REQUEST.NE.0) GOTO 999
C-----
C Set the desired parameters:
C LOGICAL parameters:
C do residual stopping test
C do not request for the user defined stopping test
C do the check of the norm of the next generated vector automatically
C DOUBLE PRECISION parameters
C set the relative tolerance to 1.0D-3 instead of default value 1.0D-6
C-----
      IPAR(9)=1
      IPAR(10)=0
      IPAR(12)=1
      DPAR(1)=1.0D-3
C-----
C Check the correctness and consistency of the newly set parameters
C-----
      CALL DFGMRES_CHECK(N, COMPUTED_SOLUTION, RHS, RCI_REQUEST,
1 IPAR, DPAR, TMP)
      IF (RCI_REQUEST.NE.0) GOTO 999
C-----
C Print the info about the RCI FGMRES method
C-----
      PRINT *, ''
      PRINT *, 'Some info about the current run of RCI FGMRES method:'
      PRINT *, ''
      IF (IPAR(8).NE.0) THEN
        WRITE(*,'(A,I1,A)') 'As IPAR(8)=' ,IPAR(8),', the automatic
1 test for the maximal number of iterations will be'
        PRINT *, 'performed'
      ELSE
        WRITE(*,'(A,I1,A)') 'As IPAR(8)=' ,IPAR(8),', the automatic test
1 for the maximal number of iterations will be'
        PRINT *, 'skipped'
      ENDIF
      PRINT *, '+++ '
      IF (IPAR(9).NE.0) THEN
        WRITE(*,'(A,I1,A)') 'As IPAR(9)=' ,IPAR(9),', the automatic
1 residual test will be performed'
      ELSE
        WRITE(*,'(A,I1,A)') 'As IPAR(9)=' ,IPAR(9),', the automatic
1 residual test will be skipped'
      ENDIF
      PRINT *, '+++ '
      IF (IPAR(10).NE.0) THEN
        WRITE(*,'(A,I1,A)') 'As IPAR(10)=' ,IPAR(10),', the user-defined
1 stopping test will be requested via'
        PRINT *, 'RCI_REQUEST=2'
      ELSE
        WRITE(*,'(A,I1,A)') 'As IPAR(10)=' ,IPAR(10),', the user-defined
1 stopping test will not be requested, thus,'
        PRINT *, 'RCI_REQUEST will not take the value 2'
      ENDIF
      PRINT *, '+++ '
      IF (IPAR(11).NE.0) THEN
        WRITE(*,'(A,I1,A)') 'As IPAR(11)=' ,IPAR(11),', the
1 Preconditioned FGMRES iterations will be performed, thus,'
        PRINT *, 'the preconditioner action will be requested via
1 RCI_REQUEST=3'
      ELSE
        WRITE(*,'(A,I1,A)') 'As IPAR(11)=' ,IPAR(11),', the
1 Preconditioned FGMRES iterations will not be performed,'
        PRINT *, 'thus, RCI_REQUEST will not take the value 3'

```

例 C-14 非対称不定値連立方程式の例 (Fortran) (続き)

```

ENDIF
PRINT *, '+++ '
IF (IPAR(12).NE.0) THEN
WRITE(*, '(A,I1,A)') 'As IPAR(12)=', IPAR(12), ', the automatic
1 test for the norm of the next generated vector is'
PRINT *, 'not equal to zero up to rounding and computational
1 errors will be performed,'
PRINT *, 'thus, RCI_REQUEST will not take the value 4'
ELSE
WRITE(*, '(A,I1,A)') 'As IPAR(12)=', IPAR(12), ', the automatic
1 test for the norm of the next generated vector is'
PRINT *, 'not equal to zero up to rounding and computational
1 errors will be skipped,'
PRINT *, 'thus, the user-defined test will be requested via
1 RCI_REQUEST=4'
ENDIF
PRINT *, '+++ '
C-----
C Compute the solution by RCI (P)FGMRES solver without preconditioning
C Reverse Communication starts here
C-----
1 CALL DFGMRES(N, COMPUTED_SOLUTION, RHS, RCI_REQUEST, IPAR,
1 DPAR, TMP)
C-----
C If RCI_REQUEST=0, then the solution was found with the required precision
C-----
IF (RCI_REQUEST.EQ.0) GOTO 3
C-----
C If RCI_REQUEST=1, then compute the vector A*TMP(IPAR(22))
C and put the result in vector TMP(IPAR(23))
C-----
IF (RCI_REQUEST.EQ.1) THEN
CALL MKL_DCSRGMV('N', N, A, IA, JA, TMP(IPAR(22)), TMP(IPAR(23)))
GOTO 1
C-----
C If RCI_REQUEST=anything else, then DFGMRES subroutine failed
C to compute the solution vector: COMPUTED_SOLUTION(N)
C-----
ELSE
GOTO 999
ENDIF
C-----
C Reverse Communication ends here
C Get the current iteration number and the FGMRES solution (DO NOT FORGET to
C call DFGMRES_GET routine as COMPUTED_SOLUTION is still containing
C the initial guess!)
C-----
3 CALL DFGMRES_GET(N, COMPUTED_SOLUTION, RHS, RCI_REQUEST, IPAR,
1 DPAR, TMP, ITERCOUNT)
C-----
C Print solution vector: COMPUTED_SOLUTION(N) and
C the number of iterations: ITERCOUNT
C-----
PRINT *, ''
PRINT *, ' The system has been SUCCESSFULLY solved'
PRINT *, ''
PRINT *, ' The following solution has been obtained:'
DO I=1,N
WRITE(*, '(A18,I1,A2,E10.3)') 'COMPUTED_SOLUTION(', I, ')=' ,
1 COMPUTED_SOLUTION(I)
ENDDO
PRINT *, ''
PRINT *, ' The expected solution is:'
DO I=1,N
WRITE(*, '(A18,I1,A2,E10.3)') 'EXPECTED_SOLUTION(', I, ')=' ,

```

例 C-14 非対称不定値連立方程式の例 (Fortran) (続き)

```

1 EXPECTED_SOLUTION(I)
  ENDDO
  PRINT *, ''
  PRINT *, ' Number of iterations: ', ITERCOUNT
  GOTO 1000

999 PRINT *, 'The solver has returned the ERROR code ', RCI_REQUEST

1000 CONTINUE
      END

```

C の例

前の例と同じ非対称不定値連立方程式の計算結果の C の例を示す。結果は、C と Fortran での表記規則に至るまで同じである。Fortran と C の配列の違いの制御をどのように推奨しているかについて特に注意を払うこと。特に、この例では、IPAR(22) から ipar[21]-1 と IPAR(23) から ipar[22]-1 のユーザー定義操作について、入力/結果のアドレスをそれぞれ調整している。ソルバーが正常に実行されると、次の結果が出力される (丸め誤差は使用したコンピューター・システムに依存する)。

```

-----
The SIMPLEST example of usage of RCI FGMRES solver
to solve a non-symmetric indefinite non-degenerate
      algebraic system of linear equations
-----

```

Some info about the current run of RCI FGMRES method:

```

As ipar[7]=1, the automatic test for the maximal number of iterations
will be performed

```

```
+++
```

```

As ipar[8]=1, the automatic residual test will be performed

```

```
+++
```

```

As ipar[9]=0, the user-defined stopping test will not be requested, thus,
RCI_request will not take the value 2

```

```
+++
```

```

As ipar[10]=0, the Preconditioned FGMRES iterations will not be
performed, thus, RCI_request will not take the value 3

```

```
+++
```

```

As ipar[11]=1, the automatic test for the norm of the next generated
vector is not equal to zero up to rounding and computational errors will
be performed, thus, RCI_request will not take the value 4

```

```
+++
```

The system has been SUCCESSFULLY solved

The following solution has been obtained:

```

computed_solution[0]=-1.000000e+000
computed_solution[1]=1.000000e+000
computed_solution[2]=3.053113e-016
computed_solution[3]=1.000000e+000
computed_solution[4]=-1.000000e+000

```

The expected solution is:

```

expected_solution[0]=-1.000000e+000
expected_solution[1]=1.000000e+000

```

```

expected_solution[2]=0.000000e+000
expected_solution[3]=1.000000e+000
expected_solution[4]=-1.000000e+000

```

Number of iterations: 5

例 C-15 非対称正定値連立方程式の例 (C)

```

/*****
/*
/*          INTEL CONFIDENTIAL
/* Copyright(C) 2005 Intel Corporation.All Rights Reserved.
/* The source code contained or described herein and all documents related to
/* the source code ("Material")are owned by Intel Corporation or its suppliers
/* or licensors.Title to the Material remains with Intel Corporation or its
/* suppliers and licensors.The Material contains trade secrets and proprietary
/* and confidential information of Intel or its suppliers and licensors.The
/* Material is protected by worldwide copyright and trade secret laws and
/* treaty provisions.No part of the Material may be used, copied, reproduced,
/* modified,published, uploaded, posted, transmitted, distributed or disclosed
/* in any way without Intel's prior express written permission.
/* No license under any patent, copyright, trade secret or other intellectual
/* property right is granted to or conferred upon you by disclosure or delivery
/* of the Materials, either expressly, by implication, inducement, estoppel or
/* otherwise.Any license under such intellectual property rights must be
/* express and approved by Intel in writing.
/*
/*****
/* Content:
/* Intel MKL RCI (P)FGMRES ((Preconditioned) Flexible Generalized Minimal
/*                               RESidual method) example
/*****

/*-----
/* Example program for solving non-symmetric indefinite system of equations
/* Simplest case: no preconditioning and no user-defined stopping tests
/*-----*/

#include <stdio.h>
#include "mkl_blas.h"
#include "mkl_spblas.h"
#include "mkl_rci.h"
#define N 5
#define size 128

int main(void)
{

/*-----
/* Define arrays for the upper triangle of the coefficient matrix
/* Compressed sparse row storage is used for sparse representation
/*-----*/

int ia[6]={1,3,6,9,12,14};
int ja[13]={ 1, 3,
            1, 2, 4,
            2, 3, 5,
            3, 4, 5,
            4, 5 };
double A[13]={ 1.0, -1.0,
              -1.0, 1.0, -1.0,
                1.0,-2.0, 1.0,
                 -1.0, 2.0,-1.0,
                  -1.0,-3.0 };

```

例 C-15 非対称正定値連立方程式の例 (C) (続き)

```

/*-----
   /* Allocate storage for the ?par parameters and the solution/rhs vectors
/*-----*/
   int ipar[size];
   double dpar[size], tmp[N*(2*N+1)+(N*(N+9))/2+1];
   double expected_solution[N]={-1.0,1.0,0.0,1.0,-1.0};
   double rhs[N];
   double computed_solution[N];

/*-----
   /* Some additional variables to use with the RCI (P)FGMRES solver
/*-----*/
   int itercount;
   int RCI_request, i, ivar;
   double dvar;
   char cvar;

   printf("-----\n");
   printf("The SIMPLEST example of usage of RCI FGMRES solver\n");
   printf("to solve a non-symmetric indefinite non-degenerate\n");
   printf("      algebraic system of linear equations\n");
   printf("-----\n\n");

/*-----
   /* Initialize variables and the right hand side through matrix-vector product
/*-----*/
   ivar=N;
   cvar='N';
   mkl_dcsrgev(&cvar, &ivar, A, ia, ja, expected_solution, rhs);

/*-----
   /* Initialize the initial guess
/*-----*/
   for(i=0;i<N;i++)
   {
       computed_solution[i]=1.0;
   }

/*-----
   /* Initialize the solver
/*-----*/
   dfgmres_init(&ivar, computed_solution, rhs, &RCI_request, ipar, dpar, tmp);
   if (RCI_request!=0) goto FAILED;

/*-----
   /* Set the desired parameters:
   /* LOGICAL parameters:
   /* do residual stopping test
   /* do not request for the user defined stopping test
   /* do the check of the norm of the next generated vector automatically
   /* DOUBLE PRECISION parameters
   /* set the relative tolerance to 1.0D-3 instead of default value 1.0D-6
/*-----*/
   ipar[8]=1;
   ipar[9]=0;

```

例 C-15 非対称正定値連立方程式の例 (C) (続き)

```

    ipar[11]=1;
    dpar[0]=1.0E-3;

/*-----
   /* Check the correctness and consistency of the newly set parameters
/*-----*/
    dfgmres_check(&ivar, computed_solution, rhs, &RCI_request, ipar, dpar, tmp);
    if (RCI_request!=0) goto FAILED;

/*-----
   /* Print the info about the RCI FGMRES method
/*-----*/
    printf("Some info about the current run of RCI FGMRES method:\n\n");
    if (ipar[7])
    {
        printf("As ipar[7]=%d, the automatic test for the maximal number of
iterations will be\n", ipar[7]);
        printf("performed\n");
    }
    else
    {
        printf("As ipar[7]=%d, the automatic test for the maximal number of
iterations will be\n", ipar[7]);
        printf("skipped\n");
    }
    printf("+++\\n");
    if (ipar[8])
    {
        printf("As ipar[8]=%d, the automatic residual test will be
performed\\n", ipar[8]);
    }
    else
    {
        printf("As ipar[8]=%d, the automatic residual test will be skipped\\n",
ipar[8]);
    }
    printf("+++\\n");
    if (ipar[9])
    {
        printf("As ipar[9]=%d, the user-defined stopping test will be requested
via\\n", ipar[9]);
        printf("RCI_request=2\\n");
    }
    else
    {
        printf("As ipar[9]=%d, the user-defined stopping test will not be
requested, thus,\\n", ipar[9]);
        printf("RCI_request will not take the value 2\\n");
    }
    printf("+++\\n");
    if (ipar[10])
    {
        printf("As ipar[10]=%d, the Preconditioned FGMRES iterations will be
performed, thus,\\n", ipar[10]);
        printf("the preconditioner action will be requested via
RCI_request=3\\n");
    }
    else
    {

```

例 C-15 非対称正定値連立方程式の例 (C) (続き)

```

printf("As ipar[10]=%d, the Preconditioned FGMRES iterations will not
be performed,\n", ipar[10]);
printf("thus, RCI_request will not take the value 3\n");
}
printf("+++\\n");
if (ipar[11])
{
printf("As ipar[11]=%d, the automatic test for the norm of the next
generated vector is\\n", ipar[11]);
printf("not equal to zero up to rounding and computational errors will
be performed,\\n");
printf("thus, RCI_request will not take the value 4\\n");
}
else
{
printf("As ipar[11]=%d, the automatic test for the norm of the next
generated vector is\\n", ipar[11]);
printf("not equal to zero up to rounding and computational errors will
be skipped,\\n");
printf("thus, the user-defined test will be requested via
RCI_request=4\\n");
}
printf("+++\\n\\n");

/*-----
/* Compute the solution by RCI (P)FGMRES solver without preconditioning
/* Reverse Communication starts here

/*-----*/
ONE: dfgmres(&ivar, computed_solution, rhs, &RCI_request, ipar, dpar, tmp);

/*-----
/* If RCI_request=0, then the solution was found with the required precision

/*-----*/
if (RCI_request==0) goto COMPLETE;

/*-----
/* If RCI_request=1, then compute the vector A*tmp[ipar[21]-1]
/* and put the result in vector tmp[ipar[22]-1]

/*-----
/* NOTE that ipar[21] and ipar[22] contain FORTRAN style addresses,
/* therefore, in C code it is required to subtract 1 from them to get
/* C style addresses

/*-----*/
if (RCI_request==1)
{
mkl_dcsrgermv(&cvar, &ivar, A, ia, ja, &tmp[ipar[21]-1],
&tmp[ipar[22]-1]);
goto ONE;
}

/*-----
/* If RCI_request=anything else, then dfgmres subroutine failed
/* to compute the solution vector: computed_solution[N]

/*-----*/
else
{

```

例 C-15 非対称正定値連立方程式の例 (C) (続き)

```
        goto FAILED;
    }

/*-----
/* Reverse Communication ends here
/* Get the current iteration number and the FGMRES solution (DO NOT FORGET
/* to call dfgmres_get routine as computed_solution is still containing
/* the initial guess!)

/*-----*/
COMPLETE:  dfgmres_get(&ivar, computed_solution, rhs, &RCI_request, ipar, dpar,
tmp, &itercount);
/*

/*-----
/* Print solution vector: computed_solution[N] and the number of iterations:
itercount

/*-----*/
printf(" The system has been SUCCESSFULLY solved \n");
printf("\n The following solution has been obtained: \n");
for (i=0;i<N;i++)
printf("computed_solution[%d]=%e\n",i,computed_solution[i]);
printf("\n The expected solution is: \n");
for (i=0;i<N;i++)
printf("expected_solution[%d]=%e\n",i,expected_solution[i]);
printf("\n Number of iterations: %d",itercount);
goto SUCCEEDED;
FAILED: printf("The solver has returned the ERROR code %d", RCI_request);

SUCCEEDED: return 0;
}
```

フーリエ変換関数のコード例

このセクションでは、「[フーリエ変換関数](#)」の章の「[DFT 関数](#)」と「[クラスター DFT 関数](#)」セクションで説明された関数のコード例を示す。例は、[DFT 計算でのマルチスレディングの使用例](#)および[クラスター DFT 関数の例](#)を含むサブセクション「[DFT 関数の例](#)」でグループ化されている。

DFT 関数の例

この例では 1 次元の計算を 2 回行っている。これらの例では、「[構成設定](#)」で指定されるすべての構成パラメーターに対してのデフォルト設定を使用する。

例 C-16 1 次元 DFT (Fortran インターフェイス)

```
! Fortran example.
! 1D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex :: X(32)
Real :: Y(34)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status
...put input data into X(1),...,X(32); Y(1),...,Y(32)

! Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
    DFTI_COMPLEX, 1, 32 )
Status = DftiCommitDescriptor( My_Desc1_Handle )
Status = DftiComputeForward( My_Desc1_Handle, X )
Status = DftiFreeDescriptor(My_Desc1_Handle)
! result is given by {X(1),X(2),...,X(32)}

! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor(My_Desc2_Handle, DFTI_SINGLE,
    DFTI_REAL, 1, 32)
Status = DftiCommitDescriptor(My_Desc2_Handle)
Status = DftiComputeForward(My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
! result is given in CCS format.
```

例 C-17 **1次元 DFT (C インターフェイス)**

```
/* C example, float _Complex is defined in C9X */
#include "mkl_dfti.h"
float _Complex x[32];
float y[34];
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle;
/* ...or alternatively
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle; */

long status;
...put input data into x[0],...,x[31]; y[0],...,y[31]
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
    DFTI_COMPLEX, 1, 32);
status = DftiCommitDescriptor( my_desc1_handle );
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is x[0], ..., x[31] */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
    DFTI_REAL, 1, 32);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is given in CCS format */
```

次に示すのは単純な 2 次元変換を 2 回行った例である。計算関数のデータと結果パラメーターは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言されている点に注意する。そのため、2 次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1 次元配列に変換しなければならない。

例 C-18 **2次元 DFT (Fortran インターフェイス)**

```
! Fortran example.
! 2D complex to complex, and real to conjugate even
Use MKL_DFTI
Complex :: X_2D(32,100)
Real :: Y_2D(34, 102)
Complex :: X(3200)
Real :: Y(3468)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc1_Handle, My_Desc2_Handle
Integer :: Status, L(2)
...put input data into X_2D(j,k), Y_2D(j,k), 1<=j=32,1<=k=100
...set L(1) = 32, L(2) = 100
...the transform is a 32-by-100

! Perform a complex to complex transform
Status = DftiCreateDescriptor( My_Desc1_Handle, DFTI_SINGLE,
                             DFTI_COMPLEX, 2, L)
Status = DftiCommitDescriptor( My_Desc1_Handle)
Status = DftiComputeForward( My_Desc1_Handle, X)
Status = DftiFreeDescriptor(My_Desc1_Handle)
! result is given by X_2D(j,k), 1<=j<=32, 1<=k<=100

! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc2_Handle, DFTI_SINGLE,
                             DFTI_REAL, 2, L)
Status = DftiCommitDescriptor( My_Desc2_Handle)
Status = DftiComputeForward( My_Desc2_Handle, Y)
Status = DftiFreeDescriptor(My_Desc2_Handle)
! result is given by the complex value z(j,k) 1<=j<=32; 1<=k<=100
! and is stored in CCS format
```

例 C-19 **2次元 DFT (C インターフェイス)**

```
/* C example */
#include "mkl_dfti.h"
float _Complex x[32][100];
float y[34][102];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle, my_desc2_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc1_handle, *my_desc2_handle; */
long status, l[2];
...put input data into x[j][k] 0<=j<=31, 0<=k<=99
...put input data into y[j][k] 0<=j<=31, 0<=k<=99
l[0] = 32; l[1] = 100;
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,
                              DFTI_COMPLEX, 2, 1);
status = DftiCommitDescriptor( my_desc1_handle);
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is the complex value x[j][k], 0<=j<=31, 0<=k<=99 */
status = DftiCreateDescriptor( &my_desc2_handle, DFTI_SINGLE,
                              DFTI_REAL, 2, 1);
status = DftiCommitDescriptor( my_desc2_handle);
status = DftiComputeForward( my_desc2_handle, y);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is the complex value z(j,k) 0<=j<=31; 0<=k<=99
/* and is stored in CCS format */
```

次の例は、`DftiSetValue` 関数を使用してデフォルト構成設定を変更する方法を示す。

例えば、入力データを **DFT** 計算後にも保全しておくには、`DFTI_PLACEMENT` の設定をデフォルトの「インプレース」から「ノット・イン・プレース」に変更する必要がある。

以下にコード例を示す。

例 C-20 デフォルト設定の変更 (Fortran)

```
! Fortran example
! 1D complex to complex, not in place
Use MKL_DFTI
Complex :: X_in(32), X_out(32)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
Integer :: Status
...put input data into X_in(j), 1<=j<=32
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
    DFTI_COMPLEX, 1, 32)
Status = DftiSetValue( My_Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X_in, X_out)
Status = DftiFreeDescriptor (My_Desc_Handle)
! result is X_out(1),X_out(2),...,X_out(32)
```

例 C-21 デフォルト設定の変更 (C)

```
/* C example */
#include "mkl_dfti.h"
float _Complex x_in[32], x_out[32];
DFTI_DESCRIPTOR_HANDLE my_desc_handle;
/* or alternatively
DFTI_DESCRIPTOR *my_desc_handle; */
long status;
...put input data into x_in[j], 0 <= j < 32
status = DftiCreateDescriptor( &my_desc_handle, DFTI_SINGLE,
DFTI_COMPLEX, 1, 32);
status = DftiSetValue( my_desc_handle, DFTI_PLACEMENT,
DFTI_NOT_INPLACE);
status = DftiCommitDescriptor( my_desc_handle);
status = DftiComputeForward( my_desc_handle, x_in, x_out);
status = DftiFreeDescriptor(&my_desc_handle);
/* result is x_out[0], x_out[1], ..., x_out[31] */
```

次の例 C-22 は、第 11 章のステータス確認関数の使用方法を説明する。

例 C-22 ステータス確認関数の使用

```
from C language:

DFTI_DESCRIPTOR_HANDLE desc;
long status, class_error, value;
char* error_message;
...descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, &value); //
//or any DFTI function

class_error = DftiErrorClass(status, DFTI_NO_ERROR);
if (! class_error) {
    printf ("DftiGetValue() fixes the wrong situation and
            returns the corresponding value n");
    error_message = DftiErrorMessage(status);
    printf("error_message = %s \n", error_message);
}
...
from Fortran:

type(DFTI_DESCRIPTOR), POINTER :: desc
integer value, status
character(DFTI_MAX_MESSAGE_LENGTH) error_message
logical class_error
...descriptor creation and other code
status = DftiGetValue( desc, DFTI_PRECISION, value)
class_error = DftiErrorClass(status, DFTI_NO_ERROR)
if (.not. class_error) then
    print *, ' DftiGetValue() fixes the wrong situation and
            returns the corresponding value '
    error_message = DftiErrorMessage(status)
    print *, 'error_message = ', error_message
endif
```

例 C-23 **1次元変換による2次元DFTの計算**

```
! Fortran
Complex :: X_2D(20,40),
Complex :: X(800)
Equivalence (X_2D, X)
INTEGER :: STRIDE(2)
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim1
type(DFTI_DESCRIPTOR), POINTER :: Desc_Handle_Dim2
...
Status = DftiCreateDescriptor( Desc_Handle_Dim1, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 20 )
Status = DftiCreateDescriptor( Desc_Handle_Dim2, DFTI_SINGLE,
                             DFTI_COMPLEX, 1, 40 )

! perform 40 one-dimensional transforms along 1st dimension
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_INPUT_DISTANCE, 20 )
Status = DftiSetValue( Desc_Handle_Dim1, DFTI_OUTPUT_DISTANCE, 20 )
Status = DftiCommitDescriptor( Desc_Handle_Dim1 )
Status = DftiComputeForward( Desc_Handle_Dim1, X )

! perform 20 one-dimensional transforms along 2nd dimension
Stride(1) = 0; Stride(2) = 20
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_DISTANCE, 1 )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_INPUT_STRIDES, Stride )
Status = DftiSetValue( Desc_Handle_Dim2, DFTI_OUTPUT_STRIDES, Stride )
```

例 C-23 1次元変換による2次元DFTの計算(続き)

```
Status = DftiCommitDescriptor( Desc_Handle_Dim2 )
Status = DftiComputeForward( Desc_Handle_Dim2, X )
Status = DftiFreeDescriptor( Desc_Handle_Dim1 )
Status = DftiFreeDescriptor( Desc_Handle_Dim2 )

/* C */
float _Complex x[20][40];
long stride[2];
long status;
DFTI_DESCRIPTOR_HANDLE desc_handle_dim1;
DFTI_DESCRIPTOR_HANDLE desc_handle_dim2;
...
status = DftiCreateDescriptor( &desc_handle_dim1, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 20 );
status = DftiCreateDescriptor( &desc_handle_dim2, DFTI_SINGLE,
                              DFTI_COMPLEX, 1, 40 );

/* perform 40 one-dimensional transforms along 1st dimension */
/* note that the 1st dimension data are not unit-stride */
stride[0] = 0; stride[1] = 40;
status = DftiSetValue( desc_handle_dim1, DFTI_NUMBER_OF_TRANSFORMS, 40 );
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_DISTANCE, 1 );
status = DftiSetValue( desc_handle_dim1, DFTI_INPUT_STRIDES, stride );
status = DftiSetValue( desc_handle_dim1, DFTI_OUTPUT_STRIDES, stride );
status = DftiCommitDescriptor( desc_handle_dim1 );
status = DftiComputeForward( desc_handle_dim1, x );

/* perform 20 one-dimensional transforms along 2nd dimension */
/* note that the 2nd dimension is unit stride */
status = DftiSetValue( desc_handle_dim2, DFTI_NUMBER_OF_TRANSFORMS, 20 );
status = DftiSetValue( desc_handle_dim2, DFTI_INPUT_DISTANCE, 40 );
status = DftiSetValue( desc_handle_dim2, DFTI_OUTPUT_DISTANCE, 40 );

status = DftiCommitDescriptor( desc_handle_dim2 );
status = DftiComputeForward( desc_handle_dim2, x );
status = DftiFreeDescriptor( &Desc_Handle_Dim1 );
status = DftiFreeDescriptor( &Desc_Handle_Dim2 );
```

多次元実数からCCE格納形式の共役偶複素行列への変換の例を次に示す。[例C-24](#)は、Fortranインターフェイスでの2次元インプレース変換、[例C-24a](#)は2次元アウトプレース変換である。[例C-25](#)は、Cインターフェイスでの3次元アウトオブプレース変換であ

る。計算関数のデータと結果パラメーターは、すべて擬寸法階数 1 の配列 DIMENSION(0:*) として宣言されている点に注意する。そのため、2次元配列は EQUIVALENCE 文か Fortran のその他の機能を使用して 1次元配列に変換しなければならない。

例 C-24 2次元 REAL インプレース DFT (Fortran インターフェイス)

```
! Fortran example.
! 2D and real to conjugate even
Use MKL_DFTI
Real :: X_2D(34,100) ! 34 = (32/2 + 1)*2
Real :: X(3400)
Equivalence (X_2D, X)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
Integer :: Status, L(2)
Integer :: strides_in(3)
Integer :: strides_out(3)
...put input data into X_2D(j,k), 1<=j<=32,1<=k<=100
...set L(1) = 32, L(2) = 100
...set strides_in(1) = 0, strides_in(2) = 1, strides_in(3) = 34
...set strides_out(1) = 0, strides_out(2) = 1, strides_out(3) = 17
...the transform is a 32-by-100

! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
DFTI_REAL, 2, L)
Status = DftiSetValue(My_Desc_Handle, DFTI_CONJUGATE_EVEN_STORAGE,
DFTI_COMPLEX_COMPLEX)
Status = DftiSetValue(My_Desc_Handle, DFTI_INPUT_STRIDES, strides_in)
Status = DftiSetValue(My_Desc_Handle, DFTI_OUTPUT_STRIDES, strides_out)
Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X)
Status = DftiFreeDescriptor(My_Desc_Handle)
! result is given by the complex value z(j,k) 1<=j<=17; 1<=k<=100 and
! is stored in real matrix X_2D in CCE format.
```

例 C-24a 2次元 REAL アウトオブプレース DFT (Fortran インターフェイス)

```
! Fortran example.
! 2D and real to conjugate even
Use MKL_DFTI
Real :: X_2D(32,100)
Complex :: Y_2D(17, 100) ! 17 = 32/2 + 1
Real :: X(3200)
Complex :: Y(1700)
Equivalence (X_2D, X)
Equivalence (Y_2D, Y)
type(DFTI_DESCRIPTOR), POINTER :: My_Desc_Handle
Integer :: Status, L(2)
Integer :: strides_out(3)

...put input data into X_2D(j,k), 1<=j=32,1<=k<=100
...set L(1) = 32, L(2) = 100

...set strides_out(1) = 0, strides_out(2) = 1, strides_out(3) = 17

...the transform is a 32-by-100
! Perform a real to complex conjugate even transform
Status = DftiCreateDescriptor( My_Desc_Handle, DFTI_SINGLE,
DFTI_REAL, 2, L)
Status = DftiSetValue(My_Desc_Handle, DFTI_CONJUGATE_EVEN_STORAGE,
DFTI_COMPLEX_COMPLEX)
Status = DftiSetValue( My_Desc_Handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE)
Status = DftiSetValue(My_Desc_Handle, DFTI_OUTPUT_STRIDES, strides_out)

Status = DftiCommitDescriptor( My_Desc_Handle)
Status = DftiComputeForward( My_Desc_Handle, X, Y)
Status = DftiFreeDescriptor(My_Desc_Handle)
! result is given by the complex value z(j,k) 1<=j<=17; 1<=k<=100 and

! is stored in complex matrix Y_2D in CCE format.
```

例 C-25 3次元 REAL DFT (C インターフェイス)

```
/* C example */
#include "mkl_dfti.h"
float x[32][100][19];
float _Complex y[32][100][10]; /* 10 = 19/2 + 1 */
DFTI_DESCRIPTOR_HANDLE my_desc_handle
/* or alternatively
DFTI_DESCRIPTOR *my_desc_handle */
long status, l[3];
long strides_out[4];

...put input data into x[j][k][s] 0<=j<=31, 0<=k<=99, 0<=s<=18
l[0] = 32; l[1] = 100; l[2] = 19;
strides_out[0] = 0; strides_out[1] = 1000;
strides_out[2] = 10; strides_out[3] = 1;

status = DftiCreateDescriptor( &my_desc_handle, DFTI_SINGLE,
DFTI_REAL, 3, l);
Status = DftiSetValue(my_desc_handle, DFTI_CONJUGATE_EVEN_STORAGE,
DFTI_COMPLEX_COMPLEX);
Status = DftiSetValue( my_desc_handle, DFTI_PLACEMENT, DFTI_NOT_INPLACE);
Status = DftiSetValue(my_desc_handle, DFTI_OUTPUT_STRIDES, strides_out);

status = DftiCommitDescriptor( my_desc_handle);
status = DftiComputeForward( my_desc_handle, x, y);
status = DftiFreeDescriptor(&my_desc_handle);
/* result is the complex value z(j,k,s) 0<=j<=31; 0<=k<=99, 0<=s<=9
and is stored in complex matrix y in CCE format.*/
```

DFT 計算でのマルチスレッディングの使用例

次のプログラムは、インテル MKL の DFT 計算で内部スレッディングを使用する方法を示す(「[ユーザースレッド数](#)」のケース 1 を参照)。

インテル MKL 内のスレッド数を指定するには、次の設定を使用する。

set OMP_NUM_THREADS = 1 (シングルスレッド・モードの場合)

set OMP_NUM_THREADS = 4 (マルチスレッド・モードの場合)

構成パラメーター `DFTI_NUMBER_OF_USER_THREADS` は、デフォルト値 1 でなければならない。

例 C-26 インテル MKL 内部スレッディング・モードの使用

