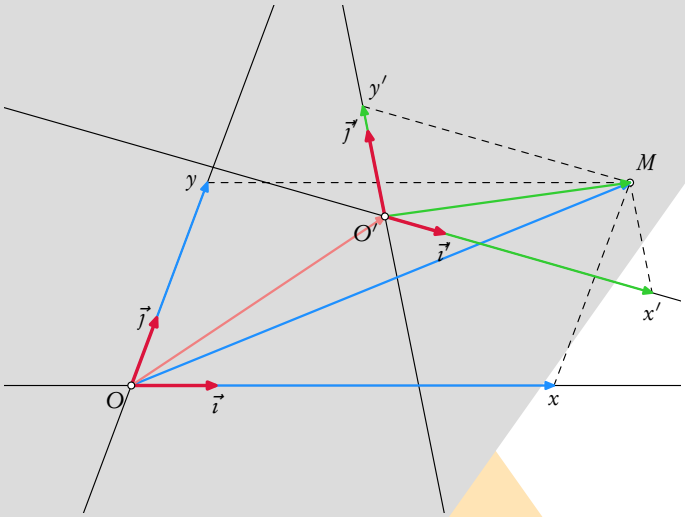
mp-geom2d

METAPOST package for plane geometry figures



Contributors
Maxime Chupin
Jean-Michel Sarlat

Version 1.3 from 11th May 2025
https://gitlab.gutenberg-asso.fr/mchupin/mp-geom2d
https://ctan.org/pkg/mp-geom2d

Contents

1 Objectif

The `mp-geom2d` package was written with the goal of providing METAPOST macros that allow the creation of geometric figures by closely following an imperative description:

> *Let A be the point with coordinates (2,3).*
> *Let B be the point with coordinates (4,5).*
> *Draw the line* (A,B).
> ....

In this context, geometric objects are most often named (A, B, etc.) or designated by their nature and attributes (line (A,B), etc.). To avoid having to go beyond this mode of description, particularly to avoid having to declare the *type* of these objects, the choice was made to identify them by an *index*[1] in tables that specify their characteristics.
Note — As of today, the goal has not been fully achieved, and the development is far from complete; it is still necessary to use METAPOST commands or *intermediate macros* to describe a figure. This will likely evolve over time as a satisfactory syntax is found...

2 Files

The `mp-geom2d` package is a set of tools for plane geometry with METAPOST [5]. This set is organized into several files:

1. `geom2d.mp` : this is the main file, which loads all the others;
2. `geom2d-main.mp` : contains the general structures and functions;
3. `geom2d-arc.mp` : contains everything related to circle arcs;
4. `geom2d-c2d.mp` : contains everything related to second-degree curves;
5. `geom2d-fct.mp` : contains some usual mathematical functions;
6. `geom2d-lbl.mp` : contains functions related to labels;
7. `geom2d-plt.mp` : contains functions facilitating the representation of mathematical functions;
8. `geom2d-rep.mp` contains various tools for drawing figures in a coordinate system;
9. `geom2d-tra.mp` contains functions for managing transparency (code borrowed from Anthony Phan);
10. `geom2d-svgnames.mp` provides the 150 colors of the SVG specification.

In the following sections, we will describe each of these functions in more detail. Some functions rely on the `graph.mp` extension, which is included in all good TEXdistributions.

[1] The `numeric` type, which is the default type in METAPOST, does not require prior declaration.

3 General Operating Principle

mp-geom2d uses tables as the main structure. Each object is numbered via the counter gddO, its type[2] is stored in the table gddT[] at the position gddT[gddO]. The properties of the objects are defined in, again, tables of type numeric which are gddA[], gddB[], ..., gddF[].

For example, for a point (type mp-geom2d), the first coordinate is found in gddA[] and the second in gddB[] (the other tables are not used for such an object).

There are three special tables: gddP[] which is of type path, gddPX[][] which is its extended version, and gddS[] which is of type string. We will see later what their utility is.

Of course, during a typical use of mp-geom2d, calling all these tables is not common practice. The functions we will describe in the rest of this document allow you to avoid having to deal too precisely with this machinery.

4 Drawing

mp-geom2d provides macros to draw objects. A command allows you to draw most of the mp-geom2d objects (which are described in section 5), as well as some METAPOSTtypes. The examples throughout the documentation will illustrate the drawing commands.

To understand how representations work with mp-geom2d, we first need to describe the unit management mechanism.

4.1 Unit

mp-geom2d defines a general unit with the global variable gddU. This numeric is by default 1 cm.

```
gddU
```

4.2 In Place

When using the drawing commands, mp-geom2d uses the macro

```
gddInPlace
```

This macro specifies the necessary geometric transformations for placing the object, particularly the scaling factor relative to the gddU unit.

Internally, this macro is a scantoken of the global variable gddW (string) which contains the transformations.

[2]The types are specific to mp-geom2d and will be described later.

4.3 Drawing Commands

The most common drawing macro is the trace macro.

gddDraw(⟨*object*⟩)

⟨***object***⟩: • for the METAPOST side, a path, a picture or a pair, in which case gddDraw will be equivalent to draw;

- for the mp-geom2d side, any of the 6 objects defined so far: triangle, segment, droite, cercle, ellipse, parabole, hyperbole, polygone, chemin, courbe or vecteur (for the latter, an arrow will be added to the end of the vector).

When gddDraw is used on an mp-geom2d object, the macro calls the METAPOST command draw with the placement mechanism described above.

mp-geom2d provides a drawing with an arrow that no longer calls draw but drawarrow .

gddArrow(⟨*object*⟩)

⟨***object***⟩: • for the METAPOST side, a path, a picture, or a pair, in which case trace will be equivalent to drawarrow;

- for the mp-geom2d side, any of the 6 objects defined so far: triangle, segment, droite, cercle, ellipse, polygone, chemin, courbe, or vecteur (for the latter, an arrow will be added to the end of the vector).

You can also color an object using the following macro.

gddFill(⟨*object*⟩)

⟨***object***⟩: can be:

- for the METAPOST side, only a path and in this case trace will be equivalent to fill;
- for the mp-geom2d side, any colorable object: triangle, cercle, ellipse, polygone, chemin, or courbe.

The drawing commands of mp-geom2d use the basic macros of METAPOST: draw and fill. Thus, to specify the color, you can use the withcolor command or the drawoptions macro.

You can also color with transparency[3] with the following command.

[3]In reality, we simulate transparency with code borrowed from Anthony Phan: http://www-math.univ-poitiers.fr/~phan/metalpha.html

`gddAlphaFill(`⟨*object*⟩`,`⟨*color*⟩`,`⟨*transparency coefficient*⟩`)`

⟨***object***⟩**:** can be:

- for the METAPOST side, only a `path` and in this case `trace` will be equivalent to `fill`;
- for the `mp-geom2d` side, any colorable object: `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, or `courbe`.

⟨***color***⟩**:** is a `color`.

⟨***transparency coefficient***⟩**:** is a `numeric` between 0 and 1, 0 for opaque coloring and 1 for invisible coloring.

You can also hatch a closed shape[4] with the following command:

`gddHatch(`⟨*object*⟩`,`⟨*angle*⟩`,`⟨*spacing*⟩`,`⟨*type*⟩`)` → `image`

⟨***object***⟩**:** can be:

- for the METAPOST side, only a `path` and in this case `trace` will be equivalent to `fill`;
- for the `mp-geom2d` side, any colorable object: `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, or `courbe`.

⟨***angle***⟩**:** is the angle (`numeric` in degrees) of the hatching lines.

⟨***spacing***⟩**:** is the space (`numeric`) between each hatching line (in the unit of `mp-geom2d`, so no unit should be specified).

⟨***type***⟩**:** is the type of line. If ⟨*type*⟩= 0, the lines will be solid, if ⟨*type*⟩= 1, the lines will be dashed evenly, if ⟨*type*⟩= 2, the lines will be axis lines, if ⟨*type*⟩= 3, the lines will be dashed with dots. Finally, if ⟨*type*⟩= 4, you can define the line pattern with the `HatchPattern` command.

Note that the `gddHatch` macro returns an `image`, so it should be used in conjunction with the `gddDraw` macro.

To use the fourth type of line for hatching, you need to define its pattern using the following command:

`HatchPattern(`⟨*pattern*⟩`)`

⟨***pattern***⟩**:** is the pattern that you would typically provide to the `dashpattern` command. We refer you to the METAPOST documentation. Note that the dimensions provided to

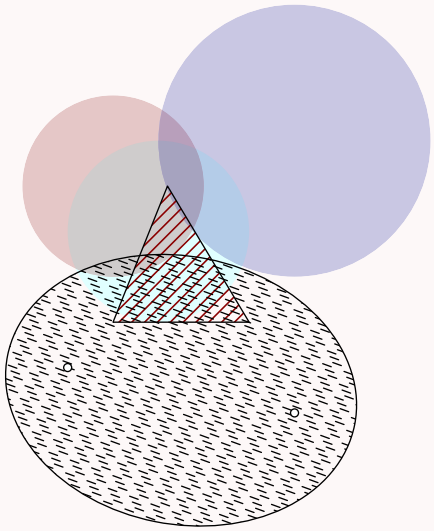
[4]The code is taken and adapted from the set of macros `geometriesyr16` by Christophe Poulain https://melusine.eu.org/syracuse/poulecl/geometriesyr16/.

the on and off commands will be scaled to the gddU unit of mp-geom2d. Therefore, they should be considered as expressed in the mp-geom2d unit.

Below, we illustrate some commands for drawing, coloring, and hatching.

Exemple 1

```
input geom2d;

beginfig(1);
gddU:=0.6cm;
A = Point(-1,1); B = Point(3,2);
C1= Circle(origine,2);
C2 = Circle(A,2); C3 = Circle(B
    ,3);

gddFill C1 withcolor LightCyan;
gddAlphaFill(C2,DarkRed,0.2);
gddAlphaFill(C3,DarkBlue,0.2);

T1 = Triangle((-1,-2),(2,-2)
    ,(0.2,1));
gddDraw gddHatch(T1,45,0.2,0)
    withPen(0.6,DarkRed);
gddDraw T1;
pE1 = Point(-2,-3);
pE2 = Point(3,-4);
E = EllipseF(pE1,pE2,3.9);
HatchPattern(on0.2 off0.3);
gddDraw gddHatch(E,-20,0.1,4);
gddDraw E;
gddDrawPoint pE1; gddDrawPoint pE
    2;
endfig;
```



The above macros call a lower-level macro that returns the path to be drawn or colored for all mp-geom2d objects.

`gddObjectDrawing(`⟨*object*⟩`)` → `path`

⟨***object***⟩: `triangle`, `segment`, `vecteur`, `cercle`, `ellipse`, `parabole`, `hyperbole`,`polygone`, `chemin`, or `courbe`.

If the object is not closed, mp-geom2d provides the following macro that adds a `--cycle` to the end of the path:

`gddClose(`⟨*object*⟩`)` → `path` (*closed with* `--cycle`)

⟨***object***⟩: • *on the METAPOSTside,* `path`*;*
• *on the* `mp-geom2d` *side,* `triangle`*,* `segment`*,* `vecteur`*,* `cercle`*,* `chemin`*, or* `courbe`*.*

You can specify the *pen* used for drawings with the following command:

`withPen(`⟨*size*⟩`,`⟨*color*⟩`)`

⟨***size***⟩: scaling factor (`numeric`) of the `pen` `pencircle` of METAPOST;

⟨***color***⟩: color `color` to use.

This macro is a shortcut for the *classic* `withpen pencircle scaled ... withcolor ...`. You can, of course, always use the METAPOST commands for drawing.
You can represent objects of type `point`. For this, there is a command that draws a small colored circle at the location of the point. The command is as follows:

`gddDrawPoint(`⟨*point*⟩`)`

⟨***point***⟩: `point` or `pair`

This macro uses three internal parameters of `mp-geom2d` for the size, color, and type of mark used (`gddTaillePoint`, `gddCouleurPoint` and `gddPointeType`). These parameters can be modified with the following commands:

`SetPointSize(`⟨*n*⟩`)`

⟨***n***⟩: `numeric`, default value 3.

`SetPointColor(`⟨*c*⟩`)`

⟨***c***⟩: `color`, default value `white`.

There are two other types of point marking: the cross and the disk. To choose another type of marking (using the same `gddDrawPoint` command), you can use the following macro:

`SetPointType(`⟨*s*⟩`)`

⟨***s***⟩: `string`, can be `"pointe"` (default behavior with a black circle filled with another

color, white by default), `"croix"` (for a simple cross), or `"disque"` (for a solid disk).

It is often necessary to mark a list of points[5]. For this, the following command is available:

`gddDrawPoints(`⟨*pointA*⟩`,` ⟨*pointB*⟩`,` ⟨*pointC*⟩`, etc.)`

⟨***pointA***⟩, ⟨***pointB***⟩, **etc.:** list of `point` or `pair` separated by commas.

4.4 Clipping the drawings

It is often necessary to restrict the drawings to a specific area of the plane $\mathbf{R}^2$. For this, `mp-geom2d` provides the command `Window`:

`Window(`⟨*Xmin*⟩`,`⟨*Ymin*⟩`,`⟨*Xmax*⟩`,`⟨*Ymax*⟩`)`

This function draws a rectangle with two opposite vertices at the points `(Xmin,Ymin)` and `(Xmax,Ymax)` with the METAPOST color `background` and clips the current picture around this frame (using the METAPOST command `clip currentpicture`).

You can also restrict drawings to the interior of a closed path derived from an `mp-geom2d` object. To do this, use the following command:

`gddClip(`⟨*object*⟩`)`

⟨***object***⟩**:** any `mp-geom2d` "closed" object.

5 Types

`mp-geom2d` provides several types of objects. The object type is stored in the `gddT[]` table, and the `gddA[]` to `gddF[]` tables, as well as the `gddX` table, contain the properties of the objects.

Here, we will describe each type of the `mp-geom2d` extension, their respective properties, as well as methods directly associated with the object.

5.1 The `point` type

This type corresponds to a point in Euclidean space.

[5]Based on an idea by Alain Matthes.

5.1.1 Constructor

To be clearer, here is the main function to create such an object:

```
vardef Point(expr a,b) =
  gddT[incr gdd0] = "point";
  gddA[gdd0] = a; gddB[gdd0] = b; gdd0
enddef;
```

`Point(`⟨*x*⟩`,`⟨*y*⟩`)` → `point`

⟨***x***⟩: `numeric`;

⟨***y***⟩: `numeric`.

This function returns the counter `gdd0` and creates an entry in the type table as `point` and the corresponding attributes (coordinates) `a` and `b` in the tables `gddA` and `gddB`.

With such a type of operation, most manipulations are done on `numeric`s. Indeed, to declare a `point`, it is sufficient to write

```
A = Point(2,3);
```

`A` then takes the current value of `gdd0`. This is the identifier of the point.

A `point` can also be defined by its *polar* coordinates with the following command.

`PolarPoint(`⟨*r*⟩`,`⟨*a*⟩`)` → `point`

⟨***r***⟩: `numeric`, modulus of the point;

⟨***a***⟩: `numeric`, angle with respect to the x-axis.

You can define a point in a coordinate system defined by an origin point and two vectors with the following function:

`PointInBasis(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*o*⟩`,`⟨*i*⟩`,`⟨*j*⟩`)` → `point`

⟨***x***⟩: `numeric` ;

⟨***y***⟩: `numeric` ;

⟨***o***⟩: `point` origin of the coordinate system ;

⟨***i***⟩: `vecteur` (or `point`) defining the x-axis ;

⟨***j***⟩: `vecteur` (or `point`) defining the y-axis.

The following code illustrates the use of this command (with `point`s instead of `vecteur`s, type described later).

```
O = Point(0,0);
I = Point(2,0);
J = PolarPoint(2,Pi/2);
A = PointInBasis(2,3,O,I,J);
```

5.1.2 Associated Macros

You can retrieve the x-coordinate and y-coordinate of a point with the following commands:

`Xcoordinate(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `point`, `vecteur` or `pair`.

`Ycoordinate(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `point`, `vecteur` or `pair`.

You can add the x-coordinates, the y-coordinates, or, like vectors, points together with the following commands.

`XcoordinateSum(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `point` or `pair`.

`YcoordinateSum(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `point` or `pair`.

and

`PointSum(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `point` or `pair`.

You can also calculate the distance between two points using the following command:

`Length(`⟨*a*⟩`,`⟨*b*⟩`)` → `numeric`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `point` or `pair`.

You can calculate the midpoint between two points:

`MidPoint(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `point` or `pair`.

You can perform the rotation of a point around the origin $(0,0)$ with the following command:

`Rotation(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `numeric`, the angle of rotation in radians.

If you want to rotate a point around another point, you will use the following command:

`CenterRotation(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`)` → `point`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `point` or `pair`, center of rotation;

⟨***c***⟩: `numeric`, the angle of rotation in radians.

We can calculate the isobarycenter of a list of points (or `pair` of METAPOST) with the following command.

`IsoBarycenter(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`, etc.)` → `point`

⟨***a***⟩: `point` or `pair`;

⟨***b***⟩: `point` or `pair`;

etc.

We can calculate the barycenter of a list of points associated with weights (but here, it is not possible to use METAPOST pairs).

`Barycenter((`⟨*A*⟩`,`⟨*a*⟩`),(`⟨*B*⟩`,`⟨*b*⟩`),(`⟨*C*⟩`,`⟨*c*⟩`), etc.)` → `point`

⟨***A***⟩: `point` ;

⟨***a***⟩: `numeric`, weight associated with ⟨*A*⟩ ;

⟨***B***⟩: `point` ;

⟨***b***⟩: `numeric`, weight associated with ⟨*B*⟩ ;

etc.

You can determine the bisector of an angular sector defined by three points. The function returns a `droite`.

`AngleBisector(`⟨*A*⟩`,`⟨*B*⟩`,`⟨*C*⟩`)` → `droite`

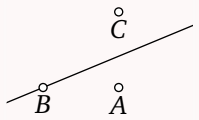
⟨***A***⟩: `point` ;

⟨***B***⟩: `point` ;

⟨***C***⟩: `point`.

Exemple 2

```
input geom2d;
beginfig(1);
A = Point(1,0);
B = Point(0,0);
C = Point(1,1);
D = AngleBisector(A,B,C);
gddDraw D;
gddMark.bot "A";
gddMark.bot "B";
gddMark.bot "C";
Window(-0.5,-0.5,2,2);
endfig;
```



The `point` type is obviously related to the METAPOST `pair` type.
`mp-geom2d` provides macros that allow converting from `point` to `pair` and vice versa.

`PointTOPair(`⟨*a*⟩`,`⟨*b*⟩`)` → `pair`

⟨***a***⟩: `numeric`, x-coordinate;

⟨***b***⟩: `numeric`, y-coordinate.

`PairTOPoint(`⟨*p*⟩`)` → `point`

⟨***p***⟩: `pair`.

These two commands are complemented by two others that ensure an element is of a given type:

`PairImp(`⟨*p*⟩`)` → `pair`

⟨***p***⟩: `point` or `pair`.

`PointImp(`⟨*p*⟩`)` → `point`

⟨***p***⟩: `point` or `pair`.

You can also retrieve a point along an `mp-geom2d` object (described in the following sections). The following macro returns a `point` along the path (cyclic or not) of the object, parameterized between 0 and 1.

`PointOf(`⟨*o*⟩`,`⟨*t*⟩`)` → `point`

⟨***o***⟩: any `mp-geom2d` object (even a `point` for which it returns itself);

⟨***t***⟩: `numeric` between 0 and 1 (which parameterizes the object's path between 0 and 1).

5.2 The `vecteur` type

This type corresponds to vectors defined using two coordinates in Euclidean space.

5.2.1 Constructors

The constructor function for such an object is:

```
vardef Vector(expr a,b) =
  save n; n = incr gddO;
  gddT[n] = "vecteur"; gddA[n] = a; gddB[n] = b; n
enddef;
```

`Vector(`⟨*a*⟩`,`⟨*b*⟩`)` → `vecteur`

⟨***a***⟩: `numeric` ;

⟨***b***⟩: `numeric`.

This function has the same structure as the corresponding one for the `point`: it returns the current value of `gddO` after incrementing it, then assigns the type `vecteur` to the corresponding entry in the `gddT` table.

A vector can also be defined from an `pair` in METAPOST.

```
vardef VectorP(expr a) =
  save n; n = incr gddO;
  gddT[n] = "vecteur"; gddA[n] = xpart a; gddB[n] = ypart a; n
enddef;
```

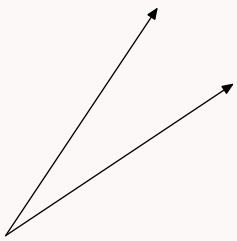
`VectorP(`⟨*a*⟩`)` → `vecteur`

⟨***a***⟩: `pair`.

A vector can then be defined as follows:

Exemple 3

```
input geom2d;
beginfig(1);
AB = Vector(2,3);
pair a;
a := (3,2);
A = VectorP(a);
gddDraw AB;
gddDraw A;
endfig;
```



5.2.2 Associated Macros

Since `mp-geom2d` objects are `numeric`s, the classic operations of the vector space cannot be written with the simple characters `+`, `-`, and `*`.

`VectorSum(`⟨*a*⟩`,`⟨*b*⟩`)` → `vecteur`

⟨***a***⟩: `vecteur` ;

⟨***b***⟩: `vecteur`.

VectorSubstract(⟨a⟩,⟨b⟩) → vecteur

⟨**a**⟩: vecteur ;

⟨**b**⟩: vecteur.

ScalarVector(⟨k⟩,⟨b⟩) → vecteur

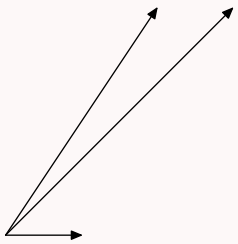
⟨**k**⟩: numeric ;

⟨**b**⟩: vecteur.

We can then perform the classic operations on vectors.

