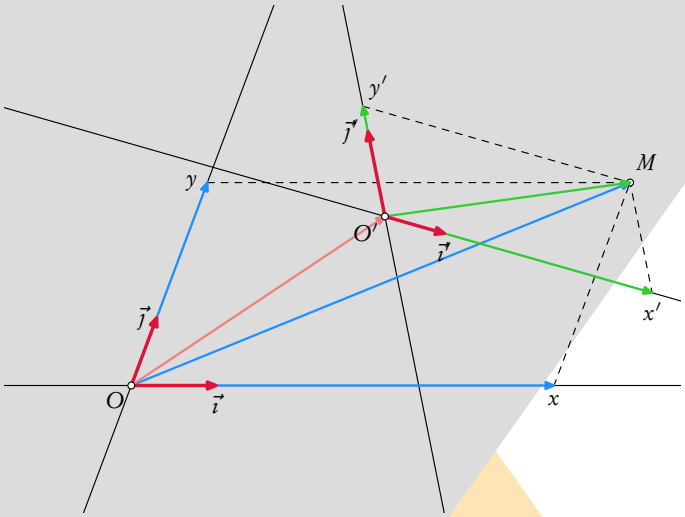
mp-geom2d

Paquet METAPOST pour des figures de géométrie plane



Contributeurs
Maxime CHUPIN
Jean-Michel SARLAT

Version 1.4 du 3 décembre 2025
https://gitlab.gutenberg-asso.fr/mchupin/mp-geom2d
https://ctan.org/pkg/mp-geom2d

Table des matières

1 Objectif

`mp-geom2d` a été écrit avec le but de proposer des macros METAPOST permettant de réaliser une figure de géométrie *en collant* d'assez près à une description impérative :

> *Soit A le point de coordonnées (2,3).*
> *Soit B le point de coordonnées (4,5).*
> *Trace la droite* (A,B).
> ....

Dans ce cadre, les objets géométriques sont le plus souvent nommés (A, B, etc.) ou désignés par leur nature et leurs attributs (droite (A,B), etc.). Pour ne pas avoir à dépasser ce mode de description, en particulier pour éviter d'avoir à déclarer le *type* de ces objets, le choix a été fait de les identifier par un *index*[1] dans des tables qui en précisent les caractéristiques.
Note — À ce jour, l'objectif n'est pas atteint, le développement est loin d'être achevé ; il est encore nécessaire de faire appel à des commandes METAPOST ou à des *macros intermédiaires* pour décrire une figure. Cela évoluera sans doute avec le temps, le temps de trouver une syntaxe satisfaisante...

2 Les fichiers

`mp-geom2d` est un ensemble d'outils pour la géométrie plane avec METAPOST [5]. Cet ensemble est organisé en plusieurs fichiers :

1. `geom2d.mp` : c'est le fichier principal, qui charge tous les autres ;
2. `geom2d-main.mp` : il contient les structures et fonctions générales ;
3. `geom2d-arc.mp` : contient tout ce qui concerne les arcs de cercles ;
4. `geom2d-c2d.mp` : contient tout ce qui concerne les courbes du second degré ;
5. `geom2d-fct.mp` : contient quelques fonctions mathématiques usuelles ;
6. `geom2d-lbl.mp` : contient les fonctions relatives aux labels ;
7. `geom2d-plt.mp` : contient des fonctions facilitant la représentation de fonctions mathématiques ;
8. `geom2d-rep.mp` contient différents outils pour le tracé de figure dans un repère ;
9. `geom2d-tra.mp` contient les fonctions permettant de gérer la transparence (code emprunté à Anthony PHAN) ;
10. `geom2d-svgnames.mp` fournit les 150 couleurs de la spécification SVG.

Nous allons, dans la suite, décrire plus en détail chacune de ces fonctions. Certaines fonctions s'appuient sur l'extension `graph.mp` présente dans toutes les bonnes distributions TEX.

3 Principe général de fonctionnement

`mp-geom2d` utilise des tables comme structure principale. Chaque objet est numéroté *via* le compteur `gdd0`, son type[2] est stocké dans la table `gddT[]` à la place `gddT[gdd0]`.

1. Le type `numeric`, qui est le type par défaut dans METAPOST, ne demande pas de déclaration préalable.
2. Les types sont propres à `mp-geom2d` et seront décrits plus tard.

Les propriétés des objets sont définies dans, là encore, des tables de type numeric qui sont gddA[], gddB[], ..., gddF[].

Par exemple, pour un Point (type mp-geom2d), la première coordonnée se trouve dans gddA[] et la seconde dans gddB[] (les autres tables ne sont pas utilisées pour un tel objet).

Il y a trois tables particulières gddP[] qui est du type path, gddPX[][] qui est sa version étendue, et gddS[] qui est du type string. Nous verrons plus tard quelle est leur utilité.

Bien entendu, lors d'une utilisation classique de mp-geom2d, l'appel à toutes ces tables n'est pas chose courante. Les fonctions que nous allons décrire dans la suite de ce document permettent de ne pas avoir recours trop précisément à cette machinerie.