```
#include "mkl_dfti.h"

void main () {

float x[200][100];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle;
long status, len[2];
//...put input data into x[j][k] 0<=j<=199, 0<=k<=99
len[0] = 200; len[1] = 100;
status = DftiCreateDescriptor( &my_desc1_handle, DFTI_SINGLE,DFTI_REAL, 2,
len);
status = DftiCommitDescriptor( my_desc1_handle);
status = DftiComputeForward( my_desc1_handle, x);
status = DftiFreeDescriptor(&my_desc1_handle);
}
```

次の例 C-27 では、各ディスクリプター・インスタンスがシングルスレッドでのみ使用される並列カスタムプログラムについて説明する（「[ユーザースレッド数](#)」のケース 3 を参照）。

スレッド数を指定するには次の設定を使用する。

set `MKL_SERIAL = yes`（または `YES`）（インテル MKL でシングルスレッド・モードの場合。推奨）

set `OMP_NUM_THREADS = 4`（カスタムプログラムでマルチスレッド・モードの場合）

構成パラメーター `DFTI_NUMBER_OF_USER_THREADS` は、デフォルト値 1 でなければならない。

この例では、プログラムはカスタムレベルではシングルスレッドに変換されるが、インテル MKL 内では並列モードで使用される。これには、パラメーター `DFTI_NUMBER_OF_TRANSFORMS = 4` および対応するパラメーター `DFTI_INPUT_DISTANCE = 5000` を設定する必要がある。

例 C-27 複数のディスクリプターを伴う並列モードの使用

```
#include "mkl_dfti.h"
#include "omp.h"
void main ()
{
    float _Complex x[200][100];
    long len[2];

    //...put input data into x[j][k] 0<=j<=199, 0<=k<=99
    len[0] = 50; len[1] = 100;

    // each thread calculates real DFT for matrix (50*100)
    #pragma omp parallel
    {
        DFTI_DESCRIPTOR_HANDLE my_desc_handle;
        long myStatus;
        int myID = omp_get_thread_num ();

        myStatus = DftiCreateDescriptor(my_desc_handle, DFTI_SINGLE, DFTI_COMPLEX, 2, len);
        myStatus = DftiCommitDescriptor (my_desc_handle);
        myStatus = DftiComputeForward (my_desc_handle, &x[myID * len[0] * len[1]]);
        myStatus = DftiFreeDescriptor (&my_desc_handle);
    } /* End OpenMP parallel region */
}
```

次の例 C-28 では、共通のディスクリプターが複数のスレッドで使用される並列カスタムプログラムについて説明する（「[ユーザースレッド数](#)」のケース 3 を参照）。

この場合、DftiCommitDescriptor() 関数によって DFT の初期化が実行された後に、スレッド数と他の構成パラメーターを変更してはならない。

例 C-28 共通のディスクリプターを伴う並列モードの使用

```
// set number of threads inside Intel MKL:
//rem set MKL_SERIAL = YES      - is not required since one-threaded mode for
Intel MKL is forced automatically
// set OMP_NUM_THREADS = 4     - multi-threaded mode for customer

#include "mkl_dfti.h"
```

例 C-28 共通のディスクリプターを伴う並列モードの使用 (続き)

```
#include "omp.h"
void main ()
{
    float _Complex x[200][100];
    long status;
    DFTI_DESCRIPTOR_HANDLE desc_handle;
    int nThread = omp_get_max_threads ();
    long len[2];
    //...put input data into x[j][k] 0<=j<=199, 0<=k<=99
    len[0] = 50; len[1] = 100;

    status = DftiCreateDescriptor (desc_handle, DFTI_SINGLE, DFTI_COMPLEX, 2, len);
    status = DftiSetValue (desc_handle, DFTI_NUMBER_OF_USER_THREADS, nThread);
    status = DftiCommitDescriptor (desc_handle);

    // each thread calculates real DFT for matrix (50*100)
    #pragma omp parallel num_threads(nThread)
    {
        long myStatus;
        int myID = omp_get_thread_num ();

        myStatus = DftiComputeForward (desc_handle, &x[myID * len[0] * len[1]]);
    } /* End OpenMP parallel region */

    status = DftiFreeDescriptor (&desc_handle);
}
```

クラスター DFT 関数の例

次の C の例は、クラスター DFT を使用して 2 次元のアウトオブプレース FFT を計算する。

例 C-29 2D アウトオブプレース・クラスター DFT 計算

```
DFTI_DESCRIPTOR_DM_HANDLE desc;
long len[2],v,i,j,n,s;
Complex *in,*out;

MPI_Init(...);
// Create descriptor for 2D FFT
len[0]=nx;
len[1]=ny;
DftiCreateDescriptorDM(MPI_COMM_WORLD,&desc,DFTI_DOUBLE,DFTI_COMPLEX,2,len);
// Ask necessary length of in and out arrays and allocate memory
DftiGetValueDM(desc,CDFT_LOCAL_SIZE,&v);
in=(Complex*)malloc(v*sizeof(Complex));
out=(Complex*)malloc(v*sizeof(Complex));
// Fill local array with initial data.Current process performs n rows, 0
row of in corresponds to s row of virtual global array
DftiGetValueDM(desc,CDFT_LOCAL_NX,&n);
DftiGetValueDM(desc,CDFT_LOCAL_X_START,&s);
// Virtual global array globalIN is defined by function f as
globalIN[i*ny+j]=f(i,j)
for(i=0;i<n;i++)
    for(j=0;j<ny;j++) in[i*ny+j]=f(i+s,j);
// Set that we want out-of-place transform (default is DFTI_INPLACE)
DftiSetValueDM(desc,DFTI_PLACEMENT,DFTI_NOT_INPLACE);
// Commit descriptor, calculate FFT, free descriptor
DftiCommitDescriptorDM(desc);
DftiComputeForwardDM(desc,in,out);

// Virtual global array globalOUT is defined by function g as
globalOUT[i*ny+j]=g(i,j)
// Now out contains result of FFT.out[i*ny+j]=g(i+s,j)
DftiFreeDescriptorDM(&desc);
free(in);
free(out);
MPI_Finalize();
```

次の C の例は、ユーザー定義のワークスペースで達成された 1 次元のインプレース・クラスター DFT 計算を示す。

例 C-30 **1D インプレース・クラスター DFT 計算**

```
DFTI_DESCRIPTOR_DM_HANDLE desc;
long len,v,i,n_out,s_out;
Complex *in,*work;

MPI_Init(...);
// Create descriptor for 1D FFT
DftiCreateDescriptorDM(MPI_COMM_WORLD,&desc,DFTI_DOUBLE,DFTI_COMPLEX,1,1
en);
// Ask necessary length of array and workspace and allocate memory
DftiGetValueDM(desc,CDFT_LOCAL_SIZE,&v);
in=(Complex*)malloc(v*sizeof(Complex));
work=(Complex*)malloc(v*sizeof(Complex));
// Fill local array with initial data.Local array has n elements, 0
element of in corresponds to s element of virtual global array
DftiGetValueDM(desc,CDFT_LOCAL_NX,&n);
DftiGetValueDM(desc,CDFT_LOCAL_X_START,&s);

// Set work array as a workspace
DftiSetValueDM(desc,CDFT_WORKSPACE,work);
// Virtual global array globalIN is defined by function f as
globalIN[i]=f(i)
for(i=0;i<n;i++) in[i]=f(i+s);
// Commit descriptor, calculate FFT, free descriptor
DftiCommitDescriptorDM(desc);
DftiComputeForwardDM(desc,in);
DftiGetValueDM(desc,CDFT_LOCAL_OUT_NX,&n_out);
DftiGetValueDM(desc,CDFT_LOCAL_OUT_X_START,&s_out);
// Virtual global array globalOUT is defined by function g as
globalOUT[i]=g(i)

// Now in contains result of FFT.Local array has n_out elements, 0
element of in corresponds to s_out element of virtual global
array.in[i]==g(i+s_out)
DftiFreeDescriptorDM(&desc);
free(in);
free(work);
MPI_Finalize();
```

区間連立線形方程式ソルバーのコード例

このセクションでは、第 12 章の「[区間線形ソルバー](#)」で説明されたルーチンを使用するコード例を示す。これらのルーチンは、区間連立線形方程式のエンクロージャと解の集合の推定を算出し、区間行列とその逆転のプロパティをチェックする。

例 C-31 区間 Gauss-Seidel 法

区間連立線形代数方程式が与えられると、区間 Gauss-Seidel 法 ([?gegss](#) ルーチンとして実装される) は、指定された区間箱によって限定される解の集合の目的部分のエンクロージャに適用される。

E. Hansen によって最初に提案された、次の区間連立線形方程式について考える。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix} x = \begin{pmatrix} [0, 120] \\ [60, 240] \end{pmatrix}$$

([Hansen92](#) を参照) この解の集合は次の区間箱と交差するか？

$$\begin{pmatrix} [0, 200] \\ [0, 200] \end{pmatrix}$$

上記の質問への答えとなるプログラムを以下に示す。

```
PROGRAM DIGEGSS_EXAMPLE
!
! Example program enclosing the solution set to a square interval
! linear system by interval Gauss-Seidel iterative method
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2
INTEGER :: NRHS, LDA, LDB, NITS, INFO, I, J
REAL(8) :: EPSILON
TYPE(D_INTERVAL) :: A(DIM,DIM), B(DIM,1), ENCL(DIM,1)
CHARACTER(1) :: TRANS
!-----!
PRINT 300
!-----!
!!
! Initializing the input data - !
```

```
!!
TRANS = 'N'
NRHS = 2
A(1,1) = DINTERVAL(2.,3.); A(1,2) = DINTERVAL(0.,1.);
A(2,1) = DINTERVAL(0.,1.); A(2,2) = DINTERVAL(2.,3.);
LDA = 2
B(1,1) = DINTERVAL(0.,120.); B(2,1) = DINTERVAL(60.,240.);
LDB = 2
EPSILON = 1.D-6
NITS = 20
!-----!
!
! Assigning the bounding box for the solution set -
DO I = 1, DIM
ENCL(I,1) = DINTERVAL(0.,200.)
END DO
!-----!
CALL DIGEGSS(TRANS, DIM, NRHS, A, LDA, B, LDB, ENCL, EPSILON, NITS, INFO )
!-----!
!
! Outputting the solution
IF( INFO /= 0 ) THEN
PRINT 400
ELSE
PRINT 600
DO I = 1, DIM
PRINT *, '[', B(I,1), ']'
END DO
END IF
!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM **** ', /, &
' by interval Gauss-Seidel method ')
400 FORMAT (/, ' The interval Gauss-Seidel method fails. ')
600 FORMAT (/, ' Outer interval estimate of the solution set:', /)
!-----!
END PROGRAM DIGEGSS_EXAMPLE
```

2 つの実数からそれぞれを終点とする区間を作成する DINTERVAL 関数によって、行列 **A** と右辺ベクトル **B** の成分に倍精度区間が割り当てられる。上記のコードを実行すると解答が得られる。

```
**** SOLVING INTERVAL LINEAR SYSTEM ****
by interval Gauss-Seidel method
Outer interval estimate of the solution set:
[ 0.0000000000000000E+000 60.00000000000000 ]
[ 0.0000000000000000E+000 120.00000000000000 ]
```

[Hansen92](#) にある対応するグラフを参照すると、作成された区間箱が解の集合の必要な部分のエンクロージャとなることを確認できる。さらに、最も厳密なエンクロージャであることも確認できる。

例 C-32 Hansen-Blied-Rohn プロシージャ

次の Fortran-90 のプログラムは、区間連立線形方程式の解の集合の外側区間を推定する、“半区間” Hansen-Blied-Rohn プロシージャを実装する digehbs ルーチンの使用方法を示す。

```
PROGRAM DIGEHBS_EXAMPLE
!
! Example program for enclosing the solution set to square interval
! interval system of equations by Hansen-Blied-Rohn procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2
INTEGER :: LDA, LDB, INFO, I, J
TYPE(D_INTERVAL), ALLOCATABLE :: A(:, :), B(:)
CHARACTER(1) :: TRANS
!-----!
PRINT 300
!-----!
!
! Initializing the input data -
!
TRANS = 'N'
ALLOCATE( A(DIM,DIM), B(DIM) )
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
```

```

A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,2.)
LDA = 2
B(1) = DINTERVAL(0.,2.); B(2) = DINTERVAL(0.,2.)
LDB = 2
!-----!
CALL DIGEHBS( TRANS, DIM, A, LDA, B, LDB, INFO )
!-----!
IF( INFO /= 0 ) THEN
PRINT 400
ELSE
PRINT 600
DO I = 1, DIM
PRINT *, I, ' ) [', B(I), ']'
END DO
END IF
!-----!
DEALLOCATE( A, B )
!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM ****', /, &
' by Hansen-Bliek-Rohn procedure ', /)
400 FORMAT (/, ' The matrix of the system is not an H-matrix,', /, &
' Hansen-Bliek-Rohn procedure fails.', /)
600 FORMAT (/, ' Enclosure of the solution set: ', /)
!-----!
END PROGRAM DIGEHBS_EXAMPLE

```

ただし、プログラムの出力は次のようになる。

```

**** SOLVING INTERVAL LINEAR SYSTEM ****
by Hansen-Bliek-Rohn procedure
The matrix of the system is not an H-matrix,
Hansen-Bliek-Rohn procedure fails.

```

これは、プログラムが次の区間連立線形方程式に適用されたためである。

$$\begin{pmatrix} [2, 4] & [-2, 1] \\ [-1, 2] & [2, 4] \end{pmatrix} x = \begin{pmatrix} [0, 2] \\ [0, 2] \end{pmatrix}$$

ここで、区間行列は H- 行列 (優対角性を持たない) ではない。

ただし、digemip ルーチンによる前処理は問題の解決に役立つ。結果に区間連立線形方程式の予備の前処理を組み込む次の変更されたプログラムは、行列に優対角性を持たせ、問題に対して許容範囲の結果を算出する。

```

PROGRAM DIGEMIP_DIGEHBS_EXAMPLE
!
! Example program for enclosing the solution set to square interval
! interval system of equations by Hansen-Bliek-Rohn procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2, NRHS = 1
INTEGER :: LDA, LDB, INFO, I, J
TYPE(D_INTERVAL), ALLOCATABLE :: A(:, :), B(:)
CHARACTER(1) :: TRANS
!-----!
PRINT 300
!-----!
!
! Initializing the input data -
!
TRANS = 'N'
ALLOCATE( A(DIM,DIM), B(DIM) )
A(1,1) = DINTERVAL(2.,4.); A(1,2) = DINTERVAL(-2.,1.)
A(2,1) = DINTERVAL(-1.,2.); A(2,2) = DINTERVAL(2.,2.)
LDA = 2
B(1) = DINTERVAL(0.,2.); B(2) = DINTERVAL(0.,2.)
LDB = 2
!-----!
CALL DIGEMIP( DIM, NRHS, A, LDA, B, LDB, INFO )
CALL DIGEHBS( TRANS, DIM, A, LDA, B, LDB, INFO )
!-----!
IF( INFO /= 0 ) THEN
PRINT 400
ELSE
PRINT 600
DO I = 1, DIM
PRINT *, I, ' ) [', B(I), ']'
END DO
END IF

```

```

DEALLOCATE( A, B )
!-----!
300 FORMAT (/, ' **** SOLVING INTERVAL LINEAR SYSTEM ****', /, &
' by Hansen-Bliek-Rohn procedure ', /)
400 FORMAT (/, ' The matrix of the system is not an H-matrix,', /, &
' Hansen-Bliek-Rohn procedure fails.', /)
600 FORMAT (/, ' Enclosure of the solution set: ', /)
!-----!
END PROGRAM DIGEMIP_DIGEHS_EXAMPLE

```

この場合、プログラムの出力は次のようになる。

```

**** SOLVING INTERVAL LINEAR SYSTEM ****
by Hansen-Bliek-Rohn procedure
Enclosure of the solution set:
1 ) [ -4.23529411764708 10.7058823529412 ]
2 ) [ -6.70588235294119 10.8235294117647 ]

```

(コンピューター・アーキテクチャーによって最後の桁は異なる。)

例 C-33 逆区間行列のエンクロージャの計算

次のような 2×2 区間行列が与えられた場合、

$$\begin{pmatrix} 3 & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix}$$

次の Fortran-90 のコードは、逆区間行列のエンクロージャを計算する。

```

PROGRAM SIGESZI_EXAMPLE
!
!Example program inverting an interval matrix by Sczulz iterative procedure
!
!-----!
USE INTERVAL_ARITHMETIC
IMPLICIT NONE
!-----!
INTEGER, PARAMETER :: DIM = 2, LDA = 2

```

```

INTEGER :: INFO, I, J
TYPE(S_INTERVAL), ALLOCATABLE :: A(:, :)
!-----!
PRINT 300
!-----!
!
! Initializing the input data -
!
ALLOCATE( A(LDA,DIM) )
A(1,1) = SINTERVAL(3.,3.); A(1,2) = SINTERVAL(0.,1.)
A(2,1) = SINTERVAL(1.,2.); A(2,2) = SINTERVAL(2.,3.)
!-----!
CALL SIGESZI ( DIM, A, LDA, INFO )
!-----!
PRINT 600
DO I = 1, DIM
PRINT *, ( '[' , A(I,J), ']' , J = 1, DIM )
END DO
DEALLOCATE( A )
!-----!
300 FORMAT ( /, ' **** INVERTING INTERVAL MATRIX ****', /, &
' by interval Schulz method ' )
400 FORMAT ( /, ' Schulz inversion procedure failed.', / )
600 FORMAT ( /, ' Enclosure of the inverse matrix ', / )
!-----!
END PROGRAM SIGESZI_EXAMPLE

```

出力は次のようになる (アーキテクチャーにより若干の違いがある)。

```

**** INVERTING INTERVAL MATRIX ****
by interval Schulz method
Enclosure of the inverse matrix
[ 0.2407409 0.5000001 ][ -0.2500000 0.1018518 ]
[ -0.5000000 5.5555239E-02 ][ 0.1388889 0.7500001 ]

```

同時に、行列の (1,1) 成分を区間 [2, 3] に変更すると、sigeszi プロシージャは新しい区間行列に対する逆の有限エンクロージャの計算に失敗する。

$$\begin{pmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{pmatrix}$$

しかし、このような行列を持つ区間連立線形方程式は、区間ガウス法や区間 Gauss-Seidel 法など、特別なルーチンを用いて解くことができる ([例 C-31](#) を参照)。

PDE サポートのコード例

このセクションでは、「[偏微分方程式のサポート](#)」の章で説明されたルーチンのコード例を示す。例は、サブセクション「[三角変換のコード例](#)」および「[ポアソン・ライブラリーのコード例](#)」でグループ化されている。

三角変換のコード例

このセクションのコードは、異なる境界条件 (DD、NN および ND) を使用して 3 つの単純な 1D ヘルムホルツ問題の解を計算する。ここで、“D” はディリクレ境界条件を表し、“N” はノイマン境界条件を意味する。

[例 C-34](#) に C のコード例を、[例 C-35](#) に Fortran-90 のコード例をそれぞれ示す。

三角変換ルーチンを使用して解を計算するアルゴリズムは、[第 13 章](#) で説明している。DD の場合、正弦変換が計算される。NN の場合、余弦変換を使用する。ND の場合、スタガード余弦変換を使用する。

解かれたヘルムホルツ問題のその他の詳細は、計算された解とともに出力される。

[例 C-34](#) が正常に実行されると、次のテキストが出力される ([例 C-35](#) も同様のテキストが出力される)。

MKL 三角変換の使用例

```
*****

This example gives the the solutions of the 1D differential problems
with the equation -u''+u=f(x), 0<x<1,
and with 3 types of boundary conditions:
DD case: u(0)=u(1)=0,
NN case: u'(0)=u'(1)=0,
ND case: u'(0)=u(1)=0.

-----

In general, the error should be of order O(1.0/n**2)
For this example, the value of n is 8
The approximation error should be of order 5.0e-002 if everything is OK
-----

Note that n should be even to use Trigonometric Transforms !
-----

                        DOUBLE PRECISION COMPUTATIONS
=====

The computed solution of DD problem is
```

```
u[0]= 0.000
u[1]= 0.153
u[2]= 0.524
u[3]= 0.895
u[4]= 1.049
u[5]= 0.895
u[6]= 0.524
u[7]= 0.153
u[8]= 0.000
```

```
Error=4.873e-002
```

The computed solution of NN problem is

```
u[0]=-0.026
u[1]= 0.128
u[2]= 0.500
u[3]= 0.872
u[4]= 1.026
u[5]= 0.872
u[6]= 0.500
u[7]= 0.128
u[8]=-0.026
```

```
Error=2.583e-002
```

The computed solution of ND problem is

```
u[0]=-0.009
u[1]= 0.145
u[2]= 0.517
u[3]= 0.890
u[4]= 1.045
u[5]= 0.892
u[6]= 0.522
u[7]= 0.152
u[8]= 0.000
```

```
Error=4.470e-002
```

計算用のCコードは次のようになる。

例 C-34 **1D ヘルムホルツ問題の計算例 (C)**

```
*****
!
!                               INTEL CONFIDENTIAL
!
!   Copyright(C) 2005 Intel Corporation.All Rights Reserved.
!
!   The source code contained or described herein and all documents related to
!   the source code ("Material") are owned by Intel Corporation or its suppliers
!   or licensors.Title to the Material remains with Intel Corporation or its
!   suppliers and licensors.The Material contains trade secrets and proprietary
!   and confidential information of Intel or its suppliers and licensors.The
!   Material is protected by worldwide copyright and trade secret laws and
!   treaty provisions.No part of the Material may be used, copied, reproduced,
!   modified, published, uploaded, posted, transmitted, distributed or disclosed
!   in any way without Intel's prior express written permission.
!
!   No license under any patent, copyright, trade secret or other intellectual
!   property right is granted to or conferred upon you by disclosure or delivery
!   of the Materials, either expressly, by implication, inducement, estoppel or
!   otherwise.Any license under such intellectual property rights must be
!   express and approved by Intel in writing.
!
!
*****
```

例 C-34 1D ヘルムホルツ問題の計算例 (C) (続き)

```
!*****
! Content:
! Double precision C test example for trigonometric transforms
!*****
!
! This example gives the solution of the 1D differential problems
! with the equation  $-u''+u=f(x)$ ,  $0<x<1$ , and with 3 types of boundary conditions:
!  $u(0)=u(1)=0$  (DD case), or  $u'(0)=u'(1)=0$  (NN case), or  $u'(0)=u(1)=0$  (ND case)
*/

#include <stdio.h>
#include <malloc.h>
#include <math.h>
#include "mkl_dfti.h"
#include "mkl_trig_transforms.h"

int main(void)
{
    int n=8, i, k, tt_type;
    int ir, ipar[128];
    /* Note that the size of the transform n must be even !!! */
    double pi=3.14159265358979324, xi, c;
    double c1, c2, c3, c4, c5, c6;
    double *u, *f, *dpar, *lambda;
    DFTI_DESCRIPTOR_HANDLE handle = 0;

    /* Printing the header for the example */
    printf("\n Example of use of MKL Trigonometric Transforms\n");
    printf(" *****\n\n");
    printf(" This example gives the the solutions of the 1D differential problems\n");
    printf(" with the equation  $-u''+u=f(x)$ ,  $0<x<1$ , \n");
    printf(" and with 3 types of boundary conditions:\n");
    printf(" DD case:  $u(0)=u(1)=0$ ,\n");
```

例 C-34 1D ヘルムホルツ問題の計算例 (C) (続き)

```

printf(" NN case: u'(0)=u'(1)=0,\n");
printf(" ND case: u'(0)=u(1)=0.\n");

printf(" -----\n");
printf(" In general, the error should be of order O(1.0/n**2)\n");
printf(" For this example, the value of n is %li\n", n);
printf(" The approximation error should be of order 5.0e-002 if
everything is OK\n");
printf("
-----\n");
printf(" Note that n should be even to use Trigonometric Transforms !\n");
printf("
-----\n");
printf("          DOUBLE PRECISION COMPUTATIONS          \n");

printf("=====\n\n");

u=(double*)malloc((n+1)*sizeof(double));
f=(double*)malloc((n+1)*sizeof(double));
dpar=(double*)malloc((3*n/2+1)*sizeof(double));
lambda=(double*)malloc((n+1)*sizeof(double));

for(i=0;i<=2;i++)
{
    /* Varying the type of the transform */
    tt_type=i;

    /* Computing test solutions u(x) */
    for(k=0;k<=n;k++)
    {
        xi=1.0E0*k/n;
        u[k]=pow(sin(pi*xi),2.0E0);
    }
}

```

例 C-34 **1D ヘルムホルツ問題の計算例 (C) (続き)**

```
/* Computing the right-hand side f(x) */
for(k=0;k<=n;k++)
{
    f[k]=(4.0E0*(pi*pi)+1.0E0)*u[k]-2.0E0*(pi*pi);
}
/* Computing the right-hand side for the algebraic system */
for(k=0;k<=n;k++)
{
    f[k]=f[k]/(n*n);
}
if (tt_type==0)
{
    /* The Dirichlet boundary conditions */
    f[0]=0.0E0;
    f[n]=0.0E0;
}
if (tt_type==2)
{
    /* The mixed Neumann-Dirichlet boundary conditions */
    f[n]=0.0E0;
}
/* Computing the eigenvalues for the three-point finite-difference
problem */
if (tt_type==0||tt_type==1)
{
    for(k=0;k<=n;k++)
    {
        lambda[k]=pow(2.0E0*sin(0.5E0*pi*k/n),2.0E0)+1.0E0/(n*n);
    }
}
}
```

例 C-34 1D ヘルムホルツ問題の計算例 (C) (続き)

```
    if (tt_type==2)
    {
        for(k=0;k<=n;k++)
        {
lambda[k]=pow(2.0E0*sin(0.25E0*pi*(2*k+1)/n),2.0E0)+1.0E0/(n*n);
        }
    }

    /* Computing the solution of 1D problem using trigonometric
transforms
    First we initialize the transform */
    d_init_trig_transform(&n,&tt_type,ipar,dpar,&ir);
    if (ir!=0) goto FAILURE;
    /* Then we commit the transform.Note that the data in f will be
changed at this stage !
    If you want to keep them, save them in some other array before the
call to the routine */
    d_commit_trig_transform(f,&handle,ipar,dpar,&ir);
    if (ir!=0) goto FAILURE;
    /* Now we can apply trigonometric transform */
    d_forward_trig_transform(f,&handle,ipar,dpar,&ir);
    if (ir!=0) goto FAILURE;

    /* Scaling the solution by the eigenvalues */
    for(k=0;k<=n;k++)
    {
        f[k]=f[k]/lambda[k];
    }

    /* Now we can apply trigonometric transform once again as ONLY
input vector f has changed */
    d_backward_trig_transform(f,&handle,ipar,dpar,&ir);
```

例 C-34 **1D ヘルムホルツ問題の計算例 (C) (続き)**

```
if (ir!=0) goto FAILURE;

/* Cleaning the memory used by handle
Now we can use handle for other kind of trigonometric transform */
free_trig_transform(&handle,ipar,&ir);
if (ir!=0) goto FAILURE;

/* Performing the error analysis */
c1=0.0E0;
c2=0.0E0;
c3=0.0E0;
for(k=0;k<=n;k++)
{
    /* Computing the absolute value of the exact solution */
    c4=fabs(u[k]);
    /* Computing the absolute value of the computed solution
Note that the solution is now in place of the former right-hand
side ! */
    c5=fabs(f[k]);
    /* Computing the absolute error */
    c6=fabs(f[k]-u[k]);
    /* Computing the maximum among the above 3 values c4-c6 */
    if (c4>c1) c1=c4;
    if (c5>c2) c2=c5;
    if (c6>c3) c3=c6;
}

/* Printing the results */
if (tt_type==0)
{
    printf("The computed solution of DD problem is\n\n");
    for(k=0;k<=n;k++)
```

例 C-34 1D ヘルムホルツ問題の計算例 (C) (続き)

```
        {
            printf("u[%1i]=%6.3f\n",k,f[k]);
        }
        printf("\nError=%6.3e\n\n",c3/c1);
    }
    if (tt_type==1)
    {
        printf("The computed solution of NN problem is\n\n");
        for(k=0;k<=n;k++)
        {
            printf("u[%1i]=%6.3f\n",k,f[k]);
        }
        printf("\nError=%6.3e\n\n",c3/c1);
    }
    if (tt_type==2)
    {
        printf("The computed solution of ND problem is\n\n");
        for(k=0;k<=n;k++)
        {
            printf("u[%1i]=%6.3f\n",k,f[k]);
        }
        printf("\nError=%6.3e\n\n",c3/c1);
    }
    /* End of the loop over the different kind of transforms and problems */

}

/* Jumping over failure message */
goto SUCCESS;
/* Failure message to print if something went wrong */
FAILURE: printf("Failed to compute the solution(s)...");

SUCCESS: return 0;
    /* End of the example code */
}
```

計算用の Fortran コードは次のようになる。

例 C-35 1D ヘルムホルツ問題の計算例 (Fortran-90)

```
!*****
!
!                               INTEL CONFIDENTIAL
!
!   Copyright(C) 2005 Intel Corporation.All Rights Reserved.
!
!   The source code contained or described herein and all documents related to
!   the source code ("Material") are owned by Intel Corporation or its suppliers
!   or licensors.Title to the Material remains with Intel Corporation or its
!   suppliers and licensors.The Material contains trade secrets and proprietary
!   and confidential information of Intel or its suppliers and licensors.The
!   Material is protected by worldwide copyright and trade secret laws and
!   treaty provisions.No part of the Material may be used, copied, reproduced,
!   modified, published, uploaded, posted, transmitted, distributed or disclosed
!   in any way without Intel's prior express written permission.
!
!   No license under any patent, copyright, trade secret or other intellectual
!   property right is granted to or conferred upon you by disclosure or delivery
!   of the Materials, either expressly, by implication, inducement, estoppel or
!   otherwise.Any license under such intellectual property rights must be
!   express and approved by Intel in writing.
!
```

例 C-35 1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)

```

!
!*****
! Content:
! Double precision Fortran90 test example for trigonometric transforms
!*****
! This example gives the solution of the 1D differential problems
! with the equation  $-u''+u=f(x)$ ,  $0<x<1$ , and with 3 types of boundary conditions:
!  $u(0)=u(1)=0$  (DD case), or  $u'(0)=u'(1)=0$  (NN case), or  $u'(0)=u(1)=0$  (ND case)

program d_tt_example_bvp

    use mkl_dfti
    use mkl_trig_transforms

    implicit none

        integer n, i, k,j, tt_type
        integer ir, ipar(128)
! Note that the size of the transform n must be even !!!
        parameter (n=8)
        double precision pi, xi
        double precision c1, c2, c3, c4, c5, c6
        double precision u(n+1), f(n+1), dpar(3*n/2+1), lambda(n+1)
        parameter (pi=3.14159265358979324D0)
        type(dfti_descriptor), pointer :: handle
! Printing the header for the example
        print *, ''
        print *, ' Example of use of MKL Trigonometric Transforms'
        print *, ' *****'
        print *, ''

```

例 C-35 1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)

```

print *, ' This example gives the solution of the 1D differential problems'
print *, ' with the equation -u''+u=f(x), 0<x<1, '
print *, ' and with 3 types of boundary conditions:'
print *, ' DD case: u(0)=u(1)=0,'
print *, ' NN case: u''(0)=u''(1)=0,'
print *, ' ND case: u''(0)=u(1)=0.'
print *, '
-----'

print *, ' In general, the error should be of order O(1.0/n**2)'
print *, ' For this example, the value of n is', n
print *, ' The approximation error should be of order 0.5E-01, if
everything is OK'
print *, '
-----'

print *, ' Note that n should be even to use Trigonometric Transforms !'
print *, '
-----'

print *, '                                DOUBLE PRECISION COMPUTATIONS
,
print*, '=====
'

print *, ''

do i=0,2
! Varying the type of the transform
tt_type=i
! Computing test solution u(x)
do k=1,n+1
xi=1.0D0*(k-1)/n
u(k)=dsin(pi*xi)**2
end do

```

例 C-35 1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)

```
! Computing the right-hand side f(x)
      do k=1,n+1
        f(k)=(4.0D0*(pi**2)+1.0D0)*u(k)-2.0D0*(pi**2)
      end do
! Computing the right-hand side for the algebraic system
      do k=1,n+1
        f(k)=f(k)/(n**2)
      end do
      if (tt_type.eq.0) then
! The Dirichlet boundary conditions
        f(1)=0.0D0
        f(n+1)=0.0D0
      end if
      if (tt_type.eq.2) then
! The mixed Neumann-Dirichlet boundary conditions
        f(n+1)=0.0D0
      end if

! Computing the eigenvalues for the three-point finite-difference problem
      if (tt_type.eq.0.or.tt_type.eq.1) then
        do k=1,n+1
          lambda(k)=(2.0D0*dsin(0.5D0*pi*(k-1)/n))**2+1.0D0/(n**2)
        end do
      end if
      if (tt_type.eq.2) then
        do k=1,n+1
          lambda(k)=(2.0D0*dsin(0.25D0*pi*(2*k-1)/n))**2+1.0D0/(n**2)
        end do
      end if
```

例 C-35 **1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)**

```
! Computing the solution of 1D problem using trigonometric transforms
! First we initialize the transform
      CALL D_INIT_TRIG_TRANSFORM(n,tt_type,ipar,dpar,ir)
      if (ir.ne.0) goto 99

! Then we commit the transform.Note that the data in f will be changed at this
stage !
! If you want to keep them, save them in some other array before the call to the
routine

      CALL D_COMMIT_TRIG_TRANSFORM(f,handle,ipar,dpar,ir)
      if (ir.ne.0) goto 99

! Now we can apply trigonometric transform
      CALL D_FORWARD_TRIG_TRANSFORM(f,handle,ipar,dpar,ir)
      if (ir.ne.0) goto 99

! Scaling the solution by the eigenvalues
      do k=1,n+1
        f(k)=f(k)/lambda(k)
      end do

! Now we can apply trigonometric transform once again as ONLY input vector f has
changed

      CALL D_BACKWARD_TRIG_TRANSFORM(f,handle,ipar,dpar,ir)
      if (ir.ne.0) goto 99

! Cleaning the memory used by handle
! Now we can use handle for other KIND of trigonometric transform
      CALL FREE_TRIG_TRANSFORM(handle,ipar,ir)
      if (ir.ne.0) goto 99
```

例 C-35 1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)

```
! Performing the error analysis
      c1=0.0D0
      c2=0.0D0
      c3=0.0D0
      do k=1,n+1
! Computing the absolute value of the exact solution
          c4=dabs(u(k))
! Computing the absolute value of the computed solution
! Note that the solution is now in place of the former right-hand side !
          c5=dabs(f(k))
! Computing the absolute error
          c6=dabs(f(k)-u(k))
! Computing the maximum among the above 3 values c4-c6
          if (c4.gt.c1) c1=c4
          if (c5.gt.c2) c2=c5
          if (c6.gt.c3) c3=c6
      end do

! Printing the results
      if (tt_type.eq.0) then
          print *, 'The computed solution of DD problem is'
          print *, ''
          do k=1,n+1
              write(*,11) k,f(k)
          end do
          print *, ''
          write(*,12) c3/c1
          print *, ''
      end if
```

例 C-35 **1D ヘルムホルツ問題の計算例 (Fortran-90) (続き)**

```
        if (tt_type.eq.1) then
            print *, 'The computed solution of NN problem is'
            print *, ''
            do k=1,n+1
                write(*,11) k,f(k)
            end do
            print *, ''
            write(*,12) c3/c1
            print *, ''
        end if
        if (tt_type.eq.2) then
            print *, 'The computed solution of ND problem is'
            print *, ''
            do k=1,n+1
                write(*,11) k,f(k)
            end do
            print *, ''
            write(*,12) c3/c1
            print *, ''
        end if
! End of the loop over the different kind of transforms and problems
        end do

! Jumping over failure message
        go to 1
! Failure message to print if something went wrong
99      continue
        print *, 'Failed to compute the solution(s)...'