Exemple 4

```
input geom2d;
beginfig(1);
AB = Vector(2,3);
BC = Vector(1,0);
AC = VectorSum(AB,BC);
kAC = ScalarVector(3.0,AC);
gddDraw AB;
gddDraw BC;
gddDraw AC;
endfig;
```



We can also calculate the dot product of two vectors with the following command.

ScalarProduct(⟨a⟩,⟨b⟩) → numeric

⟨**a**⟩: vecteur ;

⟨**b**⟩: vecteur.

We can also calculate the Euclidean norm of a vector.

Norm(⟨a⟩) → numeric

⟨**a**⟩: vecteur.

The following command returns the angle, in radians, between $[0,\pi]$ formed between two vectors. If we denote u and v as the two vectors in $\mathbf{R}^2$, the angle calculated

$$\theta = \arccos\left(\frac{u \cdot v}{\|u\|\|v\|}\right).$$

`AngleBetweenVectors(`⟨*a*⟩`,`⟨*b*⟩`)` → `numeric` (dans $[0,\pi]$)

⟨***a***⟩: `vecteur` ;

⟨***b***⟩: `vecteur`.

5.3 The `segment` type

Segments are defined by two points in $\mathbf{R}^2$.

5.3.1 Constructor

The constructor function for this object is:

```
vardef Segment (expr a,b) =
  save n; n = incr gdd0;
  gddT[n] = "segment"; gddA[n] = PointImp(a); gddB[n] = PointImp(
     b); n
enddef;
```

`Segment(`⟨*a*⟩`,`⟨*b*⟩`)` → `segment`

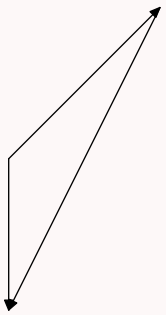
⟨***a***⟩: `point` ;

⟨***b***⟩: `point`.

A segment is then defined as follows:

Exemple 5

```
input geom2d;
beginfig(1);
A = Point(2,3);
B = Point(4,5);
C = Point(2,1);
AB = Segment(A,B);
BC = Segment(B,C);
AC = Segment(A,C);
gddArrow AB;
gddArrow BC;
gddArrow AC;
endfig;
```



5.3.2 Associated Macros

You can calculate the length of a segment with the following macro.

SegmentLength(⟨*a*⟩) → numeric

⟨***a***⟩: segment.

You can also convert a segment into a vector with the following macro, which calculates the difference in coordinates of the two points defining the segment.

SegmentTOVector(⟨*a*⟩) → vecteur

⟨***a***⟩: segment.

5.4 The droite type

A line is simply defined by two points.

5.4.1 Constructor

The constructor function for this object is as follows:

```
vardef Line (expr a,b) =
  save n; n = incr gddO;
  gddT[n] = "droite"; gddA[n] = PointImp(a); gddB[n] = PointImp(b
      ); n
enddef;
```

`Line(`⟨*a*⟩`,`⟨*b*⟩`)` → `droite`

⟨***a***⟩: `point` ;

⟨***b***⟩: `point`.

When representing lines (see section 4.3), the *infinite* nature of the line is managed by the global variable `gddExtensionDroite` which defaults to 10.

You can modify it with the following command:

`SetLineExtension(`⟨*d*⟩`)`

⟨***d***⟩: `numeric` (default value 10).

5.4.2 Associated Macros

The following macro allows, in the form of a triplet, and thus a `color`, to obtain the coefficients of a given line. For a line (D), the macro gives the triplet $(u,v,w) \in \mathbf{R}^3$ such that $\forall (x,y) \in (D)$, $ux+vy+w=0$.

`LineEquation(`⟨*a*⟩`)` → `color`

⟨***a***⟩: `droite`.

You can calculate the projection of a point onto a line with the following command.

`PointOnLineProjection(`⟨*p*⟩`,`⟨*a*⟩`)` → `point`

⟨***p***⟩: `point` ;

⟨***a***⟩: `droite`.

You can obtain the relative distance of a point on a line with the following macro.

`PointLineRelativeDistance(`⟨*p*⟩`,`⟨*a*⟩`)` → `numeric`

⟨***p***⟩: `point` ;

⟨***a***⟩: `droite`.

The calculation of the distance from a point to a line is done with the following macro.

PointLineDistance(⟨p⟩,⟨a⟩) → numeric

⟨**p**⟩: point ;

⟨**a**⟩: droite.

The following macro allows obtaining the perpendicular line to a given line passing through a point.

PerpendicularLine(⟨a⟩,⟨p⟩) → droite

⟨**a**⟩: droite ;

⟨**p**⟩: point.

Similarly, the following macro allows obtaining the parallel line to another passing through a point.

ParallelLine(⟨a⟩,⟨p⟩) → droite

⟨**a**⟩: droite ;

⟨**p**⟩: point.

You can obtain the point of intersection of two lines with the following macro.

LinesIntersection(⟨a⟩,⟨b⟩) → point

⟨**a**⟩: droite ;

⟨**b**⟩: droite.

The following macro allows you to plot a point on a line with a certain length (signed), similar to a compass.

ReportOnLine(⟨P⟩,⟨D⟩,⟨l⟩) → point

⟨**P**⟩: point ;

⟨**D**⟩: droite ;

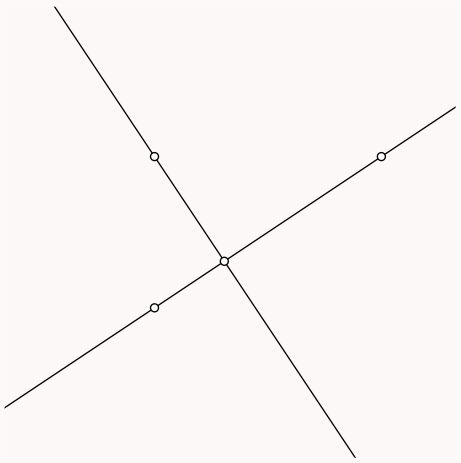
⟨**l**⟩: numeric (can be negative).

The direction of the plot relative to point ⟨*P*⟩ depends on the definition of the line: for a line defined by two `point`s *A* and *B*, the direction of the plot will follow the vector $\overrightarrow{AB}$. To change direction, simply multiply the distance parameter ⟨*l*⟩ by −1.

Here is an example illustrating the use of some commands related to lines.

Exemple 6

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Line(A,B);
gddDraw AB;
gddDrawPoint A;
gddDrawPoint B;
C = Point(0,2);
DC = PerpendicularLine(AB,C);
gddDraw DC;
gddDrawPoint C;
D = PointOnLineProjection(C,AB);
gddDrawPoint D;
Window(-2,-2,4,4);
endfig;
```



5.5 The `cercle` type

A circle could be defined by a point and a radius.

5.5.1 Constructors

The basic constructor function for this object is as follows:

```
vardef Circle (expr a,b) =
  save n; n = incr gddO;
  gddT[n] = "cercle"; gddA[n] = PointImp(a); gddB[n] = b; n
enddef;
```

`Circle(`⟨*a*⟩`,`⟨*b*⟩`)` → `cercle`

⟨***a***⟩: `point` or `pair`, center of the circle;

⟨***b***⟩: `numeric`, radius of the circle.

Two other functions allow defining circles. The `CircleCP` function defines the circle by a center and a point through which the circle passes.

`CircleCP(`⟨*a*⟩`,`⟨*b*⟩`)` → `cercle`

⟨***a***⟩: `point` or `pair`, center of the circle;

⟨***b***⟩: `point`, such that ⟨*b*⟩−⟨*a*⟩ is the radius.

We can also define a circle by two points defining its diameter. The corresponding function is as follows:

`CircleD(`⟨*a*⟩`,`⟨*b*⟩`)` → `cercle`

⟨***a***⟩: `point` or `pair` ;

⟨***b***⟩: `point` or `pair`, such that (⟨*a*⟩−⟨*b*⟩) is the diameter of the circle.

You can also define a circle as the circle passing through three points with the following macro.

`CircleThreePoints(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`)` → `cercle`

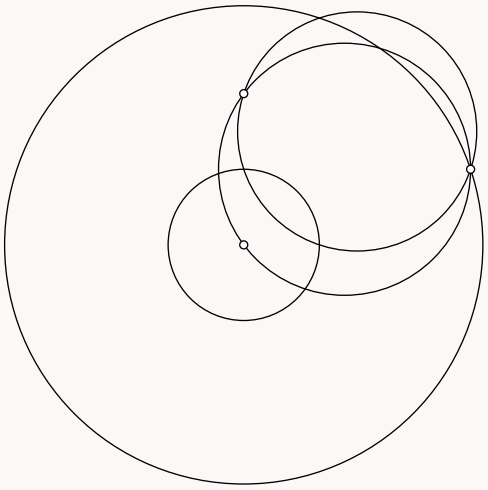
⟨***a***⟩: `point` ou `pair` ;

⟨***b***⟩: `point` ou `pair` ;

⟨***c***⟩: `point` ou `pair`.

Exemple 7

```
input geom2d;
beginfig(1);
A = Point(3,1);
B = Point(0,2);
O = Point(0,0);
r = 1.0;
COr = Circle(O,r);
COA = CircleCP(O,A);
CAB = CircleD(A,B);
COAB = CircleThreePoints(O,A,B);
gddDraw COr;
gddDraw COA;
gddDraw CAB;
gddDraw COAB;
gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint O;
endfig;
```



5.5.2 Associated Macros

Many macros are associated with circles.

You can retrieve the radius of a circle with the following command.

`Radius(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `cercle`.

You can obtain the center (`point`) of a circle with the following command:

`Center(`⟨*a*⟩`)` → `point`

⟨***a***⟩: `cercle`, `ellipse` or `hyperbole`.

We can compute the intersection between two circles using the following macro.

`CirclesIntersection(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

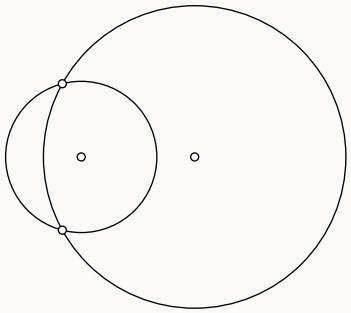
⟨***a***⟩: `cercle` ;

⟨***b***⟩: `cercle`.

This macro will only give one intersection point. To obtain both intersection points, you will need to reverse the order of the circles in the call, as illustrated in the following example.

Exemple 8

```
input geom2d;
beginfig(1);
A = Point(0,0);
CA = Circle(A,1);
B = Point(1.5,0);
CB = Circle(B,2);
D1 = CirclesIntersection(CA,CB);
D2 = CirclesIntersection(CB,CA);
gddDraw CA; gddDraw CB;
gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint D1; gddDrawPoint
     D2;
endfig;
```



If there is no intersection, the compilation will fail with the error `! Pythagorean subtraction X.XXXX+-+X.XXXX has been replaced by 0`. If the two circles are coincident, the error will be `! Division by zero.`.

`mp-geom2d` also provides a macro to obtain the intersections between a line and a circle.

`LineCircleIntersection(`⟨*d*⟩`,`⟨*c*⟩`,`⟨*n*⟩`)` → `point`

⟨***d***⟩: `line` ;

⟨***c***⟩: `circle` ;

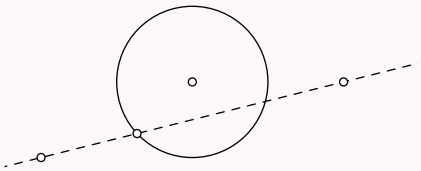
⟨***n***⟩: `numeric` which is 1 or 2 depending on the intersection point desired (if there is only one intersection point, both values return the same point).

If there is no intersection between the line and the circle, you will get the error `! Intersection between line and circle does not exist`.

The following example illustrates this macro.

Exemple 9

```
input geom2d;
beginfig(1);
A = Point(2,2);
C_A = Circle(A,1);
B = Point(0,1);
C = Point(4,2);
BC = Line(B,C);
E1 = LineCircleIntersection(BC,
    C_A,1);
gddDraw C_A;
gddDraw BC dashed evenly;
gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint C; gddDrawPoint
    E1;
Window(-0.5,-0.5,5,4);
endfig;
```



You can obtain the common internal and external tangents to two circles with the following two macros. Again, since there are two tangents, to obtain both, you will reverse the order of the two circles in the call.

`ExternalCommonTangent(`⟨*a*⟩`,`⟨*b*⟩`)` → `droite`

⟨***a***⟩: `cercle` ;

⟨***b***⟩: `cercle`.

`InternalCommonTangent(`⟨*a*⟩`,`⟨*b*⟩`)` → `droite`

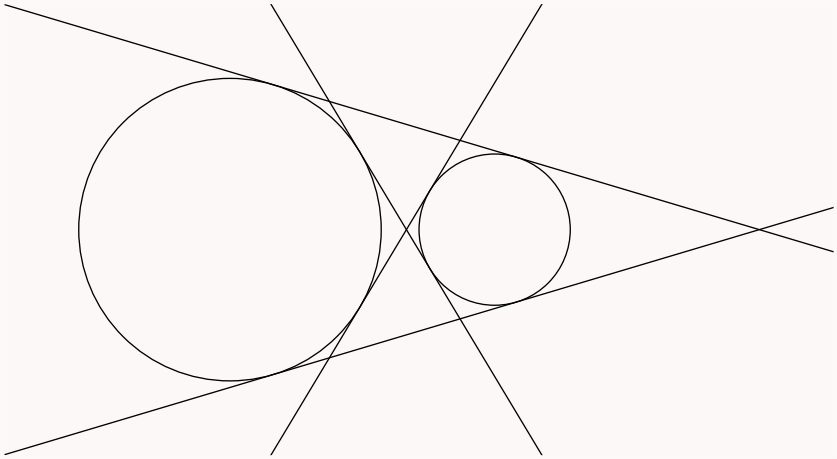
⟨***a***⟩: `cercle` ;

⟨***b***⟩: `cercle`.

The following example illustrates the use of these two macros.

Exemple 10

```
input geom2d;
beginfig(1);
C1 = Circle((-2,0),2);
C2 = Circle((1.5,0),1);
T1 = ExternalCommonTangent(C1,C2);
T2 = ExternalCommonTangent(C2,C1);
T3 = InternalCommonTangent(C1,C2);
T4 = InternalCommonTangent(C2,C1);
gddDraw C1; gddDraw C2;
gddDraw T1; gddDraw T2;
gddDraw T3; gddDraw T4;

Window(-5,-3,6,3);
endfig;
```



The following macro calculates the radical axis (`droite`) of two circles[6].

`RadicalAxis(`⟨*a*⟩`,`⟨*b*⟩`)` → `droite`

⟨***a***⟩: `cercle` ;

⟨***b***⟩: `cercle`.

The following macro calculates the radical center of three circles.

`RadicalCenter(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`)` → `point`

⟨***a***⟩: `cercle` ;

[6]The construction method was largely inspired by T. Thurston's code in his document *Drawing with METAPOST* [6].

⟨*b*⟩: cercle ;

⟨*c*⟩: cercle.

The following macro calculates the axis of similarity of three circles.

AxisOfSimilitude(⟨*a*⟩,⟨*b*⟩,⟨*c*⟩) → droite

⟨*a*⟩: cercle ;

⟨*b*⟩: cercle ;

⟨*c*⟩: cercle.

5.6 The ellipse type

5.6.1 Constructors

Ellipses can be defined in several ways. First, they can be defined with their center, one of the points on the major axis (called the vertex), and one of the points on the minor axis (called the co-vertex). However, when creating an ellipse, many attributes are calculated.

The constructor code is as follows:

```
vardef Ellipse(expr C,A,B) =
% C : centre
% A : vertex
% B : co-vertex
save n,a,b,c,e,_tmp,slope,_D,_K,_h,_ff;
pair _tmp;
n = incr gddO;
gddT[n] = "ellipse"; gddA[n] = PointImp(C); gddB[n] = PointImp(
    A);
gddC[n] = PointImp(B);
% calcul des deux foyers
a = Longueur(C,A);
b = Longueur(C,B);
c = sqrt(a**2-b**2);
e = c/a;
_tmp = e*(Pt(A)-Pt(C));
% les foyers
gddD[n] = PairTOPoint(Pt(C)+_tmp);
gddE[n] = PairTOPoint(Pt(C)-_tmp);
gddX[n][1] = a;
gddX[n][2] = b;
```

```
  gddX[n][3] = e;
  % angle du demi grand axe
  slope = angle(PairImp(A)-PairImp(C));
  gddX[n][4] = slope;
  % directrices
  _h = (a*a-c*c)/c;
  % prejection sur la directice
  _ff = Droite(gddE[n],gddD[n]);
  _K = ReportSurDroite(gddD[n],_ff,_h);
  _D = DroitePerpendiculaire(_ff,_K);
  gddX[n][5] := _D; % on stock la directrice
  gddX[n][6] := SymetrieCentrale(_D,gddA[n]); % on stock la
      deuxième directrice
  n
enddef;
```

`Ellipse(`⟨*c*⟩`,`⟨*a*⟩`,`⟨*b*⟩`)` → `ellipse`

⟨***c***⟩: `point` or `pair`, the center;

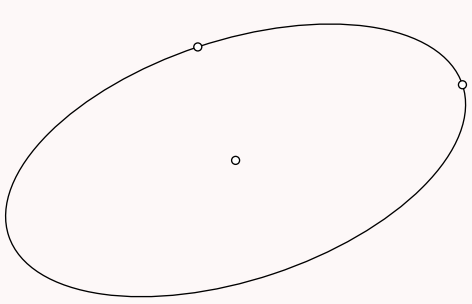
⟨***a***⟩: `point` or `pair`, the vertex;

⟨***b***⟩: `point` or `pair`, the co-vertex.

This macro is used as shown in the following example.

Exemple 11

```
input geom2d;
beginfig(1);
C = Point(0,0);
A = Point(3,1);
B = Rotation((1.5,0.5),Pi/2);
E = Ellipse(C,A,B);
gddDraw E;
gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint C;
endfig;
```



An ellipse can be defined with its two foci and the semi-major axis using the following command:

`EllipseF(`⟨*A*⟩`,`⟨*B*⟩`,`⟨*a*⟩`)` → `ellipse`

⟨***A***⟩: `point` or `pair`, first focus;

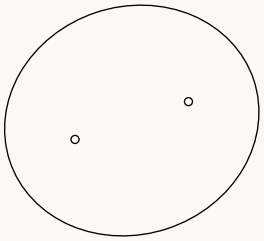
⟨***B***⟩: `point` or `pair`, second focus;

⟨***a***⟩: `numeric`, the semi-major axis.

This macro is used as shown in the following example.

Exemple 12

```
input geom2d;
beginfig(1);
F1 = Point(3,1);
F2 = Point(1.5,0.5);
E = EllipseF(F1,F2,1.7);
gddDraw E;
gddDrawPoint F1; gddDrawPoint F2;
endfig;
```



We can also define an ellipse from a focus, its directrix, and the eccentricity with the following command:

`EllipseFD(`⟨*F*⟩`,`⟨*D*⟩`,`⟨*e*⟩`)` → `ellipse`

⟨***F***⟩: `point` or `pair`, a focus;

⟨***D***⟩: `droite`, the associated directrix;

⟨***e***⟩: `numeric`, the eccentricity (< 1).

5.6.2 Associated Macros

You can obtain the center (`point`) of an ellipse with the following command:

`Center(`⟨*a*⟩`)` → `point`

⟨***a***⟩: `cercle`, `ellipse` or `hyperbole`.

You can obtain the vertex and co-vertex with the following commands:

`Vertex(`⟨*a*⟩`)` → `point`

⟨***a***⟩: `ellipse`.

`CoVertex(`⟨*a*⟩`)` → `point`

⟨***a***⟩: `ellipse`.

The following command allows obtaining the two foci based on the integer passed as an argument.

`Focus(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`

⟨***a***⟩: `ellipse` ;

⟨***n***⟩: `numeric`, integer which is 1 or 2 to obtain each focus.

You can obtain the semi-major axis and the semi-minor axis with the following commands.

`SemiMajorAxis(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `ellipse`.

`SemiMinorAxis(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `ellipse`.

We can get the excentricity, denoted by e, with the following command.

`Excentricity(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `ellipse`, `parabole` ou `hyperbole`.

To obtain the inclination (slope of the line passing through the foci of the ellipse), use the following command.

`Slope(`⟨*a*⟩`)` → `numeric`

⟨*a*⟩: `ellipse`, `parabole`, or `hyperbole`.

You can obtain the directrices of the ellipse with the following command:

`Directrix(`⟨*a*⟩`,`⟨*n*⟩`)` → `droite`

⟨*a*⟩: `ellipse`, `parabole`, or `hyperbole` ;

⟨*n*⟩: 1 or 2 (`numeric`), to choose one of the two directrices.

To obtain the tangent (`droite`) at a point on the ellipse, use the following command.

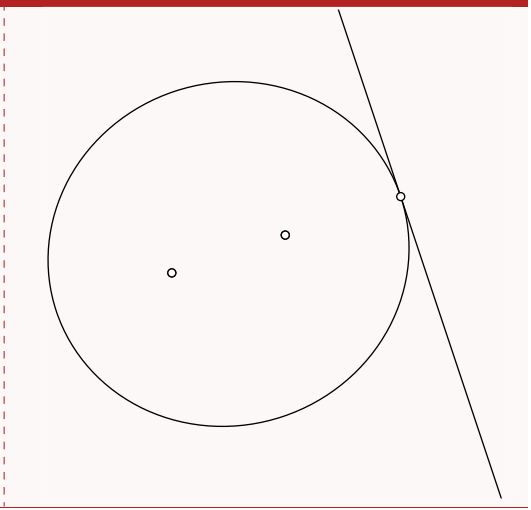
`EllipseTangent(`⟨*e*⟩`,`⟨*p*⟩`)` → `droite`

⟨*e*⟩: `ellipse`;

⟨*p*⟩: `point` or `pair`.

Exemple 13