1       continue

! Print formats
11      format(1x,'u(',I1,')=',F6.3)
12      format(1x,'Relative error =',E10.3)

! End of the example code
end
```

ポアソン・ライブラリーのコード例

以下のコードは、

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 8\pi^2 \sin 2\pi x \cdot \sin 2\pi y$$

矩形 $0 < x < 1$, $0 < y < 1$ における 2D ポアソン問題の近似解を計算する。

次の境界条件が使用される。

- ディリクレ境界条件

$$u(0, y) = u(1, y) = 1 \\ 0 \leq y \leq 1$$

- ノイマン境界条件

$$\frac{\partial u}{\partial n}(x, 0) = -2\pi \sin 2\pi x, \frac{\partial u}{\partial n}(x, 1) = 2\pi \sin 2\pi x \\ 0 < x < 1$$

正確な解は次のように知られている。

$$u(x, y) = \sin 2\pi x \cdot \sin 2\pi y + 1$$

[例 C-36](#) に C のコード例を、[例 C-37](#) に Fortran-90 のコード例をそれぞれ示す。インテル MKL DFT インターフェイスの使用による制限により、PL インターフェイスは Fortran-77 から起動できない点に注意すること。

ポアソン・ライブラリー・ルーチンを使用してポアソン問題の近似解を計算するアルゴリズムは、[第 13 章](#) で説明している。解いたポアソン問題の詳細および誤差は、計算された解と正確な解の間に出力される。

[例 C-36](#) が正常に実行されると、次のテキストが出力される ([例 C-37](#) も同様のテキストが出力される)。

MKL ポアソン・ライブラリーの使用例

```
*****
```

```
This example gives the solution of 2D Poisson problem
with the equation -u_xx-u_yy=f(x,y), 0<x<1, 0<y<1,
f(x,y)=(8*pi*pi)*sin(2*pi*x)*sin(2*pi*y),
and with the following boundary conditions:
  u(0,y)=u(1,y)=1 (Dirichlet boundary conditions),
 -u_y(x,0)=-2.0*pi*sin(2*pi*x) (Neumann boundary condition),
  u_y(x,1)= 2.0*pi*sin(2*pi*x) (Neumann boundary condition).
```

```
-----
In general, the error should be of order O(1.0/nx^2+1.0/ny^2)
For this example, the value of nx=ny is 6
The approximation error should be of order 1.0e-01, if everything is OK
-----
```

```
Note that nx should be even to use Poisson Library !
-----
```

```
DOUBLE PRECISION COMPUTATIONS
```

```
=====
```

```
The number of mesh intervals in x-direction is nx=6
```

The number of mesh intervals in y-direction is $n_y=6$

In the mesh point (0.167,0.000) the error between the computed and the true solution is equal to $-7.505e-02$

In the mesh point (0.167,0.167) the error between the computed and the true solution is equal to $4.432e-02$

In the mesh point (0.167,0.333) the error between the computed and the true solution is equal to $6.309e-02$

In the mesh point (0.167,0.500) the error between the computed and the true solution is equal to $-5.551e-16$

In the mesh point (0.167,0.667) the error between the computed and the true solution is equal to $-6.309e-02$

In the mesh point (0.167,0.833) the error between the computed and the true solution is equal to $-4.432e-02$

In the mesh point (0.167,1.000) the error between the computed and the true solution is equal to $7.505e-02$

Double precision 2D Poisson example has successfully PASSED
through all steps of computation!

誤差の実際の内容は、例を実行するために使用したアーキテクチャーおよびオペレーティング・システムによって上記の出力内容と多少異なることがある点に注意すること。

計算用のCコードは次のようになる。

例 C-36 2D ポアソン問題の計算例 (C)

```
/*
*****
/*
                                INTEL CONFIDENTIAL
/*
Copyright(C) 2006 Intel Corporation.All Rights Reserved.
/*
The source code contained or described herein and all documents related to
/*
the source code ("Material") are owned by Intel Corporation or its suppliers
/*
or licensors.Title to the Material remains with Intel Corporation or its
/*
suppliers and licensors.The Material contains trade secrets and proprietary
/*
and confidential information of Intel or its suppliers and licensors.The
/*
Material is protected by worldwide copyright and trade secret laws and
/*
treaty provisions.No part of the Material may be used, copied, reproduced,
/*
modified, published, uploaded, posted, transmitted, distributed or disclosed
/*
in any way without Intel");s prior express written permission.
/*
No license under any patent, copyright, trade secret or other intellectual
/*
property right is granted to or conferred upon you by disclosure or delivery
/*
of the Materials, either expressly, by implication, inducement, estoppel or
/*
otherwise.Any license under such intellectual property rights must be
/*
express and approved by Intel in writing.
/*
*****
/*
Content:
/*
C double precision example of solving 2D Poisson problem in a
/*
rectangular domain using MKL Poisson Library
/*
*****/

#include <stdio.h>
#include <malloc.h>
#include <math.h>
/* Include Poisson Library header files */
#include "mkl_dfti.h"
#include "mkl_poisson.h"

int main(void)
```

例 C-36 2D ポアソン問題の計算例 (C) (続き)

```

{
    /* Note that the size of the transform nx must be even !!! */
    int nx=6, ny=6;
    double pi=3.14159265358979324;

    int ix, iy, i, stat;
    int ipar[128];
    double ax, bx, ay, by, lx, ly, hx, hy, xi, yi, cx, cy;
    double *dpar, *f, *u, *bd_ax, *bd_bx, *bd_ay, *bd_by;
    double q;
    DFTI_DESCRIPTOR_HANDLE xhandle = 0;
    char *BCTYPE;

    /* Printing the header for the example */
    printf("\n Example of use of MKL Poisson Library\n");
    printf(" *****\n\n");
    printf(" This example gives the solution of 2D Poisson problem\n");
    printf(" with the equation -u_xx-u_yy=f(x,y), 0<x<1, 0<y<1,\n");
    printf(" f(x,y)=(8*pi*pi)*sin(2*pi*x)*sin(2*pi*y),\n");
    printf(" and with the following boundary conditions:\n");
    printf(" u(0,y)=u(1,y)=1 (Dirichlet boundary conditions),\n");
    printf(" -u_y(x,0)=-2.0*pi*sin(2*pi*x) (Neumann boundary condition),\n");
    printf(" u_y(x,1)= 2.0*pi*sin(2*pi*x) (Neumann boundary condition).\n");
    printf("
-----\n");
    printf(" In general, the error should be of order O(1.0/nx^2+1.0/ny^2)\n");
    printf(" For this example, the value of nx=ny is %d\n", nx);
    printf(" The approximation error should be of order 1.0e-01, if everything is
OK\n");
    printf("
-----\n");
    printf(" Note that nx should be even to use Poisson Library !\n");
    printf("
-----\n");
    printf("
                                DOUBLE PRECISION COMPUTATIONS
    printf("
=====
\n\n");

    dpar=(double*)malloc((5*nx/2+7)*sizeof(double));
    f=(double*)malloc((nx+1)*(ny+1)*sizeof(double));
    u=(double*)malloc((nx+1)*(ny+1)*sizeof(double));

```

例 C-36 2D ポアソン問題の計算例 (C) (続き)

```

bd_ax=(double*)malloc((ny+1)*sizeof(double));
bd_bx=(double*)malloc((ny+1)*sizeof(double));
bd_ay=(double*)malloc((nx+1)*sizeof(double));
bd_by=(double*)malloc((nx+1)*sizeof(double));

/* Defining the rectangular domain 0<x<1, 0<y<1 for 2D Poisson Solver */
ax=0.0E0;
bx=1.0E0;
ay=0.0E0;
by=1.0E0;

/*****
*
*   Setting the coefficient q to 0.
*   Note that this is the way to use Helmholtz Solver to solve Poisson
problem!
*****/

/
q=0.0E0;

/* Computing the mesh size hx in x-direction */
lx=bx-ax;
hx=lx/nx;
/* Computing the mesh size hy in y-direction */
ly=by-ay;
hy=ly/ny;

/* Filling in the values of the TRUE solution
u(x,y)=sin(2*pi*x)*sin(2*pi*y)+1
in the mesh points into the array u
Filling in the right-hand side
f(x,y)=(8*pi*pi+q)*sin(2*pi*x)*sin(2*pi*y)+q
in the mesh points into the array f.
We choose the right-hand side to correspond to the TRUE solution of
Poisson equation.
Here we are using the mesh sizes hx and hy computed before to compute
the coordinates (xi,yi) of the mesh points */
for(iy=0;iy<=ny;iy++)
{

```

例 C-36 2D ポアソン問題の計算例 (C) (続き)

```

        for(ix=0;ix<=nx;ix++)
        {
            xi=hx*ix/lx;
            yi=hy*iy/ly;

            cx=sin(2*pi*xi);
            cy=sin(2*pi*yi);

            u[ix+iy*(nx+1)]=1.0E0*cx*cy;
            f[ix+iy*(nx+1)]=(8.0E0*pi*pi)*u[ix+iy*(nx+1)];
            u[ix+iy*(nx+1)]=u[ix+iy*(nx+1)]+1.0E0;
        }
    }

    /* Setting the type of the boundary conditions on each side of the
    rectangular domain:
    On the boundary laying on the line x=0(=ax) Dirichlet boundary condition
    will be used
    On the boundary laying on the line x=1(=bx) Dirichlet boundary condition
    will be used
    On the boundary laying on the line y=0(=ay) Neumann boundary condition
    will be used
    On the boundary laying on the line y=1(=by) Neumann boundary condition
    will be used */
    Bctype = "DDNN";

    /* Setting the values of the boundary function G(x,y) that is equal to the
    TRUE solution
    in the mesh points laying on Dirichlet boundaries */
    for(iy=0;iy<=ny;iy++)
    {
        bd_ax[iy]=1.0E0;
        bd_bx[iy]=1.0E0;
    }
    /* Setting the values of the boundary function g(x,y) that is equal to the
    normal derivative
    of the TRUE solution in the mesh points laying on Neumann boundaries */
    for(ix=0;ix<=nx;ix++)
    {
        bd_ay[ix]=-2.0*pi*sin(2*pi*ix/nx);
        bd_by[ix]= 2.0*pi*sin(2*pi*ix/nx);
    }

```

例 C-36 2D ポアソン問題の計算例 (C) (続き)

```
/* Initializing ipar array to make it free from garbage */
for(i=0;i<128;i++)
{
    ipar[i]=0;
}

/* Initializing simple data structures of Poisson Library for 2D Poisson Solver */
d_init_Helmholtz_2D(&ax, &bx, &ay, &by, &nx, &ny, BCtype, &q, ipar, dpar, &stat);
if (stat!=0) goto FAILURE;

/* Initializing complex data structures of Poisson Library for 2D Poisson Solver
NOTE: Right-hand side f may be altered after the Commit step.If you want
to keep it,
you should save it in another memory location! */
d_commit_Helmholtz_2D(f, bd_ax, bd_bx, bd_ay, bd_by, &xhandle, ipar,
dpar, &stat);
if (stat!=0) goto FAILURE;

/* Computing the approximate solution of 2D Poisson problem
NOTE: Boundary data stored in the arrays bd_ax, bd_bx, bd_ay, bd_by should
not be changed
between the Commit step and the subsequent call to the Solver routine/*
Otherwise the results may be wrong.*/
d_Helmholtz_2D(f, bd_ax, bd_bx, bd_ay, bd_by, &xhandle, ipar, dpar, &stat);
if (stat!=0) goto FAILURE;

/* Cleaning the memory used by xhandle */
free_Helmholtz_2D(&xhandle, ipar, &stat);
if (stat!=0) goto FAILURE;
/* Now we can use xhandle to solve another 2D Poisson problem*/

/* Printing the results */
printf("The number of mesh intervals in x-direction is nx=%d\n", nx);
printf("The number of mesh intervals in y-direction is ny=%d\n\n",ny);
```

例 C-36 **2D ポアソン問題の計算例 (C) (続き)**

```
/* Watching the error along the line x=hx */
ix=1;
for(iy=0;iy<=ny;iy++)
{
    printf("In the mesh point (%5.3f,%5.3f) the error between the
computed and the true solution is equal to %10.3e\n", ix*hx, iy*hy,
f[ix+iy*(nx+1)]-u[ix+iy*(nx+1)]);
}

/* Success message to print if everything is OK */
printf("\n Double precision 2D Poisson example has successfully PASSED\n");
printf(" through all steps of computation!\n");

/* Jumping over failure message */
goto SUCCESS;

/* Failure message to print if something went wrong */
FAILURE: printf("\nDouble precision 2D Poisson example FAILED to compute
the solution...\n");

SUCCESS: return 0;

/* End of the example code */
}
```

計算用の Fortran-90 コードは次のようになる。

例 C-37 2D ポアソン問題の計算例 (Fortran-90)

```

!*****
!
!                               INTEL CONFIDENTIAL
!   Copyright(C) 2006 Intel Corporation.All Rights Reserved.
!   The source code contained or described herein and all documents related to
!   the source code ("Material") are owned by Intel Corporation or its suppliers
!   or licensors.Title to the Material remains with Intel Corporation or its
!   suppliers and licensors.The Material contains trade secrets and proprietary
!   and confidential information of Intel or its suppliers and licensors.The
!   Material is protected by worldwide copyright and trade secret laws and
!   treaty provisions.No part of the Material may be used, copied, reproduced,
!   modified, published, uploaded, posted, transmitted, distributed or disclosed
!   in any way without Intel's prior express written permission.
!   No license under any patent, copyright, trade secret or other intellectual
!   property right is granted to or conferred upon you by disclosure or delivery
!   of the Materials, either expressly, by implication, inducement, estoppel or
!   otherwise.Any license under such intellectual property rights must be
!   express and approved by Intel in writing.
!
!*****
!   Content:
!   Fortran-90 double precision example of solving 2D Poisson problem in a
!   rectangular domain using MKL Poisson Library
!
!*****

program Poisson_2D_double_precision

! Include modules defined by mkl_poisson.f90 and mkl_dfti.f90 header files
use mkl_poisson
use mkl_dfti

implicit none

integer nx,ny
! Note that the size of the transform nx must be even !!!
parameter(nx=6, ny=6)
double precision pi
parameter(pi=3.14159265358979324D0)

```

例 C-37 2D ポアソン問題の計算例 (Fortran-90) (続き)

```

integer ix, iy, i, stat
integer ipar(128)
double precision ax, bx, ay, by, lx, ly, hx, hy, xi, yi, cx, cy
double precision dpar(5*nx/2+7)
! Note that proper packing of data in right-hand side array f is
! automatically provided by the following declaration of the arrays
double precision f(nx+1,ny+1), u(nx+1,ny+1)
double precision bd_ax(ny+1), bd_bx(ny+1), bd_ay(nx+1), bd_by(nx+1)
double precision q
type(DFTI_DESCRIPTOR), pointer :: xhandle
character(4) Bctype

! Printing the header for the example
  print *, ''
  print *, ' Example of use of MKL Poisson Library'
  print *, ' *****'
  print *, ''
  print *, ' This example gives the solution of 2D Poisson problem'
  print *, ' with the equation -u_xx-u_yy=f(x,y), 0<x<1, 0<y<1,'
  print *, ' f(x,y)=(8*pi*pi)*sin(2*pi*x)*sin(2*pi*y),'
  print *, ' and with the following boundary conditions:'
  print *, ' u(0,y)=u(1,y)=1 (Dirichlet boundary conditions),'
  print *, ' -u_y(x,0)=-2.0*pi*sin(2*pi*x) (Neumann boundary condition),'
  print *, ' u_y(x,1)= 2.0*pi*sin(2*pi*x) (Neumann boundary condition).'
  print *, ' -----'
  print *, ' In general, the error should be of order O(1.0/nx^2+1.0/ny^2)'
  print *, ' (lx,a,11)', ' For this example, the value of nx=ny is ', nx
  print *, ' The approximation error should be of order 0.1E+0, if everything is OK'
  print *, ' -----'
  print *, ' Note that nx should be even to use Poisson Library !'
  print *, ' -----'
  print *, '
                                DOUBLE PRECISION COMPUTATIONS
  print *, ' ====='
  print *, ''

! Defining the rectangular domain 0<x<1, 0<y<1 for 2D Poisson Solver
ax=0.0D0
bx=1.0D0
ay=0.0D0
by=1.0D0

```

例 C-37 2D ポアソン問題の計算例 (Fortran-90) (続き)

```
!*****
! Setting the coefficient q to 0.
! Note that this is the way to use Helmholtz Solver to solve Poisson problem!
!*****
q=0.0D0

! Computing the mesh size hx in x-direction
lx=bx-ax
hx=lx/nx
! Computing the mesh size hy in y-direction
ly=by-ay
hy=ly/ny

! Filling in the values of the TRUE solution u(x,y)=sin(2*pi*x)*sin(2*pi*y)+1
! in the mesh points into the array u
! Filling in the right-hand side f(x,y)=(8*pi*pi+q)*sin(2*pi*x)*sin(2*pi*y)+q
! in the mesh points into the array f.
! We choose the right-hand side to correspond to the TRUE solution of Poisson equation.
! Here we are using the mesh sizes hx and hy computed before to compute
! the coordinates (xi,yi) of the mesh points
do iy=1,ny+1
  do ix=1,nx+1
    xi=hx*(ix-1)/lx
    yi=hy*(iy-1)/ly

    cx=dsin(2*pi*xi)
    cy=dsin(2*pi*yi)

    u(ix,iy)=1.0D0*cx*cy
    f(ix,iy)=(8.0D0*pi**2)*u(ix,iy)
    u(ix,iy)=u(ix,iy)+1.0D0
  enddo
enddo

! Setting the type of the boundary conditions on each side of the rectangular domain:
! On the boundary laying on the line x=0(=ax) Dirichlet boundary condition will be used
! On the boundary laying on the line x=1(=bx) Dirichlet boundary condition will be used
! On the boundary laying on the line y=0(=ay) Neumann boundary condition will be used
! On the boundary laying on the line y=1(=by) Neumann boundary condition will be used
BCtype = 'DDNN'
```

例 C-37 2D ポアソン問題の計算例 (Fortran-90) (続き)

```
! Setting the values of the boundary function G(x,y) that is equal to the TRUE solution
! in the mesh points laying on Dirichlet boundaries
do iy = 1,ny+1
    bd_ax(iy) = 1.0D0
    bd_bx(iy) = 1.0D0
enddo
! Setting the values of the boundary function g(x,y) that is equal to the normal
derivative
! of the TRUE solution in the mesh points laying on Neumann boundaries
do ix = 1,nx+1
    bd_ay(ix) = -2.0*pi*dsin(2*pi*(ix-1)/nx)
    bd_by(ix) = 2.0*pi*dsin(2*pi*(ix-1)/nx)
enddo

! Initializing ipar array to make it free from garbage
do i=1,128
    ipar(i)=0
enddo

! Initializing simple data structures of Poisson Library for 2D Poisson Solver
call d_init_Helmholtz_2D(ax, bx, ay, by, nx, ny, Bctype, q, ipar, dpar, stat)
if (stat.ne.0) goto 999

! Initializing complex data structures of Poisson Library for 2D Poisson Solver
! NOTE: Right-hand side f may be altered after the Commit step.If you want to keep it,
! you should save it in another memory location!
call d_commit_Helmholtz_2D(f, bd_ax, bd_bx, bd_ay, bd_by, xhandle, ipar, dpar, stat)
if (stat.ne.0) goto 999

! Computing the approximate solution of 2D Poisson problem
! NOTE: Boundary data stored in the arrays bd_ax, bd_bx, bd_ay, bd_by should not be
changed
! between the Commit step and the subsequent call to the Solver routine!
! Otherwise the results may be wrong.
call d_Helmholtz_2D(f, bd_ax, bd_bx, bd_ay, bd_by, xhandle, ipar, dpar, stat)
if (stat.ne.0) goto 999

! Cleaning the memory used by xhandle
call free_Helmholtz_2D(xhandle, ipar, stat)
if (stat.ne.0) goto 999
! Now we can use xhandle to solve another 2D Poisson problem
```

例 C-37 2D ポアソン問題の計算例 (Fortran-90) (続き)

```
! Printing the results
write(*,10) nx
write(*,11) ny
print *, ''
! Watching the error along the line x=hx
ix=2
do iy=1,ny+1
  write(*,12) (ix-1)*hx, (iy-1)*hy, f(ix,iy)-u(ix,iy)
enddo
print *, ''

! Success message to print if everything is OK
  print *, ' Double precision 2D Poisson example has successfully PASSED'
  print *, ' through all steps of computation!'
! Jumping over failure message
  go to 1
! Failure message to print if something went wrong
999 print *, 'Double precision 2D Poisson example FAILED to compute the
solution...'

1 continue

10  format(1x,'The number of mesh intervals in x-direction is nx=',I1)
11  format(1x,'The number of mesh intervals in y-direction is ny=',I1)
12  format(1x,'In the mesh point (' ,F5.3,' ,',F5.3,') the error between the
computed and the true solution is equal to ' , E10.3)

! End of the example code
end
```

BLAS に対する CBLAS インターフェイス



本付録では、CBLAS について説明する。CBLAS は、インテル[®] MKL に導入された BLAS (Basic Linear Algebra Subprograms) に対する C インターフェイスである。

BLAS の場合と同じように、CBLAS インターフェイスには、以下に示すレベルの関数が含まれる。

- 「[レベル 1 の CBLAS](#)」(ベクトル - ベクトル演算)
- 「[レベル 2 の CBLAS](#)」(行列 - ベクトル演算)
- 「[レベル 3 の CBLAS](#)」(行列 - 行列演算)
- 「[スパース CBLAS](#)」(スパースベクトルでの演算)

C インターフェイスを得るには、Fortran ルーチン名の先頭に `cblas_` を付ける (例えば、`dasum` は `cblas_dasum` となる)。すべての CBLAS 関数は、小文字で表す。

複素関数 `?dotc` と `?dotu` は、CBLAS のサブルーチン (void 関数) になる。これらの関数は、最後のパラメーターとして追加される void ポインターを介して複素数型の結果を返す。これらの関数の CBLAS 名には、末尾に `_sub` を付ける。例えば、BLAS 関数 `cdotc` は `cblas_cdotc_sub` に対応する。

CBLAS の引数

CBLAS 関数の引数は、以下の規則に従る。

- 入力引数は、`const` 修飾子で宣言する。
- 非複素数のスカラー入力引数は、値で渡される。
- 複素数のスカラー入力引数は、void ポインターとして渡される。
- 配列の引数は、アドレスで渡される。
- 出力のスカラー引数は、アドレスで渡される。
- BLAS のキャラクター引数は、該当する列挙型で置き換えられる。
- レベル 2 とレベル 3 のルーチンは、最初の引数として型 `CBLAS_ORDER` の追加パラメーターをとる。このパラメーターは、2次元配列が行主体 (`CblasRowMajor`) か列主体 (`CblasColMajor`) のどちらであるかを指定する。

列挙型

CBLAS インターフェイスは、以下に示す列挙型を使用する。

```
enum CBLAS_ORDER {
    CblasRowMajor=101, /* row-major arrays */
    CblasColMajor=102}; /* column-major arrays */
```

D

```
enum CBLAS_TRANSPOSE {
    CblasNoTrans=111,      /* trans='N' */
    CblasTrans=112,       /* trans='T' */
    CblasConjTrans=113}; /* trans='C' */

enum CBLAS_UPLO {
    CblasUpper=121,       /* uplo ='U' */
    CblasLower=122};    /* uplo ='L' */

enum CBLAS_DIAG {
    CblasNonUnit=131,    /* diag ='N' */
    CblasUnit=132};    /* diag ='U' */

enum CBLAS_SIDE {
    CblasLeft=141,       /* side ='L' */
    CblasRight=142};    /* side ='R' */
```

レベル 1 の CBLAS

基本的なベクトル-ベクトル演算を実行する「[BLAS レベル 1 のルーチンと関数](#)」へのインターフェイスである。

[ipps?asum](#)

```
float cblas_sasum(const int N, const float *X, const int incX);
double cblas_dasum(const int N, const double *X, const int incX);
float cblas_scasum(const int N, const void *X, const int incX);
double cblas_dzasum(const int N, const void *X, const int incX);
```

[ipps?axpy](#)

```
void cblas_saxpy(const int N, const float alpha, const float *X, const int incX,
float *Y, const int incY);
void cblas_daxpy(const int N, const double alpha, const double *X, const int
incX, double *Y, const int incY);
void cblas_caxpy(const int N, const void *alpha, const void *X, const int incX,
void *Y, const int incY);
void cblas_zaxpy(const int N, const void *alpha, const void *X, const int incX,
void *Y, const int incY);
```

[ipps?copy](#)

```
void cblas_scopy(const int N, const float *X, const int incX, float *Y, const int
incY);
void cblas_dcopy(const int N, const double *X, const int incX, double *Y, const
int incY);
void cblas_ccopy(const int N, const void *X, const int incX, void *Y, const int
incY);
void cblas_zcopy(const int N, const void *X, const int incX, void *Y, const int
incY);
```

[ipps?dot](#)

```
float cblas_sdot(const int N, const float *X, const int incX,
const float *Y, const int incY);
double cblas_ddot(const int N, const double *X, const int incX,
const double *Y, const int incY);
```

[ipps?sdot](#)

```
float cblas_sdsdot(const int N, const float *SB, const float *SX, const int incX,
const float *SY, const int incY);
double cblas_dsdot(const int N, const float *SX, const int incX, const float *SY,
const int incY);
```

[ipps?dotc](#)

```
void cblas_cdotc_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotc);
void cblas_zdotc_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotc);
```

[ipps?dotu](#)

```
void cblas_cdotu_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotu);
void cblas_zdotu_sub(const int N, const void *X, const int incX, const void *Y,
const int incY, void *dotu);
```

[ipps?nrm2](#)

```
float cblas_snrm2(const int N, const float *X, const int incX);
double cblas_dnrm2(const int N, const double *X, const int incX);
float cblas_scnrm2(const int N, const void *X, const int incX);
double cblas_dznrm2(const int N, const void *X, const int incX);
```

[ipps?rot](#)

```
void cblas_srot(const int N, float *X, const int incX, float *Y, const int incY,
const float c, const float s);
void cblas_drot(const int N, double *X, const int incX, double *Y, const int incY,
const double c, const double s);
```

[ipps?rotg](#)

```
void cblas_srotg(float *a, float *b, float *c, float *s);
void cblas_drotg(double *a, double *b, double *c, double *s);
```

[ipps?rotm](#)

```
void cblas_srotm(const int N, float *X, const int incX, float *Y, const int incY,
const float *P);
void cblas_drotm(const int N, double *X, const int incX, double *Y, const int
incY, const double *P);
```

[ipps?rotmg](#)

```
void cblas_srotmg(float *d1, float *d2, float *b1, const float b2, float *P);
void cblas_drotmg(double *d1, double *d2, double *b1, const double b2, double
*P);
```

[ipps?scal](#)

```
void cblas_sscal(const int N, const float alpha, float *X, const int incX);
void cblas_dscal(const int N, const double alpha, double *X, const int incX);
void cblas_cscal(const int N, const void *alpha, void *X, const int incX);
void cblas_zscal(const int N, const void *alpha, void *X, const int incX);
void cblas_csscal(const int N, const float alpha, void *X, const int incX);
void cblas_zdscal(const int N, const double alpha, void *X, const int incX);
```

[ipps?swap](#)

```
void cblas_sswap(const int N, float *X, const int incX, float *Y, const int incY);
void cblas_dswap(const int N, double *X, const int incX, double *Y, const int
incY);
void cblas_cswap(const int N, void *X, const int incX, void *Y, const int incY);
void cblas_zswap(const int N, void *X, const int incX, void *Y, const int incY);
```

[ippsi?amax](#)

```
CBLAS_INDEX cblas_isamax(const int N, const float *X, const int incX);
CBLAS_INDEX cblas_idamax(const int N, const double *X, const int incX);
CBLAS_INDEX cblas_icamax(const int N, const void *X, const int incX);
CBLAS_INDEX cblas_izamax(const int N, const void *X, const int incX);
```

[ippsi?amin](#)

```
CBLAS_INDEX cblas_isamin(const int N, const float *X, const int incX);
CBLAS_INDEX cblas_idamin(const int N, const double *X, const int incX);
CBLAS_INDEX cblas_icamin(const int N, const void *X, const int incX);
CBLAS_INDEX cblas_izamin(const int N, const void *X, const int incX);
```

レベル 2 の CBLAS

基本的な行列 - ベクトル演算を実行する「[BLAS レベル 2 のルーチン](#)」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS_ORDER の追加パラメーター (最初の引数) をとる。このパラメーターは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

[ipps?gbmv](#)

```
void cblas_sgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const float alpha, const
float *A, const int lda, const float *X, const int incX, const float beta, float
*Y, const int incY);
```

```
void cblas_dgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const double alpha, const
double *A, const int lda, const double *X, const int incX, const double beta,
double *Y, const int incY);
```

```
void cblas_cgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const void *alpha, const
void *A, const int lda, const void *X, const int incX, const void *beta, void *Y,
const int incY);
```

```
void cblas_zgbmv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const int KL, const int KU, const void *alpha, const
void *A, const int lda, const void *X, const int incX, const void *beta, void *Y,
const int incY);
```

[ipps?gemv](#)

```
void cblas_sgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const float alpha, const float *A, const int lda, const
float *X, const int incX, const float beta, float *Y, const int incY);
```

```
void cblas_dgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const double alpha, const double *A, const int lda,
const double *X, const int incX, const double beta, double *Y, const int incY);
```

```
void cblas_cgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zgemv(const enum CBLAS_ORDER order, const enum CBLAS_TRANSPOSE TransA,
const int M, const int N, const void *alpha, const void *A, const int lda, const
void *X, const int incX, const void *beta, void *Y, const int incY);
```

[ipps?ger](#)

```
void cblas_sger(const enum CBLAS_ORDER order, const int M, const int N, const
float alpha, const float *X, const int incX, const float *Y, const int incY, float
*A, const int lda);
```

```
void cblas_dger(const enum CBLAS_ORDER order, const int M, const int N, const
double alpha, const double *X, const int incX, const double *Y, const int incY,
double *A, const int lda);
```

[ipps?gerc](#)

```
void cblas_cgerc(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

```
void cblas_zgerc(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

[ipps?geru](#)

```
void cblas_cgeru(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

```
void cblas_zgeru(const enum CBLAS_ORDER order, const int M, const int N, const
void *alpha, const void *X, const int incX, const void *Y, const int incY, void
*A, const int lda);
```

[ipps?hbmV](#)

```
void cblas_chbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const void *alpha, const void *A, const int lda, const void
*X, const int incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zhbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const void *alpha, const void *A, const int lda, const void
*X, const int incX, const void *beta, void *Y, const int incY);
```

[ipps?hemv](#)

```
void cblas_chemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *A, const int lda, const void *X, const int
incX, const void *beta, void *Y, const int incY);
```

```
void cblas_zhemv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *A, const int lda, const void *X, const int
incX, const void *beta, void *Y, const int incY);
```

[ipps?her](#)

```
void cblas_cher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const void *X, const int incX, void *A, const int lda);
```

```
void cblas_zher(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const void *X, const int incX, void *A, const int lda);
```

[ipps?her2](#)

```
void cblas_cher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *A, const int lda);
```

```
void cblas_zher2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *A, const int lda);
```

[ipps?hpmv](#)

```
void cblas_chpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *Ap, const void *X, const int incX, const
void *beta, void *Y, const int incY);
```

```
void cblas_zhpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *Ap, const void *X, const int incX, const
void *beta, void *Y, const int incY);
```

[ipps?hpr](#)

```
void cblas_chpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const void *X, const int incX, void *A);
```

```
void cblas_zhpr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const void *X, const int incX, void *A);
```

[ipps?hpr2](#)

```
void cblas_chpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *Ap);
```

```
void cblas_zhpr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const void *alpha, const void *X, const int incX, const void *Y, const int
incY, void *Ap);
```

[ipps?sbmv](#)

```
void cblas_ssbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const float alpha, const float *A, const int lda, const float
*X, const int incX, const float beta, float *Y, const int incY);
void cblas_dsbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const int K, const double alpha, const double *A, const int lda, const
double *X, const int incX, const double beta, double *Y, const int incY);
```

[ipps?spmv](#)

```
void cblas_sspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *Ap, const float *X, const int incX, const
float beta, float *Y, const int incY);
void cblas_dspmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *Ap, const double *X, const int incX,
const double beta, double *Y, const int incY);
```

[ipps?spr](#)

```
void cblas_sspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, float *Ap);
void cblas_dspr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, double *Ap);
```

[ipps?spr2](#)

```
void cblas_sspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, const float *Y, const
int incY, float *A);
void cblas_dspr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, const double *Y, const
int incY, double *A);
```

[ipps?symv](#)

```
void cblas_ssymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *A, const int lda, const float *X, const int
incX, const float beta, float *Y, const int incY);
void cblas_dsymv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *A, const int lda, const double *X, const
int incX, const double beta, double *Y, const int incY);
```

[ipps?syr](#)

```
void cblas_ssyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, float *A, const int
lda);
void cblas_dsyr(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, double *A, const int
lda);
```

[ipps?syr2](#)

```
void cblas_ssyr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const float alpha, const float *X, const int incX, const float *Y, const
int incY, float *A, const int lda);
```

```
void cblas_dsyrr2(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
int N, const double alpha, const double *X, const int incX, const double *Y, const
int incY, double *A, const int lda);
```

[ipps?tbmv](#)

```
void cblas_stbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const float *A, const int lda, float *X, const int incX);
```

```
void cblas_dtbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const double *A, const int lda, double *X, const int incX);
```

```
void cblas_ctbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

```
void cblas_ztbmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

[ipps?tbsv](#)

```
void cblas_stbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const float *A, const int lda, float *X, const int incX);
```

```
void cblas_dtbbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const double *A, const int lda, double *X, const int incX);
```

```
void cblas_ctbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

```
void cblas_ztbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const int
K, const void *A, const int lda, void *X, const int incX);
```

[ipps?tpmv](#)

```
void cblas_stpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*Ap, float *X, const int incX);
```

```
void cblas_dtpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const
double *Ap, double *X, const int incX);
```

```
void cblas_ctpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

```
void cblas_ztpmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

[ipps?tpsv](#)

```
void cblas_stpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*Ap, float *X, const int incX);
```

```
void cblas_dtpsbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const double
*Ap, double *X, const int incX);
```

```
void cblas_ctpsbsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

```
void cblas_ztpsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*Ap, void *X, const int incX);
```

[ipps?trmv](#)

```
void cblas_strmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*A, const int lda, float *X, const int incX);
```

```
void cblas_dtrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const double
*A, const int lda, double *X, const int incX);
```

```
void cblas_ctrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*A, const int lda, void *X, const int incX);
```

```
void cblas_ztrmv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*A, const int lda, void *X, const int incX);
```

[ipps?trsv](#)

```
void cblas_strsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const float
*A, const int lda, float *X, const int incX);
```

```
void cblas_dtrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const double
*A, const int lda, double *X, const int incX);
```

```
void cblas_ctrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*A, const int lda, void *X, const int incX);
```

```
void cblas_ztrsv(const enum CBLAS_ORDER order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG Diag, const int N, const void
*A, const int lda, void *X, const int incX);
```

レベル 3 の CBLAS

基本的な行列 - 行列演算を実行する「[BLAS レベル 3 のルーチン](#)」へのインターフェイスである。このグループに属する各 C ルーチンは、型 CBLAS_ORDER の追加パラメーター (最初の引数) をとる。このパラメーターは、2 次元の配列が、列主体か行主体のどちらの格納方式を使用するかを決定する。

[ipps?gemm](#)

```
void cblas_sgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
float alpha, const float *A, const int lda, const float *B, const int ldb, const
float beta, float *C, const int ldc);
```

```
void cblas_dgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
double alpha, const double *A, const int lda, const double *B, const int ldb,
const double beta, double *C, const int ldc);
```

```
void cblas_cgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
void *alpha, const void *A, const int lda, const void *B, const int ldb, const
void *beta, void *C, const int ldc);
```

```
void cblas_zgemm(const enum CBLAS_ORDER Order, const enum CBLAS_TRANSPOSE TransA,
const enum CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const
void *alpha, const void *A, const int lda, const void *B, const int ldb, const
void *beta, void *C, const int ldc);
```

[ipps?hemm](#)

```
void cblas_chemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

```
void cblas_zhemm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

[ipps?herk](#)

```
void cblas_cherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
void *A, const int lda, const float beta, void *C, const int ldc);
```

```
void cblas_zherk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
void *A, const int lda, const double beta, void *C, const int ldc);
```

[ipps?her2k](#)

```
void cblas_cher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const float beta, void *C,
const int ldc);
```

```
void cblas_zher2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const double beta, void *C,
const int ldc);
```

[ipps?symm](#)

```
void cblas_ssymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const float alpha, const float *A,
const int lda, const float *B, const int ldb, const float beta, float *C, const
int ldc);
```

```
void cblas_dsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const double alpha, const double
*A, const int lda, const double *B, const int ldb, const double beta, double *C,
const int ldc);
```

```
void cblas_csymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

```
void cblas_zsymm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const int M, const int N, const void *alpha, const void *A,
const int lda, const void *B, const int ldb, const void *beta, void *C, const int
ldc);
```

[ipps?syrk](#)

```
void cblas_ssyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
float *A, const int lda, const float beta, float *C, const int ldc);
```

```
void cblas_dsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
double *A, const int lda, const double beta, double *C, const int ldc);
```

```
void cblas_csyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *beta, void *C, const int ldc);
```

```
void cblas_zsyrk(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *beta, void *C, const int ldc);
```

[ipps?syr2k](#)

```
void cblas_ssyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const float alpha, const
float *A, const int lda, const float *B, const int ldb, const float beta, float
*C, const int ldc);
```

```
void cblas_dsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const double alpha, const
double *A, const int lda, const double *B, const int ldb, const double beta,
double *C, const int ldc);
```

```
void cblas_csyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const void *beta, void *C,
const int ldc);
```

```
void cblas_zsyr2k(const enum CBLAS_ORDER Order, const enum CBLAS_UPLO Uplo, const
enum CBLAS_TRANSPOSE Trans, const int N, const int K, const void *alpha, const
void *A, const int lda, const void *B, const int ldb, const void *beta, void *C,
const int ldc);
```

[ipps?trmm](#)

```
void cblas_strmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const float alpha, const float *A, const int lda,
float *B, const int ldb);
```

```
void cblas_dtrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const double alpha, const double *A, const int
lda, double *B, const int ldb);
```

```
void cblas_ctrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

```
void cblas_ztrmm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

[ipps?trsm](#)

```
void cblas_strsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const float alpha, const float *A, const int lda,
float *B, const int ldb);
```

```
void cblas_dtrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const double alpha, const double *A, const int
lda, double *B, const int ldb);
```

```
void cblas_ctrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

```
void cblas_ztrsm(const enum CBLAS_ORDER Order, const enum CBLAS_SIDE Side, const
enum CBLAS_UPLO Uplo, const enum CBLAS_TRANSPOSE TransA, const enum CBLAS_DIAG
Diag, const int M, const int N, const void *alpha, const void *A, const int lda,
void *B, const int ldb);
```

スパース CBLAS

「[スパース BLAS レベル 1 のルーチンと関数](#)」に対するインターフェイスであり、圧縮形式で格納されたスパースベクトルに共通的なベクトル演算を実行する。

すべてのインデックス・パラメーター *indx* は C タイプで表記され [0..*N*-1] の範囲を取る。

[ipps?axpyi](#)

```
void cblas_saxpyi(const int N, const float alpha,
const float *X, const int *indx, float *Y);
void cblas_daxpyi(const int N, const double alpha,
const double *X, const int *indx, double *Y);
void cblas_caxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
void cblas_zaxpyi(const int N, const void *alpha,
const void *X, const int *indx, void *Y);
```

[ipps?doti](#)

```
float cblas_sdoti(const int N, const float *X,
const int *indx, const float *Y);
double cblas_ddoti(const int N, const double *X,
const int *indx, const double *Y);
```

[ipps?dotci](#)

```
void cblas_cdotci_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
void cblas_zdotci_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
```

[ipps?dotui](#)

```
void cblas_cdotui_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
void cblas_zdotui_sub(const int N, const void *X, const int *indx, const void
*Y, void *dotui);
```

[ipps?gthr](#)

```
void cblas_sgthr(const int N, const float *Y, float *X,
const int *indx);
void cblas_dgthr(const int N, const double *Y, double *X,
const int *indx);
void cblas_cgthr(const int N, const void *Y, void *X,
const int *indx);
void cblas_zgthr(const int N, const void *Y, void *X,
const int *indx);
```

[ipps?gthrz](#)

```
void cblas_sgthrz(const int N, float *Y, float *X,
const int *indx);
void cblas_dgthrz(const int N, double *Y, double *X,
const int *indx);
void cblas_cgthrz(const int N, void *Y, void *X,
const int *indx);
```

```
void cblas_zgthrz(const int N, void *Y, void *X,  
const int *indx);
```

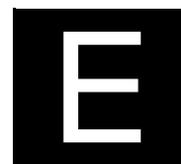
[ipps?roti](#)

```
void cblas_sroti(const int N, float *X, const int *indx,  
float *Y, const float c, const float s);  
void cblas_droti(const int N, double *X, const int *indx,  
double *Y, const double c, const double s);
```

[ipps?sctr](#)

```
void cblas_ssctr(const int N, const float *X, const int *indx, float *Y);  
void cblas_dsctr(const int N, const double *X, const int *indx, double *Y);  
void cblas_csctr(const int N, const void *X, const int *indx, void *Y);  
void cblas_zsctr(const int N, const void *X, const int *indx, void *Y);
```

LAPACK ルーチン用 Fortran-95 インターフェイス 特有の機能



インテル® MKL は、MKL Fortran-77 LAPACK ルーチンのすべての機能を提供する、LAPACK パッケージ用の Fortran-95 インターフェイス (MKL LAPACK-95) を実装している。これは LAPACK 用の Netlib Fortran-95 実装との主要な相違である。

インテル MKL LAPACK-77 実装と比較した MKL LAPACK-95 の新機能は、ラッパーとともにソース・インターフェイスのパッケージが含まれ、コンパイラに依存しないように実装されていることである。その結果、MKL LAPACK パッケージは、Fortran-95 に対応しているすべてのプログラム環境で使用することができる。

Netlib 実装との相違の程度および種類によって、MKL LAPACK-95 インターフェイスは異なる変換が必要なくつかのグループに分けられる ([「MKL LAPACK ルーチン用 Fortran-95 インターフェイスと Netlib 実装」](#)を参照)。各グループでは、ルーチンの呼び出しシーケンスと、Netlib アナログとの相違を示す。

以下の規則が使用されている。

```
<interface> ::= <name of interface> '(' <arguments list> ')'  
<arguments list> ::= <first argument> {<argument>}*  
<first argument> ::= < identifier >  
<argument> ::= <required argument>|<optional argument>  
<required argument> ::= \,' <identifier>  
<optional argument> ::= \,[' <identifier> \]'  
<name of interface> ::= <identifier>
```

ここで定義されている概念は ::= によって定義と区切られている。概念名は鍵括弧で囲まれ、終点は引用符で示される。{...}* はゼロの繰り返しを示す。

<first argument> および各 <required argument> は、すべての呼び出しに含まれる。<optional argument> は省略される場合がある。必要な部分では、インターフェイス定義へのコメントが記述されている。コメント行は、! 文字で始まる。

(引数の種類によって分けられた) 2つのサブルーチン呼び出しがある場合、1つの名前と2つのインターフェイスが示される。

Netlib と同一のインターフェイス

```

GETRI(A, IPIV [, INFO])
GEEQU(A,R,C[,ROWCND][,COLCND][,AMAX][,INFO])
GESV(A,B[,IPIV][,INFO])
GESVX(A,B,X[,AF][,IPIV][,FACT][,TRANS][,EQUED][,R][,C][,FERR][,BERR]
[,RCOND][,RPVGRW][,INFO])
GBSV(A,B[,KL][,IPIV][,INFO])
GTSV(DL,D,DU,B[,INFO])
GTSVX(DL,D,DU,B,X[,DLF][,DF][,DUF][,DU2][,IPIV][,FACT][,TRANS][,FERR]
[,BERR][,RCOND][,INFO])
POSV(A,B[,UPLO][,INFO])
POSVX(A,B,X[,UPLO][,AF][,FACT][,EQUED][,S][,FERR][,BERR][,RCOND][,INFO])
PPSV(A,B[,UPLO][,INFO])
PPSVX(A,B,X[,UPLO][,AF][,FACT][,EQUED][,S][,FERR][,BERR][,RCOND][,INFO])
PBSV(A,B[,UPLO][,INFO])
PBSVX(A,B,X[,UPLO][,AF][,FACT][,EQUED][,S][,FERR][,BERR][,RCOND][,INFO])
PTSV(D,E,B[,INFO])
PTSVX(D,E,B,X[,DF][,EF][,FACT][,FERR][,BERR][,RCOND][,INFO])
SYSV(A,B[,UPLO][,IPIV][,INFO])
SYSVX(A,B,X[,UPLO][,AF][,IPIV][,FACT][,FERR][,BERR][,RCOND][,INFO])
HESVX(A,B,X[,UPLO][,AF][,IPIV][,FACT][,FERR][,BERR][,RCOND][,INFO])
SYTRD(A,TAU[,UPLO][,INFO])
ORGTR(A,TAU[,UPLO][,INFO])
HETRD(A,TAU[,UPLO][,INFO])
UNGTR(A,TAU[,UPLO][,INFO])
SYGST(A,B[,ITYPE][,UPLO][,INFO])
HEGST(A,B[,ITYPE][,UPLO][,INFO])
GELS(A,B[,TRANS][,INFO])
GELSY(A,B[,RANK][,JPVT][,RCOND][,INFO])
GELSS(A,B[,RANK][,S][,RCOND][,INFO])
GELSD(A,B[,RANK][,S][,RCOND][,INFO])
GGLSE(A,B,C,D,X[,INFO])
GGGLM(A,B,D,X,Y[,INFO])
SYEV(A,W[,JOBZ][,UPLO][,INFO])
HEEV(A,W[,JOBZ][,UPLO][,INFO])
SYEVD(A,W[,JOBZ][,UPLO][,INFO])
SPEV(A,W[,UPLO][,Z][,INFO])
HPEV(A,W[,UPLO][,Z][,INFO])
SPEVD(A,W[,UPLO][,Z][,INFO])
HPEVD(A,W[,UPLO][,Z][,INFO])
SPEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])

```

```
HPEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
SBEV(A,W[,UPLO][,Z][,INFO])
HBEV(A,W[,UPLO][,Z][,INFO])
SBEVD(A,W[,UPLO][,Z][,INFO])
HBEVD(A,W[,UPLO][,Z][,INFO])
SBEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,Q][,ABSTOL]
[,INFO])
HBEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,Q][,ABSTOL]
[,INFO])
HPGV(A,B,W[,ITYPE][,UPLO][,Z][,INFO])
STEV(D,E[,Z][,INFO])
STEVD(D,E[,Z][,INFO])
STEVX(D,E,W[,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
STEVX(D,E,W[,Z][,VL][,VU][,IL][,IU][,M][,ISUPPZ][,ABSTOL][,INFO])
GEES(A,WR,WI[,VS][,SELECT][,SDIM][,INFO])
GEES(A,W[,VS][,SELECT][,SDIM][,INFO])
GEESX(A,WR,WI[,VS][,SELECT][,SDIM][,RCONDE][,RCONDV][,INFO])
GEESX(A,W[,VS][,SELECT][,SDIM][,RCONDE][,RCONDV][,INFO])
GEEV(A,WR,WI[,VL][,VR][,INFO])
GEEV(A,W[,VL][,VR][,INFO])
GEEVX(A,WR,WI[,VL][,VR][,BALANC][,ILO][,IHI][,SCALE][,ABNRM][,RCONDE]
[,RCONDV][,INFO])
GEEVX(A,W[,VL][,VR][,BALANC][,ILO][,IHI][,SCALE][,ABNRM][,RCONDE]
[,RCONDV][,INFO])
GESVD(A,S[,U][,VT][,WW][,JOB][,INFO])
GGSVD(A,B,ALPHA,BETA[,K][,L][,U][,V][,Q][,IWORK][,INFO])
SYGV(A,B,W[,ITYPE][,JOBZ][,UPLO][,INFO])
HEGV(A,B,W[,ITYPE][,JOBZ][,UPLO][,INFO])
SYGVD(A,B,W[,ITYPE][,JOBZ][,UPLO][,INFO])
HEGVD(A,B,W[,ITYPE][,JOBZ][,UPLO][,INFO])
SPGV(A,B,W[,ITYPE][,UPLO][,Z][,INFO])
SBGV(A,B,W[,UPLO][,Z][,INFO])
HBGV(A,B,W[,UPLO][,Z][,INFO])
GGES(A,B,ALPHAR,ALPHAI,BETA[,VSL][,VSR][,SELECT][,SDIM][,INFO])
GGES(A,B,ALPHA,BETA[,VSL][,VSR][,SELECT][,SDIM][,INFO])
GGESX(A,B,ALPHAR,ALPHAI,BETA[,VSL][,VSR][,SELECT][,SDIM][,RCONDE]
[,RCONDV][,INFO])
GGESX(A,B,ALPHA,BETA[,VSL][,VSR][,SELECT][,SDIM][,RCONDE][,RCONDV]
[,INFO])
GGEV(A,B,ALPHAR,ALPHAI,BETA[,VL][,VR][,INFO])
GGEV(A,B,ALPHA,BETA[,VL][,VR][,INFO])
```

```

GGEVX(A,B,ALPHAR,ALPHAI,BETA[,VL][,VR][,BALANC][,ILO][,IHI][,LSCALE]
[,RSCALE][,ABNRM][,BBNRM][,RCONDE][,RCONDV][,INFO])
GGEVX(A,B,ALPHA,BETA[,VL][,VR][,BALANC][,ILO][,IHI][,LSCALE][,RSCALE]
[,ABNRM][,BBNRM][,RCONDE][,RCONDV][,INFO])
GERFS(A,AF,IPIV,B,X[,TRANS][,FERR][,BERR][,INFO])

```

引数名が置換されるインターフェイス

このグループのルーチンでの引数名は、以下のように置換される。

Netlib 引数名	MKL 引数名
AP	A
AB	A
AFP	AF
BP	B
BB	B

```

SPSV(A,B[,UPLO][,IPIV][,INFO])
! netlib: (AP,B,UPLO,IPIV,INFO)
SPSVX(A,B,X[,UPLO][,AF][,IPIV][,FACT][,FERR][,BERR][,RCOND][,INFO])
! netlib: (A,B,X,UPLO,AFP,IPIV,FACT,FERR,BERR,RCOND,INFO)
HPSVX(A,B,X[,UPLO][,AF][,IPIV][,FACT][,FERR][,BERR][,RCOND][,INFO])
! netlib: (A,B,X,UPLO,AFP,IPIV,FACT,FERR,BERR,RCOND,INFO)
HEEVD(A,W[,JOB][,UPLO][,INFO])
! netlib: (A,W,JOBZ,UPLO,INFO)
SPGVD(A,B,W[,ITYPE][,UPLO][,Z][,INFO])
! netlib: (AP,BP,W,ITYPE,UPLO,Z,INFO)
HPGVD(A,B,W[,ITYPE][,UPLO][,Z][,INFO])
! netlib: (AP,BP,W,ITYPE,UPLO,Z,INFO)
SPGVX(A,B,W[,ITYPE][,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL]
[,INFO])
! netlib: (AP,BP,W,ITYPE,UPLO,Z,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
HPGVX(A,B,W[,ITYPE][,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL]
[,INFO])
! netlib: (AP,BP,W,ITYPE,UPLO,Z,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
SBGVD(A,B,W[,UPLO][,Z][,INFO])
! netlib: (AB,BB,W,UPLO,Z,INFO)
HBGVD(A,B,W[,UPLO][,Z][,INFO])
! netlib: (AB,BB,W,UPLO,Z,INFO)
SBGVX(A,B,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,Q][,ABSTOL]
[,INFO])
! netlib: (AB,BB,W,UPLO,Z,VL,VU,IL,IU,M,IFAIL,Q,ABSTOL,INFO)
HBGVX(A,B,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,Q][,ABSTOL]
[,INFO])
! netlib: (AB,BB,W,UPLO,Z,VL,VU,IL,IU,M,IFAIL,Q,ABSTOL,INFO)

```

```

GBSVX(A,B,X[,KL][,AF][,IPIV][,FACT][,TRANS][,EQUED][,R][,C][,FERR]
[,BERR][,RCOND][,RPVGRW][,INFO])
! netlib:
!(A,B,X,KL,AFB,IPIV,FACT,TRANS,EQUED,R,C,FERR,BERR,RCOND,RPVGRW,INFO)

```

修正された Netlib インターフェイス

```

SYEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,W,JOBZ,UPLO,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 4, mkl: 3
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

HEEVX(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,W,JOBZ,UPLO,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 4, mkl: 3
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

SYEVR(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,ISUPPZ][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,W,JOBZ,UPLO,VL,VU,IL,IU,M,ISUPPZ,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 4, mkl: 3
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

HEEVR(A,W[,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,ISUPPZ][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,W,JOBZ,UPLO,VL,VU,IL,IU,M,ISUPPZ,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 4, mkl: 3
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

GESDD(A,S[,U][,VT][,JOBZ][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,S,U,VT,WW,JOB,INFO)
! Different number of parameters, netlib: 7, mkl: 6
! Absent mkl parameter: WW
! Absent mkl parameter: JOB
! Different order for parameter INFO, netlib: 7, mkl: 6
! Extra mkl parameter: JOBZ

SYGVX(A,B,W[,ITYPE][,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,B,W,ITYPE,JOBZ,UPLO,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 6, mkl: 5
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

HEGVX(A,B,W[,ITYPE][,UPLO][,Z][,VL][,VU][,IL][,IU][,M][,IFAIL][,ABSTOL][,INFO])
! Interface netlib95 exists, parameters:
! netlib: (A,B,W,ITYPE,JOBZ,UPLO,VL,VU,IL,IU,M,IFAIL,ABSTOL,INFO)
! Different order for parameter UPLO, netlib: 6, mkl: 5
! Absent mkl parameter: JOBZ
! Extra mkl parameter: Z

```

```

GETRS(A,IPIV,B[,TRANS][,INFO])
!   Interface netlib95 exists:
!   Different intents for parameter A, netlib: INOUT, mkl: IN

```

Netlib にないインターフェイス

```

GTTRF(DL,D,DU,DU2[,IPIV][,INFO])
PPTRF(A[,UPLO][,INFO])
PBTRF(A[,UPLO][,INFO])
PTTRF(D,E[,INFO])
SYTRF(A[,UPLO][,IPIV][,INFO])
HETRF(A[,UPLO][,IPIV][,INFO])
SPTRF(A[,UPLO][,IPIV][,INFO])
HPTRF(A[,UPLO][,IPIV][,INFO])
GBTRS(A,B,IPIV[,KL][,TRANS][,INFO])
GTTRS(DL,D,DU,DU2,B,IPIV[,TRANS][,INFO])
POTRS(A,B[,UPLO][,INFO])
PPTRS(A,B[,UPLO][,INFO])
PBTRS(A,B[,UPLO][,INFO])
PTTRS(D,E,B[,INFO])
PTTRS(D,E,B[,UPLO][,INFO])
SYTRS(A,B,IPIV[,UPLO][,INFO])
HETRS(A,B,IPIV[,UPLO][,INFO])
SPTRS(A,B,IPIV[,UPLO][,INFO])
HPTRS(A,B,IPIV[,UPLO][,INFO])
TRTRS(A,B[,UPLO][,TRANS][,DIAG][,INFO])
TPTRS(A,B[,UPLO][,TRANS][,DIAG][,INFO])
TBTRS(A,B[,UPLO][,TRANS][,DIAG][,INFO])
GECON(A,ANORM,RCOND[,NORM][,INFO])
GBCON(A,IPIV,ANORM,RCOND[,KL][,NORM][,INFO])
GTCON(DL,D,DU,DU2,IPIV,ANORM,RCOND[,NORM][,INFO])
POCON(A,ANORM,RCOND[,UPLO][,INFO])
PPCON(A,ANORM,RCOND[,UPLO][,INFO])
PBCON(A,ANORM,RCOND[,UPLO][,INFO])
PTCON(D,E,ANORM,RCOND[,INFO])
SYCON(A,IPIV,ANORM,RCOND[,UPLO][,INFO])
HECON(A,IPIV,ANORM,RCOND[,UPLO][,INFO])
SPCON(A,IPIV,ANORM,RCOND[,UPLO][,INFO])
HPCON(A,IPIV,ANORM,RCOND[,UPLO][,INFO])
TRCON(A,RCOND[,UPLO][,DIAG][,NORM][,INFO])
TPCON(A,RCOND[,UPLO][,DIAG][,NORM][,INFO])
TBCON(A,RCOND[,UPLO][,DIAG][,NORM][,INFO])
GBRFS(A,AF,IPIV,B,X[,KL][,TRANS][,FERR][,BERR][,INFO])

```

```
GTRFS(DL,D,DU,DLF,DF,DUF,DU2,IPIV,B,X[,TRANS][,FERR][,BERR][,INFO])
PORFS(A,AF,B,X[,UPLO][,FERR][,BERR][,INFO])
PPRFS(A,AF,B,X[,UPLO][,FERR][,BERR][,INFO])
PBRFS(A,AF,B,X[,UPLO][,FERR][,BERR][,INFO])
PTRFS(D,DF,E,EF,B,X[,FERR][,BERR][,INFO])
PTRFS(D,DF,E,EF,B,X[,UPLO][,FERR][,BERR][,INFO])
SYRFS(A,AF,IPIV,B,X[,UPLO][,FERR][,BERR][,INFO])
HERFS(A,AF,IPIV,B,X[,UPLO][,FERR][,BERR][,INFO])
SPRFS(A,AF,IPIV,B,X[,UPLO][,FERR][,BERR][,INFO])
HPRFS(A,AF,IPIV,B,X[,UPLO][,FERR][,BERR][,INFO])
TRRFS(A,B,X[,UPLO][,TRANS][,DIAG][,FERR][,BERR][,INFO])
TPRFS(A,B,X[,UPLO][,TRANS][,DIAG][,FERR][,BERR][,INFO])
TBRFS(A,B,X[,UPLO][,TRANS][,DIAG][,FERR][,BERR][,INFO])
POTRI(A[,UPLO][,INFO])
PPTRI(A[,UPLO][,INFO])
SYTRI(A,IPIV[,UPLO][,INFO])
HETRI(A,IPIV[,UPLO][,INFO])
SPTRI(A,IPIV[,UPLO][,INFO])
HPTRI(A,IPIV[,UPLO][,INFO])
TRTRI(A[,UPLO][,DIAG][,INFO])
TPTRI(A[,UPLO][,DIAG][,INFO])
GBEQU(A,R,C[,KL][,ROWCND][,COLCND][,AMAX][,INFO])
POEQU(A,S[,SCOND][,AMAX][,INFO])
PPEQU(A,S[,SCOND][,AMAX][,UPLO][,INFO])
PBEQU(A,S[,SCOND][,AMAX][,UPLO][,INFO])
HESV(A,B[,UPLO][,IPIV][,INFO])
HPSV(A,B[,UPLO][,IPIV][,INFO])
GEQRF(A[,TAU][,INFO])
GEQPF(A,JPVT[,TAU][,INFO])
GEQP3(A,JPVT[,TAU][,INFO])
ORGQR(A,TAU[,INFO])
ORMQR(A,TAU,C[,SIDE][,TRANS][,INFO])
UNGQR(A,TAU[,INFO])
UNMQR(A,TAU,C[,SIDE][,TRANS][,INFO])
GELQF(A[,TAU][,INFO])
ORGLQ(A,TAU[,INFO])
ORMLQ(A,TAU,C[,SIDE][,TRANS][,INFO])
UNGLQ(A,TAU[,INFO])
UNMLQ(A,TAU,C[,SIDE][,TRANS][,INFO])
GEQLF(A[,TAU][,INFO])
ORGQL(A,TAU[,INFO])
```

```

UNGQL(A,TAU[,INFO])
ORMQL(A,TAU,C[,SIDE][,TRANS][,INFO])
UNMQL(A,TAU,C[,SIDE][,TRANS][,INFO])
GERQF(A[,TAU][,INFO])
ORGRQ(A,TAU[,INFO])
UNGRQ(A,TAU[,INFO])
ORMRQ(A,TAU,C[,SIDE][,TRANS][,INFO])
UNMRQ(A,TAU,C[,SIDE][,TRANS][,INFO])
TZRZF(A[,TAU][,INFO])
ORMRZ(A,TAU,C,L[,SIDE][,TRANS][,INFO])
UNMRZ(A,TAU,C,L[,SIDE][,TRANS][,INFO])
GGQRF(A,B[,TAUA][,TAUB][,INFO])
GGRQF(A,B[,TAUA][,TAUB][,INFO])
GEBRD(A[,D][,E][,TAUQ][,TAUP][,INFO])
GBBRD(A[,C][,D][,E][,Q][,PT][,KL][,M][,INFO])
ORGBR(A,TAU[,VECT][,INFO])
ORMBR(A,TAU,C[,VECT][,SIDE][,TRANS][,INFO])
ORMTR(A,TAU,C[,SIDE][,UPLO][,TRANS][,INFO])
UNGBR(A,TAU[,VECT][,INFO])
UNMBR(A,TAU,C[,VECT][,SIDE][,TRANS][,INFO])
BDSQR(D,E[,VT][,U][,C][,UPLO][,INFO])
BDSDC(D,E[,U][,VT][,Q][,IQ][,UPLO][,INFO])
UNMTR(A,TAU,C[,SIDE][,UPLO][,TRANS][,INFO])
SPTRD(A,TAU[,UPLO][,INFO])
OPGTR(A,TAU,Q[,UPLO][,INFO])
OPMTR(A,TAU,C[,SIDE][,UPLO][,TRANS][,INFO])
HPTRD(A,TAU[,UPLO][,INFO])
UPGTR(A,TAU,Q[,UPLO][,INFO])
UPMTR(A,TAU,C[,SIDE][,UPLO][,TRANS][,INFO])
SBTRD(A[,Q][,VECT][,UPLO][,INFO])
HBTRD(A[,Q][,VECT][,UPLO][,INFO])
STERF(D,E[,INFO])
STEQR(D,E[,Z][,COMPZ][,INFO])
STEDC(D,E[,Z][,COMPZ][,INFO])
STEGR(D,E,W[,Z][,VL][,VU][,IL][,IU][,M][,ISUPPZ][,ABSTOL][,INFO])
PTEQR(D,E[,Z][,COMPZ][,INFO])
STEBZ(D,E,M,NSPLIT,W,IBLOCK,ISPLIT[,ORDER][,VL][,VU][,IL][,IU][,ABSTOL][,INFO])
STEIN(D,E,W,IBLOCK,ISPLIT,Z[,IFAILV][,INFO])
DISNA(D,SEP[,JOB][,MINMN][,INFO])
SPGST(A,B[,ITYPE][,UPLO][,INFO])