```
input geom2d;
beginfig(1);
F1 = Point(3,1);
F2 = Point(1.5,0.5);
E = EllipseF(F1,F2,2.4);
gddDrawPoints F1, F2;
gddDraw E;
M' = PointOf(E,0.5);
D = EllipseTangent(E,M');
gddDraw D;
gddDrawPoints M', Foyer(E,1),
    Foyer(E,2);
Window(-0.2,-2.5,6,4);
endfig;
```



To obtain the tangents (`droite`) passing through a point outside the ellipse, use the following command. If the chosen point is not outside the ellipse, a compilation error will occur.

`ExternalEllipseTangent(`⟨*e*⟩`,`⟨*p*⟩`,`⟨*n*⟩`)` → `droite`

⟨*e*⟩: `ellipse` ;

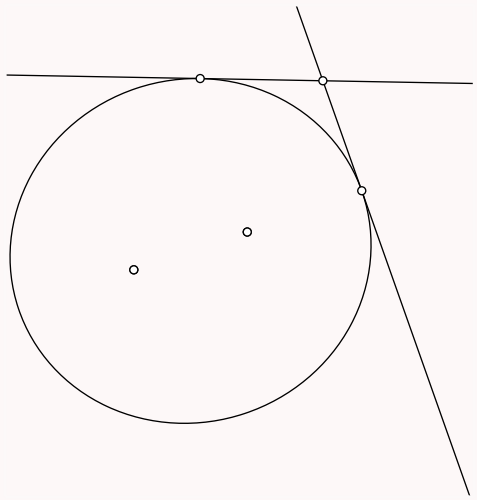
⟨*p*⟩: `point` or `pair` ;

⟨*n*⟩: `numeric` which is 1 or 2 to choose the tangent among the two possible ones.

The points of tangency can be accessed as the second point in the definition of the `droite` using the `gddB[]` table.

Exemple 14

```
input geom2d;
beginfig(1);
F1 = Point(3,1);
F2 = Point(1.5,0.5);
E = EllipseF(F1,F2,2.4);
gddDrawPoint F1; gddDrawPoint F2;
gddDraw E;
M = Point(4,3);
D1 = ExternalEllipseTangent(E,M
    ,1);
gddDraw D1;
gddDrawPoint gddB[D1];
D2 = ExternalEllipseTangent(E,M
    ,2);
gddDraw D2;
gddDrawPoint gddB[D2];
gddDrawPoint M;
gddDrawPoint Focus(E,1);
gddDrawPoint Focus(E,2);
Window(-0.2,-2.5,6,4);
endfig;
```



5.7 The `parabole` type

The basic constructor for the parabola defines this object using the focus and the directrix.

The constructor code is as follows:

```
vardef ParaboleFD(expr F,D) =
  % F : foyer (point)
  % D : directrice (droite)
  save u, v, w, d, i, n,_tmp,slope;
  pair _tmp;
  n = incr gddO;
  (u,v,w) = EquationDroite(D);
  % ordonnée relative
  d := u * gddA[F] + v * gddB[F] + w;
  gddT[n] := "parabole";
  gddX[n][1] := D; % on stocke la directrice
```

```
  gddX[n][2] := D; % on stocke la directrice (compatibilité avec
     hyperbole)
  % sommet
  _tmp := (((-d/2)*(u,v)) shifted PairImp(F));
  gddB[n] = PointImp(_tmp);
  gddC[n] = PointImp(_tmp);
  % le foyer (doublé pour compatibilité)
  gddD[n] := F;
  gddE[n] := F;
  gddX[n][3] := 1.0; % excentricité
  % angle du demi-grand axe
  slope = angle(PairImp(gddA[D])-PairImp(gddB[D]))+90;
  gddX[n][4] = slope;
  i := -gddC2Dparam-1;
  gddP[n] := (
    ((i*(v,-u)+((i*i-d*d)/(2d))*(u,v))
    for i:= -gddC2Dparam upto gddC2Dparam:
      ..(i*(v,-u)+((i*i-d*d)/(2d))*(u,v))
    endfor)) shifted PairImp(F);
   n
enddef;
```

`ParabolaFD(`⟨*f*⟩`,`⟨*d*⟩`)` → `path`

⟨***f***⟩: is the focus (`point`) of the parabola;

⟨***d***⟩: is the directrix (`droite`) of the parabola.

When representing a parabola, the *infinite* nature is managed by the global variable `gddC2Dparam`, which defaults to 15.

To modify it, use the following command:

`SetConicCoef(`⟨*n*⟩`)`

⟨***n***⟩: `numeric`, parameter for drawing hyperbolas and parabolas (default value 15).

5.7.1 Associated Functions

The eccentricity, often denoted as *e*, can be obtained using the following command:

`Excentricity(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `ellipse`, `parabole`, or `hyperbole`.

To obtain the slope (the slope of the axis of symmetry of the parabola), use the following command:

`Slope(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `ellipse`, `parabole`, or `hyperbole`.

The vertex of the parabola can be obtained using the following command:

`ParabolaVertex(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`

⟨***a***⟩: `parabole` or `hyperbole`;

⟨***n***⟩: 1 or 2 (`numeric`), an argument useful for the hyperbola which has two vertices. Here, the value of ⟨*n*⟩ does not matter, as the unique vertex will be returned.

The parabola's directrix can be obtained using the following command:

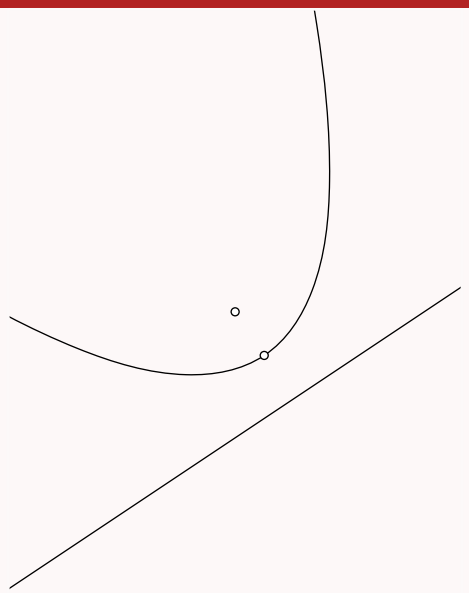
`Directrix(`⟨*a*⟩`,`⟨*n*⟩`)` → `droite`

⟨***a***⟩: `ellipse`, `parabole`, or `hyperbole`;

⟨***n***⟩: 1 or 2 (`numeric`), an argument useful for the hyperbola which has two directrices. Here, the value of ⟨*n*⟩ does not matter, as the unique directrix will be returned.

Exemple 15

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Line(A,B);
F = Point(-1,1);
Para = ParabolaFD(F,AB);
gddDraw AB;
gddDraw Para;
gddDrawPoint ParabolaVertex(Para
    ,1);
gddDrawPoint F;
Window(-4,-5,2,5);
endfig;
```



5.8 The hyperbole type

The basic constructor for the hyperbola defines this object using the focus and the directrix.
The constructor code is as follows:

```
vardef HyperboleFD(expr F,D,e) =
  % F : foyer (point)
  % D : directrice (droite)
  % e : exentricité (numeric)
  % pm : +1 ou -1 pour les deux branches
  save u, v, w, d, i, n,_tmp,slope,aa,bb;
  pair _tmp;
  n = incr gddO;

  (u,v,w) = EquationDroite(D);
  d := u * gddA[F] + v * gddB[F] + w;
  gddT[n] := "hyperbole";

  % sommets
  _tmp := -(d+1*sqrt(e*e*d*d))/(e*e-1)*(u,v)shifted (
     ProduitScalaire(F,Vecteur(v,-u))*(v,-u));
  gddB[n] := PointImp(_tmp);
```

```
  _tmp := -(d-1*sqrt(e*e*d*d))/(e*e-1)*(u,v)shifted
  (ProduitScalaire(F,Vecteur(v,-u))*(v,-u));
  gddC[n] := PointImp(_tmp);
  % centre comme milieu des deux sommets
  gddA[n] = Milieu(gddB[n],gddC[n]);
  % le foyer (doublé pour compatibilité)
  gddD[n] := F;
  gddE[n] := RotationCentre(F, gddA[n], Pi);
  gddX[n][3] := e; % excentricité
  % directrices
  gddX[n][1] := D; % on stocke la directrice
  gddX[n][2] := Droite(RotationCentre(gddA[D], gddA[n], Pi),
     RotationCentre(gddB[D], gddA[n], Pi));
  % angle de l'axe
  slope = angle(PairImp(gddA[D])-PairImp(gddB[D]))+90;
  gddX[n][4] = slope;
  % cercle principale
  gddX[n][5] = CercleD(gddB[n],gddC[n]);
  % asymptotes
  aa = IntersectionDroiteCercle(D,gddX[n][5],1);
  bb = IntersectionDroiteCercle(D,gddX[n][5],2);
  gddX[n][6] = Droite(gddA[n],aa);
  gddX[n][7] = Droite(gddA[n],bb);
  % tracés des moitiés
  i := -gddC2Dparam-1;
  gddPX[n][1] := (
    (
    (i*(v,-u)-(d+sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
    for i:= -gddC2Dparam upto gddC2Dparam:
      ..(i*(v,-u)-(d+sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
    endfor
  ) shifted (ProduitScalaire(F,Vecteur(v,-u))*(v,-u))
  );
  gddPX[n][2] := (
    (
    (i*(v,-u)-(d-sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
    for i:= -gddC2Dparam upto gddC2Dparam:
      ..(i*(v,-u)-(d-sqrt(e*e*d*d+i*i*(e*e-1)))/(e*e-1)*(u,v))
    endfor
  ) shifted (ProduitScalaire(F,Vecteur(v,-u))*(v,-u))
  );
  n
enddef;
```

In this constructor, we can see the use of the extended `path` table `gddPX[][]`, which allows associating multiple `path`s with a single object.

`HyperbolaFD(`⟨*f*⟩`,`⟨*d*⟩`,`⟨*e*⟩`)` → `hyperbole`

⟨***f***⟩**:** is the focus (`point`) of the hyperbola;

⟨***d***⟩**:** is the directrix (`droite`) of the hyperbola;

⟨***e***⟩**:** is the eccentricity of the hyperbola.

You can also define a hyperbola using its two foci and the distance between a vertex and the center of the hyperbola (bifocal definition).

`HyperbolaF(`⟨*f1*⟩`,`⟨*f2*⟩`,`⟨*a*⟩`)` → `hyperbole`

⟨***f1***⟩**:** is one focus (`point`) of the hyperbola;

⟨***f2***⟩**:** is the second focus (`point`) of the hyperbola;

⟨***a***⟩**:** is half the distance between the two vertices of the hyperbola.

5.8.1 Associated Functions

You can obtain the center (`point`) of a hyperbola with the following command:

`Center(`⟨*a*⟩`)` → `point`

⟨***a***⟩**:** `cercle`, `ellipse`, or `hyperbole`.

The eccentricity, often denoted as e, can be obtained using the following command.

`Excentricity(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩**:** `ellipse`, `parabole`, or `hyperbole`.

To obtain the inclination (slope of the line passing through the foci of the hyperbola), use the following command.

`Slope(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩**:** `ellipse`, `parabole`, or `hyperbole`.

The vertices of the hyperbola can be obtained using the following command:

`HyperbolaVertex(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`

⟨***a***⟩: `parabola` or `hyperbola`;

⟨***n***⟩: 1 or 2 (`numeric`), to choose one of the two vertices.

You can obtain the directrices of the hyperbola with the following command:

`Directrix(`⟨*a*⟩`,`⟨*n*⟩`)` → `droite`

⟨***a***⟩: `ellipse`, `parabole`, or `hyperbole` ;

⟨***n***⟩: 1 or 2 (`numeric`), to choose one of the two directrices.

To obtain the *principal circle* of the hyperbola, you can use the following command:

`PrincipalCircle(`⟨*h*⟩`)` → `cercle`

⟨***h***⟩: `hyperbole`.

You can also obtain the two asymptotes of the hyperbola with the following command:

`HyperbolaAsymptote(`⟨*h*⟩`,`⟨*n*⟩`)` → `droite`

⟨***h***⟩: `hyperbole` ;

⟨***n***⟩: 1 or 2 (`numeric`), to choose one of the two asymptotes.

Because a hyperbola consists of two disjoint parts, you cannot directly use the `gddDraw` command on the object as with other `mp-geom2d` objects. Instead, you must use the following command to obtain the `path` associated with one of the two parts of the hyperbola.

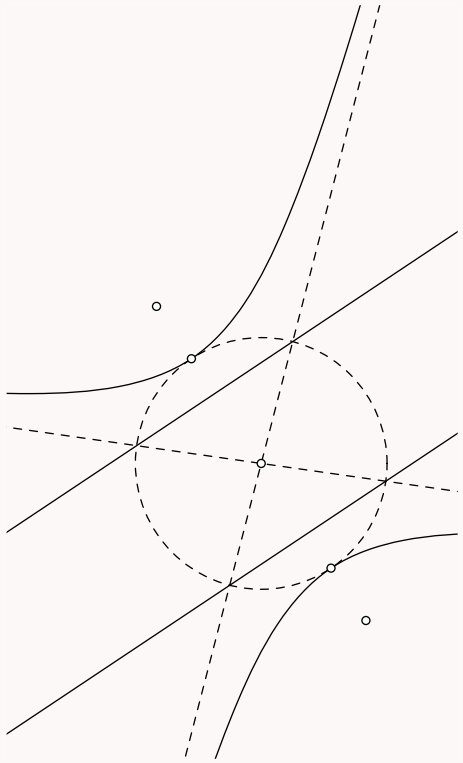
`SemiHyperbola(`⟨*h*⟩`,`⟨*n*⟩`)` → `path`

⟨***h***⟩: `hyperbole`;

⟨***n***⟩: 1 or 2 (`numeric`), to choose one of the two parts.

Exemple 16

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Line(A,B);
F = Point(-1,1);
Hyper = HyperbolaFD(F,AB,1.5);
gddDraw AB;
gddDraw SemiHyperbola(Hyper,1);
gddDraw SemiHyperbola(Hyper,2);
gddDraw Directrix(Hyper,2);
gddDraw PrincipalCircle(Hyper)
    dashed evenly;
gddDraw HyperbolaAsymptote(Hyper
    ,1) dashed evenly;
gddDraw HyperbolaAsymptote(Hyper
    ,2) dashed evenly;
gddDrawPoint Center(Hyper);
gddDrawPoint Focus(Hyper,1);
gddDrawPoint Focus(Hyper,2);
gddDrawPoint HyperbolaVertex(
    Hyper,1);
gddDrawPoint HyperbolaVertex(
    Hyper,2);
Window(-3,-5,3,5);
endfig;
```



5.9 The `triangle` type

Triangles are defined by three points in $\mathbf{R}^2$.

5.9.1 Constructor

The constructor function for this object is as follows:

```
vardef Triangle (expr a,b,c) =
  save n; n = incr gddO; gddT[n] = "triangle";
  gddA[n] = PointImp(a); gddB[n] = PointImp(b); gddC[n] =
     PointImp(c); n
enddef;
```

`Triangle(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`)` → `numeric`

⟨***a***⟩: `point` ;

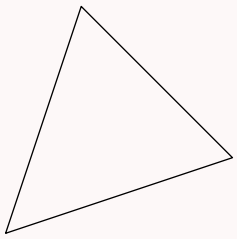
⟨***b***⟩: `point` ;

⟨***c***⟩: `point`.

Here, we see the use of the third table gddC to store the third point.
A triangle is then defined as follows:

Exemple 17

```
input geom2d;
beginfig(1);
A = Point(3,1);
B = Point(1,3);
C = Point(0,0);
ABC = Triangle(A,B,C);
gddDraw ABC;
endfig;
```



5.9.2 Associated Macros

The following macro calculates the area of a triangle.

`TriangleArea(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `triangle`.

We can also calculate the orthocenter of a triangle using the following command.

`Orthocenter(`⟨*a*⟩`)` → `point`

⟨***a***⟩: `triangle`.

The following macro calculates the inscribed circle of a triangle.

`InscribedCircle(`⟨*a*⟩`)` → `cercle`

⟨***a***⟩: `triangle`.

You can also obtain the circumscribed circle with the following macro.

`CircumscribedCircle(`⟨*a*⟩`)` → `cercle`

⟨***a***⟩: `triangle`.

We can also calculate the escribed circles. Let A, B, and C be three points defining a triangle ABC[7]. The following command allows obtaining one of the three escribed circles,

[7]Of course, the points can have different names.

as desired.

`EscribedCircle(`⟨*a*⟩`,`⟨*n*⟩`)` → `cercle`

⟨***a***⟩: `triangle`;

⟨***n***⟩: `numeric` which can be 1, 2, or 3. If ⟨*n*⟩= 1, it is the escribed circle tangent to the side [*BC*] of the triangle; if ⟨*n*⟩= 2, it is the one tangent to the side [*AC*]; and if ⟨*n*⟩= 3, it is the one tangent to the side [*AB*].

You can also obtain the Euler circle of a triangle with the following command.

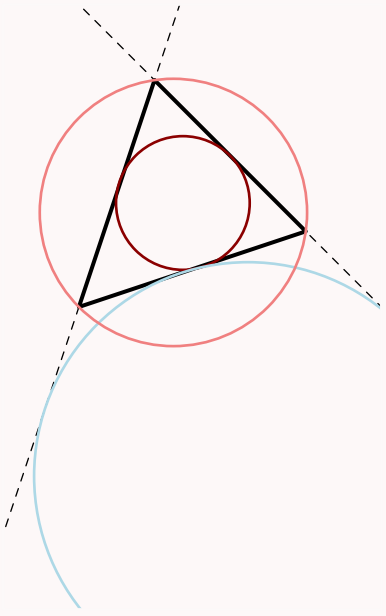
`EulerCircle(`⟨*a*⟩`)` → `cercle`

⟨***a***⟩: `triangle`.

Here is an example illustrating some of the macros related to triangles.

Exemple 18

```
input geom2d;
beginfig(1);
A = Point(3,1);
B = Point(1,3);
C = Point(0,0);
AB = Line(A,B);
BC = Line(B,C);
ABC = Triangle(A,B,C);
gddDraw AB dashed evenly;
gddDraw BC dashed evenly;
gddDraw ABC withPen(1.5,black);
Euler = EulerCircle(ABC);
gddDraw Euler withPen(1,DarkRed);
C_E1 = EscribedCircle(ABC,2);
gddDraw C_E1 withPen(1,LightBlue)
    ;
C_C = CircumscribedCircle(ABC);
gddDraw C_C withPen(1,LightCoral)
    ;
Window(-1,-4,4,4);
endfig;
```



5.10 The `polygon` type

Polygons are defined by N points in $\mathbf{R}^2$.

5.10.1 Constructors

The constructor function for this object is as follows:

```
vardef Polygon e(text plist) =
  save n,_point,i; n = incr gddO; gddT[n] = "polygone";
  i:=1;
  for _point = plist:
    gddX[gddO][i] = PointImp(_point);
    i:=i+1;
  endfor
  gddA[n] = i-1; % number of sides
  gddB[n] = IsoBarycenter(plist); % center
  n
enddef;
```

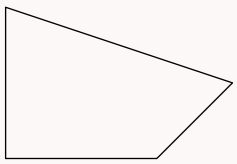
`Polygon(`⟨*list*⟩`)` → `polygone`

⟨***list***⟩: is a list of `point` or `pair`.

Here, we see the use of the *double* table `gddX` to store the *N* points.
A polygon is then defined as follows:

Exemple 19

```
input geom2d;
beginfig(1);
A = Point(2,0);
B = Point(3,1);
C = Point(0,2);
D = Point(0,0);
P = Polygon(A,B,C,D);
gddDraw P;
endfig;
```



You can also construct regular polygons with the following command.

`RegularPolygon(`⟨*N*⟩`,`⟨*radius*⟩`,`⟨*rotation*⟩`,`⟨*translation*⟩`)` → `polygone`

⟨***N***⟩: integer (`numeric`) indicating the number of points;

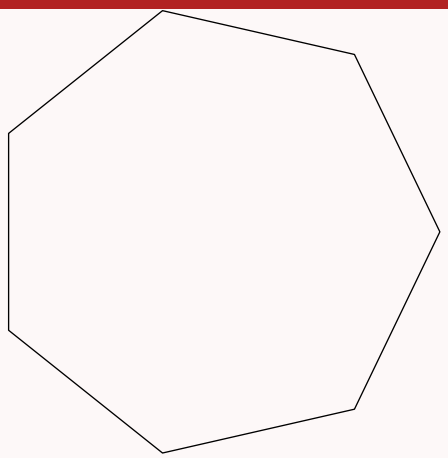
⟨***radius***⟩: `numeric`, radius of the circumscribed circle;

⟨***rotation***⟩: `numeric`, rotation of the polygon;

⟨***translation***⟩: (point) or `vecteur` (or even `pair`), displacement of the polygon's center.

Exemple 20

```
input geom2d;
beginfig(1);
P = RegularPolygon(7,3,0,origine)
    ;
gddDraw P;
endfig;
```



5.10.2 Associated Macros

You can obtain the number of sides of a polygon with the following command.

`PolygonNumberOfSides(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩: `polygone`.

You can access the different vertices of the polygon, numbered starting from 1, using the following macro.

`PolygonPoint(`⟨*a*⟩`,`⟨*i*⟩`)` → `point`

⟨***a***⟩: `polygone`;

⟨***i***⟩: `numeric`, an integer greater than or equal to 1, allowing access to the i^{th} point of the polygon.

5.11 The `chemin` type

For this particular type (french translation of *path*), `mp-geom2d` stores the `path` in the reserved `gddP` table. Thus, the constructor function for this type of object is as follows:

```
vardef Chemin (expr p) =
gddT[incr gddO] = "chemin"; gddP[gddO] = p; gddO
enddef;
```

We defined the english macro `Path` for the english interface.

`Path(`⟨*p*⟩`)` → `chemin`

⟨***p***⟩: a `path` from METAPOST

5.11.1 Associated Macros

The following macro constructs, from a list of `point`s, a polyline of type `chemin`.

`Polyline(`⟨*list*⟩`)` → `chemin`

⟨***list***⟩: is a list of `point` or `pair`.

5.12 The `courbe` type

For this particular type, `mp-geom2d` stores a string (a filename to be used with the `graph.mp` extension) in the reserved `gddS` table.

5.12.1 Constructor

The constructor function for this type of object is as follows:

```
vardef CourbeDat (expr s) =
  gddT[incr gddO] = "courbe"; gddS[gddO] = s; gddO
enddef;
```

The engilsh interface uses the macro `CurveData`.

`CurveData(`⟨*s*⟩`)` → `courbe`

⟨***s***⟩: a filename in the form of a `string` in METAPOST

6 Circle Arc

With `mp-geom2d`, several commands allow working with circle arcs. First, you can choose to represent only a circle arc from a previously defined `cercle`. This is done using the following command, which takes as arguments a `cercle` and two angles in radians, and returns a `path` in METAPOST corresponding to the circle arc between the two given angles (angle 0 being parallel to the Ox axis).

`gddDrawCircleArc(`⟨*C*⟩`,`⟨*a*⟩`,`⟨*b*⟩`)` → `path`

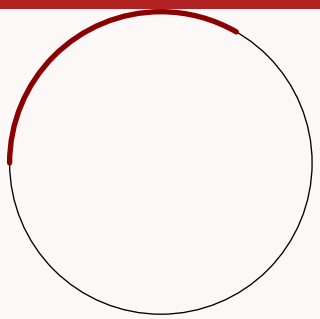
⟨***C***⟩: `cercle` from which we want to represent an arc;

⟨***a***⟩: first angle in radians (`numeric`);

⟨***b***⟩: second angle in radians (`numeric`).

Exemple 21

```
input geom2d;
beginfig(1);
C = Circle(origin,2);
gddDraw C;
gddDraw gddDrawCircleArc(C,Pi/3,
    Pi) withPen(2,DarkRed);
endfig;
```



You can also define a `path` object. For this, you can use the following two commands. The first allows you to define a circular arc from a point, a radius, and two angles.

`Arc(`⟨*P*⟩`,`⟨*r*⟩`,`⟨*a*⟩`,`⟨*b*⟩`)` → `path`

⟨***P***⟩: `point` or `pair`, center of the circular arc;

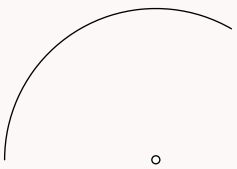
⟨***r***⟩: radius of the circular arc (`numeric`);

⟨***a***⟩: first angle in radians (`numeric`);

⟨***b***⟩: second angle in radians (`numeric`).

Exemple 22

```
input geom2d;
beginfig(1);
A = Point(2,2);
Ac = Arc(A,2,Pi/3,Pi);
gddDrawPoint A;
gddDraw Ac;
endfig;
```



The second command defining a `chemin` is as follows. It takes as arguments three points, denoted as C, A, and B: the center of the circular arc C, and the points A and B. It then takes as an argument the radius of the arc defined between the segments $[C,A]$ and $[C,B]$.

`ArcBetweenPoints(`⟨*P*⟩`,`⟨*r*⟩`,`⟨*A*⟩`,`⟨*B*⟩`,`⟨*s*⟩`)` → `path`

⟨***P***⟩: `point` or `pair`, center of the circular arc;

⟨***r***⟩: radius of the circular arc (`numeric`);

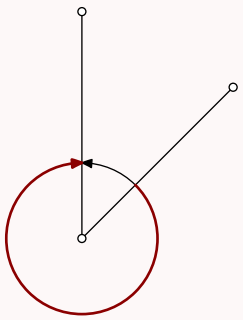
⟨***A***⟩: first `point` or `pair`;

⟨***B***⟩: second `point` or `pair`;

⟨***s***⟩: `numeric`, −1 or 1 depending on the chosen direction.

Exemple 23

```
input geom2d;
beginfig(1);
C = Point(0,0);
A = Point(2,2);
B = Point(0,3);
Cab = ArcBetweenPoints(C,1,A,B,1)
    ;
Cba = ArcBetweenPoints(C,1,A,B
    ,-1);
gddDraw Segment(C,A);
gddDraw Segment(C,B);
gddDrawPoint A; gddDrawPoint B;
    gddDrawPoint C;
gddArrow Cab;
gddArrow Cba withPen(1,DarkRed);
endfig;
```



7 Some Macros

This section groups together some macros that are not specific to a type.

The following macro allows calculating an intersection point between two `mp-geom2d` objects. It is essentially a wrapper around the METAPOST primitive `intersectionpoint` applied to the *drawn* objects (using `gddTraceObjet`).

`PointIntersection(`⟨*o*⟩`,`⟨*b*⟩`)` → `point`

⟨***o***⟩: a `mp-geom2d` object;

⟨***b***⟩: a `mp-geom2d` object.

8 Some Transformations

8.1 Homothety

You can perform a homothety on any `mp-geom2d` object using the following command.

`Homothety(`⟨*p*⟩`,`⟨*o*⟩`,`⟨*k*⟩`)` → same type as ⟨*p*⟩

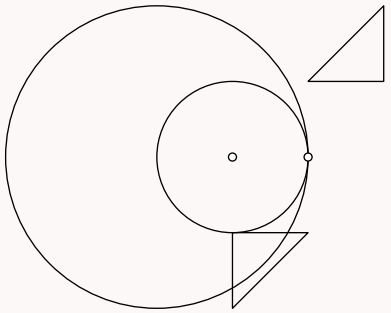
⟨***p***⟩: a `mp-geom2d` object;

⟨***o***⟩: `point` or `pair`, center of the homothety;

⟨***k***⟩: `numeric`, scaling factor of the homothety.

Exemple 24

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Circle(B,1);
T = Triangle((1,1),(0,0),(0,1));
C_H = Homothety(C_B, A, 2);
T_H = Homothety(T, A ,-1);
gddDraw C_B; gddDraw C_H;
gddDraw T; gddDraw T_H;
gddDrawPoint A; gddDrawPoint B;
endfig;
```



8.2 Axial Symmetry

You can perform an axial symmetry on any `mp-geom2d` object using the following command.

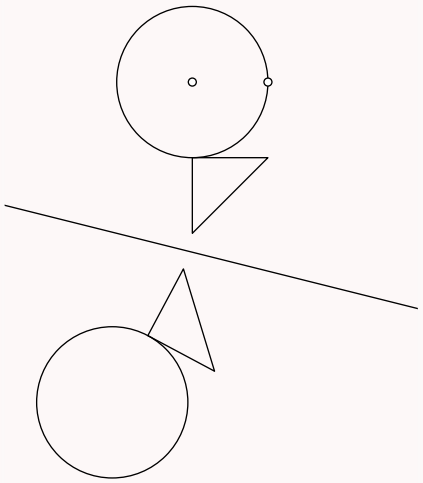
`AxialSymmetry(`⟨*p*⟩`,`⟨*d*⟩`)` → same type as ⟨*p*⟩

⟨***p***⟩: a `mp-geom2d` object;

⟨***d***⟩: `droite` defining the axis of symmetry.

Exemple 25

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Circle(B,1);
T = Triangle((1,1),(0,0),(0,1));
D = Line((-1,0),(3,-1));
C_H = AxialSymmetry(C_B, D);
T_H = AxialSymmetry(T, D);
gddDraw C_B; gddDraw C_H;
gddDraw T; gddDraw T_H;
gddDraw D;
gddDrawPoint A; gddDrawPoint B;
Window(-2.5,-3.5,3,3.2);
endfig;
```



8.3 Central Symmetry

You can perform a central symmetry on any `mp-geom2d` object using the following command.

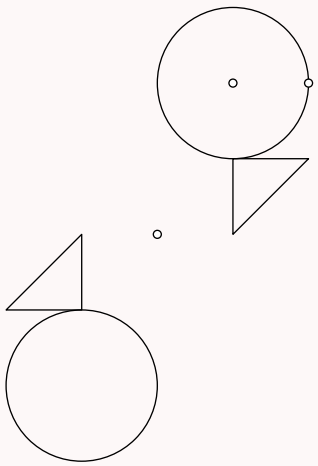
`CentralSymmetry(`⟨*p*⟩`,`⟨*d*⟩`)` → same type as ⟨*p*⟩

⟨***p***⟩: a `mp-geom2d` object;

⟨***d***⟩: `point` defining the center of symmetry.

Exemple 26

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Circle(B,1);
T = Triangle((1,1),(0,0),(0,1));
D = Point(-1,0);
C_H = CentralSymmetry(C_B, D);
T_H = CentralSymmetry(T, D);
gddDraw C_B; gddDraw C_H;
gddDraw T; gddDraw T_H;
gddDrawPoint D;
gddDrawPoint A; gddDrawPoint B;
Window(-3.5,-3.5,3,3.2);
endfig;
```



8.4 Inversion

You can compute the inversion of a point, a circle, or a line with respect to a circle using the following macro:

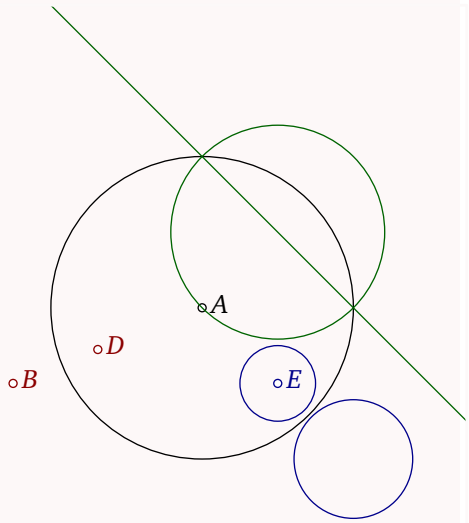
`Inverse(`⟨*p*⟩`,`⟨*c*⟩`)` → `point`

⟨***p***⟩: `point`, `cercle`, or `droite`;

⟨***c***⟩: `cercle`.

Exemple 27

```
input geom2d;
beginfig(1);
  A = Point(2,2);
  C = Circle(A,2);
  gddDraw C;
  gddMark.rt "A";
  B = Point(-0.5,1);
  D = Inverse(B,C);
  drawoptions(withcolor DarkRed);
  gddMark.rt "B"; gddMark.rt "D";

  drawoptions(withcolor DarkBlue)
      ;
  E = Point(3,1);
  CE = Circle(E,0.5);
  gddDraw CE;
  gddMark.rt "E";
  iCE = Inverse(CE,C);
  gddDraw iCE;

  drawoptions(withcolor DarkGreen
      );
  d = Line((3,3),(4,2));
  gddDraw d;
  Cd = Inverse(d,C);
  gddDraw Cd;
  Window(-1,-1,5.5,6);
endfig;
```



9 Annotations and Labels

9.1 Right Angle Marks

`mp-geom2d` provides the following macro to mark a right angle formed by three points:

`OrthoSign(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*c*⟩`,`⟨*x*⟩`)`

⟨***a***⟩,⟨***b***⟩,⟨***c***⟩: are the three `point`s forming the right angle $\widehat{ABC}$;

⟨***x***⟩: is the "size" (`numeric`) of the right angle mark.

You can also create marks between two points or on a segment, a vector, or any type of object. For this, the first macro is as follows:

`Mark(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*n*⟩`)`

⟨***a***⟩: first `point` forming the segment to be marked;

⟨***b***⟩: second `point` forming the segment to be marked;

⟨***n***⟩: is the type (`numeric`) of mark, there are four ⟨*n*⟩= 1, 2, 3, or 4.

You can also mark any type of line with the following macro:

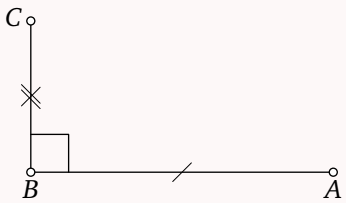
`LineMark(`⟨*a*⟩`,`⟨*n*⟩`)`

⟨***a***⟩: can be any `mp-geom2d` object or even a `path` in METAPOST;

⟨***n***⟩: is the type (`numeric`) of mark, there are four types: ⟨*n*⟩= 1, 2, 3, or 4.

Exemple 28

```
input geom2d;
beginfig(1);
A = Point(4,0);
B = Point(0,0);
C = Point(0,2);
gddDraw Segment(B,A);
gddDraw Segment(B,C);
gddMark.bot "A";
gddMark.bot "B";
gddMark.lft "C";
gddDraw OrthoSign(A,B,C,0.5);
gddDraw Mark(A,B,1);
gddDraw LineMark(Segment(B,C),3);
endfig;
```



9.2 Labels

Since the code produced with `mp-geom2d` is METAPOST code [5], we can use the classic labeling tools. However, to allow more flexibility (and particularly compatibility with

LuaLaTeX and luamplib [1] or minim-mp [4]), if the code is compiled with METAPOST, then the package latexmp [2] is loaded and provides the `textext()` command.

Additionally, `mp-geom2d` provides some commands to make life easier.

Points can be marked using the following command. This command *marks* the point (using the `gddDrawPoint` command, see section 4.3) and adds a label.

`gddMark.`⟨*place*⟩ `"`⟨*name*⟩`"`

⟨***place***⟩: can be the classic placements of METAPOST: `top`, `bot`, `rt`, `lft`, `urt`, `ulft`, `lrt`, `llft`.

⟨***name***⟩: enclosed in double quotes, must be the name of a point variable. The name will be typeset in math mode (enclosed in $). If the name contains an `_`, everything after it will be subscripted (e.g., `A_be` will become A_{be}). If the name is `alpha`, `beta`, `gamma`, or `delta`, the result will be the corresponding Greek letter typeset in math mode.

`mp-geom2d` also provides an adaptation to the classic `label` of METAPOST.

`gddLabel.`⟨*place*⟩`(`⟨*content*⟩`,`⟨*point*⟩`)`

⟨***place***⟩: can be the classic placements of METAPOST: `top`, `bot`, `rt`, `lft`, `urt`, `ulft`, `lrt`, `llft`.

⟨***content***⟩: typically what is given to `label`, a string, or a `picture` (which can be produced, for example, with `btex ... etex` or, since latexmp is loaded by `mp-geom2d`, `textext()`).

⟨***point***⟩: must be a `point` from `mp-geom2d`.

You can also label a `chemin`, a `courbe`, or a `path` using the following macro:

`PathTag.`⟨*place*⟩`(`⟨*content*⟩`,`⟨*path*⟩`,`⟨*position*⟩`)`

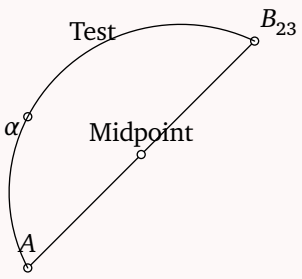
⟨***place***⟩: can be the classic placements of METAPOST: `top`, `bot`, `rt`, `lft`, `urt`, `ulft`, `lrt`, `llft`.

⟨***content***⟩: typically what is given to `label`, a string, or a `picture` (which can be produced, for example, with `btex ... etex` or, since latexmp is loaded by `mp-geom2d`, `textext()`).

⟨***path***⟩: must be a `chemin`, a `courbe`, or a `path`.

Exemple 29

```
input geom2d;
beginfig(1);
A = Point(0,0);
B_23 = Point(3,3);
alpha = Point(0,2);
gddDraw Segment(A,B_23);
C = MidPoint(A,B_23);
gddDrawPoint C;
gddLabel.top(textext("Midpoint"),
    C);
gddMark.top "A";
gddMark.urt "B_23";
gddMark.llft "alpha";
P = Path(Pt(A)..Pt(alpha)..Pt(B_
    23));
gddDraw P;
PathTag.top("Test",P,0.6);
endfig;
```



10 Coordinate System

`mp-geom2d` provides a set of commands to facilitate the drawing of coordinate systems.

The main command is the definition of the coordinate system, that is, the box in which the drawing will be represented.

`Frame(`⟨*l*⟩`,`⟨*h*⟩`,`⟨*ox*⟩`,`⟨*oy*⟩`,`⟨*ux*⟩`,`⟨*uy*⟩`)`

⟨***l***⟩: `numeric`, width of the coordinate system (in `gddU` units);

⟨***h***⟩: `numeric`, height of the coordinate system (in `gddU` units);

⟨***ox***⟩: `numeric`, distance (in `gddU` units) from the origin (`point` (0,0)) to the left edge;

⟨***oy***⟩: `numeric`, distance (in `gddU` units) from the origin (`point` (0,0)) to the bottom edge;

⟨***ux***⟩: `numeric`, unit of the (Ox) axis (in `gddU` units);

⟨***uy***⟩: `numeric`, unit of the (Oy) axis (in `gddU` units).

This command does not return or draw anything. It serves to specify some internal variables for drawing definition. It also modifies the behavior of the `gddInPlace` macro (see page 5). Note that this command requires that the origin (i.e., the `point` (0,0)) is inside the coordinate system.

This macro is accompanied by two others, also *silent*, which serve only to:

- save the current `picture`;
- build a `picture` with the content found between `gddBegin` and `gddEnd`;

- crop (with `clip`) the `picture` to the frame of the constructed coordinate system;
- add the current `picture` to the saved one;
- finally, restore the behavior of `gddInPlace` as it was before using `Frame`.

These two commands are called `gddBegin` and `gddEnd`.

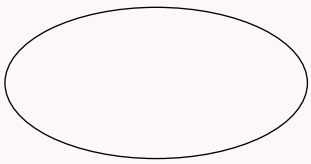
`gddBegin`

`gddEnd`

Thus, it is very easy to illustrate that, up to a change of orthogonal basis, a circle is an ellipse.

Exemple 30

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
Frame(5,5,1,1,1,0.5);
gddBegin;
gddDraw CA;
gddEnd;
endfig;
```



There are two commands available to draw the axes of the coordinate system. The first draws the axes passing through the origin.

`Axis` *(draws a set of elements on the coordinate system)*

This command also adds the labels for the x-axis and y-axis, which are stored in the following dedicated global variables:

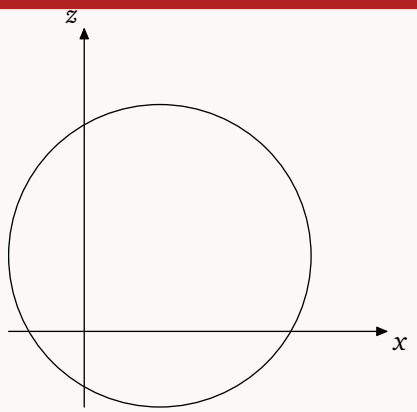
`gddXlabel` (*`string`, default value `"$x$"`*)

`gddYlabel` (*`string`, default value `"$y$"`*)

Note, these commands must be used before calling `gddBegin`.
The following example illustrates how to draw the axes.

Exemple 31

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
Frame(5,5,1,1,1,1);
gddYlabel:="$z$";
Axis;
gddBegin;
gddDraw CA;
gddEnd;
endfig;
```



You can also define a coordinate system using a syntax that allows you to specify the minimum and maximum abscissas and ordinates.

`FrameMinMax(`⟨*xmin*⟩`,`⟨*xmax*⟩`,`⟨*ymin*⟩`,`⟨*ymax*⟩`,`⟨*ux*⟩`,`⟨*uy*⟩`)`

⟨***xmin***⟩: `numeric`, minimum abscissa (in `gddU` units).

⟨***xmax***⟩: `numeric`, maximum abscissa (in `gddU` units).

⟨***ymin***⟩: `numeric`, minimum ordinate (in `gddU` units).

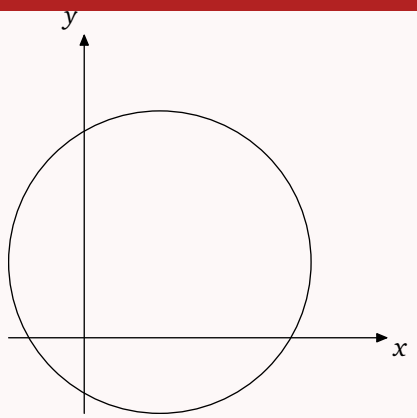
⟨***ymax***⟩: `numeric`, maximum ordinate (in `gddU` units).

⟨***ux***⟩: `numeric`, unit of the (Ox) axis (in `gddU` units).

⟨***uy***⟩: `numeric`, unit of the (Oy) axis (in `gddU` units).

Exemple 32

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
Axis;
gddBegin;
gddDraw CA;
gddEnd;
endfig;
```

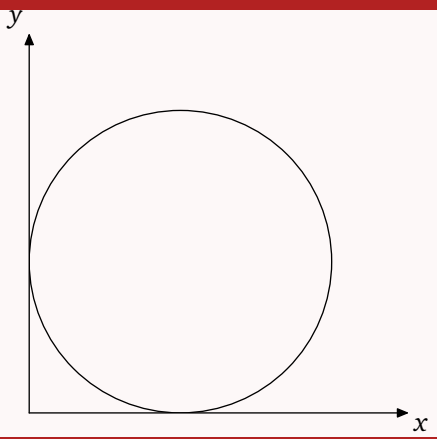


`mp-geom2d` also provides a way to draw axes on the edge of the frame with the following command.

`SideAxis` *(draws a set of elements on the coordinate system)*

Exemple 33

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
SideAxis;
gddBegin;
gddDraw CA;
gddEnd;
endfig;
```



The following commands allow you to add graduations to the axes (either classic or on the edge). Again, these commands take no arguments, return nothing, and simply draw.

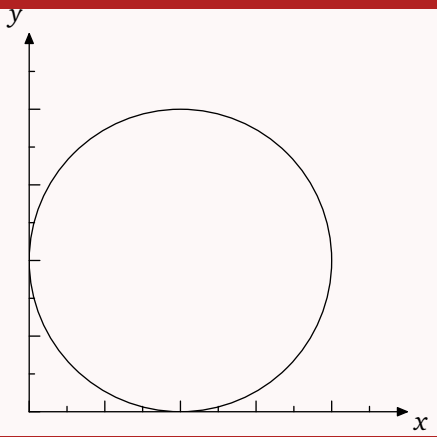
`Graduations` *(draws a set of elements on the coordinate system)*

`SideGraduations` *(draws a set of elements on the coordinate system)*

Use the version consistent with the chosen axes.