```

```
HPGST(A,B[,ITYPE][,UPLO][,INFO])
SBGST(A,B[,X][,UPLO][,INFO])
HBGST(A,B[,X][,UPLO][,INFO])
PBSTF(B[,UPLO][,INFO])
GEHRD(A[,TAU][,ILO][,IHI][,INFO])
ORGHR(A,TAU[,ILO][,IHI][,INFO])
ORMHR(A,TAU,C[,ILO][,IHI][,SIDE][,TRANS][,INFO])
UNGHR(A,TAU[,ILO][,IHI][,INFO])
UNMHR(A,TAU,C[,ILO][,IHI][,SIDE][,TRANS][,INFO])
GEBAL(A[,SCALE][,ILO][,IHI][,JOB][,INFO])
GEBAK(V,SCALE[,ILO][,IHI][,JOB][,SIDE][,INFO])
HSEQR(H,WR,WI[,ILO][,IHI][,Z][,JOB][,COMPZ][,INFO])
HSEQR(H,W[,ILO][,IHI][,Z][,JOB][,COMPZ][,INFO])
HSEIN(H,WR,WI,SELECT[,VL][,VR][,IFAILL][,IFAILR][,INITV][,EIGSRC][,M][,INFO])
HSEIN(H,W,SELECT[,VL][,VR][,IFAILL][,IFAILR][,INITV][,EIGSRC][,M][,INFO])
TREVC(T[,HOWMNY][,SELECT][,VL][,VR][,M][,INFO])
TRSNA(T[,S][,SEP][,VL][,VR][,SELECT][,M][,INFO])
TREXC(T,IFST,ILST[,Q][,INFO])
TRSEN(T,SELECT[,WR][,WI][,M][,S][,SEP][,Q][,INFO])
TRSEN(T,SELECT[,W][,M][,S][,SEP][,Q][,INFO])
TRSYL(A,B,C,SCALE[,TRANA][,TRANB][,ISGN][,INFO])
GGHRD(A,B[,ILO][,IHI][,Q][,Z][,COMPQ][,COMPZ][,INFO])
GGBAL(A,B[,ILO][,IHI][,LSCALE][,RSCALE][,JOB][,INFO])
GGBAK(V[,ILO][,IHI][,LSCALE][,RSCALE][,JOB][,INFO])
HGEQZ(H,T[,ILO][,IHI][,ALPHAR][,ALPHAI][,BETA][,Q][,Z][,JOB][,COMPQ][,COMPZ][,INFO])
HGEQZ(H,T[,ILO][,IHI][,ALPHA][,BETA][,Q][,Z][,JOB][,COMPQ][,COMPZ][,INFO])
TGEVC(S,P[,HOWMNY][,SELECT][,VL][,VR][,M][,INFO])
TGEXC(A,B[,IFST][,ILST][,Z][,Q][,INFO])
TGSEN(A,B,SELECT[,ALPHAR][,ALPHAI][,BETA][,IJOB][,Q][,Z][,PL][,PR][,DIF][,M][,INFO])
TGSEN(A,B,SELECT[,ALPHA][,BETA][,IJOB][,Q][,Z][,PL][,PR][,DIF][,M][,INFO])
TGSYL(A,B,C,D,E,F[,IJOB][,TRANS][,SCALE][,DIF][,INFO])
TGSNA(A,B[,S][,DIF][,VL][,VR][,SELECT][,M][,INFO])
GGSVP(A,B,TOLA,TOLB[,K][,L][,U][,V][,Q][,INFO])
TGSJA(A,B,TOLA,TOLB,K,L[,U][,V][,Q][,JOBV][,JOBQ][,ALPHA][,BETA][,NCYCLE][,INFO])
```

新機能のインターフェイス

```
GETRF(A[,IPIV][,INFO])
!   Interface netlib95 exists, parameters:
!   netlib: (A,IPIV,RCOND,NORM,INFO)
!   Different number of parameters, netlib: 5, mkl: 3
!   Different order for parameter INFO, netlib: 5, mkl: 3
!   Absent mkl parameter: NORM
!   Absent mkl parameter: RCOND

GBTRF(A[,KL][,M][,IPIV][,INFO])
!   Interface netlib95 exists, parameters:
!   netlib: (A,K,M,IPIV,RCOND,NORM,INFO)
!   Different number of parameters, netlib: 7, mkl: 5
!   Different order for parameter INFO, netlib: 7, mkl: 5
!   Absent mkl parameter: NORM
!   Replace parameter name: netlib: K: mkl: KL
!   Absent mkl parameter: RCOND

POTRF(A[,UPLO][,INFO])
!   Interface netlib95 exists, parameters:
!   netlib: (A,UPLO,RCOND,NORM,INFO)
!   Different number of parameters, netlib: 5, mkl: 3
!   Different order for parameter INFO, netlib: 5, mkl: 3
!   Absent mkl parameter: NORM
!   Absent mkl parameter: RCOND
```

用語集

A^H	一般行列 A の共役を表す。 共役行列も参照。
A^T	一般行列 A の転置を表す。 転置も参照。
BLAS	Basic Linear Algebra (基本線形代数) の略。これらのサブプログラムには、ベクトル演算、行列 - ベクトル演算、行列 - 行列演算が導入されている。
BRNG	Basic Random Number Generator (基本乱数生成器) の略。基本乱数生成器は、一様分布の i.i.d. 乱数列を模倣した擬似乱数生成器である。一様分布以外の分布は、一様分布の乱数列に異なる変換手法を用いて生成される。
BRNG 登録	ユーザ一定義の BRNG を VSL に含め、定義済み VSL 基本生成器とともに使用することのできる標準化されたメカニズム。
Bunch-Kaufman 因子分解	実数称行列あるいは複素エルミート行列 A を形式 $A = PUDU^H P^T$ (または $A = PLDL^H P^T$) で表したものの。 P は置換行列、 U と L は単位対角成分を持つ上および下三角行列、 D は 1×1 と 2×2 の対角ブロックを持つエルミートブロック対角行列である。 U と L は、 D の 2×2 のブロックに対応する 2×2 の単位対角ブロックを持つ。
c	ルーチン名の最初の文字に使われている場合、c は単精度の複素数データ型の使用を示す。
CBLAS	BLAS に対する C インターフェイス。BLAS を参照。
CDF	累積分布関数。一変量または多変量のランダム変数 X における確率分布を決定する関数。一変量分布において累積分布関数は、実引数 x の関数であり、それぞれの x は事象 $A: X \leq x$ の確率と等しい値をとる。多変量分布において累積分布関数は、実ベクトル $\mathbf{x} = (x_1, x_2, \dots, x_n)$ の関数であり、それぞれの \mathbf{x} は事象 $A = (X_1 \leq x_1 \& X_2 \leq x_2, \& \dots, \& X_n \leq x_n)$ の確率と等しい値をとる。
d	ルーチン名の最初の文字に使われている場合、d は倍精度の実数データ型を使用することを示す。
DFT	Discrete Fourier Transforms (離散フーリエ変換) の略。本書の第 11 章を参照。

FFT	Fast Fourier Transforms (高速フーリエ変換) の略。本書の第 11 章を参照。
I	単位行列を参照。
i.i.d.	Independent Identically Distributed (独立同一分布) の略。
LQ 因子分解	<p>$m \times n$ の行列 A を、$A = LQ$ あるいは $A = (L \ 0) Q$ として表したものである。Q は、$n \times n$ の直交 (ユニタリー) 行列である。$m \leq n$ に対しては、L は実数の対角成分を持つ $m \times m$ の下三角行列である。$m > n$ に対しては、次のようになる。</p> $L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \quad L_1 \text{ は、} n \times n \text{ の下三角行列、}$ <p>L_2 は矩形行列である。</p>
LU 因子分解	<p>$m \times n$ の一般行列 A を、$A = PLU$ として表したものである。P は置換行列、L は単位対角成分を持つ下三角行列 ($m > n$ の場合は、下台形行列)、U は上三角行列 ($m < n$ の場合は上台形行列) である。</p>
MPI	Message Passing Interface (メッセージ・パッシング・インターフェイス) の略。並列計算処理でのさまざまなメッセージ通信機能におけるユーザー・インターフェイスと機能を標準化した規格。
MPICH	移植可能な無償の MPI の実装ライブラリー。
PDF	<p>Probability Density Function (確率密度関数) の略。一変量または多変量の連続ランダム変数 X における確率分布を決定する関数。確率密度関数 $f(x)$ は、累積分布関数 $F(x)$ と密接な関係を持つ。一変量分布における関係を次に示す。</p> $F(x) = \int_{-\infty}^x f(t) dt。$ <p>多変量分布における関係を次に示す。</p> $F(x_1, x_2, \dots, x_n) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \dots \int_{-\infty}^{x_n} f(t_1, t_2, \dots, t_n) dt_1 dt_2 \dots dt_n$
QR 因子分解	<p>$m \times n$ の行列 A を、$A = QR$ として表したものである。Q は $m \times m$ の直交 (ユニタリー) 行列、R は実数の対角成分を持つ $n \times n$ の上三角行列 ($m \geq n$ の場合) あるいは上台形行列 ($m < n$ の場合) である。</p>
RNG	Random Number Generator (乱数生成器) の略。本書における乱数生成器とは、真にランダムな数列を模倣する、完全な決定論的アルゴリズムに基づいた擬似乱数生成器を意味する。
s	ルーチン名の最初の文字に使われている場合、s は、単精度の実数データ型を使用することを示す。
ScaLAPACK	Scalable Linear Algebra PACKage の略。

Schur 因子分解	正方行列 A を $A = ZTZ^H$ の形式で表したものの。 T は、Schur 形式と呼ばれる A の上疑似三角行列 (複素数 A に対しては、三角行列) である。行列 Z は、直交行列 (複素数 A に対してはユニタリー行列) である。 Z の列は、Schur ベクトルと呼ばれる。
SMP	Symmetric MultiProcessing (対称型マルチプロセッシング) の略。MKL では、SMP によって提供される並列化によって性能を改善している。
SVD	Singular Value Decomposition (特異値分解) の略。第 5 章の特異値分解も参照。
VML	Vector Mathematical Library (ベクトル数学ライブラリー) の略。 本書の第 9 章を参照。
VSL	Vector Statistical Library (ベクトル統計ライブラリー) の略。 本書の第 10 章を参照。
z	ルーチン名の最初の文字に使われている場合、 z は倍精度の複素数データ型を使用することを示す。
圧縮格納	対称行列、エルミート行列、三角行列をよりコンパクトに格納できる格納形式。行列の上三角または下三角が、列ごとに 1 次元配列に圧縮される。
因子分解	行列を行列同士の積として表すこと。Bunch-Kaufman 因子分解、コレスキー因子分解、LU 因子分解、LQ 因子分解、QR 因子分解、Schur 因子分解も参照。
インテル MKL	Intel [®] Math Kernel Library (インテル [®] マス・カーネル・ライブラリー) の略。
インプレース	演算の修飾子。演算をインプレースで実行する関数では、その入力を配列から読み取り、その出力を同じ配列に返す。
エルミート行列	共役行列 A^H に等しい正方行列 A 。共役 A^H は、次のように定義される： $(A^H)_{ij} = (a_{ji})^*$ 。
帯格納	帯行列に対する特殊な格納形式。 行列は、2 次元配列に格納される。このとき、行列の各列は配列の対応する列に、行列の各対角成分は配列の各行に格納される。
帯行列	$ i - j > l$ に対して $a_{ij} = 0$ となる $m \times n$ の一般行列 A 。 ここで、 $1 < l < \min(m, n)$ 。例えば、三角行列は、いずれも帯行列である。
格納形式	行列を格納する方法。フル格納、圧縮格納、帯格納を参照。
擬似乱数生成器	真にランダムな数列を模倣する完全な決定論的アルゴリズム。
基本リフレクター (Householder 行列)	一般形式 $H = I - \tau v v^T$ の行列。 v は列ベクトル、 τ はスカラーである。 LAPACK では、基本リフレクターは、例えば行列 Q を QR 因子分解で表すために使用する (行列 Q は、基本リフレクターの積として表す)。

逆行列	A^{-1} のように表し、所定の正方行列 A に対して次のように定義される行列: $AA^{-1} = A^{-1}A = I$ 。 特異行列 A に対しては、 A^{-1} は存在しない。
共役行列	所定の一般行列 A に対して、次のように定義される行列 A^H : $(A^H)_{ij} = (a_{ji})^*$ 。
共役数	複素数 $z = a + bi$ の共役は、 $z^* = a - bi$ になる。
固有値	固有値問題を参照。
固有値問題	所定の正方行列 A に対して、 $Ax = \lambda x$ となるような非ゼロのベクトル x と値 λ を求める問題。値 λ は、行列 A の固有値と呼ばれ、ベクトル x は行列 A の固有ベクトルと呼ばれる。
固有ベクトル	固有値問題を参照。
コレスキー因子分解	対称 (複素データの場合はエルミート) 正定値行列 A を、形式 $A = U^H U$ か $A = LL^H$ で表したものの。 L は下三角行列、 U は上三角行列である。
三角行列	劣対角成分 (優対角成分) すべてがゼロである場合、行列 A は上 (下) 三角行列と呼ばれる。したがって、上三角行列に対しては $i > j$ の場合に $a_{ji} = 0$ に、下三角行列に対しては $i < j$ の場合に $a_{ji} = 0$ になる。
三重対角行列	非ゼロの成分が、3つの対角成分 (主対角成分、最初の劣対角成分、最初の優対角成分) だけに存在する行列。
正定値行列	非ゼロの任意のベクトル x に対して、 $Ax \cdot x > 0$ であるような正方行列 A 。は、ドット積を表す。
条件数	所定の正方行列 A に対して、次のように定義される数 $\kappa(A)$: $\kappa(A) = \ A\ \ A^{-1}\ $ 。
スパース BLAS	スパースベクトルに対して基本的なベクトル演算を実行するルーチン。スパース BLAS ルーチンは、ベクトルが粗であるのを利用し、ベクトルの非ゼロの成分だけを格納できる。 BLAS を参照。
スパースベクトル	成分の大半がゼロであるベクトル。
台形行列	$A = (A_1 A_2)$ である行列 A 。 A_1 は上三角行列、 A_2 は矩形行列である。
対称行列	$a_{ij} = a_{ji}$ である正方行列 A 。
単位行列	対角成分が 1、非対角成分が 0 である正方行列 I 。任意の行列 A に対し、 $AI = A$ と $IA = A$ 。
単精度	浮動小数点データ型の 1 つ。インテル® プロセッサでは、このデータ型は $1.18 \cdot 10^{-38} < x < 3.40 \cdot 10^{38}$ の範囲にある実数 x を格納できる。 このデータ型に対するマシン精度 ϵ は、およそ 10^{-7} になる。つまり、単精度の数は、通常は、7 以下の有効 10 進桁を格納できる。詳細は、『Pentium® Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual』 (英語) を参照。

直交行列	転置とその逆が等しくなる、つまり $A^T = A^{-1}$ であるような実数の正方行列 A 。したがって $AA^T = A^T A = I$ 。直交行列では、すべての固有値は絶対値 1 を持つ。
転置	所定の行列 A の転置は、 $(A^T)_{ij} = a_{ji}$ となる行列 A^T になる (A の行が A^T の列に、 A の列が A^T の行になる)。
特異行列	行列式がゼロであるような行列。 A が特異行列とすると、逆 A^{-1} は存在せず、また連立方程式 $Ax = b$ は独自の解を持たない (つまり、解が存在しないか無限数の解が存在する)。
特異値	所定の一般行列 A に対し、行列 AA^H の固有値として定義される数。SVD も参照。
ドット積	$x \cdot y$ で表され、特定のベクトル x と y に対して次のように定義される数: $x \cdot y = \sum_i x_i y_i$ 。 x_i と y_i はそれぞれ、 x と y の i 番目の成分を表す。
倍精度	浮動小数点データ型の 1 つ。インテル® プロセッサでは、このデータ型は $2.23 \cdot 10^{-308} < x < 1.79 \cdot 10^{308}$ の範囲にある実数 x を格納できる。 このデータ型に対するマシン精度 ϵ は、およそ 10^{-15} になる。つまり、倍精度の数は、通常は、15 以下の有効 10 進桁を格納できる。 詳細は、『 <i>Pentium® Processor Family Developer's Manual, Volume 3: Architecture and Programming Manual</i> 』(英語)を参照。
フル格納	あらゆる種類の行列の格納が可能な格納形式。行列 A は、2次元の配列 a に格納される。このとき、行列の成分 a_{ij} は、配列の成分 $a(i, j)$ に格納される。
マシン精度	マシンによる実数表現の精度を決定する数 ϵ 。インテル® アーキテクチャーでは、マシン精度は単精度のデータに対しては約 10^{-7} に、倍精度のデータに対しては約 10^{-15} になる。この精度は、マシンによる実数の表現における有効 10 進桁数も決定する。倍精度と単精度 も参照。
ユニタリー行列	共役とその逆が等しくなる複素数型の正方行列 A 。つまり、 $A^H = A^{-1}$ となり、したがって $AA^H = A^H A = I$ となる。ユニタリー行列のすべての固有値は、絶対値 1 を持つ。
ランダムストリーム	一様分布の独立同一分布乱数の抽象的なソース。本書におけるランダムストリームとは、指定されたランダムストリームに関連付けられた基本生成器によって生成される乱数列を、一意に定義する構造を指す。

参考文献

BLAS、スパース BLAS、LAPACK、ScaLAPACK、スパースソルバー、区間ソルバー、VML、VSL、および DFT の機能のさらに詳しい説明は、以下の資料を参照。

- **BLAS レベル 1**

C. Lawson, R. Hanson, D. Kincaid, F. Krough 著。『*Basic Linear Algebra Subprograms for Fortran Usage*』。ACM Transactions on Mathematical Software, Vol.5, No.3 (1979 年 9 月) 308-325。

- **BLAS レベル 2**

J. Dongarra, J. Du Croz, S. Hammarling, R. Hanson 著。『*An Extended Set of Fortran Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software, Vol.14, No.1 (1988 年 3 月) 1-32。

- **BLAS レベル 3**

J. Dongarra, J. DuCroz, I. Duff, S. Hammarling 著。『*A Set of Level 3 Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software (1989 年 12 月)。

- **スパース BLAS**

D. Dodson, R. Grimes, J. Lewis 著。『*Sparse Extensions to the FORTRAN Basic Linear Algebra Subprograms*』。ACM Transactions on Math Software, Vol.17, No.2 (1991 年 6 月)。

D. Dodson, R. Grimes, J. Lewis 著。『*Algorithm 692: Model Implementation and Test Package for the Sparse Basic Linear Algebra Subprograms*』。ACM Transactions on Mathematical Software, Vol.17, No.2 (1991 年 6 月)。

[Duff86] I.S.Duff, A.M.Erisman, J.K.Reid 著。『*Direct Methods for Sparse Matrices*』。Clarendon Press, Oxford, UK, 1986 年。

[CXML01] 『*Compaq Extended Math Library Reference Guide*』、2001 年 10 月。

[Rem05] K.Remington 著。『*A NIST FORTRAN Sparse Blas User's Guide*』。
(<http://math.nist.gov/~KRemington/fspblas/> を参照のこと)

[Saad94] Y.Saad 著。『*SPARSKIT: A Basic Tool-kit for Sparse Matrix Computation*』。Version 2, 1994 年。(<http://www.cs.umn.edu/~saad>)

[Saad96] Y.Saad 著。『*Iterative Methods for Linear Systems*』。PWS Publishing, Boston, 1996 年。

- **LAPACK**

- [LUG] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen 著。『*LAPACK Users' Guide*』。Third Edition。Society for Industrial and Applied Mathematics (SIAM)、1999 年。
- [Golub96] G. Golub, C. Van Loan 著。『*Matrix Computations*』。Johns Hopkins University Press, Baltimore, third edition, 1996 年。

- **ScaLAPACK**

- [SLUG] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley 著。『*ScaLAPACK Users' Guide*』。Society for Industrial and Applied Mathematics (SIAM)、1997 年。

- **スパースソルバー**

- [Duff99] I. S. Duff, J. Koster 著。『*The Design and Use of Algorithms for Permuting Large Entries to the Diagonal of Sparse Matrices*』。SIAM J. Matrix Analysis and Applications, 20(4):889–901、1999 年。
- [Dong95] J. Dongarra, V. Eijkhout, A. Kalhan 著。『*Reverse Communication Interface for Linear Algebra Templates for Iterative Methods*』。UT-CS-95-291、1995 年 5 月。
<http://www.netlib.org/lapack/lawnspdf/lawn99.pdf> を参照のこと。
- [Karypis98] G. Karypis, V. Kumar 著。『*A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*』。SIAM Journal on Scientific Computing, 20(1):359–392、1998 年。
- [Li99] X.S.Li, J.W. Demmel 著。『*A Scalable Sparse Direct Solver Using Static Pivoting*』。9th SIAM conference on Parallel Processing for Scientific Computing 発表論文。San Antonio, Texas、1999 年 3 月 22-24 日。
- [Liu85] J.W.H.Liu 著。『*Modification of the Minimum-Degree algorithm by multiple elimination*』。ACM Transactions on Mathematical Software, 11(2):141–153、1985 年。
- [Menon98] R. Menon L. Dagnum 著。『*OpenMP: An Industry-Standard API for Shared-Memory Programming*』。IEEE Computational Science & Engineering, 1:46–55、1998 年。
<http://www.openmp.org> を参照のこと。
- [Saad03] Y. Saad 著。『*Iterative Methods for Sparse Linear Systems*』。2nd edition、SIAM、Philadelphia, PA、2003 年。
- [Schenk00] O. Schenk 著。『*Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors*』。博士論文。ETH Zurich、2000 年。

- [Schenk00-2] O. Schenk, K. Gartner, W. Fichtner 著。『Efficient Sparse LU Factorization with Left-right Looking Strategy on Shared Memory Multiprocessors』。BIT, 40(1):158–176、2000 年。
- [Schenk01] O. Schenk, K. Gartner 著。『Sparse Factorization with Two-Level Scheduling in PARDISO』。10th SIAM conference on Parallel Processing for Scientific Computing 発表論文。Portsmouth, Virginia、2001 年 3 月 12-14 日。
- [Schenk02] O. Schenk, K. Gartner 著。『Two-Level Scheduling in PARDISO: Improved Scalability on Shared Memory Multiprocessing Systems』。Parallel Computing, 28:187–197、2002 年。
- [Schenk03] O. Schenk, K. Gartner 著。『Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO』。Journal of Future Generation Computer Systems, 20(3):475-487、2004 年。
- [Schenk04] O. Schenk, K. Gartner 著。『On Fast Factorization Pivoting Methods for Sparse Symmetric Indefinite Systems』。Technical Report, Department of Computer Science, University of Basel、2004 年。提出。
- [Sonn89] P. Sonneveld 著。『CGS, a Fast Lanczos-Type Solver for Nonsymmetric Linear Systems』。SIAM Journal on Scientific and Statistical Computing, 10:36–52、1989 年。
- [Young71] D.M.Young 著。『Iterative Solution of Large Linear Systems』。New York, Academic Press, Inc.、1971 年。
- **VSL**
- [VSL Notes] インテル[®] MKL 製品に同梱されるドキュメント (ファイル名: vslnotes.pdf)。
- [Bratley87] Bratley P.、Fox B.L.、Schrage L.E. 著。『A Guide to Simulation』。2nd edition。Springer-Verlag, New York、1987 年。
- [Bratley88] Bratley P.、Fox B.L. 著。『Implementing Sobol's Quasirandom Sequence Generator』。ACM Transactions on Mathematical Software, Vol.14, No.1, 88-100、1988 年 3 月。
- [Bratley92] Bratley P.、Fox B.L.、Niederreiter H. 著。『Implementation and Tests of Low-Discrepancy Sequences』。ACM Transactions on Modeling and Computer Simulation, Vol.2, No.3, 195-213、1992 年 7 月。
- [Coddington94] Coddington, P. D. 『Analysis of Random Number Generators Using Monte Carlo Simulation』。Int. J. Mod. Phys. C-5, 547、1994 年。
- [Gentle98] Gentle, James E 著。『Random Number Generation and Monte Carlo Methods』。Springer-Verlag New York, Inc.、1998 年。

- [L'Ecuyer94] L'Ecuyer, Pierre 著。『*Uniform Random Number Generation*』。Annals of Operations Research, 53, 77–120、1994 年。
- [L'Ecuyer99] L'Ecuyer, Pierre 著。『*Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure*』。Mathematics of Computation, 68, 225, 249-260、1999 年。
- [L'Ecuyer99a] L'Ecuyer, Pierre 著。『*Good Parameter Sets for Combined Multiple Recursive Random Number Generators*』。Operations Research, 47, 1, 159-164、1999 年。
- [L'Ecuyer01] L'Ecuyer, Pierre 著。『*Software for Uniform Random Number Generation: Distinguishing the Good and the Bad*』。2001 Winter Simulation Conference 発表論文。IEEE Press, 95–105、2001 年 12 月。
- [Kirkpatrick81] Kirkpatrick, S., Stoll, E. 著。『*A Very Fast Shift-Register Sequence Random Number Generator*』。Journal of Computational Physics, V. 40.517–526、1981 年。
- [Knuth81] Knuth, Donald E. 著。『*The Art of Computer Programming, Volume 2, Seminumerical Algorithms*』。2nd edition。Addison-Wesley Publishing Company, Reading, Massachusetts、1981 年。
- [Matsumoto98] Matsumoto, M., Nishimura, T. 著。『*Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*』。ACM Transactions on Modeling and Computer Simulation, Vol.8, No.1, 3–30、1998 年 1 月。
- [Matsumoto2000] Matsumoto, M., Nishimura, T. 著。『*Dynamic Creation of Pseudorandom Number Generators*』。56-69: Monte Carlo and Quasi-Monte Carlo Methods 1998 に収録。Niederreiter, H., Spanier, J. 編、Springer 2000。
<http://www.math.sci.hiroshima-u.ac.jp/%7Em-mat/MT/DC/dc.html> を参照のこと。
- [NAG] NAG Numerical Libraries。
http://www.nag.co.uk/numeric/numerical_libraries.asp を参照のこと。
- [Sobol76] Sobol, I.M., Levitan, Yu.L. 著。『*The production of points uniformly distributed in a multidimensional cube*』。Preprint 40, Institute of Applied Mathematics, USSR Academy of Sciences、1976 年 (ロシア)。

• **DFT**

- [1] E. Oran Brigham 著。『*The Fast Fourier Transform and Its Applications*』。Prentice Hall, New Jersey、1988 年。
- [2] Athanasios Papoulis 著。『*The Fourier Integral and its Applications*』。2nd edition。McGraw-Hill, New York、1984 年。

- [3] Ping Tak Peter Tang 著。『*DFTI, a New API for DFT: Motivation, Design, and Rationale*』。2002年7月。
- [4] Charles Van Loan 著。『*Computational Frameworks for the Fast Fourier Transform*』。SIAM, Philadelphia、1992年。
- VML

J.M.Muller 著。『*Elementary functions: algorithms and implementation*』。Birkhauser Boston、1997年。

『IEEE Standard for Binary Floating-Point Arithmetic』。
ANSI/IEEE Std 754-1985。
 - 区間ソルバー

[Alefeld83] G. Alefeld、J. Herzberger 著。『*Introduction to Interval Computations*』。 – Academic Press, New York, 1983年。

[Bentbib02] A.H.Bentbib 著。『*Solving the full rank interval least squares problem*』 // *Applied Numerical Mathematics*。 – 2002年。 – Vol. 41.– P.283–294。

[Bliek92] Ch. Bliek 著。『*Computer methods for design automation*』。
博士論文。 – Dept. of Ocean Engineering, Massachusetts Institute of Technology、1992年。

[Hammer95] R. Hammer、M. Hocks、U. Kulisch、D. Ratz 著。『*C++ Toolbox for Verified Computing I. Basic Numerical Problems*』。 – Berlin-Heidelberg: Springer, 1995年。

[Hansen92] E.Hansen 著。『*Bounding the solution of interval linear equations*』 // *SIAM Journal on Numerical Analysis*。 – 1992年。 – Vol.29, No. 5.– P.1493–1503。

[Herzberger94] J. Herzberger 著。『*Iterative methods for the inclusion of the inverse of a matrix*』 // *Topics in Validated Computations*』。
J. Herzberger 編。 – Amsterdam: Elsevier、1994年。 –

[Jansson91] Ch.Jansson 著。『*Interval linear systems with symmetric matrices, skew-symmetric matrices, and dependencies in the right hand side*』 // *Computing*。 – 1991年。 – Vol. 46.– P.265 – 274。

[Kearfott96] R.B.Kearfott 著。『*Rigorous Global Search: Continuous Problems*』。 – Dordrecht, Kluwer、1996年。

[Kearfott] R.B.Kearfott、M.T. Nakao、A. Neumaier、S.M.Rump、S.P. Shary、P. van Hentenryck 著。『*Standardized notation in interval analysis*』。
<http://www.mat.univie.ac.at/~neum/software/int/> を参照のこと。

- [Kreinovich97] V. Kreinovich, A. Lakeyev, J. Rohn, P. Kahl 著。『*Computational Complexity and Feasibility of Data Processing and Interval Computations*』。 – Kluwer, Dordrecht, 1997 年。
- [Neumaier90] A. Neumaier 著。『*Interval Methods for Systems of Equations*』。 – Cambridge, Cambridge University Press, 1990 年。
- [Neumaier99] A. Neumaier 著。『A simple derivation of Hansen-Blik-Rohn-Ning-Kearfott enclosure for linear interval equations』// *Reliable Computing*。 – 1999 年。 – Vol.5, No. 2.– P.131–136.
- [Ning97] S. Ning, R.B.Kearfott 著。『A comparison of some methods for solving linear interval equations』// *SIAM Journal on Numerical Analysis*。 – 1997 年。 – Vol.34, No. 4.– P.1289–1305。
- [Rex99] G.Rex, J.Rohn 著。『Sufficient conditions for regularity and singularity of interval matrices』// *SIAM Journal on Numerical Analysis*。 – 1999 年。 – Vol. 20.– P.437–445。
- [Rohn93] J. Rohn 著。Cheap and tight bounds: the recent result by E. Hansen can be made more efficient』// *Interval Computations*。 – 1993 年。 – No. 4.– P.13–21。
- [Rump83] S. M.Rump 著。『Solving algebraic problems with high accuracy』// *A New Approach to Scientific Computation*; Kulisch U. W. , Miranker W. L. 編。 – New York: Academic Press, 1983 年。 – P.51–120。
- [Rump84] S. M.Rump 著。『Solution of linear and nonlinear algebraic problems with sharp guaranteed bounds』// *Computing Supplement*。 – 1984 年。 – Vol. 5.– P.147–168。
- [Rump80] S. M.Rump, Kaucher E 著。『Small bounds for the solution of systems of linear equations』// *Computing Supplement*。 – 1980 年。 – Vol. 2.– P.157–164。
- [Rump] S.Rump 著。『INTLAB — INTerval LABoratory』。 – 21 p. – <http://www.ti3.tu-harburg.de/~rump/intlab/> を参照のこと。
- [Shary92] S.P. Shary 著。『A new class of algorithms for optimal solution of interval linear systems』// *Interval Computations*。 – 1992, No. 2(4). – P.18–29。
- [Shary95] S.P.Shary 著。『On optimal solution of interval linear equations』// *SIAM Journal on Numerical Analysis*。 – 1995 年。 – Vol.32, No. 2.– P.610–630。
- [Shary02] S. P. Shary 著。『A new technique in systems analysis under interval uncertainty and ambiguity』// *Reliable Computing*。 – 2002 年。 – Vol. 8.– 321–419。

[Shary] S.P.Shary 著。『A new class of methods for optimal enclosing solution sets to interval linear systems』// *Journal of Computational Mathematics*。 - 出版予定。

BLAS、スパース BLAS、LAPACK、および ScaLAPACK パッケージ (プラットフォーム固有の最適化なし) を参考用に導入するには、www.netlib.org を参照のこと。

索引

記号

?asum 2-5
?axpy 2-6
?axpyi 2-105
?_backward_trig_transform 13-9
?bdsdc 4-86
?bdsqr 4-83
?combamax1 7-7
?_commit_Helmholtz_2D 13-28
?_commit_Helmholtz_3D 13-28
?_commit_trig_transform 13-6
?ConvExec 10-102
?ConvExec1D 10-104
?ConvExecX 10-106
?ConvExecX1D 10-107
?ConvNewTask 10-88
?ConvNewTask1D 10-90
?ConvNewTaskX 10-91
?ConvNewTaskX1D 10-94
?copy 2-7
?CorrExec 10-102
?CorrExec1D 10-104
?CorrExecX 10-106
?CorrExecX1D 10-107
?CorrNewTask 10-88
?CorrNewTask1D 10-90
?CorrNewTaskX 10-91
?CorrNewTaskX1D 10-94
?dbtf2 7-149
?dbtrf 7-150
?disna 4-139
?dot 2-9
?dotc 2-11
?dotci 2-107
?doti 2-106
?dotu 2-12
?dotui 2-108
?dttrf 7-152
?dttrsv 7-153
?_forward_trig_transform 13-8
?gbbdr 4-69
?gbcon 3-63
?gbequ 3-142
?gbmv 2-27
?gbrfs 3-90
?gbsv 3-157
?gbsvx 3-159
?gbtf2 5-19
?gbtrf 3-11
?gbtrs 3-34
?gebak 4-171
?gebal 4-168
?gebd2 5-21
?gebrd 4-66
?gecon 3-61
?geequ 3-140
?gees 4-332
?geesx 4-336
?geev 4-341
?geevx 4-344
?gegas 12-4
?gegss 12-9
?gehbs 12-10
?gehd2 5-22
?gehrd 4-157
?gehss 12-6
?gekws 12-7
?gelq2 5-24
?gelqf 4-22
?gels 4-243
?gelsd 4-252
?gelss 4-249
?gelsy 4-246
?gemip 12-21
?gemm 2-78

?gemv 2-30	?gtts2 5-33
?gepps 12-12	?hbev 4-305
?gepss 12-13	?hbevd 4-309
?geql2 5-25	?hbevz 4-316
?geqlf 4-32	?hbgst 4-151
?geqp3 4-11	?hbgv 4-402
?geqpf 4-9	?hbgvd 4-407
?geqr2 5-27	?hbgvx 4-413
?geqrf 4-6	?hbtrd 4-117
?ger 2-32	?hecon 3-76
?gerbr 12-18	?heev 4-264
?gerc 2-34	?heevd 4-268
?gerfs 3-88	?heevr 4-282
?gerq2 5-28	?heevz 4-274
?gerqf 4-42	?heft2 5-248
?geru 2-35	?hegst 4-143
?gesc2 5-29	?hegv 4-366
?gesdd 4-354	?hegvd 4-371
?gesv 3-150	?hegvz 4-378
?gesvd 4-350	?_Helmholtz_2D 13-32
?gesvr 12-19	?_Helmholtz_3D 13-32
?gesvx 3-152	?hemm 2-81
?geszi 12-17	?hemv 2-39
?getc2 5-30	?her 2-41
?getf2 5-32	?her2 2-43
?getrf 3-10	?her2k 2-86
?getri 3-124	?herfs 3-108
?getrs 3-32	?herk 2-84
?ggbak 4-205	?hesv 3-199
?ggbal 4-203	?hesvx 3-201
?gges 4-418	?hetrd 4-99
?ggesx 4-423	?hetrf 3-24
?ggev 4-429	?hetri 3-131
?ggevz 4-433	?hetrs 3-48
?ggglm 4-258	?hgeqz 4-207
?gghrd 4-200	?hpcon 3-80
?gglse 4-256	?hpev 4-289
?ggqrf 4-58	?hpevd 4-293
?ggrqf 4-61	?hpevx 4-299
?ggsvd 4-357	?hpgst 4-147
?ggsvp 4-233	?hpgv 4-385
?gtcon 3-65	?hpgvd 4-390
?gthr 2-110	?hpgvx 4-396
?gthrz 2-111	?hpmv 2-45
?gtrfs 3-93	?hpr 2-47
?gtsv 3-164	?hpr2 2-49
?gtsvx 3-166	?hprfs 3-113
?gttrf 3-13	?hpsvx 3-212
?gttrs 3-36	?hptrd 4-110

?hptrf 3-29	?lamc4 5-264
?hptri 3-134	?lamc5 5-265
?hptrs 3-52	?lamch 5-261
?hsein 4-177	?lamrg 5-94
?hseqr 4-173	?lamsh 7-141
?_init_Helmholtz_2D 13-26	?langb 5-95
?_init_Helmholtz_3D 13-26	?lange 5-96
?_init_trig_transform 13-5	?langt 5-97
?labrd 5-34	?lanhb 5-101
?lacgv 5-8	?lanhe 5-108
?lacon 5-36	?lanhp 5-104
?lacpy 5-37	?lanhs 5-98
?lacrm 5-8	?lansb 5-100
?lacrt 5-9	?lansp 5-103
?ladiv 5-38	?lanst/?lanht 5-105
?lae2 5-39	?lansy 5-106
?laebz 5-40	?lantb 5-109
?laed0 5-44	?lantp 5-110
?laed1 5-46	?lantr 5-112
?laed2 5-48	?lanv2 5-113
?laed3 5-50	?lapll 5-114
?laed4 5-51	?lapmt 5-115
?laed5 5-53	?lapy2 5-116
?laed6 5-54	?lapy3 5-117
?laed7 5-55	?laqgb 5-117
?laed8 5-58	?laqge 5-119
?laed9 5-60	?laqp2 5-120
?laeda 5-62	?laqps 5-122
?laein 5-63	?laqsb 5-123
?laesy 5-10	?laqsp 5-125
?laev2 5-66	?laqsy 5-126
?laexc 5-67	?laqtr 5-128
?lag2 5-68	?lar1v 5-130
?lags2 5-70	?lar2v 5-132
?lagtf 5-72	?laref 7-143
?lagtm 5-73	?larf 5-133
?lagts 5-75	?larfb 5-134
?lagv2 5-76	?larfg 5-136
?lahef 5-206	?larft 5-137
?lahqr 5-78	?larfx 5-140
?lahrd 5-80	?largv 5-141
?laic1 5-82	?larnv 5-142
?lahn2 5-84	?larrb 5-143
?lals0 5-86	?larre 5-145
?lalsa 5-89	?larrf 5-146
?lalasd 5-92	?larrv 5-148
?lamc1 5-262	?lartg 5-150
?lamc2 5-263	?lartv 5-151
?lamc3 5-264	?laruv 5-152

?larz 5-153	?orgl2/?ungl2 5-226
?larzb 5-154	?orglq 4-24
?larzt 5-156	?orgql 4-34
?las2 5-159	?orgqr 4-13
?lascl 5-160	?orgr2/?ungr2 5-227
?lasd0 5-161	?orgrq 4-44
?lasd1 5-162	?orgtr 4-95
?lasd2 5-165	?orm2l/?unm2l 5-229
?lasd3 5-168	?orm2r/?unm2r 5-231
?lasd4 5-170	?ormbr 4-74
?lasd5 5-171	?ormhr 4-161
?lasd6 5-172	?orml2/?unml2 5-233
?lasd7 5-176	?ormlq 4-26
?lasd8 5-179	?ormql 4-37
?lasd9 5-181	?ormqr 4-15
?lasda 5-182	?ormr2/?unmr2 5-235
?lasdq 5-186	?ormr3/?unmr3 5-237
?lasdt 5-188	?ormrq 4-47
?laset 5-189	?ormrz 4-53
?lasorte 7-145	?ormtr 4-96
?lasq1 5-190	?pbcon 3-71
?lasq2 5-191	?pbequ 3-147
?lasq3 5-192	?pbrfs 3-101
?lasq4 5-193	?pbstf 4-153
?lasq5 5-194	?pbsv 3-182
?lasq6 5-195	?pbsvx 3-183
?lasr 5-196	?pbt2 5-239
?lasrt 5-198	?pbt3 3-18
?lasrt2 7-146	?pbtrs 3-42
?lassq 5-198	?pocon 3-67
?lasv2 5-200	?poequ 3-144
?laswp 5-201	?porfs 3-96
?lasy2 5-202	?posv 3-170
?lasyf 5-204	?posvx 3-171
?latbs 5-208	?potf2 5-240
?latdf 5-210	?potrf 3-15
?latps 5-212	?potri 3-126
?latrd 5-214	?potrs 3-38
?latrs 5-216	?ppcon 3-69
?latrz 5-219	?ppequ 3-145
?lauu2 5-221	?pprfs 3-98
?lauum 5-222	?ppsv 3-176
?nrm2 2-13	?ppsvx 3-177
?opgtr 4-106	?pptrf 3-17
?opmtr 4-108	?pptri 3-127
?org2l/?ung2l 5-223	?pptrs 3-40
?org2r/?ung2r 5-225	?ptcon 3-73
?orgbr 4-72	?pteqr 4-131
?orghr 4-159	?ptrfs 3-103

?ptsv 3-188	?steqr2 7-155
?ptsvx 3-189	?sterf 4-119
?ptrf 3-20	?stev 4-320
?ptrs 3-44	?stevd 4-321
?ptrsv 7-154	?stevr 4-327
?ptts2 5-241	?stevx 4-324
?rot 2-14	?sum1 5-19
?rot (複素数) 5-11	?swap 2-22
?rotg 2-16	?sycon 3-74
?roti 2-112	?syev 4-262
?rotm 2-17	?syevd 4-266
?rotmg 2-19	?syevr 4-278
?rscl 5-243	?syevx 4-271
?sbev 4-303	?sygs2/?hegs2 5-244
?sbevd 4-307	?sygst 4-141
?sbevx 4-312	?sygv 4-363
?sbgst 4-149	?sygvd 4-368
?sbgv 4-400	?sygvx 4-374
?sbgvd 4-404	?symm 2-89
?sbgvx 4-410	?symv 2-59
?sbmv 2-51	?symv (複素数) 5-15
?sbrd 4-115	?syr 2-61
?scal 2-20	?syr (複素数) 5-17
?sctr 2-113	?syr2 2-62
?sdot 2-10	?syr2k 2-94
?spcon 3-78	?syrrs 3-106
?spev 4-287	?syrrk 2-92
?spevd 4-291	?sysv 3-192
?spevx 4-296	?sysvx 3-195
?spgst 4-145	?sytd2/?hetd2 5-245
?spgv 4-382	?sytf2 5-247
?spgvd 4-387	?sytrd 4-93
?spgvx 4-393	?sytrf 3-21
?spm 2-53, 5-12	?sytri 3-129
?spr 2-55, 5-14	?sytrs 3-46
?spr2 2-57	?tbcon 3-85
?sprfs 3-111	?tbmv 2-64
?spsv 3-205	?tbsv 2-67
?spsvx 3-207	?tbtrs 3-58
?sptrd 4-105	?tgevc 4-213
?sptrf 3-27	?tgex2 5-250
?sptri 3-132	?tgexc 4-217
?sptrs 3-50	?tgensen 4-220
?stebz 4-134	?tgsja 4-237
?stedc 4-123	?tgsna 4-228
?stegr 4-127	?tgsy2 5-252
?stein 4-137	?tgsyl 4-225
?stein2 7-147	?tpcon 3-83
?steqr 4-120	?tpmv 2-69

?tprfs 3-118
 ?tpsv 2-71
 ?tptri 3-137
 ?tprts 3-56
 ?trcon 3-81
 ?trevc 4-182
 ?trexc 4-190
 ?trmm 2-97
 ?trmv 2-73
 ?trrfs 3-116
 ?trsen 4-192
 ?trsm 2-100
 ?trsna 4-186
 ?trsv 2-75
 ?trsyl 4-196
 ?trti2 5-255
 ?trtri (LAPACK) 3-136
 ?trtri (区間線形ソルバー) 12-16
 ?trtrs (LAPACK) 3-54
 ?trtrs (区間線形ソルバー) 12-3
 ?tzrzf 4-51
 ?ungbr 4-77
 ?unghr 4-164
 ?unglq 4-28
 ?ungql 4-36
 ?ungqr 4-18
 ?ungrq 4-45
 ?ungtr 4-101
 ?unmbr 4-80
 ?unmhr 4-166
 ?unmlq 4-30
 ?unmq1 4-39
 ?unmqr 4-20
 ?unmrq 4-49
 ?unmrz 4-56
 ?unmtr 4-103
 ?upgtr 4-111
 ?upmtr 4-113

数字

1次元FFT
 格納効果 11-27, 11-28

1次方程式の解の誤差
 一般行列 3-88, 6-41
 帯格納 3-90
 エルミート行列 3-108
 圧縮格納 3-113
 エルミート正定値行列 3-96, 3-103, 6-44
 圧縮格納 3-98
 帯格納 3-101

三角行列 3-116
 圧縮格納 3-118
 帯格納 3-121
 三重対角行列 3-93
 対称行列 3-106
 圧縮格納 3-111
 対称正定値行列 3-96, 3-103, 6-44
 圧縮格納 3-98
 帯格納 3-101
 分散三重対角係数行列 6-47

1次方程式の解の算出。1次方程式を参照

1次方程式の解の精度の改善
 一般行列 3-88, 6-41
 エルミート行列 3-108
 圧縮格納 3-113
 エルミート正定値行列 3-96, 3-103
 圧縮格納 3-98
 帯格納 3-101
 帯行列 3-90
 三重対角行列 3-93
 対称/エルミート正定値分散行列 6-44
 対称行列 3-106
 圧縮格納 3-111
 対称正定値行列 3-96, 3-103
 圧縮格納 3-98
 帯格納 3-101

1次方程式の誤差推定
 分散三重対角係数行列 6-47

1次方程式、解の算出
 エルミート正定値三重対角1次方程式 7-154
 三角行列
 帯格納 3-58, 7-126
 三重対角行列
 ScaLAPACK 補助 7-153
 対称正定値三重対角1次方程式 7-154

1次方程式、算出 5-84, 5-128
 一般帯行列
 ScaLAPACK 6-179
 一般行列 3-32, 6-18
 帯格納 3-34, 6-20
 一般三重対角行列
 ScaLAPACK 6-182
 エルミート行列 3-48
 圧縮格納 3-52, 3-210, 3-212
 誤差範囲 3-201, 3-212
 エルミート正定値行列 3-38, 6-22
 LAPACK 3-170, 3-171
 ScaLAPACK 6-186
 圧縮格納 3-40, 3-176, 3-177
 帯格納 3-42, 3-183, 6-23
 LAPACK 3-182
 ScaLAPACK 6-191
 誤差範囲 3-177, 3-183
 LAPACK 3-171
 ScaLAPACK 6-186
 エルミート正定値三重対角行列 3-189, 6-25
 LAPACK 3-188

- ScaLAPACK 6-193
 - 誤差範囲 3-189
 - 帯行列
 - LAPACK 3-157, 3-159
 - ScaLAPACK 6-177
 - 過大評価問題または過小評価問題 6-195
 - コレスキー因子分解された行列
 - LAPACK 3-42
 - ScaLAPACK 6-23
 - 三角行列 3-54, 6-32
 - 圧縮格納 3-56
 - 三重対角行列
 - LAPACK 3-36, 3-44, 3-164, 3-166
 - LAPACK 補助 5-130
 - 誤差範囲 3-166
 - 正方向列
 - LAPACK 3-150, 3-152
 - ScaLAPACK 6-170, 6-172
 - 誤差範囲
 - LAPACK 3-152, 3-159
 - ScaLAPACK 6-172
 - 対角優位行列
 - 帯行列 6-30
 - 三重対角行列 6-27
 - 対称行列 3-46
 - 圧縮格納 3-50, 3-205, 3-207
 - 誤差範囲 3-195, 3-207
 - 対称正定値行列 3-38, 6-22, 6-25
 - LAPACK 3-170, 3-171
 - ScaLAPACK 6-184, 6-186
 - 圧縮格納 3-40, 3-176, 3-177
 - 帯格納 3-42, 3-183, 6-23
 - LAPACK 3-182
 - ScaLAPACK 6-191
 - 誤差範囲 3-177, 3-183
 - LAPACK 3-171
 - ScaLAPACK 6-186
 - 対称正定値三重対角行列 3-189
 - LAPACK 3-188
 - ScaLAPACK 6-193
 - 誤差範囲 3-189
 - 複数の右辺
 - エルミート行列 3-199, 3-210
 - エルミート正定値行列 3-170, 3-176
 - 帯格納 3-182
 - 帯行列
 - LAPACK 3-157, 3-159
 - ScaLAPACK 6-177
 - 三重対角行列 3-164, 3-166
 - 正方向列
 - LAPACK 3-150, 3-152
 - ScaLAPACK 6-170, 6-172
 - 対称行列 3-192, 3-205
 - 対称正定値行列 3-170, 3-176
 - 帯格納 3-182
 - 1- ノルムの値
 - 一般矩形行列 5-96, 7-45
 - 一般三重対角行列 5-97
 - 上 Hessenberg 行列 5-98, 7-47
 - エルミート帯行列 5-101
 - 三角帯行列 5-109
 - 三角行列 5-112, 7-51
 - 圧縮格納 5-110
 - 実対称行列 5-106, 7-49
 - 実対称三重対角行列 5-105
 - 台形行列 5-112
 - 対称帯行列 5-100
 - 対称行列
 - 圧縮格納 5-103
 - 複素エルミート行列 5-108, 7-49
 - 圧縮格納 5-104
 - 複素エルミート三重対角行列 5-105
 - 複素対称行列 5-106
 - 2つの行列
 - QR 因子分解
 - LAPACK 4-58
 - ScaLAPACK 6-114
- B**
- Bernoulli 10-63
 - Beta 10-58
 - Binomial 10-66
 - BLACS 6-1
 - BLAS ルーチン
 - 行列の引数 B-3
 - ベクトルの引数 B-1
 - ルーチングループ 2-1
 - BLAS レベル 1 の関数
 - ?asum 2-4, 2-5
 - ?dot 2-4, 2-9
 - ?dotc 2-4, 2-11
 - ?dotu 2-4, 2-12
 - ?nrm2 2-4, 2-13
 - ?sdot 2-4, 2-10
 - dcabs1 2-5, 2-25
 - i?amax 2-5, 2-23
 - i?amin 2-5, 2-24
 - コード例 C-1
 - BLAS レベル 1 のルーチン
 - ?axpy 2-4, 2-6
 - ?copy 2-4, 2-7
 - ?rot 2-4, 2-14
 - ?rotg 2-5, 2-16
 - ?rotm 2-5, 2-17
 - ?rotmg 2-19
 - ?rotmq 2-5
 - ?scal 2-5, 2-20
 - ?swap 2-5, 2-22
 - コード例 C-2
 - BLAS レベル 2 のルーチン
 - ?gbmv 2-26, 2-27
 - ?gemv 2-26, 2-30

?ger 2-26, 2-32
 ?gerc 2-26, 2-34
 ?geru 2-26, 2-35
 ?hbmV 2-26, 2-37
 ?hemv 2-26, 2-39
 ?her 2-26, 2-41
 ?her2 2-26, 2-43
 ?hpmv 2-26, 2-45
 ?hpr 2-26, 2-47
 ?hpr2 2-26, 2-49
 ?sbmv 2-26, 2-51
 ?spmv 2-26, 2-53
 ?spr 2-26, 2-55
 ?spr2 2-26, 2-57
 ?symv 2-26, 2-59
 ?syr 2-26, 2-61
 ?syr2 2-26, 2-62
 ?tbmv 2-26, 2-64
 ?tbsv 2-26, 2-67
 ?tpmv 2-26, 2-69
 ?tpsv 2-26, 2-71
 ?trmv 2-26, 2-73
 ?trsv 2-26, 2-75
 コード例 C-2

BLAS レベル 3 のルーチン
 ?gemm 2-78
 ?hemm 2-78, 2-81
 ?her2k 2-78, 2-86
 ?herk 2-78, 2-84
 ?symm 2-78, 2-89
 ?syr2k 2-78, 2-94
 ?syrk 2-78, 2-92
 ?trmm 2-78, 2-97
 ?trsm 2-78, 2-100
 コード例 C-3

BRNG 10-1, 10-7

Bunch-Kaufman 因子分解 3-9, 6-5
 エルミート行列 3-24
 圧縮格納 3-29
 対称行列 3-21
 圧縮格納 3-27

C

Cauchy 10-48
 CBLAS D-1
 スパース BLAS D-13
 引数 D-1
 レベル 1 (ベクトル演算) D-3
 レベル 2 (行列 - ベクトル演算) D-5
 レベル 3 (行列 - 行列演算) D-10
 CommitDescriptor 11-8
 CommitDescriptorDM 11-44
 ComputeBackward 11-13
 ComputeBackwardDM 11-48
 ComputeForward 11-11
 ComputeForwardDM 11-46
 ConvCopyTask 10-110
 ConvDeleteTask 10-109
 ConvInternalPrecision 10-98
 ConvSetDecimation 10-100
 ConvSetMode 10-97
 ConvSetStart 10-99
 CopyDescriptor 11-9
 CopyStream 10-21
 CopyStreamState 10-22
 CorrCopyTask 10-110
 CorrDeleteTask 10-109
 CorrSetInternalDecimation 10-100
 CorrSetInternalPrecision 10-98
 CorrSetMode 10-97
 CorrSetStart 10-99
 Cray 7-158
 CreateDescriptor 11-6
 CreateDescriptorDM 11-42

D

dcabs1 2-25
 DeleteStream 10-20
 DFT インターフェイス 11-3
 DFT 計算 11-3
 クラスター DFTI 11-42
 DFT ルーチン
 DFT 計算
 ComputeBackward 11-13
 ComputeBackwardDM 11-48
 ComputeForward 11-11
 ComputeForwardDM 11-46
 ステータス確認 11-3
 ErrorClass 11-4
 ErrorMessage 11-5
 ディスクリプター構成
 GetValue 11-16
 GetValueDM 11-52
 SetValueDM 11-51
 SetValue 11-15
 ディスクリプター操作
 CommitDescriptor 11-8
 CommitDescriptorDM 11-44
 CopyDescriptor 11-9
 CreateDescriptor 11-6
 CreateDescriptorDM 11-42
 FreeDescriptor 11-10
 FreeDescriptorDM 11-45
 dNewAbstractStream 10-16
 DSS インターフェイス, スパースソルバー 8-13

E

ErrorClass 11-4
 ErrorMessage 11-5
 ESSL ライブラリー 10-82
 Exponential 10-42

F

Fortran-95 LAPACK インターフェイスと Netlib 3-4
 Fortran-95 インターフェイス規則
 BLAS、スパース BLAS 2-3
 LAPACK 3-3
 FreeDescriptor 11-10
 FreeDescriptorDM 11-45
 free_Helmholtz_2D 13-36
 free_Helmholtz_3D 13-36
 free_trig_transform 13-11
 Frobenius の値
 一般矩形行列 5-96, 7-45
 Frobenius ノルムの値
 一般三重対角行列 5-97
 上 Hessenberg 行列 5-98, 7-47
 エルミート帯行列 5-101
 三角帯行列 5-109
 三角行列 5-112, 7-51
 圧縮格納 5-110
 実対称行列 5-106, 7-49
 実対称三重対角行列 5-105
 台形行列 5-112
 対称帯行列 5-100
 対称行列
 圧縮格納 5-103
 複素エルミート行列 5-108, 7-49
 圧縮格納 5-104
 複素エルミート三重対角行列 5-105
 複素対称行列 5-106

G

Gamma 10-56
 Gaussian 10-36
 GaussianMV 10-38
 Gauss-Seidel の反復法、区間方程式 12-9, 12-22
 Geometric 10-65
 GetBrngProperties 10-77
 GetNumRegBrngs 10-32
 GetStreamStateBrng 10-31
 GetValue 11-16
 GetValueDM 11-52
 GFSR 10-3
 Givens 回転
 スパースベクトル 2-112
 パラメーター 2-16

変形 Givens 変換パラメーター 2-19
 Gumbel 10-54

H

Hansen-Blik-Rohn プロシージャ、区間方程式 12-11
 Householder 行列
 LAPACK 5-136
 ScaLAPACK 7-70
 Householder 法、区間方程式 12-6, 12-22
 Householder リフレクター 7-143
 Hypergeometric 10-68

I

i?amax 2-23
 i?amin 2-24
 i?max1 5-18
 IEEE 演算 7-44
 IEEE スタンダード
 実装 7-159
 符号ビットの位置 7-161
 ilaenv 5-257
 iNewAbstractStream 10-14

K

Krawczyk 反復法、区間方程式 12-7

L

LAPACK 用 Fortran-95 インターフェイス
 Netlib と同一 E-2
 Netlib にない E-6
 修正された Netlib インターフェイス E-5
 新機能 E-10
 置換された Netlib 引数名 E-4
 LAPACK ルーチン
 1 次方程式の解の算出
 ?gbtrs 3-34
 ?getrs 3-32
 ?gttrs 3-36
 ?hetrs 3-48
 ?hptrs 3-52
 ?laln2 5-84
 ?laqtr 5-128
 ?pbtrs 3-42
 ?potrs 3-38
 ?pptrs 3-40
 ?pttrs 3-44
 ?sptrs 3-50
 ?sytrs 3-46
 ?tbtrs 3-58
 ?tptrs 3-56

- ?trrs 3-54
- 2×2 一般固有値問題 5-68
- 2×2 エルミート行列
 - 面回転 5-132
- 2×2 三角行列
 - SVD 5-200
 - 特異値 5-159
- 2×2 実数行列
 - 汎用 Schur 因子分解 5-76
- 2×2 実数非対称行列
 - Schur 因子分解 5-113
- 2×2 直交行列 5-70
- 2×2 対称行列
 - 面回転 5-132
- dqd 変換 5-195
- dqds 変換 5-194
- Householder 行列
 - 基本リフレクター 5-136
- LQ 因子分解
 - ?gelq2 5-24
 - ?gelqf 4-22
 - ?orglq 4-24
 - ?ormlq 4-26
 - ?unglq 4-28
 - ?unmlq 4-30
- LU 因子分解
 - 一般帯行列 5-19
- QL 因子分解
 - ?geql2 5-25
 - ?geqlf 4-32
 - ?orgql 4-34
 - ?ormql 4-37
 - ?ungql 4-36
 - ?unmq1 4-39
- QR 因子分解
 - ?geqpf 4-9
 - ?geqr2 5-27
 - ?geqp3 4-11
 - ?geqrf 4-6
 - ?ggqrf 4-58
 - ?ggrqf 4-61
 - ?laqp2 5-120
 - ?laqps 5-122
 - ?orgqr 4-13
 - ?ormqr 4-15
 - ?ungqr 4-18
 - ?unmqr 4-20
 - p?geqrf 6-60
- RQ 因子分解
 - ?geqr2 5-28
 - ?gerqf 4-42
 - ?orgrq 4-44
 - ?ormrq 4-47
 - ?ungrq 4-45
 - ?unmrq 4-49
- RZ 因子分解
 - ?tzrzf 4-51
 - ?ormrz 4-53
- ?unmrz 4-56
- 圧縮形式のエルミート帯行列
 - 平衡化 5-125
- 圧縮形式の対称帯行列
 - 平衡化 5-125
- 一様分布 5-152
- 一般帯行列 5-95
 - 平衡化 5-117
- 一般行列
 - 基本リフレクター 5-153
 - 二重対角形式への縮退 5-21, 5-34
 - ブロック・リフレクター 5-154
- 一般矩形行列
 - LQ 因子分解 5-24
- 一般三重対角行列 5-72, 5-73, 5-75, 5-97, 5-145, 5-148
- 一般正方行列 5-80, 5-96, 5-160
 - QL 因子分解 5-25
 - QR 因子分解 5-27
 - RQ 因子分解 5-28
 - 上 Hessenberg 形式への縮退 5-22
 - 基本リフレクター 5-133, 5-140
 - 行変換 5-201
 - ブロック・リフレクター 5-134
 - 平衡化 5-119
 - 面回転 5-196
- 上 Hessenberg 行列 5-98
 - Schur 因子分解 5-78
 - 固有値 5-78
 - 指定された固有ベクトルの計算 5-63
- エルミート帯行列 5-101
 - 平衡化 5-123, 5-126
- エルミート行列
 - 固有値および固有ベクトルの計算 5-66
- 解の精度の改善と誤差の推定
 - ?gbrfs 3-90
 - ?gerfs 3-88
 - ?gtrfs 3-93
 - ?herfs 3-108
 - ?hprfs 3-113
 - ?pbrfs 3-101
 - ?porfs 3-96
 - ?pprfs 3-98
 - ?ptrfs 3-103
 - ?sprfs 3-111
 - ?syrf 3-106
 - ?tbrfs 3-121
 - ?tprfs 3-118
 - ?trrf 3-116
- 数のソート 5-198
- 環境問い合わせ 5-257
- 行列 - 行列の積
 - ?lagtm 5-73
- 行列の反転
 - ?getri 3-124
 - ?hetri 3-131
 - ?hptri 3-134
 - ?potri 3-126

- ?pptri 3-127
- ?sptri 3-132
- ?sytri 3-129
- ?tptri 3-137
- ?trtri 3-136
- 行列の平衡化
 - ?gbequ 3-142
 - ?geequ 3-140
 - ?laqgb 5-117
 - ?laqge 5-119
 - ?laqsb 5-123
 - ?laqsp 5-125
 - ?laqsy 5-126
 - ?pbequ 3-147
 - ?poequ 3-144
 - ?ppequ 3-145
- 行列の列の置換 5-115
- 更新された上二重対角行列
 - SVD 5-172
- 最小固有値に対する近似 5-193
- 三角行列 5-112
 - 圧縮格納 5-110
- 三角分解
 - ?gbtrf 3-11
 - ?getrf 3-10
 - ?gttrf 3-13
 - ?hetrf 3-24
 - ?hptrf 3-29
 - ?pbtrf 3-18
 - ?potrf 3-15
 - ?pptrf 3-17
 - ?pttrf 3-20
 - ?sprtf 3-27
 - ?sytrf 3-21
 - p?dbtrf 6-8
- 三角連立方程式 5-212, 5-216
- 三重帯行列 5-109
- 実数上二重対角行列
 - SVD 5-162, 5-182, 5-186
 - 特異値 5-161
- 実数対称三重対角行列 5-40, 5-105
- 実数の上準三角行列
 - 直交相似変換 5-67
- 実数の下二重対角行列
 - SVD 5-186
- 実数平方二重対角行列
 - 特異値 5-190
- 実対称行列 5-106
- 縮退されていない対称三重対角行列 5-44
- 条件数推定
 - ?disna 4-139
 - ?gbcon 3-63
 - ?gecon 3-61
 - ?gtcon 3-65
 - ?hecon 3-76
 - ?hpcon 3-80
 - ?pocon 3-67
 - ?ppcon 3-69
- ?ptcon 3-73
- ?spcon 3-78
- ?sycon 3-74
- ?tbcon 3-85
- ?tpcon 3-83
- ?trcon 3-81
- シルベスター式
 - ?lasy2 5-202
 - ?tgsy2 5-252
 - ?trsyl 4-196
- 増加条件推定 5-82
- 台形行列 5-112, 5-219
- 対称帯行列 5-100
 - 平衡化 5-123, 5-126
- 対称行列
 - 圧縮格納 5-103
 - 固有値および固有ベクトルの計算 5-66
- 対称固有値問題
 - ?disna 4-139
 - ?hbtrd 4-117
 - ?hetrd 4-99
 - ?hptrd 4-110
 - ?opgtr 4-106
 - ?opmtr 4-108
 - ?orgtr 4-95
 - ?ormtr 4-96
 - ?pteqr 4-131
 - ?sbrtd 4-115
 - ?sprtd 4-105
 - ?stebz 4-134
 - ?stedc 4-123
 - ?stegr 4-127
 - ?stein 4-137
 - ?steqr 4-120
 - ?sterf 4-119
 - ?sytrd 4-93
 - ?ungtr 4-101
 - ?unmtr 4-103
 - ?upgtr 4-111
 - ?upmtr 4-113
- 補助
 - ?lae2 5-39
 - ?laebz 5-40
 - ?laed0 5-44
 - ?laed1 5-46
 - ?laed2 5-48
 - ?laed3 5-50
 - ?laed4 5-51
 - ?laed5 5-53
 - ?laed6 5-54
 - ?laed7 5-55
 - ?laed8 5-58
 - ?laed9 5-60
 - ?laeda 5-62
- 対称正定値三重対角行列
 - 固有値 5-191
- 置換リストの作成 5-94
- 特異値のセットの併合 5-165, 5-176