Exemple 34

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
SideAxis;
SideGraduations;
gddBegin;
gddDraw CA;
gddEnd;
endfig;
```



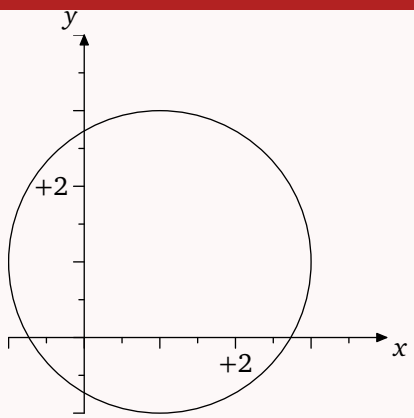
If you want to mark the units, `mp-geom2d` provides the following macro (which can only be used if the axes are not placed on the border).

`Units(`⟨*unit*⟩`)` *(draws a set of elements on the coordinate system)*

⟨***unit***⟩: `numeric`, *value to be displayed on both axes.*

Exemple 35

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
Axis;
gddBegin;
  Graduations;
  Units(2);
  gddDraw CA;
gddEnd;
endfig;
```



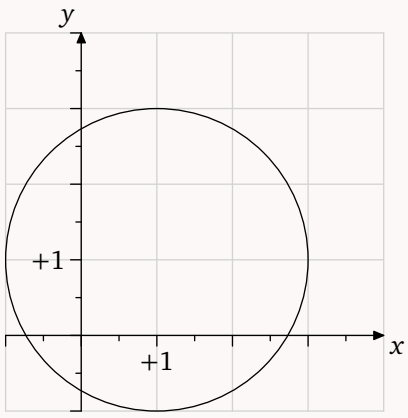
You can also add a grid to your coordinate system with the following macro.

`FrameGrid` *(draws a set of elements on the coordinate system)*

In the following example, the color of the grid is set using the MetaPost `drawoptions` command.

Exemple 36

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
drawoptions(withcolor LightGrey);
FrameGrid;
drawoptions();
Axis;
gddBegin;
  Graduations;
  Units(1);
  gddDraw CA;
gddEnd;
endfig;
```



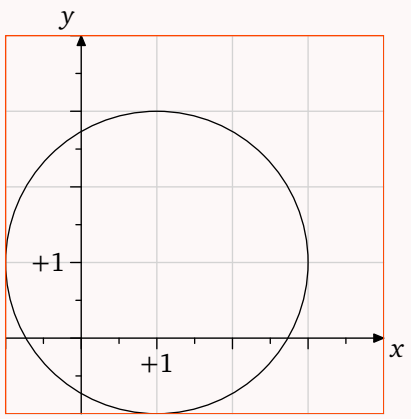
You can also add a frame to the coordinate system with the following macro.

FrameBox → path

In the following example, the color of the grid is set using the MetaPost drawoptions command.

Exemple 37

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Circle(A,2);
FrameMinMax(-1,4,-1,4,1,1);
drawoptions(withcolor LightGrey);
FrameGrid;
drawoptions();
Axis;
gddBegin;
  Graduations;
  Units(1);
  gddDraw CA;
  gddDraw FrameBox withcolor
      OrangeRed;
gddEnd;
endfig;
```



11 Some Mathematical Constants and Functions

mp-geom2d defines two mathematical constants, Pi and _E , for the constants $\pi \simeq 3.14159265$ and $e = 2.71828183$.

In addition, the package defines some mathematical functions of a real variable:

sin(⟨x⟩)

cos(⟨x⟩)

tan(⟨x⟩)

exp(⟨x⟩)

ln(⟨x⟩)

cosh(⟨x⟩)

sinh(⟨x⟩)

tanh(⟨x⟩)

`arcsin(`⟨*x*⟩`)`

`arccos(`⟨*x*⟩`)`

`arctan(`⟨*x*⟩`)`

12 Representation of Curves and Functions

`mp-geom2d` also provides some macros to facilitate the simple representation of curves and mathematical functions.

12.1 Function of a Real Variable

To represent a function of a real variable, use the following macro:

`Plot(`⟨*function*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***function***⟩**:** is a METAPOST macro that defines the mathematical function of a real variable to be represented;

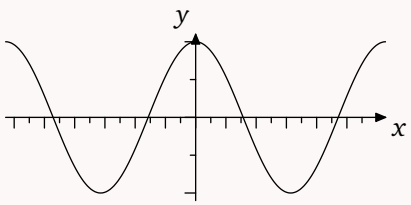
⟨***ti***⟩**:** is the initial value (`numeric`) of the variable from which to start building the representation of the function;

⟨***tf***⟩**:** is the final value (`numeric`) of the variable up to which to build the representation of the function;

⟨***n***⟩**:** is the number of steps (`numeric`) used for the discretization of the representation.

Exemple 38

```
input geom2d;
beginfig(1);
FrameMinMax(-2Pi,2Pi
    ,-1.1,1.1,0.4,1);
Axis;
gddBegin;
  Graduations;
  gddDraw Plot(cos,-2Pi,2Pi,100);
gddEnd;
endfig;
```



12.2 Parametric Curve

To represent a plane curve defined by two functions describing the abscissa and ordinate as functions of a parameter, use the following macro:

`Curve(`⟨*abscissa_function*⟩`)(`⟨*ordinate_function*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***abscissa_function***⟩**:** is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the abscissa of the points of the curve to be represented;

⟨***ordinate_function***⟩**:** is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the ordinate of the points of the curve to be represented;

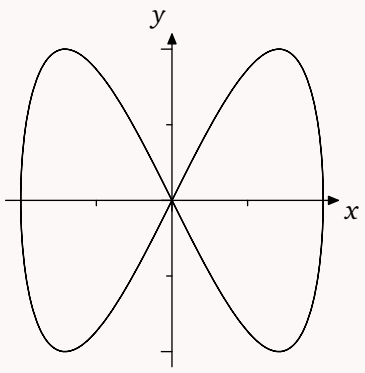
⟨***ti***⟩**:** is the initial value (`numeric`) of the variable from which to start building the representation of the function;

⟨***tf***⟩**:** is the final value (`numeric`) of the variable up to which to build the representation of the function;

⟨***n***⟩**:** is the number of steps (`numeric`) used for the discretization of the representation.

Exemple 39

```
input geom2d;
vardef f_x(expr t)= cos(t) enddef
    ;
vardef f_y(expr t)= sin(2*t)
    enddef;

beginfig(1);
FrameMinMax
    (-1.1,1.1,-1.1,1.1,2,2);
Axis;
gddBegin;
  Graduations;
  gddDraw Curve(f_x,f_y,-2Pi,2Pi
      ,100);
gddEnd;
endfig;
```



To represent a plane curve defined by a function describing the polar coordinates as a function of a parameter, use the following macro:

`PolarCurve(`⟨*function*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***function***⟩**:** is a METAPOST macro that defines the mathematical function of a real variable describing the evolution of the polar coordinates of the points of the curve to be represented;

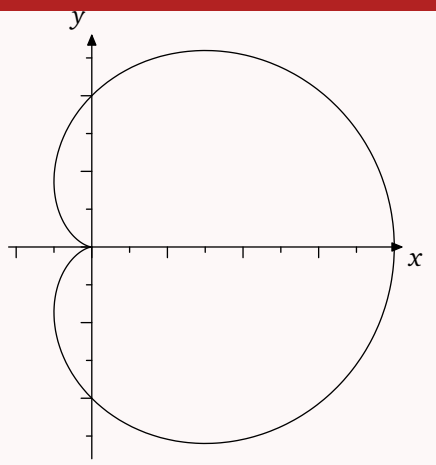
⟨***ti***⟩**:** is the initial value (`numeric`) of the variable from which to start building the representation of the function;

⟨***tf***⟩: is the final value (numeric) of the variable up to which to build the representation of the function;

⟨***n***⟩: is the number of steps (numeric) used for the discretization of the representation.

Exemple 40

```
input geom2d;
a := 2;
vardef cardioid(expr t)= a*(1+cos
    (t)) enddef;
beginfig(1);
FrameMinMax
    (-1.1,4.1,-2.8,2.8,1,1);
Axis;
gddBegin;
 Graduations;
 gddDraw PolarCurve(cardioid,0,2
     Pi,100);
gddEnd;
endfig;
```



12.3 Vector Fields

mp-geom2d provides macros for representing vector fields.

First, you can plot vector fields associated with a first-order differential equation for a function y of the variable x:

$$y' = F(x, y).$$

The associated macro is as follows:

VectorField(⟨*function*⟩)(⟨*x*⟩,⟨*y*⟩,⟨*px*⟩,⟨*py*⟩,⟨*dx*⟩,⟨*color*⟩) → path

⟨***function***⟩: is a METAPOST macro that defines a mathematical function from $\mathbf{R}^2$ to $\mathbf{R}$;

⟨***x***⟩: is a value (numeric), in gddU units, which allows shifting the grid of vectors along the x direction, with points at all ⟨*x*⟩+*i*⟨*px*⟩ for integer i;

⟨***y***⟩: is a value (numeric), in gddU units, which allows shifting the grid of vectors along the y direction, with points at all ⟨*y*⟩+*i*⟨*py*⟩ for integer i;

⟨***px***⟩: is the value (numeric), in gddU units, of the grid step along the x-axis for the representation of vectors;

⟨***py***⟩: is the value (numeric), in gddU units, of the grid step along the y-axis for the representation of vectors;

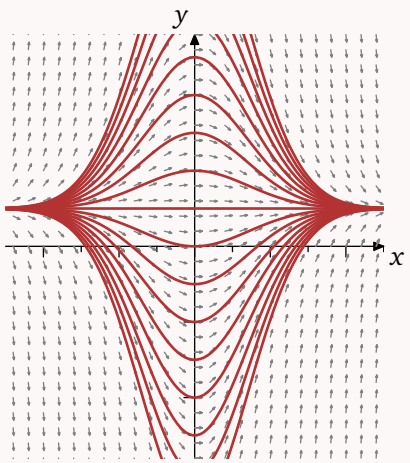
⟨***dx***⟩: is the norm (numeric), in gddU units, of the vectors in the vector field;

⟨***color***⟩: is the (color) used to draw the vectors.

Since this macro uses the drawarrow macro from METAPOST, you will need to adjust the ahlength parameter to set the size of the arrowhead triangle (see [5]).

Exemple 41

```
input geom2d;
vardef F(expr x,y) = x - 2 * x *
    y enddef;
vardef f(expr x) = 1/2 + a * exp
    (- x*x) enddef;
beginfig(1);
FrameMinMax
    (-2.5,2.5,-2.8,2.8,1,1);
Axis;
gddBegin;
Graduations;
ahlength := 1;
VectorField(F,0,0,0.2,0.2,0.1,0.5
    white);
% Integral curves
for n = 0 upto 16:
  a := (n/2) - 4;
  gddDraw Plot(f,-2.5,2.5,50)
  withPen(1,(0.7,0.2,0.2));
endfor
gddEnd;
endfig;
```



Similarly, you can plot vector fields of a function from $\mathbf{R}^2$ to $\mathbf{R}^2$. The associated macro is as follows:

VectorFieldDD(⟨*function*⟩)(⟨*x*⟩,⟨*y*⟩,⟨*px*⟩,⟨*py*⟩,⟨*dx*⟩,⟨*color*⟩) → path

⟨***function***⟩: is a METAPOST macro that defines a mathematical function from $\mathbf{R}^2$ to $\mathbf{R}^2$ (thus returning a pair);

⟨***x***⟩: is a value (numeric), in gddU units, which shifts the grid of vectors along the x direction, with points at all $\langle x\rangle+i\langle px\rangle$ for integer i;

⟨***y***⟩: is a value (numeric), in gddU units, which shifts the grid of vectors along the y direction, with points at all $\langle y\rangle+i\langle py\rangle$ for integer i;

⟨***px***⟩: is the value (numeric), in gddU units, of the grid step along the x-axis for the representation of vectors;

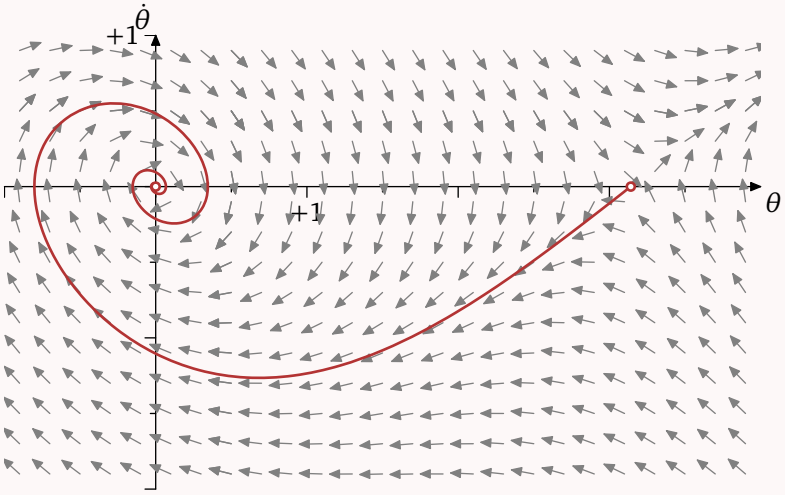
⟨***py***⟩: is the value (numeric), in gddU units, of the grid step along the y-axis for the representation of vectors;

⟨***dx***⟩: is the norm (numeric), in gddU units, of the vectors in the vector field;

⟨***color***⟩: is the (color) used to draw the vectors.

Exemple 42

```
input geom2d;
gddXlabel := "$\theta$";
gddYlabel := "$\dot\theta$";
numeric b,c;
b:=0.5;
c=1.0;

vardef F(expr x,y) = (y,-b*y-c*sin(x)) enddef;

beginfig(1);
Frame(10,6,2,4,2,2);
Axis;
Units(1);
gddBegin;
Graduations;
trajectory := CurveData("solution0",0);
VectorFieldDD(F,0.5,0.5,0.2,0.2,0.15,0.5white);
gddDraw trajectory withPen(1,(0.7,0.2,0.2));
gddDrawPoint Point(0,0) withPen(1,(0.7,0.2,0.2));
gddDrawPoint Point(3.1415,0) withPen(1,(0.7,0.2,0.2));
gddEnd;
endfig;
```



13 SVG Colors (`svgnames`)

AliceBlue
Azure
BlanchedAlmond
BurlyWood
Coral
Cyan
DarkGray
DarkMagenta
DarkRed
DarkSlateGray
DeepPink
DodgerBlue
Fuchsia
Goldenrod
Grey
Indigo
LavenderBlush
LightCoral
LightGray
LightSalmon
LightSlateGray
Lime
Maroon
MediumPurple
MediumTurquoise
MistyRose
NavyBlue
Orange
PaleGreen
PeachPuff
PowderBlue
RoyalBlue
SeaGreen
SkyBlue
Snow
Teal
Violet
WhiteSmoke

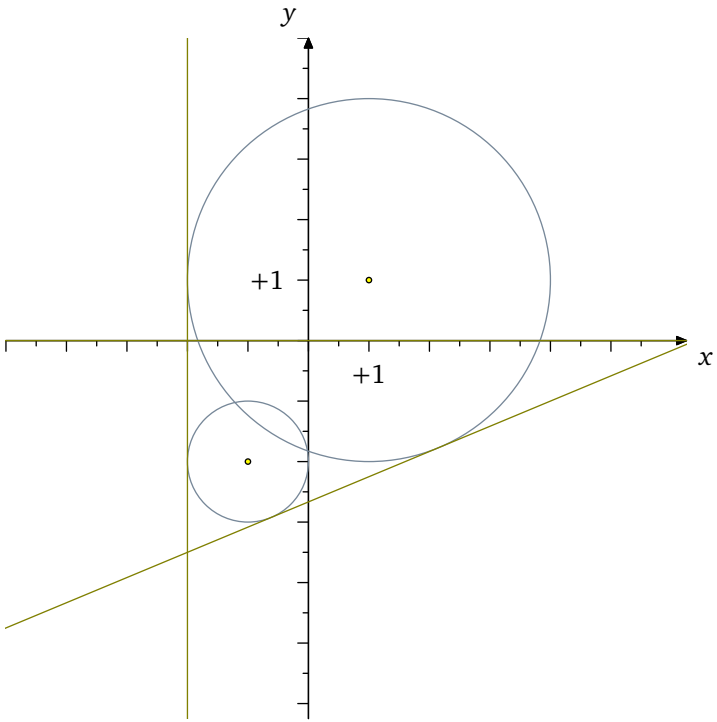
AntiqueWhite
Beige
Blue
CadetBlue
CornflowerBlue
DarkBlue
DarkGreen
DarkOliveGreen
DarkSalmon
DarkSlateGrey
DeepSkyBlue
FireBrick
Gainsboro
Gray
Honeydew
Ivory
LawnGreen
LightCyan
LightGreen
LightSeaGreen
LightSlateGrey
LimeGreen
MediumAquamarine
MediumSeaGreen
MediumVioletRed
Moccasin
OldLace
OrangeRed
PaleTurquoise
Peru
Purple
SaddleBrown
Seashell
SlateBlue
SpringGreen
Thistle
VioletRed
Yellow

Aqua
Bisque
BlueViolet
Chartreuse
Cornsilk
DarkCyan
DarkGrey
DarkOrange
DarkSeaGreen
DarkTurquoise
DimGray
FloralWhite
GhostWhite
Green
HotPink
Khaki
LemonChiffon
LightGoldenrod
LightGrey
LightSkyBlue
LightSteelBlue
Linen
MediumBlue
MediumSlateBlue
MidnightBlue
NavajoWhite
Olive
Orchid
PaleVioletRed
Pink
Red
Salmon
Sienna
SlateGray
SteelBlue
Tomato
Wheat
YellowGreen

Aquamarine
Black
Brown
Chocolate
Crimson
DarkGoldenrod
DarkKhaki
DarkOrchid
DarkSlateBlue
DarkViolet
DimGrey
ForestGreen
Gold
GreenYellow
IndianRed
Lavender
LightBlue
LightGoldenrodYellow
LightPink
LightSlateBlue
LightYellow
Magenta
MediumOrchid
MediumSpringGreen
MintCream
Navy
OliveDrab
PaleGoldenrod
PapayaWhip
Plum
RosyBrown
SandyBrown
Silver
SlateGrey
Tan
Turquoise
White

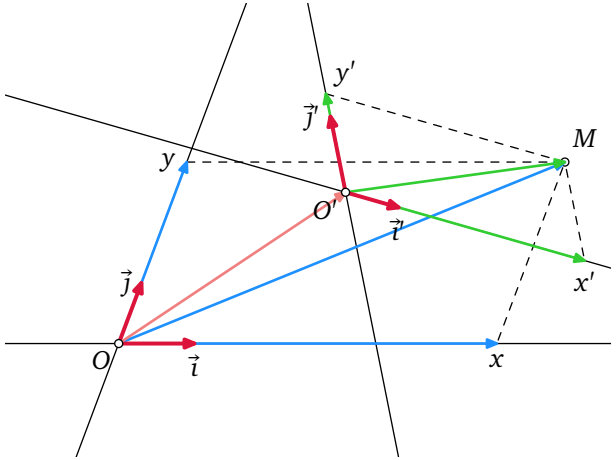
14 Gallery

14.1 Coordinate System and External Tangents




```
input geom2d;

labeloffset := 6;
SetPointSize(2);
SetPointColor(Yellow);

beginfig(1);
 Frame(9,9,4,5,0.8,0.8);
 Axis;
 gddBegin;
   Axis;
   Graduations;
   Units(1);

   C1 = Circle((1,1),3);
   C2 = Circle((-1,-2),1);

   T1 = ExternalCommonTangent(C1,C2);
   T2 = ExternalCommonTangent(C2,C1);

   drawoptions(withcolor LightSlateGrey);
   gddDraw C1;
   gddDraw C2;

   drawoptions(withcolor Olive);
   gddDraw T1;
   gddDraw T2;
   gddDraw Xaxis;

   drawoptions();
   gddDrawPoint Point(1,1);
   gddDrawPoint Point(-1,-2);
 gddEnd;
endfig;
end
```

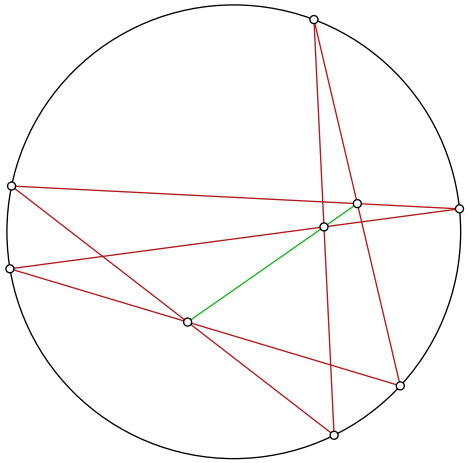
14.2 Vector in a Coordinate System




```
input geom2d;

labeloffset := 4;
gddU:=1.cm;

beginfig(1);
  O = Point(0,0);
  I = Point(1,0);
  J = Point(0.3,0.8);
  M = PointInBasis(5,3,O,I,J);
  H = PointInBasis(5,0,O,I,J);
  K = PointInBasis(0,3,O,I,J);
  O' = Point(3,2);
  I' = Point(3.7,1.8);
  J' = Point(2.8,3);

  pair Mt;
  Mt = CoordinatesInBasis(M,O',I',J');
  H' = PointInBasis(xpart Mt,0,O',I',J');
  K' = PointInBasis(0,ypart Mt,O',I',J');

  Frame(8,6,1.5,1.5,1,1);
  gddBegin;
    gddDraw Line(O,I);
    gddDraw Line(O,J);
    gddDraw Line(O',I');
    gddDraw Line(O',J');

    gddDraw Pt(K)--Pt(M)--Pt(H) dashed evenly;
    gddDraw Pt(K')--Pt(M)--Pt(H') dashed evenly;

    gddMark.urt "M";

    drawoptions(withpen pencircle scaled 1pt withcolor DodgerBlue);
    gddArrow Segment(O,K);
    gddArrow Segment(O,H);
    gddArrow Segment(O,M);

    drawoptions(withpen pencircle scaled 1pt withcolor LimeGreen);
    gddArrow Segment(O',M);
    gddArrow Segment(O',H');
    gddArrow Segment(O',K');

    drawoptions(withpen pencircle scaled 1.5pt withcolor Crimson);
    gddArrow Segment(O,I);
    gddArrow Segment(O,J);
    gddArrow Segment(O',I');
    gddArrow Segment(O',J');

    drawoptions(withpen pencircle scaled 1pt withcolor LightCoral);
    gddArrow Segment(O,O');

    drawoptions();
    gddMark.llft "O";
    gddMark.llft "O'";

    label.bot(textext("\(x\)"), PtR(H));
    label.lft(textext("\(y\)"), PtR(K));
    label.bot(textext("\(\vec\imath\)"), PtR(I));
    label.lft(textext("\(\vec\jmath\)"), PtR(J));
```

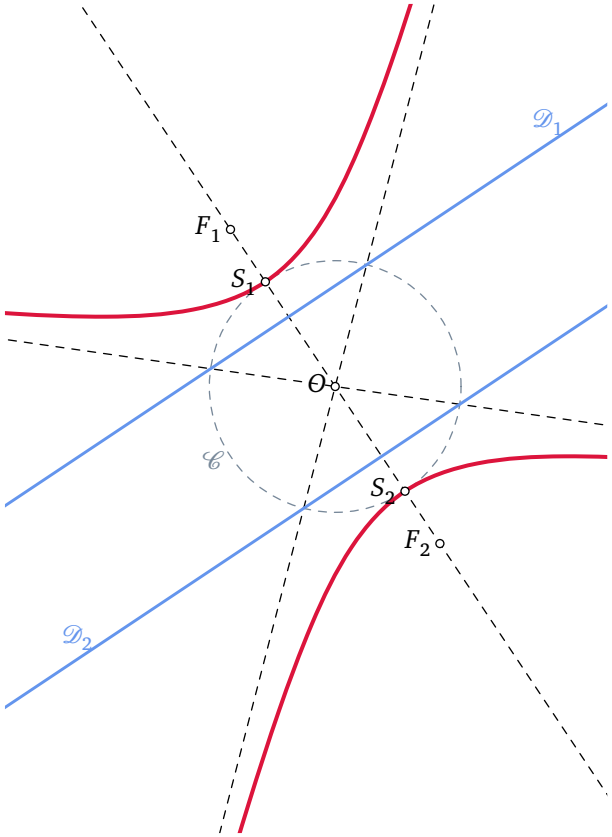
```
61
62    label.bot(textext("\(x'\)"), PtR(H'));
63    label.urt(textext("\(y'\)"), PtR(K'));
64    label.bot(textext("\(\vec\imath'\)"), PtR(I'));
65    label.lft(textext("\(\vec\jmath'\)"), PtR(J'));
66
67    gddEnd;
68  endfig;
69  end
```

14.3 Pascal's Theorem



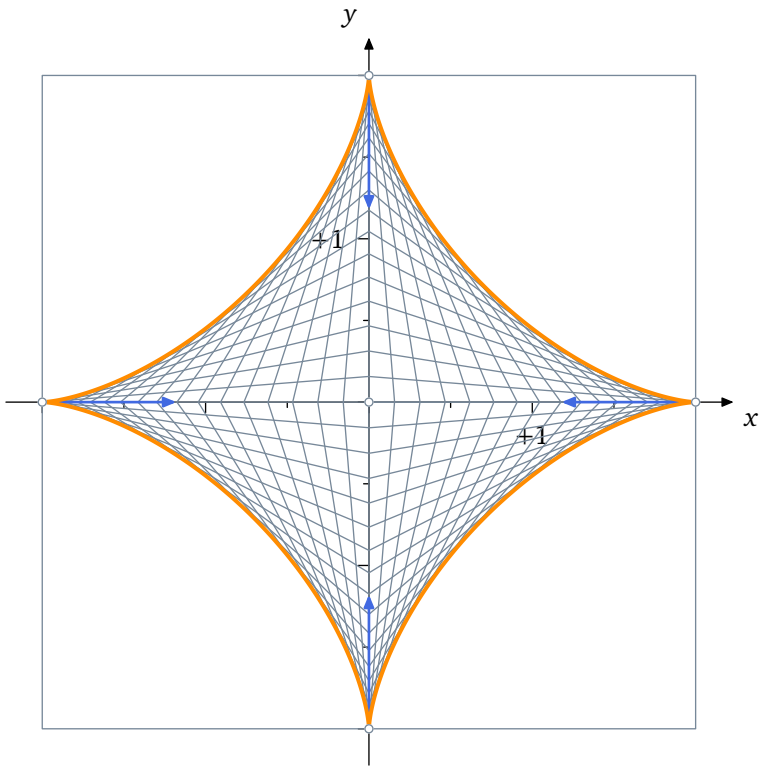
```
input geom2d;

beginfig(1);
C = Circle(origine,3);
for i:=1 upto 6:
   rd := uniformdeviate(1.0/6)+(i-1)/6;
   P[i] := PointOf(C,rd);
endfor;
D1 = Line(P1,P3); S1 = Segment(P1,P3);
D2 = Line(P3,P5); S2 = Segment(P3,P5);
D3 = Line(P6,P2); S3 = Segment(P6,P2);
D4 = Line(P4,P6); S4 = Segment(P4,P6);
D5 = Line(P5,P2); S5 = Segment(P5,P2);
D6 = Line(P1,P4); S6 = Segment(P1,P4);
I1 = LinesIntersection(D1,D3);
I2 = LinesIntersection(D2,D4);
I3 = LinesIntersection(D5,D6);
PL = Segment(I1,I2);
gddDraw C;
drawoptions(withcolor (0.7,0.1,0.1));
gddDraw S1; gddDraw S2; gddDraw S3; gddDraw S4; gddDraw S5; gddDraw S
    6;
drawoptions(withcolor (0.1,0.7,0.1));
gddDraw PL;
drawoptions();
gddDrawPoint P1; gddDrawPoint P2; gddDrawPoint P3; gddDrawPoint P4;
    gddDrawPoint P5; gddDrawPoint P6;
gddDrawPoint I1; gddDrawPoint I2; gddDrawPoint I3;
endfig;
end.
```

14.4 Hyperbola



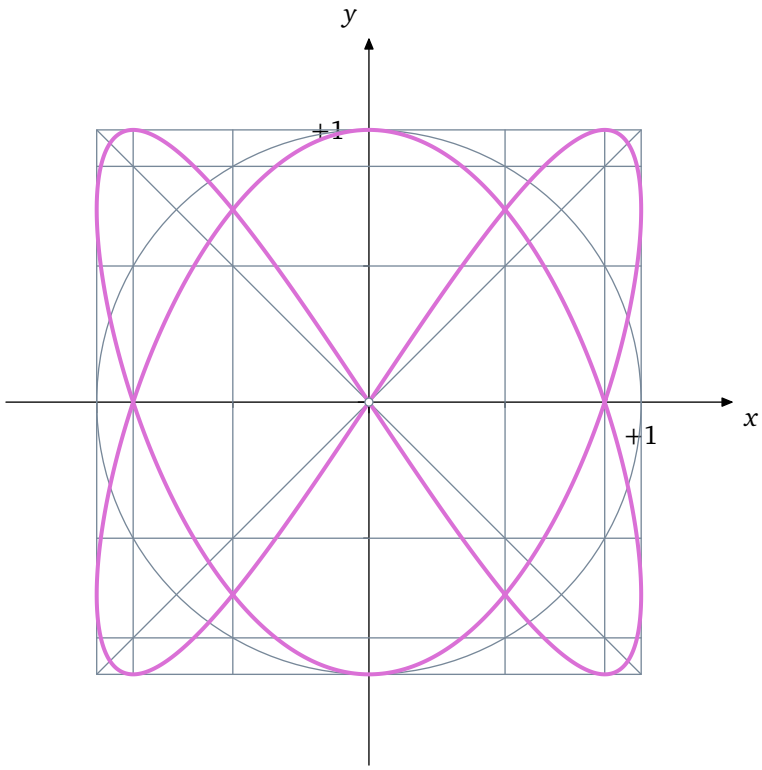

```
input geom2d;

beginfig(1);
  A = Point(0,0);
  B = Point(3,2);
  AB = Line(A,B);
  F_1 = Point(-1,1);
  Hyper = HyperbolaFD(F_1,AB,1.5);
  O  = Center(Hyper);
  F_2 = Focus(Hyper,2);
  Axe = Line(F_1,F_2);
  S_1 = Sommet(Hyper,1);
  S_2 = Sommet(Hyper,2);

  gddDraw Axe dashed evenly;
  C = PrincipalCircle(Hyper) ;
  gddDraw C withPen(0.5,LightSlateGray) dashed evenly;
  A_1 = HyperbolaAsymptote(Hyper,1);
  A_2 = HyperbolaAsymptote(Hyper,2);
  gddDraw A_1 dashed evenly;
  gddDraw A_2 dashed evenly;

  D_1 = Directrix(Hyper,1);
  D_2 = Directrix(Hyper,2);

  gddDraw D_1 withPen(1.1,CornflowerBlue);
  gddDraw D_2 withPen(1.1,CornflowerBlue);
  gddDraw SemiHyperbola(Hyper,1) withPen(1.5,Crimson);
  gddDraw SemiHyperbola(Hyper,2) withPen(1.5,Crimson);

  gddMark.lft "O";
  gddMark.lft "S_1";
  gddMark.lft "S_2";
  gddMark.lft "F_1";
  gddMark.lft "F_2";
  gddLabel.lft(textext("\(\mathcal{C}\)"),PointDe(C,0.6)) withcolor
       LightSlateGray;
  gddLabel.top(textext("\(\mathcal{D}_1\)"),PointDe(D_1,0.47))
       withcolor CornflowerBlue;
  gddLabel.top(textext("\(\mathcal{D}_2\)"),PointDe(D_2,0.46))
       withcolor CornflowerBlue;
  Window(-4,-7,4,4);
endfig;


end.
```

14.5 Astroid as an Envelope of Lines



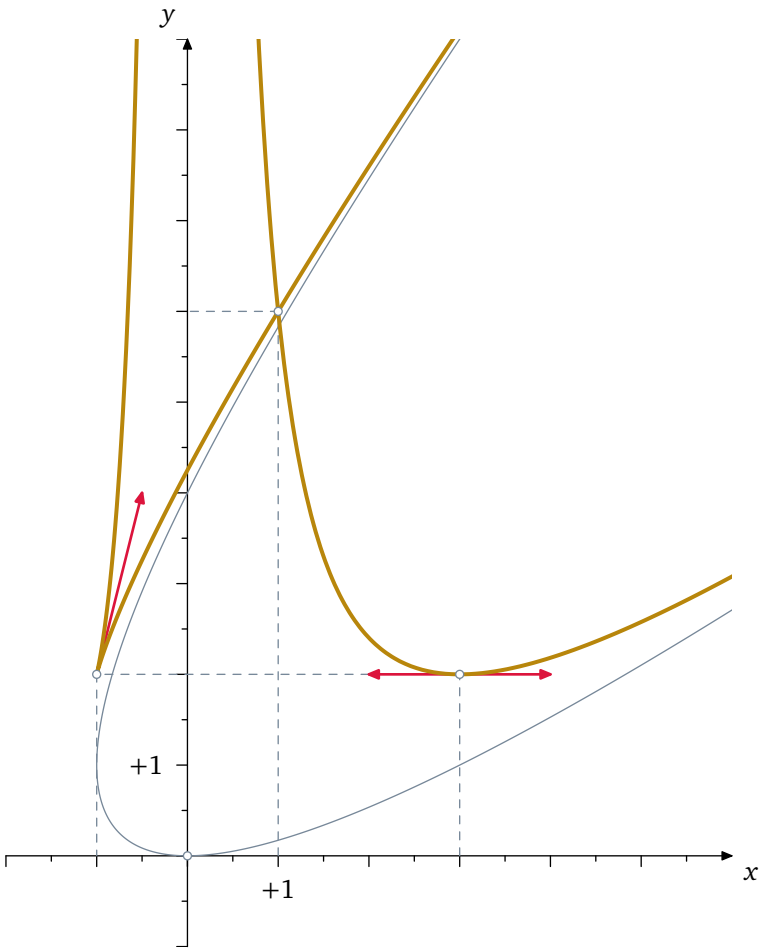

```
input geom2d;

labeloffset := 6;
gddU:=1.2cm;

vardef f(expr t) = 2*cos(t)*cos(t)*cos(t) enddef;
vardef g(expr t) = 2*sin(t)*sin(t)*sin(t) enddef;


beginfig(1);
 Frame(8,8,4,4,1.8,1.8);
 Axis;
 gddBegin;
  Graduations; Units(1);
  drawoptions(withpen pencircle scaled 0.5 withcolor LightSlateGrey);
  gddDraw ((-2,-2)--(-2,2)--(2,2)--(2,-2)--cycle);

  nb = 80;
  pas = 2*Pi/nb;
  for i=0 upto nb: gddDraw (0,2*sin(i*pas))--(2*cos(i*pas),0); endfor
       ;

  gddArrow Segment((2,0),(1.2,0)) withPen(1,RoyalBlue);
  gddArrow Segment((-2,0),(-1.2,0)) withPen(1,RoyalBlue);
  gddArrow Segment((0,2),(0,1.2)) withPen(1,RoyalBlue);
  gddArrow Segment((0,-2),(0,-1.2)) withPen(1,RoyalBlue);

  gddDraw Curve(f,g,0,2*Pi,500) withPen(1.5,DarkOrange);

  gddDrawPoint Point(0,0);
  gddDrawPoint Point(2,0);
  gddDrawPoint Point(0,2);
  gddDrawPoint Point(0,-2);
  gddDrawPoint Point(-2,0);

 gddEnd;
endfig;
end
```

14.6 Lissajous Curve



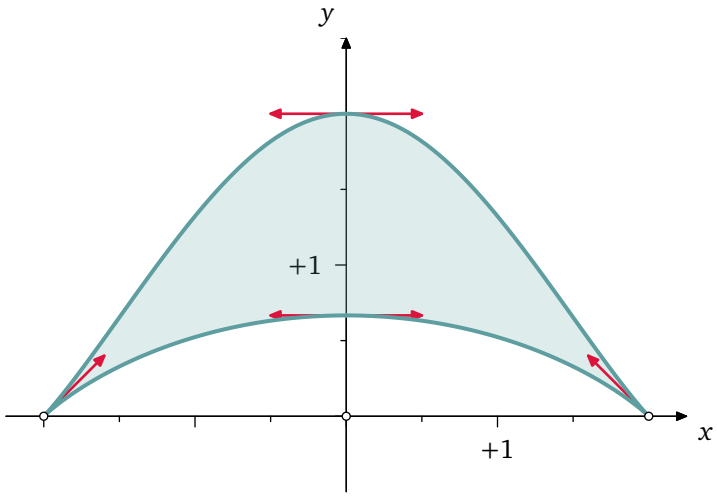

```
input geom2d;

labeloffset := 6;
gddU:=1.2cm;

vardef f(expr t) = sin(2*t+Pi/3) enddef;
vardef g(expr t) = sin(3*t) enddef;

beginfig(1);
 Frame(8,8,4,4,3,3);
 Axis;
 gddBegin;
  Graduations; Units(1);

  drawoptions(withcolor LightSlateGrey);
  gddDraw ((-1,-1)--(-1,1)--(1,1)--(1,-1)--cycle);
  gddDraw ((-1,-1)--(1,1));
  gddDraw (1,-1)--(-1,1);
  gddDraw Circle(origine,1);
  gddDraw ((-1,0.5)--(1,0.5));
  gddDraw ((-1,-0.5)--(1,-0.5));
  gddDraw ((0.5,-1)--(0.5,1));
  gddDraw ((-0.5,-1)--(-0.5,1));
  gddDraw ((-1,sqrt(3)/2)--(1,sqrt(3)/2));
  gddDraw ((-1,-sqrt(3)/2)--(1,-sqrt(3)/2));
  gddDraw ((sqrt(3)/2,-1)--(sqrt(3)/2,1));
  gddDraw ((-sqrt(3)/2,-1)--(-sqrt(3)/2,1));

  gddDraw Curve(f,g,0,2*Pi,500) withPen(1.5,Orchid);

  pointe Point(0,0);

 gddEnd;
endfig;
end
```

14.7 Function Study



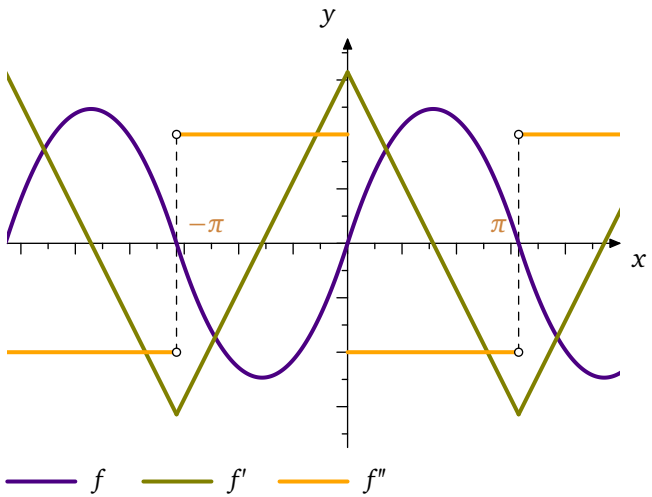

```
input geom2d;

labeloffset := 6;
gddU:=1.2cm;

vardef f(expr t) = t*t-2*t enddef;
vardef g(expr t) = t*t + 1/(t*t) enddef;
vardef fp(expr t) = 2*t+t*t enddef;
vardef gp(expr t) = t*t enddef;

beginfig(1);

 Frame(8,10,2,1,1,1);
 Axis;
 gddBegin;
  Graduations; Units(1);

  drawoptions(withcolor LightSlateGrey);
  gddDraw (3,0)--(3,2)--(-1,2)--(-1,0) dashed evenly;
  gddDraw (1,0)--(1,6)--(0,6) dashed evenly;
  gddDraw Curve(fp,gp,-5,5,500);

  gddArrow Segment((-1,2),(-1,2)+(0.5,2)) withPen(1,Crimson);
  gddArrow Segment((3,2),(4,2)) withPen(1,Crimson);
  gddArrow Segment((3,2),(2,2)) withPen(1,Crimson);

  gddDraw Curve(f,g,-10,-0.05,300) withPen(1.5,DarkGoldenrod);
  gddDraw Curve(f,g,0.05,10,300) withPen(1.5,DarkGoldenrod);

  gddDrawPoint Point(0,0);
  gddDrawPoint Point(-1,2);
  gddDrawPoint Point(3,2);
  gddDrawPoint Point(1,6);

 gddEnd;
endfig;
end
```

14.8 Bicorne



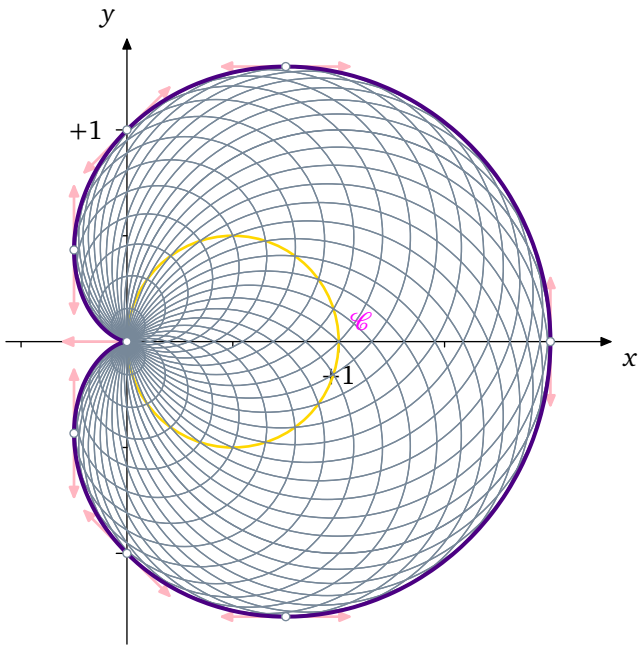

```
input geom2d;

labeloffset := 6;
gddU:=1cm;

vardef f(expr t) = 2*sin(t) enddef;
vardef g(expr t) = 2*cos(t)*cos(t)/(2-cos(t)) enddef;
vardef h(expr t) = -t*(1+f(t)) enddef;

def DrawDoubleVector(expr o,d)= drawdblarrow ((o-d)--(o+d))
     gddEnPlace enddef;

path bicorne;
bicorne = Curve(f,g,-Pi,Pi,200)--cycle;

beginfig(1);

 Frame(9,6,4.5,1,2,2);
 Axis;
 gddBegin;
  Axis;
  Graduations; Units(1);

  gddAlphaFill(bicorne,CadetBlue,0.2);

  DrawDoubleVector((0,2),(0.5,0)) withPen(1,Crimson);
  DrawDoubleVector((0,2/3),(0.5,0)) withPen(1,Crimson);
  gddArrow Segment((2,0),(1.6,0.4)) withPen(1,Crimson);
  gddArrow Segment((-2,0),(-1.6,0.4)) withPen(1,Crimson);

  gddDraw bicorne withPen(1.5,CadetBlue);

  gddDrawPoint Point(0,0);
  gddDrawPoint Point(2,0);
  gddDrawPoint Point(-2,0);

 gddEnd;
endfig;
end
```