特異値分解 5-86, 5-89, 5-92

?bdsdc 4-86

?bdsqr 4-83

?gbbird 4-69

?gebrd 4-66

?orgbr 4-72

?ormbr 4-74

?ungbr 4-77

?unmbr 4-80

ドライバールーチン

1 次方程式の解の算出

?gbsv 3-157

?gbsvx 3-159

?gesv 3-150

?gesvx 3-152

?gtsv 3-164

?gtsvx 3-166

?hesv 3-199

?hesvx 3-201

?hpsv 3-210

?hpsvx 3-212

?pbsv 3-182

?pbsvx 3-183

?posv 3-170

?posvx 3-171

?ppsv 3-176

?ppsvx 3-177

?ptsvx 3-189

?ptsv 3-188

?spsv 3-205

?spsvx 3-207

?sysv 3-192

?sysvx 3-195

線形最小二乗問題

?gels 4-243

?gelsd 4-252

?gelss 4-249

?gelsy 4-246

?lalsd (補助) 5-92

?lals0 (補助) 5-86

?lalsa (補助) 5-89

対称固有値問題

?hbev 4-305

?hbevd 4-309

?hbevz 4-316

?heev 4-264

?heevd 4-268

?heevr 4-282

?heevx 4-274

?hpev 4-289

?hpevd 4-293

?hpevx 4-299

?sbev 4-303

?sbevd 4-307

?sbevz 4-312

?spev 4-287

?spevd 4-291

?spevx 4-296

?stev 4-320

?stevd 4-321

?stevr 4-327

?stevx 4-324

?syev 4-262

?syevd 4-266

?syevr 4-278

?syevx 4-271

特異値分解

?gesdd 4-354

?gesvd 4-350

?ggsvd 4-357

汎用 LLS 問題

?ggglm 4-258

?gglse 4-256

汎用対称固有値問題

?hbgv 4-402

?hbgvd 4-407

?hbgvx 4-413

?hegv 4-366

?hegvd 4-371

?hegvx 4-378

?hpgv 4-385

?hpgvd 4-390

?hpgvx 4-396

?sbgv 4-400

?sbgvd 4-404

?sbgvx 4-410

?spgv 4-382

?spgvd 4-387

?spgvx 4-393

?sygv 4-363

?sygvd 4-368

?sygvx 4-374

汎用非対称固有値問題

?gges 4-418

?ggesx 4-423

?ggev 4-429

?ggevx 4-433

非対称固有値問題

?gees 4-332

?geesx 4-336

?geev 4-341

?geevx 4-344

二重対角分割統治 5-188

二乗和の更新 5-198

汎用固有値問題

?hbgst 4-151

?hegst 4-143

?hpgst 4-147

?pbstf 4-153

?sbgst 4-149

?spgst 4-145

?sygst 4-141

比較的分離された固有値の検索 5-146

非対角成分と対角成分 5-189

非対称固有値問題

?gebak 4-171

?gebal 4-168	?laebz 5-40
?gehrd 4-157	?laed0 5-44
?hsein 4-177	?laed1 5-46
?hseqr 4-173	?laed2 5-48
?orghr 4-159	?laed3 5-50
?ormhr 4-161	?laed4 5-51
?trevc 4-182	?laed5 5-53
?trsen 4-192	?laed6 5-54
?trsna 4-186	?laed7 5-55
?trexc 4-190	?laed8 5-58
?unghr 4-164	?laed9 5-60
?unmhr 4-166	?laeda 5-62
複素	?laein 5-63
線形変換 5-9	?laesy 5-10
複素エルミート行列 5-108	?laev2 5-66
圧縮格納 5-104	?laexc 5-67
複素エルミート三重対角行列 5-105	?lag2 5-68
複素行列の乗算 5-8	?lags2 5-70
複素対称圧縮行列	?lagtf 5-72
対称階数 1 の更新 5-14	?lagtm 5-73
複素対称行列 5-106	?lagts 5-75
行列 - ベクトルの積 5-15	?lagv2 5-76
対称階数 1 の更新 5-17	?lahef 5-206
複素対象行列	?lahqr 5-78
固有値および固有ベクトルの計算 5-10	?lahrd 5-80
複素ベクトル	?laic1 5-82
行列 - ベクトルの積 5-12	?laln2 5-84
最大絶対値を持つ成分のインデックス 5-18	?lals0 5-86
真の絶対値を使用した 1- ノルム 5-19	?lalsa 5-89
面回転 5-11	?lalsd 5-92
複素ベクトルの共役 5-8	?lamrg 5-94
ブロック・リフレクター	?langb 5-95
三角係数 5-137, 5-156	?lange 5-96
平方根 5-116, 5-117, 5-168, 5-170, 5-171, 5-179, 5-181, 5-261	?langt 5-97
ベクトルの線形従属性 5-114	?lanhb 5-101
補助ルーチン	?lanhe 5-108
?gbtf2 5-19	?lanhp 5-104
?gebd2 5-21	?lanhs 5-98
?gehd2 5-22	?lansb 5-100
?gelq2 5-24	?lansp 5-103
?geql2 5-25	?lanst/?lanht 5-105
?geqr2 5-27	?lansy 5-106
?gerq2 5-28	?lantb 5-109
?gesc2 5-29	?lantp 5-110
?getc2 5-30	?lantr 5-112
?getf2 5-32	?lanv2 5-113
?gtts2 5-33	?lapll 5-114
?hetf2 5-248	?lapmt 5-115
?labrd 5-34	?lapy2 5-116
?lacgv 5-8	?lapy3 5-117
?lacon 5-36	?laqgb 5-117
?lacpy 5-37	?laqge 5-119
?lacrm 5-8	?laqp2 5-120
?lacrt 5-9	?laqps 5-122
?ladiv 5-38	?laqsb 5-123
?lae2 5-39	?laqsp 5-125
	?laqsy 5-126
	?laqtr 5-128

?lar1v 5-130
 ?lar2v 5-132
 ?larf 5-133
 ?larfb 5-134
 ?larfg 5-136
 ?larft 5-137
 ?larfx 5-140
 ?largv 5-141
 ?larnv 5-142
 ?larrb 5-143
 ?larre 5-145
 ?larrf 5-146
 ?larrv 5-148
 ?lartg 5-150
 ?lartv 5-151
 ?laruv 5-152
 ?larz 5-153
 ?larzb 5-154
 ?larzt 5-156
 ?las2 5-159
 ?lascl 5-160
 ?lasd0 5-161
 ?lasd1 5-162
 ?lasd2 5-165
 ?lasd3 5-168
 ?lasd4 5-170
 ?lasd5 5-171
 ?lasd6 5-172
 ?lasd7 5-176
 ?lasd8 5-179
 ?lasd9 5-181
 ?lasda 5-182
 ?lasdq 5-186
 ?lasdt 5-188
 ?laset 5-189
 ?lasq1 5-190
 ?lasq2 5-191
 ?lasq3 5-192
 ?lasq4 5-193
 ?lasq5 5-194
 ?lasq6 5-195
 ?lasr 5-196
 ?lasrt 5-198
 ?lassq 5-198
 ?lasv2 5-200
 ?laswp 5-201
 ?lasy2 5-202
 ?lasyf 5-204
 ?latbs 5-208
 ?latdf 5-210
 ?latps 5-212
 ?latrd 5-214
 ?latrs 5-216
 ?latrz 5-219
 ?lauu2 5-221
 ?lauum 5-222
 ?org2l/?ung2l 5-223
 ?org2r/?ung2r 5-225
 ?orgl2l/?ungl2 5-226
 ?orgr2/?ungr2 5-227
 ?orm2l/?unm2l 5-229
 ?orm2r/?unm2r 5-231
 ?orml2/?unml2 5-233
 ?ormr2/?unmr2 5-235
 ?ormr3/?unmr3 5-237
 ?pbt2 5-239
 ?potf2 5-240
 ?ptts2 5-241
 ?rot 5-11
 ?rscl 5-243
 ?spm2 5-12
 ?spr 5-14
 ?sum1 5-19
 ?sygs2/?hegs2 5-244
 ?symv 5-15
 ?syr 5-17
 ?sytd2/?hetd2 5-245
 ?sytf2 5-247
 ?tgex2 5-250
 ?tgsy2 5-252
 ?trti2 5-255
 i?max1 5-18
 マシン・パラメーターの決定 5-262, 5-263
 無限大演算の安全性の確認 5-259
 面回転 5-150, 5-151, 5-196
 面回転ベクトル 5-141
 文字の同一性の確認 5-260
 文字列の同一性の確認 5-260
 ユーティリティ関数とルーチン
 ?labad 5-261
 ?lamc1 5-262
 ?lamc2 5-263
 ?lamc3 5-264
 ?lamc4 5-264
 ?lamc5 5-265
 ?lamch 5-261
 ieeek 5-259
 ilaenv 5-257
 lsame 5-260
 lsamen 5-260
 second/dsecnd 5-266
 xerbla 5-266
 乱数ベクトル 5-142
 Laplace 10-44
 LeapfrogStream 10-26
 LoadStreamF 10-24
 Lognormal 10-52
 LQ 因子分解 4-5
 一般矩形行列 5-24, 7-20
 成分の計算
 実直交行列 Q 6-76
 直交行列 Q 4-24
 ユニタリー行列 Q 4-28, 6-77
 lsame 5-260

lsamen 5-260

LU 因子分解 3-10, 6-5

1 次方程式の解の算出

一般行列 5-29

三重対角行列 5-33, 5-75

正方行列 6-172

一般帯行列 5-19

一般行列 5-32, 7-28

帯行列 3-11, 6-6, 6-8

非ブロック化アルゴリズム 7-149

ブロック化アルゴリズム 7-150

完全ピボット演算 5-30, 5-210

三角帯行列 7-8

三重対角帯行列 7-11

三重対角行列 3-13, 5-72, 7-152

対角優位三重対角行列 6-16

部分的なピボット演算 5-32, 7-28

M

mk1_cvt_to_null_terminated_str 8-23

mk1_dcoogemv 2-131

mk1_dcoomm 2-160

mk1_dcoomv 2-129

mk1_dcoosm 2-171

mk1_dcoosv 2-147

mk1_dcoosymv 2-133

mk1_dcootrsv 2-149

mk1_dcscmm 2-158

mk1_dcscmv 2-127

mk1_dcscsm 2-169

mk1_dcscsv 2-145

mk1_dcsrgemv 2-124

mk1_dcsrmm 2-156

mk1_dcsrmv 2-122

mk1_dcsrsm 2-167

mk1_dcsrsv 2-141

mk1_dcsrsymv 2-126

mk1_dcsrtrsv 2-143

mk1_ddiagemv 2-136

mk1_ddiamm 2-162

mk1_ddiamv 2-134

mk1_ddiasm 2-173

mk1_ddiasv 2-150

mk1_ddiasymv 2-138

mk1_ddiatsrv 2-152

mk1_dskymm 2-165

mk1_dskymv 2-139

mk1_dskysm 2-175

mk1_dskysv 2-154

MPI 6-1

N

Negbinomial 10-73

NewStream 10-12

NewStreamEx 10-13

P

p?dbsv 6-179

p?dbtrf 6-8

p?dbtrs 6-30

p?dbtrsv 7-8

p?dtsv 6-182

p?dttrf 6-16

p?dttrs 6-27

p?dttrsv 7-11

p?gbsv 6-177

p?gbtrf 6-6

p?gbtrs 6-20

p?gebd2 7-14

p?gebrd 6-154

p?gecon 6-34

p?geequ 6-56

p?gehd2 7-17

p?gehrrd 6-143

p?gelq2 7-20

p?gelqf 6-74

p?gels 6-195

p?geql2 7-22

p?geqlf 6-85

p?geqpf 6-62

p?geqr2 7-24

p?geqrf 6-60

p?gerfs 6-41

p?gerq2 7-26

p?gerqf 6-96

p?gesv 6-170

p?gesvd 6-213

p?gesvx 6-172

p?getf2 7-28

p?getrf 6-5

p?getri 6-51

p?getrs 6-18

p?ggqrf 6-114

p?ggrqf 6-118

p?heevx 6-206

p?hegst 6-168

p?hegvx 6-223

p?hetrd 6-129

p?labad 7-158

p?labrd 7-29
p?lacgv 7-5
p?lachkieee 7-159
p?lacon 7-33
p?laconsb 7-35
p?lacp2 7-36
p?lacp3 7-37
p?lacpy 7-39
p?laevswp 7-40
p?lahqr 6-151
p?lahrd 7-42
p?laiect 7-44
p?lamch 7-160
p?lange 7-45
p?lanhs 7-47
p?lansy, p?lanhe 7-49
p?lantr 7-51
p?lapiv 7-53
p?laqge 7-55
p?laqsy 7-57
p?lared1d 7-59
p?lared2d 7-60
p?larf 7-61
p?larfb 7-64
p?larfc 7-67
p?larfg 7-70
p?larft 7-71
p?larz 7-74
p?larzb 7-77
p?larzc 7-80
p?larzt 7-83
p?lascl 7-86
p?laset 7-88
p?lasmsub 7-89
p?lasnbt 7-161
p?lassq 7-90
p?laswp 7-92
p?latra 7-93
p?latrd 7-94
p?latrs 7-98
p?latrz 7-100
p?lauu2 7-102
p?lauum 7-103
p?lawil 7-104
p?max1 7-6
p?org2l/p?ung2l 7-105
p?org2r/p?ung2r 7-107
p?orgl2/p?ungl2 7-109
p?orglq 6-76
p?orgql 6-87
p?orgqr 6-65
p?org2r/p?ungr2 7-111
p?orgrq 6-98
p?orm2l/p?unm2l 7-113
p?orm2r/p?unm2r 7-116
p?ormbr 6-158
p?ormhr 6-146
p?orml2/p?unml2 7-119
p?ormlq 6-79
p?ormql 6-90
p?ormqr 6-68
p?ormr2/p?unmr2 7-122
p?ormrq 6-101
p?ormrz 6-109
p?ormtr 6-126
p?pbsv 6-191
p?pbtrf 6-12
p?pbtrs 6-23
p?pbtrsv 7-126
p?pocon 6-36
p?poequ 6-58
p?porfs 6-44
p?posv 6-184
p?posvx 6-186
p?potf2 7-132
p?potrf 6-10
p?potri 6-53
p?potrs 6-22
p?ptsv 6-193
p?pttrf 6-14
p?pttrs 6-25
p?pttrsv 7-129
p?rscl 7-133
p?stebz 6-135
p?stein 6-139
p?sum1 7-7
p?syev 6-198
p?syevx 6-201
p?sygs2/p?hegs2 7-134
p?sygst 6-166
p?sygvx 6-216
p?sytd2/p?hetd2 7-137
p?sytrd 6-123
p?trcon 6-39
p?trrfs 6-47
p?trti2 7-140
p?trtri 6-54
p?trtrs 6-32

p?tzrzf 6-107
 p?unglq 6-77
 p?ungql 6-88
 p?ungqr 6-66
 p?ungrq 6-99
 p?unmbr 6-161
 p?unmhr 6-149
 p?unmlq 6-82
 p?unmql 6-93
 p?unmqr 6-71
 p?unmrq 6-104
 p?unmrz 6-112
 p?unmtr 6-132
 PARDISO 8-1
 pardiso 関数 8-2
 pdlaiectb 7-44
 pdlaiectl 7-44
 Poisson 10-70
 PoissonV 10-71
 pslaiect 7-44
 pxerbla 7-161

Q

QL 因子分解
 一般行列との掛け合わせ
 直交行列 Q 6-90
 ユニタリー行列 Q 6-93
 一般矩形行列
 ScaLAPACK 7-22
 一般正方行列
 LAPACK 5-25
 成分の計算
 実行列 Q 4-34
 直交行列 Q 6-87
 複素行列 Q 4-36
 ユニタリー行列 Q 6-88
 QR 因子分解 4-5
 一般矩形行列
 ScaLAPACK 7-24, 7-26
 一般正方行列
 LAPACK 5-27, 5-28
 成分の計算
 直交行列 Q 4-13, 6-65
 ユニタリー行列 Q 4-18, 6-66
 ピボット演算 4-9, 4-11, 5-120, 5-122
 ScaLAPACK 6-62
 QZ 法 4-207

R

Rayleigh 10-50
 RCI CG 8-27

RCI CG スパース・ソルバー・ルーチン
 dcg 8-40
 dcg_check 8-39
 dcg_get 8-42
 dcg_init 8-38
 RCI CG ルーチン 8-38
 RCI FGMRES 8-27
 RCI FGMRES ルーチン 8-42
 RCI Flexible Generalized Minimal RESidual Solver 8-27
 RCI GFMRES スパース・ソルバー・ルーチン
 dgmres 8-45
 dgmres_check 8-44
 dgmres_get 8-47
 dgmres_init 8-42
 RCI ISS インターフェイス 8-27
 RCI 共役勾配ソルバー 8-27
 RegisterBrng 10-76
 Rex-Rohn テスト 12-18, 12-19
 Ris-Beeck スペクトル条件 12-18
 RQ 因子分解
 成分の計算
 実行列 Q 4-44
 直交行列 Q 6-98
 複素行列 Q 4-45
 ユニタリー行列 Q 6-99
 Rump 判断 12-19

S

SaveStreamF 10-23
 ScaLAPACK 6-1
 ScaLAPACK ルーチン
 1 次元配列の再分配 7-59, 7-60
 1 次方程式の解の算出
 ?dtrsv 7-153
 ?pttrsv 7-154
 p?dbtrs 6-30
 p?dtrrs 6-27
 p?gbtrs 6-20
 p?getrs 6-18
 p?potrs 6-22
 p?pttrrs 6-25
 p?trtrrs 6-32
 Householder 行列
 基本リフレクター 7-70
 LQ 因子分解
 p?gelq2 7-20
 p?gelqf 6-74
 p?orglq 6-76
 p?ormlq 6-79
 p?unglq 6-77
 p?unmlq 6-82
 LU 因子分解
 p?getf2 7-28
 p?dbtrsv 7-8

p?dtrf 6-16
 p?dtrsv 7-11
 QL 因子分解
 ?geqlf 6-85
 ?ungql 6-88
 p?geql2 7-22
 p?orgql 6-87
 p?ormql 6-90
 p?unmql 6-93
 QR 因子分解
 p?geqpf 6-62
 p?geqr2 7-24
 p?ggqrf 6-114
 p?orgqr 6-65
 p?ormqr 6-68
 p?ungqr 6-66
 p?unmqr 6-71
 RQ 因子分解
 p?gerq2 7-26
 p?gerqf 6-96
 p?ggrqf 6-118
 p?orgrq 6-98
 p?ormrq 6-101
 p?ungrq 6-99
 p?unmrq 6-104
 RZ 因子分解
 p?ormrz 6-109
 p?tzzf 6-107
 p?unmrz 6-112
 一般行列
 LU 因子分解 7-28
 上 Hessenberg 形式への縮退 7-17
 基本リフレクター 7-74
 ブロック・リフレクター 7-77
 一般矩形行列 7-86
 LQ 因子分解 7-20
 QL 因子分解 7-22
 QR 因子分解 7-24
 RQ 因子分解 7-26
 基本リフレクター 7-61
 行交換 7-92
 実二重対角形式への縮退 7-14
 二重対角形式への縮退 7-29
 エラー制御
 pxerbla 7-161
 解の精度の改善と誤差の推定
 p?gerfs 6-41
 p?porfs 6-44
 行列の反転
 p?getri 6-51
 p?potri 6-53
 p?trtri 6-54
 行列の平衡化
 p?geequ 6-56
 p?poequ 6-58
 誤差推定
 p?trrfs 6-47
 コレスキー因子分解 6-14

三角分解
 ?dtrf 7-152
 ?dbtrf 7-150
 p?dbtrsv 7-8
 p?dtrsv 7-11
 p?gbtrf 6-6
 p?getrf 6-5
 p?pbtrf 6-12
 p?potrf 6-10
 p?ptrf 6-14
 三角連立方程式 7-98
 条件数推定
 p?gecon 6-34
 p?pocon 6-36
 p?trcon 6-39
 台形行列 7-100
 対称固有値問題
 ?stein2 7-147
 ?steqr2 7-155
 p?hetrd 6-129
 p?ormtr 6-126
 p?stebz 6-135
 p?stein 6-139
 p?sytrd 6-123
 p?unmtr 6-132
 特異値分解
 p?gebrd 6-154
 p?ormbr 6-158
 p?unmbr 6-161
 ドライバルーチン
 p?dbsv 6-179
 p?dtsv 6-182
 p?gbsv 6-177
 p?gels 6-195
 p?gesv 6-170
 p?gesvd 6-213
 p?gesvx 6-172
 p?heevx 6-206
 p?hegvx 6-223
 p?pbsv 6-191
 p?posv 6-184
 p?posvx 6-186
 p?ptsv 6-193
 p?syeve 6-198
 p?syevx 6-201
 p?sygvx 6-216
 二乗和の更新 7-90
 汎用固有値問題
 p?hegst 6-168
 p?sygst 6-166
 非対称固有値問題
 p?gehrd 6-143
 p?lahqr 6-151
 p?ormhr 6-146
 p?unmhr 6-149
 複素行列
 複素基本リフレクター 7-80
 複素ベクトル

- 真の絶対値を使用した 1- ノルム 7-7
 - 複素ベクトルの共役 7-5
 - ブロック・リフレクター
 - 三角係数 7-71, 7-83
 - 補助ルーチン
 - ?combamax1 7-7
 - ?dbtrf 7-150
 - ?dbtf2 7-149
 - ?dtrf 7-152
 - ?dtrsv 7-153
 - ?lamsh 7-141
 - ?laref 7-143
 - ?lasorte 7-145
 - ?lasrt 7-146
 - ?pttrsv 7-154
 - ?stein2 7-147
 - ?steqr2 7-155
 - p?dbtrsv 7-8
 - p?gebd2 7-14
 - p?gehd2 7-17
 - p?gelq2 7-20
 - p?geql2 7-22
 - p?geqr2 7-24
 - p?gerq2 7-26
 - p?getf2 7-28
 - p?labrd 7-29
 - p?lacgv 7-5
 - p?lacon 7-33
 - p?laconsb 7-35
 - p?lacp2 7-36
 - p?lacp3 7-37
 - p?lacpy 7-39
 - p?laevswp 7-40
 - p?lahrd 7-42
 - p?laiect 7-44
 - p?lange 7-45
 - p?lanhs 7-47
 - p?lansy、 p?lanhe 7-49
 - p?lantr 7-51
 - p?lapiv 7-53
 - p?laqge 7-55
 - p?laqsy 7-57
 - p?lared1d 7-59
 - p?lared2d 7-60
 - p?larf 7-61
 - p?larfb 7-64
 - p?larfc 7-67
 - p?larfg 7-70
 - p?larft 7-71
 - p?larz 7-74
 - p?larzb 7-77
 - p?larzc 7-80
 - p?larzt 7-83
 - p?lascl 7-86
 - p?lasel 7-88
 - p?lasmsub 7-89
 - p?lassq 7-90
 - p?laswp 7-92
 - p?latra 7-93
 - p?latrd 7-94
 - p?latrs 7-98
 - p?latrz 7-100
 - p?lauu2 7-102
 - p?lauum 7-103
 - p?lawil 7-104
 - p?max1 7-6
 - p?org2l/p?ung2l 7-105
 - p?org2r/p?ung2r 7-107
 - p?orgl2/p?ungl2 7-109
 - p?orgr2/p?ungr2 7-111
 - p?orm2l/p?unm2l 7-113
 - p?orm2r/p?unm2r 7-116
 - p?orml2/p?unml2 7-119
 - p?ormr2/p?unmr2 7-122
 - p?pbtrsv 7-126
 - p?potf2 7-132
 - p?pttrsv 7-129
 - p?rscl 7-133
 - p?sum1 7-7
 - p?sygs2/p?hegs2 7-134
 - p?sytd2/p?hetd2 7-137
 - p?trti2 7-140
 - pdlaiectb 7-44
 - pdlaiectl 7-44
 - pslaiect 7-44
 - ユーティリティ関数とルーチン
 - p?labad 7-158
 - p?lachkieee 7-159
 - p?lamch 7-160
 - p?lasnbt 7-161
 - p?xerbla 7-161
 - Schulz 区間プロシージャー 12-17
 - Schur 因子分解
 - 2 × 2 実数行列 5-76
 - 2 × 2 非対称行列 5-113
 - 一般行列 4-190
 - 上 Hessenberg 行列 5-78
 - Schur 特異値分解 4-217, 4-220
 - SetValue 11-15
 - SetValueDM 11-51
 - SkipAheadStream 10-28
 - sNewAbstractStream 10-18
 - SVD (特異値分解)
 - LAPACK 4-64
 - ScaLAPACK 6-154
- T**
- TT インターフェイス 13-1, 13-19
 - 三角変換インターフェイスを参照
 - TT ルーチン 13-5
 - 三角変換インターフェイスを参照

U

Uniform (離散) 10-60
Uniform (連続) 10-34
UniformBits 10-62

V

VML 9-1

VML 関数

pack/unpack 関数

Pack 9-48

Unpack 9-50

サービス関数

ClearErrorCallBack 9-61

ClearErrStatus 9-58

GetErrorCallBack 9-60

GetErrStatus 9-57

GetMode 9-55

SetErrorCallBack 9-58

SetErrStatus 9-56

SetMode 9-53

数学関数

Acos 9-26

Acosh 9-34

Asin 9-27

Asinh 9-35

Atan 9-28

Atan2 9-29

Atanh 9-36

Cbrt 9-12

Ceil 9-42

Cos 9-22

Cosh 9-30

Div 9-9

Erf 9-38

Erfc 9-39

ErfInv 9-40

Exp 9-18

Floor 9-41

Inv 9-8

InvCbrt 9-13

InvSqrt 9-11

Ln 9-20

Log10 9-21

Modf 9-46

NearbyInt 9-44

Pow 9-14

Powx 9-15, 9-17

Rint 9-45

Round 9-43

Sin 9-23

SinCos 9-24

Sinh 9-32

Sqrt 9-10

Tan 9-25

Tanh 9-33

Trunc 9-42

VSL ルーチン

生成器サブルーチン

Gumbel 10-54

アドバンスド・サービス・サブルーチン

GetBrngProperties 10-77

RegisterBrng 10-76

サービス・サブルーチン

CopyStream 10-21

CopyStreamState 10-22

DeleteStream 10-20

dNewAbstractStream 10-16

GetNumRegBrngs 10-32

GetStreamStateBrng 10-31

iNewAbstractStream 10-14

LeapfrogStream 10-26

LoadStreamF 10-24

NewStream 10-12

NewStreamEx 10-13

SaveStreamF 10-23

SkipAheadStream 10-28

sNewAbstractStream 10-18

生成器サブルーチン

Bernoulli 10-63

Beta 10-58

Binomial 10-66

Cauchy 10-48

Exponential 10-42

Gamma 10-56

Geometric 10-65

Gaussian 10-36

GaussianMV 10-38

Hypergeometric 10-68

Laplace 10-44

Lognormal 10-52

Negbinomial 10-73

Poisson 10-70

PoissonV 10-71

Rayleigh 10-50

Uniform (離散) 10-60

Uniform (連続) 10-34

UniformBits 10-62

Weibull 10-46

畳み込み / 相関

CopyTask 10-110

DeleteTask 10-109

Exec 10-102

Exec1D 10-104

ExecX 10-106

ExecX1D 10-107

NewTask 10-88

NewTask1D 10-90

NewTaskX 10-91

NewTaskX1D 10-94

SetInternalDecimation 10-100

SetInternalPrecision 10-98

SetMode 10-97

SetStart 10-99

W

Weibull 10-46

Wilkinson 変換 7-104

X

xerbla

エラー報告ルーチン 2-1, 5-266, 9-5

あ

圧縮格納体系 B-3

圧縮形式 11-23

圧縮形式の三角行列

1- ノルムの値 5-110

Frobenius ノルムの値 5-110

最大絶対値の成分 5-110

無限ノルムの値 5-110

圧縮形式のスパースベクトル 2-103

圧縮形式のスパースベクトルの成分のフル圧縮形式への分散 2-113

圧縮形式のスパースベクトルのフル格納形式への変換 2-113

圧縮形式の対称行列

1- ノルムの値 5-103

Frobenius ノルムの値 5-103

最大絶対値の成分 5-103

無限ノルムの値 5-103

圧縮形式の複素エルミート行列

1- ノルムの値 5-104

Frobenius ノルムの値 5-104

最大絶対値の成分 5-104

無限ノルムの値 5-104

い

一般行列

LQ 因子分解 4-22, 6-74

LU 因子分解 3-10, 5-32, 6-5, 7-28

帯格納 3-11, 5-19, 6-6, 6-8, 7-149, 7-150

QL 因子分解

LAPACK 4-32

ScaLAPACK 6-85

QR 因子分解 4-6, 4-61, 6-60

ピボット演算 4-9, 4-11, 6-62

RQ 因子分解

LAPACK 4-42

ScaLAPACK 6-118

上 Hessenberg 形式への縮退 7-17

階数 1 の更新 2-32

階数 1 の更新、共役 2-34

階数 1 の更新、非共役 2-35

基本リフレクター 5-153, 7-74

行列の逆転

LAPACK 3-124

ScaLAPACK 6-51

行列 - ベクトルの積 2-30

帯格納 2-27

固有値問題 4-155, 4-199, 6-143

条件数の推定 3-61, 6-34, 6-36, 6-39

帯格納 3-63

スカラー - 行列 - 行列の積 2-78

直交行列による乗算

LQ 因子分解から 5-233, 7-119

QR 因子分解から 5-231, 7-116

RQ 因子分解から 5-235, 7-122

RZ 因子分解から 5-237

二重対角形式への縮退 5-21, 5-34, 7-14

ブロック・リフレクター 5-154, 7-77

ユニタリー行列による乗算

LQ 因子分解から 5-233, 7-119

QR 因子分解から 5-231, 7-116

RQ 因子分解から 5-235, 7-122

RZ 因子分解 5-237

連立 1 次方程式の解の算出 3-32, 6-18

帯格納

LAPACK 3-34

ScaLAPACK 6-20

一般矩形行列

1- ノルムの値

ScaLAPACK 7-45

Frobenius ノルムの値

ScaLAPACK 7-45

LQ 因子分解

LAPACK 5-24

ScaLAPACK 7-20

QL 因子分解

ScaLAPACK 7-22

QR 因子分解

ScaLAPACK 7-24

RQ 因子分解

ScaLAPACK 6-96, 7-26

基本リフレクター

LAPACK 5-133, 7-67

ScaLAPACK 7-61

行交換

ScaLAPACK 7-92

最初の列への縮退

ScaLAPACK 7-42

最大絶対値の成分

ScaLAPACK 7-45

乗算

ScaLAPACK 7-86

スケールリング 7-55

二重対角形式への縮退 7-29

ブロック・リフレクター

ScaLAPACK 7-64

無限ノルムの値

ScaLAPACK 7-45

一般矩形分散行列
スケール係数の計算 6-56
平衡化 6-56
一般三角行列
LU 因子分解
帯格納 7-8
一般三重対角行列
1- ノルムの値 5-97
Frobenius ノルムの値 5-97
LU 因子分解
帯格納 7-11
最大絶対値の成分 5-97
無限ノルムの値 5-97
一般正方行列
1- ノルムの値
LAPACK 5-96
Frobenius ノルムの値
LAPACK 5-96
QL 因子分解
LAPACK 5-25
QR 因子分解
LAPACK 5-27
RQ 因子分解
LAPACK 5-28
上 Hessenberg 形式への縮退 5-22
基本リフレクター 5-140
行変換
LAPACK 5-201
最大絶対値の成分
LAPACK 5-96
乗算
LAPACK 5-160
先頭列の縮退
LAPACK 5-80
対角和 7-93
ブロック・リフレクター
LAPACK 5-134
無限ノルムの値
LAPACK 5-96
因子分解
Bunch-Kaufman
LAPACK 3-9
ScaLAPACK 6-5
LU
LAPACK 3-9
ScaLAPACK 6-5
上台形行列 5-219
コレスキー 6-5
LAPACK 3-9, 5-239, 5-240
ScaLAPACK 7-132
三角分解を参照
対角ピボット
エルミート行列 3-201
圧縮 3-212
対称行列 3-195
不定値 5-247
エルミート行列

複素数 5-248
直交
LAPACK 4-6
ScaLAPACK 6-60
部分
実数 / 複素対称行列 5-204
複素エルミート無限行列 5-206
対角ピボット
対称行列
圧縮 3-207

う

上 Hessenberg 行列 4-155, 4-199
1- ノルムの値
LAPACK 5-98
ScaLAPACK 7-47
Frobenius ノルムの値
LAPACK 5-98
ScaLAPACK 7-47
ScaLAPACK 6-143
最大絶対値の成分
LAPACK 5-98
ScaLAPACK 7-47
無限ノルムの値
LAPACK 5-98
ScaLAPACK 7-47
上 Hessenberg 形式への縮退
一般行列 7-17
一般正方行列 5-22

え

エラー診断
VML 9-5
エラー制御
pxerbla 7-161
xerbla 2-1, 5-266, 9-5
エルミート帯行列
1- ノルムの値 5-101
Frobenius ノルムの値 5-101
最大絶対値の成分 5-101
無限ノルムの値 5-101
エルミート行列 4-89, 4-141
Bunch-Kaufman 因子分解 3-24
圧縮格納 3-29
階数 1 の更新 2-41
圧縮格納 2-47
階数 2 の更新 2-43
圧縮格納 2-49
階数 2k の更新 2-86
階数 n の更新 2-84
行列の逆転 3-131
圧縮格納 3-134
行列 - ベクトルの積 2-39
帯格納 2-37
固有値と固有ベクトル 6-206