14.9 A function et its derivatives



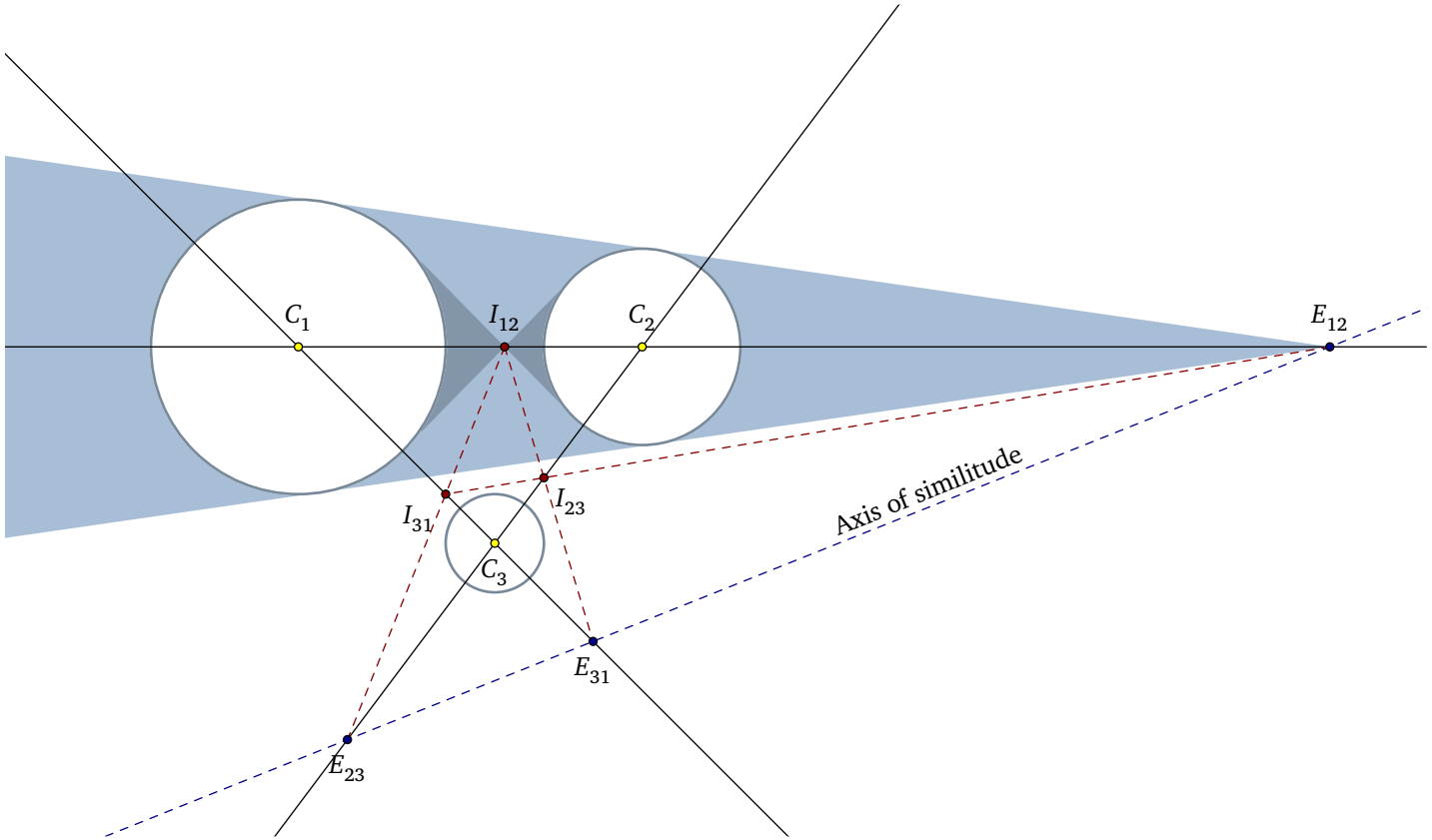

```
input geom2d;
labeloffset := 6;
gddU:=0.9cm;

vardef f(expr x) = x*(Pi-x) enddef; % f
vardef g(expr x) = Pi-2*x enddef; % f'
vardef h(expr x) = -2 enddef; % f''

beginfig(1);

Frame(9,6,5,3,.8,0.8);
Axis;
 gddBegin;
  Graduations;

  gddDraw (Pi,-2)--(Pi,2) dashed evenly;
  gddDraw (-Pi,-2)--(-Pi,2) dashed evenly;

  gddDraw Plot(f,0,Pi,100) withPen(1.5,Indigo);
  gddDraw (Plot(f,0,Pi,100) scaled -1) withPen(1.5,Indigo);
  gddDraw (Plot(f,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Indigo);
  gddDraw (Plot(f,0,Pi,100) scaled -1 shifted (2*Pi,0)) withPen(1.5,
      Indigo);

  gddDraw Plot(g,0,Pi,100) withPen(1.5,Olive);
  gddDraw (Plot(g,0,Pi,100) xscaled -1) withPen(1.5,Olive);
  gddDraw (Plot(g,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Olive);
  gddDraw (Plot(g,0,Pi,100) xscaled -1 shifted (2*Pi,0)) withPen(1.5,
      Olive);

  gddDraw Plot(h,0,Pi,100) withPen(1.5,Orange);
  gddDraw (Plot(h,0,Pi,100) scaled -1) withPen(1.5,Orange);
  gddDraw (Plot(h,0,Pi,100) shifted (-2*Pi,0)) withPen(1.5,Orange);
  gddDraw (Plot(h,0,Pi,100) scaled -1 shifted (2*Pi,0)) withPen(1.5,
      Orange);

  gddDrawPoint Point(Pi,2);
  gddDrawPoint Point(Pi,-2);
  gddDrawPoint Point(-Pi,2);
  gddDrawPoint Point(-Pi,-2);

  gddLabel.urt(textext("\(-\pi\)"), (-Pi,0)) withcolor Peru;
  gddLabel.ulft(textext("\(\pi\)"), (Pi,0)) withcolor Peru;
gddEnd;

gddDraw (0,-0.5)--(1,-0.5) withPen(1.5,Indigo);
gddLabel.rt(textext("\(f\)"), (1,-0.5));
gddDraw (2,-0.5)--(3,-0.5) withPen(1.5,Olive);
gddLabel.rt(textext("\(f'\)"), (3,-0.5));
gddDraw (4,-0.5)--(5,-0.5) withPen(1.5,Orange);
gddLabel.rt(textext("\(f''\)"), (5,-0.5));
endfig;
end
```

14.10 Cardioid




```
input geom2d;

labeloffset := 6;
gddU:=0.8cm;

vardef r(expr t) = 1+cos(t) enddef;
vardef rp(expr t) = (r(t)*cos(t),r(t)*sin(t)) enddef;

def DrawDoubleVector(expr o,d)= drawdblarrow ((o-d)--(o+d))
     gddInPlace enddef;

beginfig(1);
 Frame(10,10,2,5,3.5,3.5);
 Axis;
 gddBegin;
  Graduations; Units(1);

  drawoptions(withcolor LightSlateGrey);
  draw fullcircle shifted (0.5,0) gddInPlace withPen(1,Gold);

  nb = 80;
  pas = 2Pi / nb;
  for i=0 upto nb:
    theta := i * pas;
    draw (fullcircle scaled 2cos(theta)
      shifted (cos(theta)*cos(theta),cos(theta)*sin(theta)))
          gddInPlace;
  endfor;


  DrawDoubleVector(rp(0),(0,0.3)) withPen(1,LightPink);
  DrawDoubleVector(rp(Pi/3),(0.3,0)) withPen(1,LightPink);
  DrawDoubleVector(rp(Pi/2),(0.2,0.2)) withPen(1,LightPink);
  DrawDoubleVector(rp(2Pi/3),(0,0.3)) withPen(1,LightPink);
  DrawDoubleVector(rp(-Pi/3),(0.3,0)) withPen(1,LightPink);
  DrawDoubleVector(rp(-Pi/2),(0.2,-0.2)) withPen(1,LightPink);
  DrawDoubleVector(rp(-2Pi/3),(0,0.3)) withPen(1,LightPink);

  gddArrow Segment(origine,(-0.3,0)) withPen(1,LightPink);
  gddDraw PolarCurve(r,-Pi,Pi,100) withPen(1.5,Indigo);

  gddDrawPoint Point(0,0);
  gddDrawPoint Point(2,0);
  gddDrawPoint PairTOPoint(rp(Pi/3));
  gddDrawPoint PairTOPoint(rp(-Pi/3));
  gddDrawPoint PairTOPoint(rp(2Pi/3));
  gddDrawPoint PairTOPoint(rp(-2Pi/3));
  gddDrawPoint Point(0,1);
  gddDrawPoint Point(0,-1);

  gddLabel.urt(textext("\(\mathcal{C}\)"),(1,0)) withcolor Magenta;
 gddEnd;
endfig;
end
```

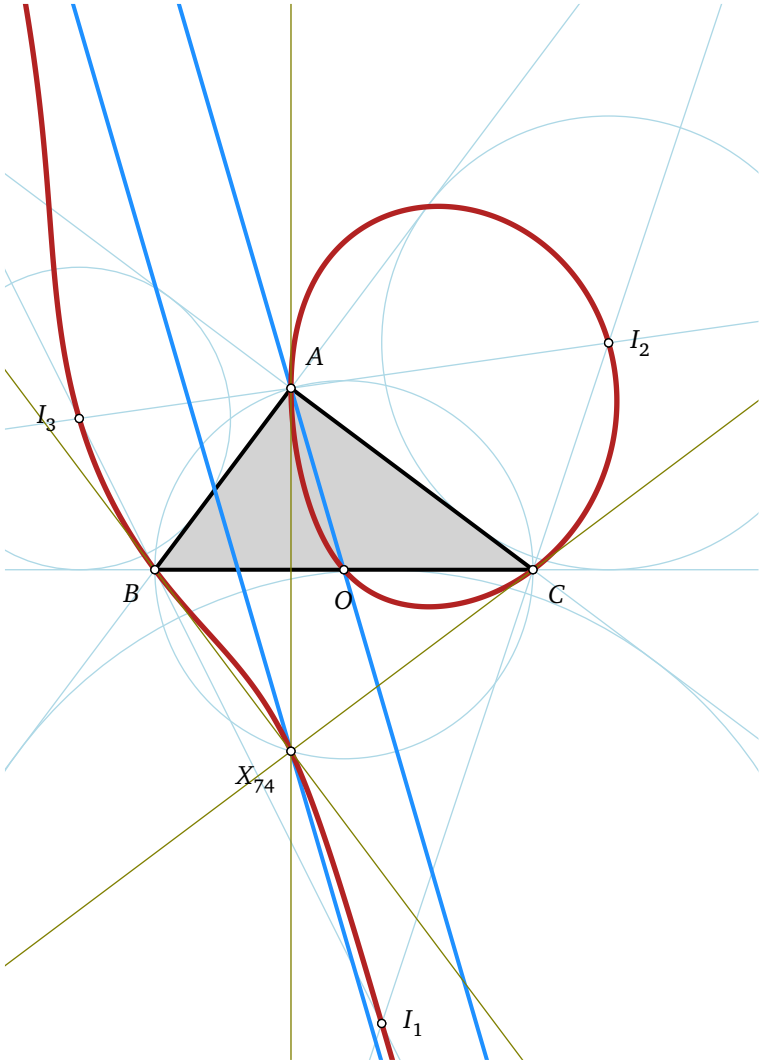
14.11 Axis of similitude



```
input geom2d;
%%% from Drawing with Metapost de Toby Thurston
labeloffset := 6;
SetPointSize(3);
SetPointColor(Yellow);
gddU:=0.65cm;
beginfig(1);
  C1 = Circle((-4,0),3);
  C2 = Circle((3,0),2);
  C3 = Circle((0,-4),1);

  T1 = ExternalCommonTangent(C1,C2);
  T2 = ExternalCommonTangent(C2,C1);
  T3 = InternalCommonTangent(C1,C2);
  T4 = InternalCommonTangent(C2,C1);

  T5 = ExternalCommonTangent(C2,C3);
  T6 = ExternalCommonTangent(C3,C2);
  T7 = InternalCommonTangent(C2,C3);
  T8 = InternalCommonTangent(C3,C2);

  T9 = ExternalCommonTangent(C1,C3);
  T10 = ExternalCommonTangent(C3,C1);
  T11 = InternalCommonTangent(C1,C3);
  T12 = InternalCommonTangent(C3,C1);

  E12 = LinesIntersection(T1,T2);
  E23 = LinesIntersection(T5,T6);
  E31 = LinesIntersection(T9,T10);
  I12 = LinesIntersection(T3,T4);
  I23 = LinesIntersection(T7,T8);
  I31 = LinesIntersection(T11,T12);

  path t[];
  t1 :=(gddTraceObjet T1) gddEnPlace;
  t2 := (gddTraceObjet T2) gddEnPlace;
  t3 := (-10*gddU,-10*gddU)--(-10gddU,10*gddU);
  fill buildcycle(t1, t3,reverse t2) withcolor 1.4*LightSlateGrey;

  t4 :=(gddTraceObjet T3) gddEnPlace;
  t5 := (gddTraceObjet T4) gddEnPlace;
  t6 := (1.5*gddU,-10*gddU)--(1.5*gddU,10*gddU);
  fill buildcycle(t4, t5,reverse t6) withcolor 1.1*LightSlateGrey;

  t7 :=(gddTraceObjet T3) gddEnPlace;
  t8 := (gddTraceObjet T4) gddEnPlace;
  t9 := (-1.5*gddU,-10*gddU)--(-1.5*gddU,10*gddU);
  fill buildcycle(t7, t8,reverse t9) withcolor 1.1*LightSlateGrey;

  drawoptions(withpen pencircle scaled 1pt withcolor LightSlateGrey);
  gddFill C1 withcolor white;
  gddFill C2 withcolor white;
  gddFill C3 withcolor white;
  gddDraw C1; gddDraw C2; gddDraw C3;

  drawoptions();
  gddDraw Line(Center(C1),Center(C2));
  gddDraw Line(Center(C3),Center(C2));
  gddDraw Line(Center(C1),Center(C3));

  D_E = Line(E12,E23);
  gddDraw D_E dashed evenly withcolor DarkBlue;

  E_I = Segment(E12,I31);
  gddDraw E_I dashed evenly withcolor DarkRed;
  I_E1 = Segment(I12,E31);
  gddDraw I_E1 dashed evenly withcolor DarkRed;
  I_E2 = Segment(I12,E23);
  gddDraw I_E2 dashed evenly withcolor DarkRed;


  drawoptions();
  gddDrawPoint Center(C1);
  gddDrawPoint Center(C2);
  gddDrawPoint Center(C3);
  SetPointColor(DarkBlue);
  gddDrawPoint E12; gddDrawPoint E31; gddDrawPoint E23;
  SetPointColor(DarkRed);
  gddDrawPoint I12; gddDrawPoint I31; gddDrawPoint I23;

  gddLabel.top(btex $E_{12}$ etex,E12);
  gddLabel.bot(btex $E_{31}$ etex,E31);
  gddLabel.bot(btex $E_{23}$ etex,E23);

  gddLabel.top(btex $I_{12}$ etex,I12);
  gddLabel.llft(btex $I_{31}$ etex,I31);
  gddLabel.lrt(btex $I_{23}$ etex,I23);
```

```
88
89
90  gddLabel.top(btex $C_{1}$ etex,Center(C1));
91  gddLabel.top(btex $C_{2}$ etex,Center(C2));
92  gddLabel.bot(btex $C_{3}$ etex,Center(C3));
93
94   draw textext("Axis of similitude") rotated (22) shifted (0.5[Pt(E
         23),Pt(E12)]
95   gddInPlace +(0,3));
96   Window(-10,-10,19,7);
97 endfig;
98 end.
```

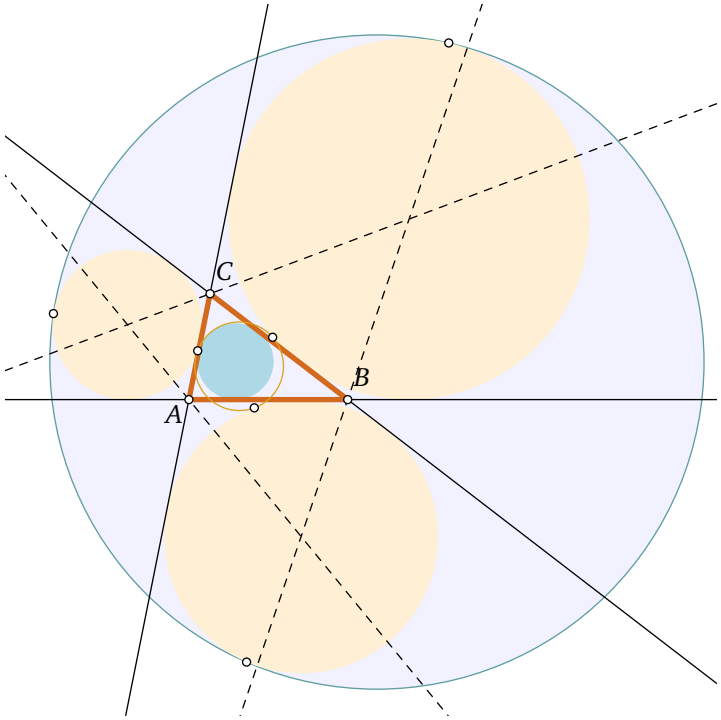
14.12 Cubic Drawing




```
input geom2d;
gddU := 0.5cm;
labeloffset := 8pt;
gddTailleLabel := 1;

beginfig(1);
  A := Point(3.6,4.8);
  B := Point(0,0);
  C := Point(10,0);
  T := Triangle(A,B,C);
  X_74 := Point(3.6,-4.8);

  C1 := CurveData("K001-1",0);
  C2 := CurveData("K001-2",1);

  CE1 := EscribedCircle(T,1);
  CE2 := EscribedCircle(T,2);
  CE3 := EscribedCircle(T,3);
  I_1 := Center(CE1);
  I_2 := Center(CE2);
  I_3 := Center(CE3);
  CC := CircumscribedCircle(T);
  O  := Center(CC);

  drawoptions(withcolor LightBlue);
  gddDraw Line(A,B);
  gddDraw Line(B,C);
  gddDraw Line(C,A);
  gddDraw Line(I_1,I_2);
  gddDraw Line(I_2,I_3);
  gddDraw Line(I_3,I_1);
  gddDraw CE1;
  gddDraw CE2;
  gddDraw CE3;
  gddDraw CC;
  colorie T withcolor LightGrey;
  drawoptions(withpen pencircle scaled 1.5);
  gddDraw T;
```

```
drawoptions(withcolor DodgerBlue withpen pencircle scaled 1.5);
gddDraw Line(O,A);
gddDraw Line(X_74, Point(2.2,0));
drawoptions(withpen pencircle scaled 2);
gddDraw C1 withcolor FireBrick;
gddDraw C2 withcolor FireBrick;
drawoptions(withcolor Olive);
gddDraw Line(A,X_74);
gddDraw Line(B,X_74);
gddDraw Line(C,X_74);
drawoptions();
gddDrawPoint A;
gddDrawPoint B;
gddDrawPoint C;
gddDrawPoint I_1;
gddDrawPoint I_2;
gddDrawPoint I_3;
gddDrawPoint O;
gddDrawPoint X_74;

gddMark.urt "A";
gddMark.llft "B";
gddMark.lrt "C";
gddMark.rt "I_1";
gddMark.rt "I_2";
gddMark.lft "I_3";
gddMark.bot "O";
gddMark.llft "X_74";

Window(-4,-13,16,15);
endfig;

end
```