- 三重対角形式への縮退
 - LAPACK 5-214, 5-245
 - ScaLAPACK 7-94, 7-137
 - 条件数の推定 3-76
 - 圧縮格納 3-80
 - スカラー - 行列 - 行列の積 2-81
 - スケールリング 7-57
 - 標準形式への縮退
 - LAPACK 5-244
 - ScaLAPACK 7-134
 - 連立 1 次方程式の解の算出 3-48
 - 圧縮格納 3-52
 - 行列 - ベクトルの積
 - 圧縮格納 2-45
 - エルミート正定値帯行列
 - コレスキー因子分解 5-239
 - エルミート正定値行列
 - 行列の逆転 3-126
 - 圧縮格納 3-127
 - コレスキー因子分解 3-15, 5-240, 6-10, 7-132
 - 圧縮格納 3-17
 - 帯格納 3-18, 6-12
 - 条件数の推定 3-67
 - 圧縮格納 3-69
 - 帯格納 3-71
 - 連立 1 次方程式の解の算出 3-38, 6-22
 - 圧縮格納 3-40
 - 帯格納 3-42, 6-23
 - エルミート正定値三重対角行列
 - 連立 1 次方程式の解の算出 6-25
 - エルミート正定値分散行列
 - 行列の逆転 6-53
 - スケール係数の計算 6-58
 - 平衡化 6-58
 - エルミート行列
 - 汎用固有値問題 4-141
- お**
- 帯格納体系 B-3
- か**
- 階数 1 の更新
 - 一般行列 2-32
 - エルミート行列 2-41
 - 圧縮格納 2-47
 - 共役、一般行列 2-34
 - 実対称行列 2-61
 - 圧縮格納 2-55
 - 非共役、一般行列 2-35
 - 複素対称行列 5-17
 - 圧縮格納 5-14
 - 階数 2 の更新
 - エルミート行列 2-43
 - 圧縮格納 2-49
 - 対称行列 2-62
 - 圧縮格納 2-57
 - 階数 2k の更新
 - エルミート行列 2-86
 - 対称行列 2-94
 - 階数 n の更新
 - エルミート行列 2-84
 - 対称行列 2-92
 - 回転
 - Givens 回転に対するパラメーター 2-16
 - スパースベクトル 2-112
 - 変形 Givens 変換のパラメーター 2-19
 - 変形面における点 2-17
 - 面における点 2-14
 - 解分割、区間方程式 12-13
 - ガウス法、区間方程式 12-4, 12-22
 - 格納、スパース行列 A-6
 - 簡易ドライバー 6-3
 - 関数命名規則
 - VML 9-2
- き**
- 擬似乱数 10-1
 - 基本準乱数生成器
 - Niederreiter 10-7
 - Sobol 10-7
 - 基本生成器の登録 10-75
 - 基本乱数生成器 10-1, 10-7
 - GFSR 10-7
 - MCG, 32 ビット 10-7
 - MCG, 59 ビット 10-7
 - Mersenne Twister
 - MT19937 10-7
 - MT2203 10-7
 - MRG 10-7
 - Wichmann-Hill 10-7
 - 基本リフレクター
 - LAPACK 生成 5-136
 - ScaLAPACK 生成 7-70
 - 一般行列 5-153, 7-74
 - 一般矩形行列
 - ScaLAPACK 7-61, 7-67
 - 一般正方行列
 - LAPACK 5-133, 5-140
 - 複素基本リフレクター
 - ScaLAPACK 7-80
 - 複素行列 7-80
 - 逆行列。行列の逆転を参照
 - 共役勾配ソルバー 8-27
 - 共役ベクトル 5-8, 7-5
 - 行列 - 行列演算
 - 階数 2k の更新
 - エルミート行列 2-86
 - 対称行列 2-94

階数 n の更新
エルミート行列 2-84
対称行列 2-92
スカラー - 行列 - 行列の積
エルミート行列 2-81
三角行列 2-97
対称行列 2-89
積
一般行列 2-78
行列の 1 次元下部構造 B-2
行列の逆転
一般行列
LAPACK 3-124
ScaLAPACK 6-51
エルミート行列 3-131
圧縮格納 3-134
エルミート正定値行列
LAPACK 3-126
ScaLAPACK 6-53
圧縮格納 3-127
三角行列 3-136
圧縮格納 3-137
三角分散行列 6-54
対称行列 3-129
圧縮格納 3-132
対称正定値行列
LAPACK 3-126
ScaLAPACK 6-53
圧縮格納 3-127
行列の行または列のピボット演算 7-53
行列の順序変更 A-3
行列の引数 B-3
行数 B-6
転置パラメーター B-6
リーディング・ディメンジョン B-6
例 B-6
列主体の順序 B-2, B-6
列数 B-6
行列の平衡化 4-168, 4-203
行列の方程式
AX = B 2-100, 3-8, 3-32, 6-4, 6-18
行列ブロック
QR 因子分解
ピボット演算 5-120
行列 - ベクトル演算
階数 1 の更新 2-32, 2-34, 2-35
エルミート行列 2-41
圧縮格納 2-47
実対称行列 2-61
圧縮格納 2-55
複素対称行列 5-17
階数 2 の更新
エルミート行列 2-43
圧縮格納 2-49
対称行列 2-62
圧縮格納 2-57

積 2-27, 2-30
エルミート行列 2-39
圧縮格納 2-45
帯格納 2-37
三角行列 2-73
圧縮格納 2-69
帯格納 2-64
実対称行列 2-59
圧縮格納 2-53
対称行列
帯格納 2-51
複素対称行列 5-15
圧縮格納 5-12
階数 1 の更新
複素対称行列
圧縮格納 5-14

<

区間ソルバールーチン

?gegas 12-4
?gegss 12-9
?gehbs 12-10
?gehss 12-6
?gekws 12-7
?gemip 12-21
?gepps 12-12
?gepss 12-13
?gerbr 12-18
?gesvr 12-19
?geszi 12-17
?trtri 12-16
?trtrs 12-3

グローバル配列 6-1

け

計算ルーチン 4-5

検索

最小絶対値を持つベクトルの成分 2-24
最大絶対値を持つベクトルの成分 2-23
実数部分が最大絶対値を持つ成分とそのグローバル・インデックス 7-7
実数部分が最大絶対値を持つベクトルの成分のインデックス 7-6

こ

合計

スパースベクトルとフル格納ベクトル 2-105
ベクトル 2-6
ベクトル成分の大きさ 2-5

更新

階数 1
一般行列 2-32
エルミート行列 2-41
圧縮格納 2-47

実対称行列 2-61
 圧縮格納 2-55
 複素対称行列 5-17
 圧縮格納 5-14
 階数 1、共役
 一般行列 2-34
 階数 1、非共役
 一般行列 2-35
 階数 2
 エルミート行列 2-43
 圧縮格納 2-49
 対称行列 2-62
 圧縮格納 2-57
 階数 2k
 エルミート行列 2-86
 対称行列 2-94
 階数 n
 エルミート行列 2-84
 対称行列 2-92
 構成パラメータ、DFTI 11-2
 高度ドライバー 6-3
 コード例
 BLAS レベル 1 の関数 C-1
 BLAS レベル 1 のルーチン C-2
 BLAS レベル 2 のルーチン C-2
 BLAS レベル 3 のルーチン C-3
 コピー
 行列
 2次元
 LAPACK 5-37
 ScaLAPACK 7-39
 グローバル並列 7-37
 分散 7-36
 ローカル複製 7-37
 ベクトル 2-7
 コミュニケーション・サブプログラム 6-1
 固有値問題 4-1
 一般行列 4-155, 4-199, 6-143
 エルミート行列 4-89
 対称行列 4-89
 対称三重対角行列 7-147, 7-155
 汎用形式 4-141
 固有値。固有値問題を参照
 固有ペア、ソート 7-145
 固有ベクトル。固有値問題を参照
 コレスキー因子分解
 エルミート正定値行列 3-15, 3-171, 6-10, 6-186
 圧縮格納 3-17, 3-177
 帯格納 3-18, 3-42, 3-183, 6-12, 6-23
 対称正定値行列 3-15, 3-171, 6-10, 6-186
 圧縮格納 3-17, 3-177
 帯格納 3-18, 3-42, 3-183, 6-12, 6-23
 分割 4-153

さ

最小二乗問題 4-1
 最大絶対値の成分
 一般矩形行列 5-96, 7-45
 一般三重対角行列 5-97
 上 Hessenberg 行列 5-98, 7-47
 エルミート帯行列 5-101
 三角帯行列 5-109
 三角行列 5-112, 7-51
 圧縮格納 5-110
 実対称行列 5-106, 7-49
 実対称三重対角行列 5-105
 台形行列 5-112
 対称帯行列 5-100
 対称行列
 圧縮格納 5-103
 複素エルミート行列 5-108, 7-49
 圧縮格納 5-104
 複素エルミート三重対角行列 5-105
 複素対称行列 5-106
 三角帯行列
 1- ノルムの値 5-109
 Frobenius ノルムの値 5-109
 最大絶対値の成分 5-109
 無限ノルムの値 5-109
 三角帯方程式
 ScaLAPACK 7-126
 三角帯連立方程式
 LAPACK 5-208
 三角行列 4-155, 4-199
 1- ノルムの値
 LAPACK 5-112
 ScaLAPACK 7-51
 Frobenius ノルムの値
 LAPACK 5-112
 ScaLAPACK 7-51
 ScaLAPACK 6-143
 行列の逆転 3-136
 LAPACK 5-255
 ScaLAPACK 7-140
 圧縮格納 3-137
 行列 - ベクトルの積 2-73
 圧縮格納 2-69
 帯格納 2-64
 最大絶対値の成分
 LAPACK 5-112
 ScaLAPACK 7-51
 条件数の推定 3-81
 圧縮格納 3-83
 帯格納 3-85
 スカラー - 行列 - 行列の積 2-97
 積
 LAPACK 5-221, 5-222
 ScaLAPACK 7-102, 7-103
 非ブロック化アルゴリズム 5-221
 ブロック化アルゴリズム 5-222, 7-103

無限ノルムの値
LAPACK 5-112
ScaLAPACK 7-51
隣接対角ブロックの交換 5-250
連立 1 次方程式の解の算出 2-75, 3-54
ScaLAPACK 6-32
圧縮格納 2-71, 3-56
帯格納 2-67, 3-58

三角分解
一般行列 3-10, 6-5
エルミート行列 3-24
圧縮格納 3-29
エルミート正定値行列 3-15, 6-10
圧縮格納 3-17
帯格納 3-18, 6-12
三重対角行列 3-20, 6-14
帯行列 3-11, 6-6, 6-8, 7-8, 7-11, 7-150
三重対角行列
LAPACK 3-13
ScaLAPACK 7-152
対称行列 3-21
圧縮格納 3-27
対称正定値行列 3-15, 6-10
圧縮格納 3-17
帯格納 3-18, 6-12
三重対角行列 3-20, 6-14

三角分散行列
行列の逆転 6-54

三角変換
逆方向スタaggerド余弦 13-2
逆方向正弦 13-2
逆方向余弦 13-2
順方向スタaggerド余弦 13-2
順方向正弦 13-2
順方向余弦 13-2

三角変換インターフェイス 13-1
コード例 C-65
ルーチン 13-3, 13-23
?_backward_trig_transform 13-9
?_commit_trig_transform 13-6
?_forward_trig_transform 13-8
?_init_trig_transform 13-5
free_trig_transform 13-11

三角連立方程式 7-129
スケール係数
LAPACK 5-216
スケール係数で解く
ScaLAPACK 7-98

三重対角行列 4-89
条件数の推定 3-65
連立 1 次方程式の解の算出 3-36, 3-44
ScaLAPACK 7-153

三重連立方程式 5-241

し

次元 B-1
字体規則 1-8
実行列
QR 因子分解
ピボット演算 5-122
実数演算での複素除算 5-38
実数準三角連立方程式 5-128
実対称行列
1- ノルムの値 5-106
Frobenius ノルムの値 5-106
最大絶対値の成分 5-106
無限ノルムの値 5-106
実対称三重対角行列
1- ノルムの値 5-105
Frobenius ノルムの値 5-105
最大絶対値の成分 5-105
無限ノルムの値 5-105
準三角行列
LAPACK 4-155, 4-199
ScaLAPACK 6-143
準乱数 10-1
条件数
エルミート行列 3-76
圧縮格納 3-80
エルミート正定値行列 3-67
圧縮格納 3-69
帯格納 3-71
三重対角 3-73
帯行列 3-63
三角行列 3-81
圧縮格納 3-83
帯格納 3-85
三重対角行列 3-65
対称行列 3-74, 4-139
圧縮格納 3-78
対称正定値行列 3-67
圧縮格納 3-69
帯格納 3-71
三重対角 3-73
条件数推定
一般行列
LAPACK 3-61
ScaLAPACK 6-34, 6-36, 6-39
乗算合同生成器 10-3
処理ノード 10-2
シルベスター方程式 4-196, 4-225

す

スカラー - 行列 - 行列の積 2-81
一般行列 2-78
三角行列 2-97
対称行列 2-89

- スカラー - 行列の積 2-78, 2-81, 2-89
- スケーリング
 - 一般矩形行列 7-55
 - 対称 / エルミート行列 7-57
- スケール係数
 - 一般矩形分散行列 6-56
 - エルミート正定値分散行列 6-58
 - 対称正定値分散行列 6-58
- ステータス確認
 - DFTI 11-3
- ストライド。増分を参照
- ストリーム 10-8
- ストリーム・ディスクリプター 10-2
- スパース BLAS レベル 1 2-103
 - データ型 2-104
 - 命名規則 2-103
- スパース BLAS レベル 1 のルーチンと関数 2-104
 - ?axpyi 2-105
 - ?dotci 2-107
 - ?doti 2-106
 - ?dotui 2-108
 - ?gthr 2-110
 - ?gthrz 2-111
 - ?roti 2-112
 - ?sctr 2-113
- スパース BLAS レベル 2 2-115
 - 命名規則 2-115
- スパース BLAS レベル 2 のルーチン
 - mkl_dcoogemv 2-131
 - mkl_dcoomv 2-129
 - mkl_dcoosv 2-147
 - mkl_dcoosymv 2-133
 - mkl_dcootrsv 2-149
 - mkl_dcscmv 2-127
 - mkl_dcscsv 2-145
 - mkl_dcsgemv 2-124
 - mkl_dcsrcmv 2-122
 - mkl_dcsrcsv 2-141
 - mkl_dcsrcsymv 2-126
 - mkl_dcsrcsv 2-143
 - mkl_ddiagemv 2-136
 - mkl_ddiamv 2-134
 - mkl_ddiasv 2-150
 - mkl_ddiasymv 2-138
 - mkl_ddiatrsv 2-152
 - mkl_dskymv 2-139
 - mkl_dskysv 2-154
- スパース BLAS レベル 3 2-115
 - 命名規則 2-115
- スパース BLAS レベル 3 のルーチン
 - mkl_dcoomm 2-160
 - mkl_dcoosm 2-171
 - mkl_dcscmm 2-158
 - mkl_dcscsm 2-169
 - mkl_dcsrcmm 2-156
 - mkl_dcsrcsm 2-167
 - mkl_ddiamm 2-162
 - mkl_ddiasm 2-173
 - mkl_dskymm 2-165
 - mkl_dskysm 2-175
- スパース行列 2-115
- スパースソルバー
 - 直接法スパース・ソルバー・インターフェイス
 - dss_create 8-16
 - dss_define_structure 8-16
 - dss_delete 8-20
 - dss_factor_real, dss_factor_complex 8-18
 - dss_reorder 8-18
 - dss_solve_real, dss_solve_complex 8-19
 - dss_statistics 8-21
 - mkl_cvt_to_null_terminated_str 8-23
 - 反復法スパース・ソルバー・インターフェイス
 - dcg_check 8-39
 - dcg_init 8-38
 - dfgmres 8-45
 - dfgmres_check 8-44
 - dfgmres_get 8-47
 - dfgmres_init 8-42
 - dcg_get 8-42
 - dcg 8-40
- スパースベクトル 2-103
 - BLAS レベル 1 のルーチンへの引渡し 2-104
 - Givens 回転 2-112
 - norm 2-104
 - scaling 2-104
 - 圧縮形式 2-103
 - 圧縮形式への変換 2-110, 2-111
 - 実ドット積 2-106
 - 追加とスケーリング 2-105
 - 複素ドット積、共役 2-107
 - 複素ドット積、非共役 2-108
 - フル格納形式 2-103
 - フル格納形式への変換 2-113
- スパースベクトルの圧縮格納形式への変換 2-110
 - 元のベクトルへのゼロの書き込み 2-111
- スパースベクトルの成分の圧縮形式への集積 2-110
 - これらの成分へのゼロの書き込み 2-111

せ

生成方法 10-2

正方行列

- 1- ノルムの推定
 - LAPACK 5-36
 - ScaLAPACK 7-33

精密モード
VML 9-1

積

- 行列 - ベクトル
 - 一般行列 2-30
 - 帯格納 2-27

エルミート行列 2-39
 圧縮格納 2-45
 帯格納 2-37
三角行列 2-73
 圧縮格納 2-69
 帯格納 2-64
実対称行列 2-59
 圧縮格納 2-53
対称行列
 帯格納 2-51
複素対称行列 5-15
 圧縮格納 5-12
スカラー - 行列
 一般行列 2-78
 エルミート行列 2-81
スカラー - 行列 - 行列
 一般行列 2-78
 エルミート行列 2-81
 三角行列 2-97
 対称行列 2-89
ドット積を参照
ベクトル - スカラー 2-20
線形合同生成器 10-3

そ

相関関数

?CorrExec 10-102
?CorrExec1D 10-104
?CorrExecX 10-106
?CorrExecX1D 10-107
?CorrNewTask 10-88
?CorrNewTask1D 10-90
?CorrNewTaskX 10-91
?CorrNewTaskX1D 10-94
?CorrNewTaskX1D 10-94
CorrSetDecimation 10-100
CorrSetInternalPrecision 10-98
CorrSetMode 10-97
CorrSetStart 10-99

増分 B-1

ソート

数の昇順 / 降順
 LAPACK 5-198
 ScaLAPACK 7-146

固有ペア 7-145

ソルバー

スパース 8-1
直接法 A-1
反復法 A-1

た

対応プラットフォーム 1-6

対称行列

三重対角形式への縮退 5-245, 7-137

対角成分

LAPACK 5-189
ScaLAPACK 7-88

対角優位三重対角行列

連立 1 次方程式の解の算出 6-27

対角優位帯行列

連立 1 次方程式の解の算出 6-30

台形行列

1- ノルムの値 5-112
Frobenius ノルムの値 5-112
RZ 因子分解
 LAPACK 4-51
 ScaLAPACK 6-107
最大絶対値の成分 5-112
三角形式への縮退 7-100
無限ノルムの値 5-112

対称帯行列

1- ノルムの値 5-100
Frobenius ノルムの値 5-100
最大絶対値の成分 5-100
無限ノルムの値 5-100

対称行列 4-89, 4-141

Bunch-Kaufman 因子分解 3-21

圧縮格納 3-27

階数 1 の更新 2-61, 5-17

圧縮格納 2-55, 5-14

階数 2 の更新 2-62

圧縮格納 2-57

階数 2k の更新 2-94

階数 n の更新 2-92

行列の逆転 3-129

圧縮格納 3-132

行列 - ベクトルの積 2-59, 5-15

帯格納 2-51

固有値と固有ベクトル 6-198, 6-201

三重対角形式への縮退

LAPACK 5-214

ScaLAPACK 7-94

条件数の推定 3-74, 4-139

圧縮格納 3-78

スカラー - 行列 - 行列の積 2-89

スケーリング 7-57

汎用固有値問題 4-141

標準形式への縮退

LAPACK 5-244

ScaLAPACK 7-134

連立 1 次方程式の解の算出 3-46

圧縮格納 3-50

対象行列

行列 - ベクトルの積

圧縮格納 2-53, 5-12

対称構造式 A-8

対称正定値行列

行列の逆転

圧縮格納 3-127

対称正定値帯行列

コレスキー因子分解 5-239
 対称正定値行列
 条件数の推定
 圧縮格納 3-69
 行列の逆転 3-126
 コレスキー因子分解
 LAPACK 3-15, 5-240
 ScaLAPACK 6-10, 7-132
 圧縮格納 3-17
 帯格納 3-18, 6-12
 条件数の推定 3-67
 帯格納 3-71
 三重対角行列 3-73
 連立 1 次方程式の解の算出
 LAPACK 3-38
 ScaLAPACK 6-22
 圧縮格納 3-40
 帯格納 3-42, 6-23
 対称正定値三重対角行列
 連立 1 次方程式の解の算出 6-25
 対称正定値分散行列
 行列の逆転 6-53
 スケール係数の計算 6-58
 平衡化 6-58
 対称不定値行列
 対角ピボット演算法での因子分解 5-247
 畳み込み関数
 ?ConvExec 10-102
 ?ConvExec1D 10-104
 ?ConvExecX 10-106
 ?ConvExecX1D 10-107
 ?ConvNewTask 10-88
 ?ConvNewTask1D 10-90
 ?ConvNewTaskX 10-91
 ?ConvNewTaskX1D 10-94
 ConvCopyTask 10-110
 ConvDeleteTask 10-109
 ConvSetDecimation 10-100
 ConvSetInternalPrecision 10-98
 ConvSetMode 10-97
 ConvSetStart 10-99
 CorrCopyTask 10-110
 CorrDeleteTask 10-109
 単一成分行列 7-141

ち

小さい劣対角成分 7-89
 置換行列 A-2
 直接法スパースソルバー (DSS) インターフェイス・
 ルーチン 8-13
 直交行列 4-64, 4-89, 4-155, 4-199, 6-143, 6-154
 LQ 因子分解から
 LAPACK 5-226
 ScaLAPACK 7-109
 QL 因子分解から

LAPACK 5-223, 5-229
 ScaLAPACK 7-105, 7-113
 QR 因子分解から
 LAPACK 5-225
 ScaLAPACK 7-107
 RQ 因子分解から
 LAPACK 5-227
 ScaLAPACK 7-111

て

ディスクリプター構成
 DFTI 11-3
 クラスター DFTI 11-42
 ディスクリプター操作
 DFTI 11-3
 クラスター DFTI 11-42
 データ型
 VML 9-1
 省略表記 1-8
 転置パラメーター B-6
 点の回転
 変形面 2-17
 面 2-14

と

特異値分解
 LAPACK 4-64, 4-350
 LAPACK ルーチン、特異値分解を参照
 ScaLAPACK 6-154, 6-213
 ドット積
 実ベクトル 2-9
 実ベクトル (拡張精度) 2-10
 スパース実ベクトル 2-106
 スパース複素ベクトル 2-108
 スパース複素ベクトル、共役 2-107
 複素ベクトル、共役 2-11
 複素ベクトル、非共役 2-12
 ドライバー
 簡易 6-3
 高度 6-3
 ドライバールーチン 3-150, 4-243

な

長さ。次元を参照

に

二重対角行列
 LAPACK 4-64
 ScaLAPACK 6-154
 二乗和
 更新

LAPACK 5-198
ScaLAPACK 7-90

二分法 5-143

は

配列ディスクリプター 6-2

パラメーター

Givens 回転 2-16

変形 Givens 変換 2-19

パラメーターの分割、区間方程式 12-12

汎用 Schur 因子分解 5-76, 5-132, 5-141, 5-142

汎用 Schur 特異値分解 4-217, 4-220

汎用固有値問題 4-141

LAPACK ルーチン、汎用固有値問題を参照
実対称の問題 4-141, 5-244, 5-245, 6-166, 7-134,
7-137

実対称問題

圧縮格納 4-145

帯格納 4-149

複素エルミート問題 4-143, 5-244, 5-245, 6-168,
7-134, 7-137

圧縮格納 4-147

帯格納 4-151

汎用固有値問題の縮退

LAPACK 4-141

ScaLAPACK 6-166

ひ

引数

行列 B-3

スパースベクトル 2-103

ベクトル B-1

非対角成分

LAPACK 5-189

ScaLAPACK 7-88

初期化 7-88

ふ

フィルイン、スパース行列 A-3

複素エルミート行列

1- ノルムの値

LAPACK 5-108

ScaLAPACK 7-49

Frobenius ノルムの値

LAPACK 5-108

ScaLAPACK 7-49

最大絶対値の成分

LAPACK 5-108

ScaLAPACK 7-49

対角ピボット演算法での因子分解 5-248

無限ノルムの値

LAPACK 5-108

ScaLAPACK 7-49

複素エルミート三重対角行列

1- ノルムの値 5-105

Frobenius ノルムの値 5-105

最大絶対値の成分 5-105

無限ノルムの値 5-105

複素対称行列

1- ノルムの値 5-106

Frobenius ノルムの値 5-106

最大絶対値の成分 5-106

無限ノルムの値 5-106

複素ベクトル

共役

LAPACK 5-8

ScaLAPACK 7-5

真の絶対値を使用した 1- ノルム

LAPACK 5-19

ScaLAPACK 7-7

複素ベクトルの共役

LAPACK 5-8

ScaLAPACK 7-5

負の固有値 7-44

プリコンディショニング、区間方程式 12-22

フル格納体系 B-3

フル格納ベクトル 2-103

付録テンプレート A-14

プロセスグリッド 6-1

ブロック分割法 10-7

ブロック・サイクリック分割 6-2

ブロック・リフレクター

一般行列

LAPACK 5-154

ScaLAPACK 7-77

一般矩形行列

ScaLAPACK 7-64

一般正方行列

LAPACK 5-134

三角係数

LAPACK 5-137, 5-156

ScaLAPACK 7-71, 7-83

分割コレスキー因子分解 (帯行列) 4-153

分割統治アルゴリズム 7-129

分散メモリー演算 6-1

へ

並列化対応直接法ソルバー (Pardiso) 8-1

ベクトル

Givens 回転 2-16

交換 2-22

コピー 2-7

最小絶対値を持つ成分 2-24

最大絶対値を持つ成分 2-23

実数部分が最大絶対値を持つ成分とそのイン

- デックス 7-7
- 実数部分が最大絶対値を持つ成分のインデックス 7-6
- スパーベクトル 2-104
- 点の回転 2-14
- ドット積
 - 実ベクトル 2-9
 - 複素ベクトル 2-12
 - 複素ベクトル、共役 2-11
- ベクトル - スカラーの積 2-20
- ベクトル成分の大きさの追加 2-5
- ベクトルの合計 2-6
- ベクトルの線形組み合わせ 2-6
- 変形 Givens 変換パラメーター 2-19
- 変形面における点の回転 2-17
- ユークリッド・ノルム 2-13
- ベクトル pack 関数 9-48
- ベクトル unpack 関数 9-50
- ベクトル乗算
 - ScaLAPACK 7-133
- ベクトル数学関数 9-6
 - 10 進対数 9-21
 - power (定数) 9-15
 - 逆誤差関数 9-40
 - 逆正弦 9-27
 - 逆正接 9-28
 - 逆双曲正弦 9-35
 - 逆双曲正接 9-36
 - 逆双曲余弦 9-34
 - 逆転 9-8
 - 逆平方根 9-11
 - 逆余弦 9-26
 - 逆立方根 9-13
 - 切り捨てられた整数値を計算する 9-46
 - 現在の丸めモードで丸めた整数値を計算する 9-44
 - 誤差関数値 9-38
 - 指数 9-18
 - 自然対数 9-20
 - 四分円逆正接 9-29
 - 除算 9-9
 - 正弦 9-23
 - 正弦と余弦 9-24
 - 整数値を計算して不正確な結果例外処理を起動する 9-45
 - 正接 9-25
 - ゼロ方向に丸める 9-42
 - 双曲正弦 9-32
 - 双曲正接 9-33
 - 双曲余弦 9-30
 - 相補誤差関数値 9-39
 - 二乗和の平方根 9-17
 - プラス無限大方向に丸める 9-42
 - 平方根 9-10
 - マイナス無限大方向に丸める 9-41
 - 最も近い整数値に丸める 9-43
 - 立方根 9-12
- 累乗 9-14
- 余弦 9-22
- ベクトル - スカラーの積 2-20
 - スパーベクトル 2-105
- ベクトル成分の大きさの追加 2-5
- ベクトル成分の最小絶対値 2-24
- ベクトル成分の絶対値
 - 最小値 2-24
 - 最大値 2-23
- ベクトル統計関数
 - Bernoulli 10-63
 - Beta 10-58
 - Binomial 10-66
 - Cauchy 10-48
 - CopyStream 10-21
 - CopyStreamState 10-22
 - DeleteStream 10-20
 - dNewAbstractStream 10-16
 - Exponential 10-42
 - Gamma 10-56
 - Gaussian 10-36
 - GaussianMV 10-38
 - Geometric 10-65
 - GetBrngProperties 10-77
 - GetNumRegBrngs 10-32
 - GetStreamStateBrng 10-31
 - Gumbel 10-54
 - Hypergeometric 10-68
 - iNewAbstractStream 10-14
 - Laplace 10-44
 - LeapfrogStream 10-26
 - LoadStreamF 10-24
 - Lognormal 10-52
 - Negbinomial 10-73
 - NewStream 10-12
 - NewStreamEx 10-13
 - Poisson 10-70
 - PoissonV 10-71
 - Rayleigh 10-50
 - RegisterBrng 10-76
 - SaveStreamF 10-23
 - SkipAheadStream 10-28
 - sNewAbstractStream 10-18
 - Uniform (離散) 10-60
 - Uniform (連続) 10-34
 - UniformBits 10-62
 - Weibull 10-46
- ベクトルの交換 2-22
- ベクトルの乗算
 - LAPACK 5-243
- ベクトルの線形組み合わせ 2-6
- ベクトルの引数 B-1
 - 行列の 1 次元下部構造 B-2
 - デフォルト B-2
 - 長さ B-1
 - 配列の次元 B-1

例 B-1

ベクトル引数

スパーベクトル 2-103

ベクトル・インデックス 9-5

偏微分方程式のサポート

2次元ヘルムホルツ問題 13-19

2次元ポアソン問題 13-20

2次元ラプラス問題 13-20

3次元ヘルムホルツ問題 13-21

3次元ポアソン問題 13-21

3次元ラプラス問題 13-22

PDEのサポート。偏微分方程式のサポートを参照

ほ

ポアソン・ライブラリー・ルーチン 13-19, 13-25

?_commit_Helmholtz_3D 13-28

?_commit_Helmholtz_2D 13-28

?_Helmholtz_2D 13-32

?_Helmholtz_3D 13-32

?_init_Helmholtz_2D 13-26

?_init_Helmholtz_3D 13-26

free_Helmholtz_2D 13-36

free_Helmholtz_3D 13-36

コード例 C-81

補助ルーチン

LAPACK 5-1

ScaLAPACK 7-1

ま

マシン・パラメーター

ScaLAPACK 7-160

LAPACK 5-261

む

無限ノルムの値

一般矩形行列 5-96, 7-45

一般三重対角行列 5-97

上 Hessenberg 行列 5-98, 7-47

エルミート帯行列 5-101

三角帯行列 5-109

三角行列 5-112, 7-51

圧縮格納 5-110

実対称行列 5-106, 7-49

実対称三重対角行列 5-105

台形行列 5-112

対称帯行列 5-100

対称行列

圧縮格納 5-103

複素エルミート行列 5-108, 7-49

圧縮格納 5-104

複素エルミート三重対角行列 5-105

複素対称行列 5-106

め

命名規則 1-8

BLAS 2-2

LAPACK 3-2, 4-2, 6-2

VML 9-2

スパー BLAS レベル 1 2-103

スパー BLAS レベル 2 2-115

スパー BLAS レベル 3 2-115

ゆ

ユークリッド・ノルム

ベクトルの 2-13

ユーザータイム 5-266

ユニタリー行列 4-64, 4-89, 4-155, 4-199

LQ 因子分解から

LAPACK 5-226

ScaLAPACK 7-109

QL 因子分解から

LAPACK 5-223, 5-229

ScaLAPACK 7-105, 7-113

QR 因子分解から

LAPACK 5-225

ScaLAPACK 7-107

RQ 因子分解から

LAPACK 5-227

ScaLAPACK 7-111

ScaLAPACK 6-143, 6-154

ら

乱数生成器 10-1

ランダムストリーム 10-8

り

リーディング・ディメンジョン B-6

リーブフロッグ法 10-7

離散フーリエ変換

CommitDescriptor 11-8

CommitDescriptorDM 11-44

ComputeBackward 11-13

ComputeBackwardDM 11-48

ComputeForward 11-11

ComputeForwardDM 11-46

CopyDescriptor 11-9

CreateDescriptor 11-6

CreateDescriptorDM 11-42

ErrorClass 11-4

ErrorMessage 11-5

FreeDescriptor 11-10

FreeDescriptorDM 11-45

GetValue 11-16

GetValueDM 11-52

SetValue 11-15

SetValueDM 11-51

離散分布生成器 10-33

リバース・コミュニケーション・インターフェイス
8-27

隣接対角ブロックの交換 5-67, 5-250

る

ルーチングループ 1-6

ルーチン命名規則

BLAS 2-2

スパース BLAS レベル 1 2-103

スパース BLAS レベル 2 2-115

スパース BLAS レベル 3 2-115

れ

連続分布生成器 9-7, 10-33

連立 1 次方程式

三角行列 2-75

圧縮格納 2-71

帯格納 2-67

連立 1 次方程式。1 次方程式を参照