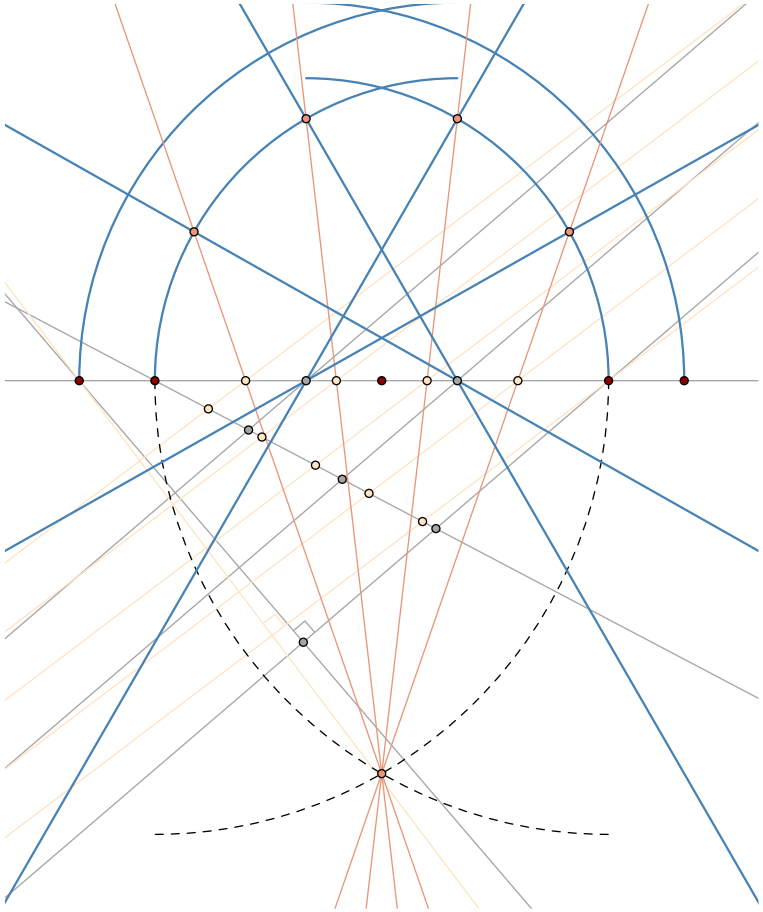
14.13 Apollonius




```
input geom2d;
gddU:=0.35cm;
beginfig(1);

A = Point(0,0);
B = Point(6,0);
C = Point(0.8,4);
T_ABC = Triangle(A,B,C);

C_I = InscribedCircle(T_ABC);
C_A = EscribedCircle(T_ABC,2);
C_B = EscribedCircle(T_ABC,3);
C_C = EscribedCircle(T_ABC,1);

d_AB = Line(A,B);
d_BC = Line(B,C);
d_CA = Line(C,A);

I = Center(C_I);
I_C_A = Center(C_A);
I_C_B = Center(C_B);
I_C_C = Center(C_C);
d_CAA = Line(A,I_C_A);
d_CAB = Line(B,I_C_B);
d_CAC = Line(C,I_C_C);

A_S = AxisOfSimilitude(C_A,C_B,C_C);
P_CA = PointOnLineProjection(I_C_A,A_S);
P_CB = PointOnLineProjection(I_C_B,A_S);
P_CC = PointOnLineProjection(I_C_C,A_S);

P_A = Inverse(P_CA,C_A);
P_B = Inverse(P_CB,C_B);
P_C = Inverse(P_CC,C_C);

C_R = RadicalCenter(C_A,C_B,C_C);
% les neuf points pour les cercles 'deuler (tangent intérieur)
% et 'dapollonius (tangent extérieur)
D1 = Line(C_R,P_A);
P1 = LineCircleIntersection(D1,C_A,1);
Q1 = LineCircleIntersection(D1,C_A,2);

D2 = Line(C_R,P_B);
P2 = LineCircleIntersection(D2,C_B,1);
Q2 = LineCircleIntersection(D2,C_B,2);

D3 = Line(C_R,P_C);
P3 = LineCircleIntersection(D3,C_C,2);
```

```
Q3 = LineCircleIntersection(D3,C_C,1);

% le cercle 'dapollonius
Apol = CircleThreePoints(P1,P2,P3);
gddFill Apol withcolor 1.05*Lavender;
gddDraw Apol withcolor CadetBlue;

% cercle inscrit
gddFill C_I withcolor LightBlue;
% cercles exinscrits
gddFill C_A withcolor PapayaWhip;
gddFill C_B withcolor PapayaWhip;
gddFill C_C withcolor PapayaWhip;

gddDraw d_AB;
gddDraw d_BC;
gddDraw d_CA;

gddDraw d_CAA dashed evenly;
gddDraw d_CAB dashed evenly;
gddDraw d_CAC dashed evenly;

gddDraw T_ABC withpen pencircle scaled 2 withcolor Chocolate;

C_E = EulerCircle(T_ABC);
gddDraw C_E withcolor Goldenrod;

gddDrawPoint P1;
gddDrawPoint P2;
gddDrawPoint P3;

gddDrawPoint Q1;
gddDrawPoint Q2;
gddDrawPoint Q3;


gddMark.llft "A";
gddMark.urt "B";
gddMark.urt "C";
Window(-7,-12,20,15)
endfig;
end.
```

14.14 Ogive Working Drawing

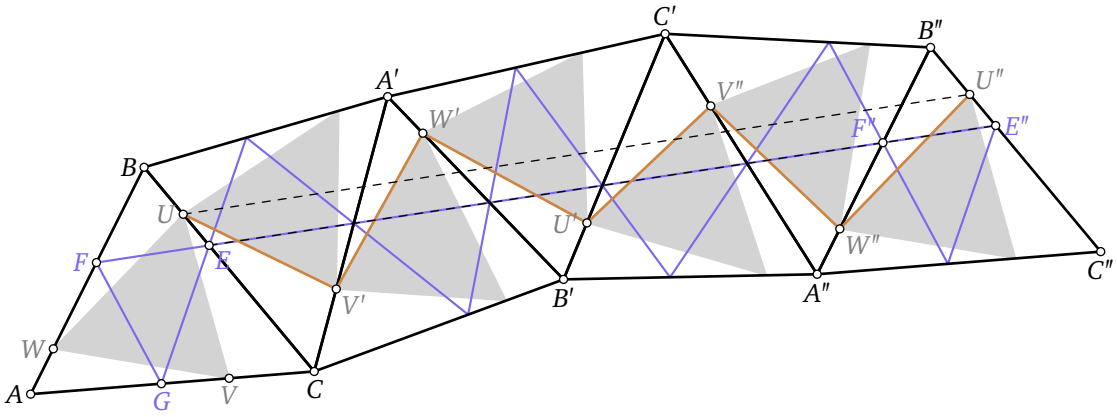


```
input geom2d;

beginfig(1);
   % base points for the ogive
   O = Point(0,0);
   A = Point(-3,0);
   A2 = Point(-4,0);
   B = Point(3,0);
   B2 = Point(4,0);
   % the base line of the ogive
   AB = Line(A,B);
   % we take an arbitrary line
   Ct = Point(0.8,-2); % arbitrary point
   AC = Line(A,Ct); % arbitrary line
   % we take three equal lengths on the line
   longueurTier := 1.4;
   C1 = ReportOnLine(A,AC,longueurTier);
   C2 = ReportOnLine(C1,AC,longueurTier);
   C3 = ReportOnLine(C2,AC,longueurTier);
   % we define the line passing through B and the last point
   BC = Line(B,C3);
   % we construct a perpendicular line to BC
   D = Point(-4,0);
   perpD = PerpendicularLine(BC,D);
   DD = LinesIntersection(perpD,BC);
   % we project C2 and C1 onto the line AB to divide [AB] into 3
        equal parts
   perpC2 = PerpendicularLine(perpD,C2);
   perpC1 = PerpendicularLine(perpD,C1);
   D1 = LinesIntersection(AB,perpC1);
   D2 = LinesIntersection(AB,perpC2);
   % we construct the envelope of the ogive
   C_DA = CircleCP(D2,A);
   C_DB = CircleCP(D1,B);
   C_DA2 = CircleCP(D2,A2);
   C_DB2 = CircleCP(D1,B2);
   % we construct the point P for the final projection
   C_AB = CircleCP(A,B);
   C_BA = CircleCP(B,A);
   P = CirclesIntersection(C_BA,C_AB);
   % we take 5 equal lengths on the initial arbitrary line
```

```
    longueurCinq :=0.8;
    F1 = ReportOnLine(A,AC,longueurCinq);
    F2 = ReportOnLine(F1,AC,longueurCinq);
    F3 = ReportOnLine(F2,AC,longueurCinq);
    F4 = ReportOnLine(F3,AC,longueurCinq);
    F5 = ReportOnLine(F4,AC,longueurCinq);
    % we project the points onto AB to divide [AB] into 5 equal parts
    BF5 = Line(B,F5);
    perpF5 = PerpendicularLine(BF5,A2);
    CC = LinesIntersection(BF5,perpF5);
    perpF4 = PerpendicularLine(perpF5,F4);
    perpF3 = PerpendicularLine(perpF5,F3);
    perpF2 = PerpendicularLine(perpF5,F2);
    perpF1 = PerpendicularLine(perpF5,F1);
    G1 = LinesIntersection(AB,perpF1);
    G2 = LinesIntersection(AB,perpF2);
    G3 = LinesIntersection(AB,perpF3);
    G4 = LinesIntersection(AB,perpF4);
    % we project the Gi points onto the ogive using point P
    PG1 = Line(P,G1);
    PG2 = Line(P,G2);
    PG3 = Line(P,G3);
    PG4 = Line(P,G4);
    I1 = LineCircleIntersection(PG1,C_DA,1);
    I2 = LineCircleIntersection(PG2,C_DA,2);
    I3 = LineCircleIntersection(PG3,C_DB,2);
    I4 = LineCircleIntersection(PG4,C_DB,2);
    % from the 2 points resulting from dividing [AB] into 3 parts
    % we draw the separations of the stones that make up the ogive
    Dvoute1 = Line(D2,I1);
    Dvoute2 = Line(D2,I2);
    Dvoute3 = Line(D1,I3);
    Dvoute4 = Line(D1,I4);

    % drawings
    drawoptions(withcolor 1.3*Grey);
    gddDraw perpC2; gddDraw perpC1;
    gddDraw AB;
    gddDraw AC;
    gddDraw BC;
    gddDraw perpD;
    gddDraw OrthoSign(B,DD,A2,0.2);

    drawoptions(withcolor Bisque);
    gddDraw BF5;
    gddDraw perpF5;gddDraw perpF4;gddDraw perpF3;gddDraw perpF2;
        gddDraw perpF1;
    gddDraw OrthoSign(B,CC,A2,0.2);

    drawoptions(withcolor SteelBlue withpen pencircle scaled 0.85pt);
    gddDraw gddDrawCircleArc(C_DA,Pi/2,Pi);
    gddDraw gddDrawCircleArc(C_DA2,Pi/2,Pi);
    gddDraw gddDrawCircleArc(C_DB2,0,Pi/2);
    gddDraw gddDrawCircleArc(C_DB,0,Pi/2);

    drawoptions();
    gddDraw gddDrawCircleArc(C_AB,0,-Pi/2) dashed evenly;
    gddDraw gddDrawCircleArc(C_BA,Pi,3Pi/2) dashed evenly;

    drawoptions(withcolor DarkSalmon);
    gddDraw PG1; gddDraw PG2; gddDraw PG3; gddDraw PG4;

    drawoptions(withcolor SteelBlue withpen pencircle scaled 0.85pt);
    gddDraw Dvoute1; gddDraw Dvoute2; gddDraw Dvoute3; gddDraw Dvoute
        4;
    drawoptions();

    SetPointColor(DarkRed);
    gddDrawPoint O;
    gddDrawPoint A;
    gddDrawPoint B; gddDrawPoint D;
    gddDrawPoint B2;

    SetPointColor(Bisque);
    gddDrawPoint F1; gddDrawPoint F2; gddDrawPoint F3; gddDrawPoint F
        4;
    gddDrawPoint F5;
    gddDrawPoint G1; gddDrawPoint G2;gddDrawPoint G3;gddDrawPoint G4;
    SetPointColor(1.3*Grey);
    gddDrawPoint DD;
    gddDrawPoint C1; gddDrawPoint C2; gddDrawPoint C3;
    gddDrawPoint D1; gddDrawPoint D2;

    SetPointColor(DarkSalmon);
    gddDrawPoint P;
    gddDrawPoint I1; gddDrawPoint I2; gddDrawPoint I3; gddDrawPoint I
        4;
    Window(-5,-7,5,5);
endfig;
end.
```

14.15 Pedal Triangle

Figure 21 of [3].

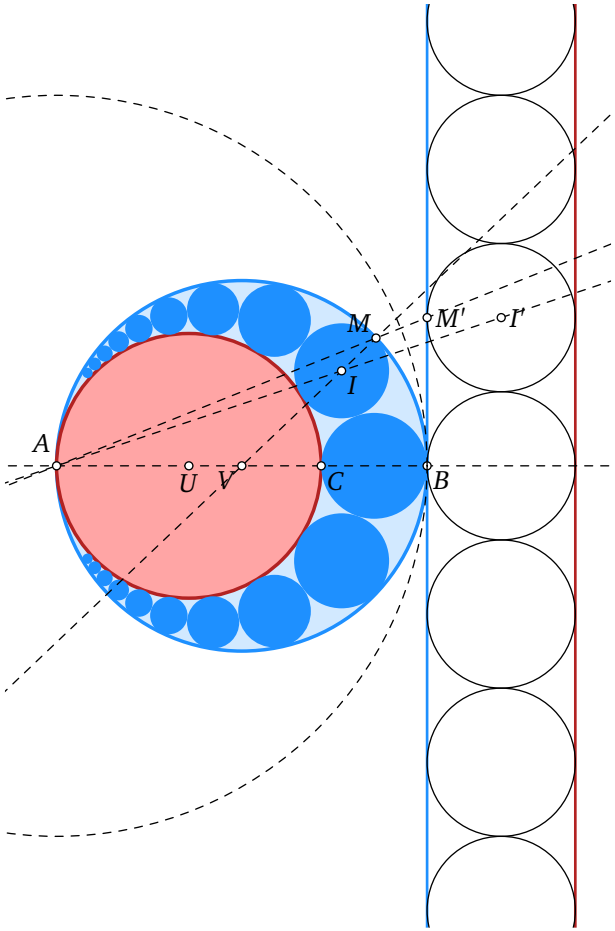



```
input geom2d;
gddU:=1.5cm;
beginfig(1);
numeric A[],B[],C[],E[],F[],G[],U[],W[],V[],T[],R[],P[];
A[1] = Point(0,0);
B1 = Point(1,2);
C1 = Point(2.5,0.2);
W1 = PointOf(Segment(A1,B1),0.2);
U1 = PointOf(Segment(B1,C1),0.23);
V1 = PointOf(Segment(C1,A1),0.3);
G1 = PointOnLineProjection(B1,Line(A1,C1));
F1 = PointOnLineProjection(C1,Line(A1,B1));
E1 = PointOnLineProjection(A1,Line(B1,C1));


for i:=0 step 3 until 3:

D[1+i] = Line(B[1+i],C[1+i]);
A[2+i] = AxialSymmetry(A[1+i],D[1+i]);

B[2+i] = B[1+i];
C[2+i] = C[1+i];

W[2+i] = AxialSymmetry(W[1+i],D[1+i]);
V[2+i] = AxialSymmetry(V[1+i],D[1+i]);
U[2+i] = U[1+i];

G[2+i] = AxialSymmetry(G[1+i],D[1+i]);
F[2+i] = AxialSymmetry(F[1+i],D[1+i]);
E[2+i] = E[1+i];

D[2+i] = Line(A[2+i],C[2+i]);
B[3+i] = AxialSymmetry(B[2+i],D[2+i]);
A[3+i] = A[2+i];
C[3+i] = C[2+i];
W[3+i] = AxialSymmetry(W[2+i],D[2+i]);
U[3+i] = AxialSymmetry(U[2+i],D[2+i]);
```

```
V[3+i] = V[2+i];
E[3+i] = AxialSymmetry(E[2+i],D[2+i]);
F[3+i] = AxialSymmetry(F[2+i],D[2+i]);
G[3+i] = G[2+i];

D[3+i] = Line(A[3+i],B[3+i]);
C[4+i] = AxialSymmetry(C[3+i],D[3+i]);
A[4+i] = A[3+i];
B[4+i] = B[3+i];
U[4+i] = AxialSymmetry(U[3+i],D[3+i]);
V[4+i] = AxialSymmetry(V[3+i],D[3+i]);
W[4+i] = W[3+i];
E[4+i] = AxialSymmetry(E[3+i],D[3+i]);
G[4+i] = AxialSymmetry(G[3+i],D[3+i]);
F[4+i] = F[3+i];
endfor;

for i:=1 upto 7:
   T[i] = Triangle(A[i],B[i],C[i]);
   Q[i] = Triangle(U[i],V[i],W[i]);
   P[i] = Triangle(E[i],F[i],G[i]);
   gddFill Q[i] withPen(0.5,LightGray);
   gddDraw P[i] withPen(0.8,MediumSlateBlue);
endfor;
for i:=1 upto 7:
   gddDraw T[i] withPen(1,black);
endfor;

gddDraw Polyline(U1,V2,W3,U4,V5,W6,U7) withPen(1,Peru) ;
gddDraw Segment(U1,U7) dashed evenly;
gddDraw Segment(E1,E7) dashed evenly;
gddDrawPoint A1; gddDrawPoint B1; gddDrawPoint C1;
gddDrawPoint A2; gddDrawPoint B3; gddDrawPoint C4;
gddDrawPoint A5; gddDrawPoint B6; gddDrawPoint C7;
gddLabel.lft(textext("$A$"),A1);
gddLabel.lft(textext("$B$"),B1);
gddLabel.bot(textext("$C$"),C1);
gddLabel.top(textext("$A'$"),A2);
gddLabel.bot(textext("$B'$"),B3);
gddLabel.top(textext("$C'$"),C4);
gddLabel.bot(textext("$A''$"),A5);
gddLabel.top(textext("$B''$"),B7);
gddLabel.bot(textext("$C''$"),C7);

gddDrawPoint U1; gddDrawPoint V1; gddDrawPoint W1;
gddLabel.lft(textext("$U$"),U1) withcolor Gray;
gddLabel.bot(textext("$V$"),V1) withcolor Gray;
gddLabel.lft(textext("$W$"),W1) withcolor Gray;

gddLabel.lft(textext("$U'$"),U4) withcolor Gray;
gddLabel.lrt(textext("$V'$"),V2) withcolor Gray;
gddLabel.urt(textext("$W'$"),W3) withcolor Gray;
gddLabel.urt(textext("$U''$"),U7) withcolor Gray;
gddLabel.urt(textext("$V''$"),V5) withcolor Gray;
gddLabel.lrt(textext("$W''$"),W7) withcolor Gray;
gddDrawPoint U4; gddDrawPoint V2; gddDrawPoint W3;
gddDrawPoint U7; gddDrawPoint V5; gddDrawPoint W7;
gddDrawPoint E1; gddDrawPoint F1; gddDrawPoint G1;
gddLabel.lrt(textext("$E$"),E1) withcolor MediumSlateBlue;
gddLabel.lft(textext("$F$"),F1) withcolor MediumSlateBlue;
gddLabel.bot(textext("$G$"),G1) withcolor MediumSlateBlue;

gddLabel.ulft(textext("$F''$"),F7) withcolor MediumSlateBlue;
gddDrawPoint F7;
gddLabel.rt(textext("$E''$"),E7) withcolor MediumSlateBlue;
gddDrawPoint E7;
endfig;
end.
```

14.16 Pappus Chain

Pappus chain with geometric inverse.




```
input geom2d;

gddU:=0.7cm;

beginfig(1);
   r = 2.5;
   R = 3.5;
   A = Point(0,0);
   C_A = Circle(A,2*R);
   C = Point(2r,0);
   B = Point(2R,0);
   C_V = CircleD(A,B);
   C_U = CircleD(A,C);
   C_CB=CircleD(C,B);
   V = Center(C_V);
   U = Center(C_U);
   AB = Line(A,B);
   I_CU = Inverse(C_U,C_A);
   I_CV = Inverse(C_V,C_A);

   L_AB = Line(A,B);
   Up = LinesIntersection(L_AB,I_CU);
   Vp = LinesIntersection(L_AB,I_CV);
   Cp0 = CircleD(Vp,Up);
   Rp= Radius(Cp0);


   gddAlphaFill(C_V,DodgerBlue,0.2);
   gddDraw C_V withPen(1.3,DodgerBlue);

   gddFill C_U withcolor 4.9*FireBrick;
   gddDraw C_U withPen(1.3,FireBrick);
   gddFill C_CB withcolor DodgerBlue;
   gddDraw I_CV withPen(1,DodgerBlue);
   gddDraw I_CU withPen(1,FireBrick);

   Ip[0] = MidPoint(Vp,Up);
   N=9;
   gddDraw Cp0;
```

```
for i:=-N upto N:
   if(i<>0):
   Ip[i] = Addition(Ip[0],Point(0,i*2*Rp));
   Mp[i] = Addition(Ip[i],Point(-Rp,0));
   DAMp[i] = Line(A,Mp[i]);
   DVIp[i] = Line(V,Ip[i]);
   DAIp[i] = Line(A,Ip[i]);
   M[i] = LineCircleIntersection(DAMp[i],C_V,2);
   DVM[i] = Line(V,M[i]);
   I[i] = LinesIntersection(DAIp[i],DVM[i]);
   gddFill CircleCP(I[i],M[i]) withcolor DodgerBlue;
   gddDraw CircleCP(Ip[i],Mp[i]);
   if(i=1):
      drawoptions(dashed evenly);
      gddDraw DAIp[i];gddDraw DVM[i];gddDraw DAMp[i];
      drawoptions();
      gddDrawPoint Ip[i]; gddDrawPoint Mp[i];
      gddDrawPoint M[i]; gddDrawPoint I[i];
      gddLabel.rt(textext("$I'$"),Ip[i]);
      gddLabel.lrt(textext("$I$"),I[i]);
      gddLabel.rt(textext("$M'$"),Mp[i]);
      gddLabel.ulft(textext("$M$"),M[i]);
   fi
   fi
endfor;
gddDraw AB dashed evenly;
gddMark.bot "U";
gddMark.llft "V";
gddMark.ulft "A"; gddMark.lrt "B"; gddMark.lrt "C";
gddDraw C_A dashed evenly;

Window(-1,-2.5R,3*R,2.5R);
endfig;

end.
```

15 Historique

may 2025: translation into English of macros, documentation and examples.

References

[1] Hans Hagen et al. *The `luamplib` package. Use LuaTeX's built-in MetaPost interpreter*. Version 2.34.5. Aug. 3, 2024. URL: https://ctan.org/pkg/luamplib.

[2] Jens-Uwe Morawski. *The `latexMP` package. Interface for LaTeX-based typesetting in MetaPost*. Version 1.2.1. June 21, 2020. URL: https://ctan.org/pkg/latexmp.

[3] Hans Rademacher and Otto Toeplitz. *The enjoyment of math. Translated from the German. With a new foreword by Alex Kontorovich*. English. Reprint. Princeton Sci. Libr. Princeton, NJ: Princeton University Press, 2023. ISBN: 978-0-691-24154-8; 978-0-691-24153-1. DOI: 10.1515/9780691241531.

[4] Esger Renkema. *The `minim-mp` package. Low-level mplib integration for LuaTeX*. Version 2024/1.6. Apr. 6, 2024. URL: https://ctan.org/pkg/minim-mp.

[5] The MetaPost Team and John Hobby. *The `metapost` package. A development of Metafont for creating graphics*. Aug. 26, 2021. URL: https://ctan.org/pkg/metapost.

[6] Toby Thurston. *The `Drawing-with-Metapost` package. How to draw technical diagrams with MetaPost*. Apr. 16, 2023. URL: https://ctan.org/pkg/drawing-with-metapost.

Index