4 Tracés

mp-geom2d fournit des macros pour tracer des objets. Une commande permet de tracer la plupart des objets de mp-geom2d (qui sont décrits dans la section 5), ainsi que certains type METAPOST. Les exemples tout au long de la documentation permettront d'illustrer les commandes de tracé.

Pour comprendre comment fonctionnent les représentations avec mp-geom2d, il nous faut tout d'abord décrire le mécanisme de gestion des unités.

4.1 Unité

mp-geom2d définit une unité générale avec la variable globale gddU. Ce numeric vaut par défaut 1 cm.

```
gddU
```

4.2 En place

Lors de l'utilisation des commandes de tracé, mp-geom2d utilise la macro

```
gddEnPlace
```

Cette macro permet de spécifier les transformations géométriques nécessaires au placement de l'objet, notamment le facteur d'échelle relatif à l'unité gddU.

En interne, cette macro est un scantoken de la variable globale gddW (string) qui contient les transformations.

4.3 Commandes de tracés

La macro de tracé la plus courante est la macro trace.

```
trace(⟨objet⟩)
```

⟨***objet***⟩ **:** — pour le côté METAPOST, un `path`, une `picture` ou un `pair`, et dans ce cas `trace` sera équivalent à `draw` ;
— du côté de `mp-geom2d`, n'importe quel des 6 objets définis jusque là : `triangle`, `segment`, `droite`, `cercle`, `ellipse`, `parabole`, `hyperbole`, `polygone`, `chemin`, `courbe` ou `vecteur` (pour ce dernier, une flèche sera ajoutée à l'extrémité du vecteur).

Lorsque `trace` est utilisée sur un objet de `mp-geom2d`, la macro fait appel à la commande METAPOST `draw` avec le mécanisme de mise en place décrit plus haut.

`mp-geom2d` fournit un tracé avec flèche faisant non plus appel à `draw` mais à `drawarrow` .

`fleche(`⟨*objet*⟩`)`

⟨***objet***⟩ **:** — pour le côté METAPOST, un `path`, une `picture` ou un `pair`, et dans ce cas `trace` sera équivalent à `drawarrow` ;
— du côté de `mp-geom2d`, n'importe quel des 6 objets définis jusque là : `triangle`, `segment`, `droite`, `cercle`, `ellipse`, `polygone`, `chemin`, `courbe` ou `vecteur` (pour ce dernier, une flèche sera ajoutée à l'extrémité du vecteur).

On peut aussi colorier un objet grâce à la macro suivante.

`colorie(`⟨*objet*⟩`)`

⟨***objet***⟩ **:** peut être :
— pour le côté METAPOST, uniquement un `path` et dans ce cas `trace` sera équivalent à `fill` ;
— du côté de `mp-geom2d`, n'importe quel objet coloriable : `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, ou `courbe`.

Les commandes de tracés de `mp-geom2d` font appel aux macros de base de METAPOST : `draw` et `fill`. Ainsi, pour spécifier la couleur, on pourra utiliser la commande `withcolor` ou bien la macro `drawoptions`.

On peut aussi colorier avec de la transparence avec la commande suivante.

`colorieAvecTransparence(`⟨*objet*⟩`,`⟨*couleur*⟩`,`⟨*coefficient de transparence*⟩`)`

⟨***objet***⟩ **:** peut être :
— pour le côté METAPOST, uniquement un `path` et dans ce cas `trace` sera équivalent à `fill` ;
— du côté de `mp-geom2d`, n'importe quel objet coloriable : `triangle`, `cercle`, `ellipse`, `polygone`, `chemin`, ou `courbe`.

⟨***couleur***⟩ **:** est une `color`.

⟨***coefficient de transparence***⟩ **:** est un `numeric` compris entre 0 et 1, 0 pour un coloriage opaque et 1 pour un coloriage invisible.

En réalité, avec une utilisation *classique*, on simule de la transparence avec le code emprunté à Anthony Phan : http://www-math.univ-poitiers.fr/~phan/metalpha.html. Si on utilise mp-geom2d avec le package luamplib, la macro colorieAvecTransparence fait appel à la macro withtransparency qui utilise la transparence de PDF et est bien plus légère en calcul. Ainsi, un code pourra être compilable avec luatex et pas avec mpost.

On pourra aussi hachurer une forme fermée[3] avec la commande suivante :

hachure(⟨*objet*⟩,⟨*angle*⟩,⟨*espace*⟩,⟨*type*⟩) → image

⟨***objet***⟩ **:** peut être :
- — pour le côté METAPOST, uniquement un path et dans ce cas trace sera équivalent à fill ;
- — du côté de mp-geom2d, n'importe quel objet « coloriable » : triangle, cercle, ellipse, polygone, chemin, ou courbe.

⟨***angle***⟩ **:** est l'angle (numeric en degré) des lignes de hachurage.

⟨***espace***⟩ **:** est l'espace (numeric) entre chaque ligne de hachurage (dans l'unité de mp-geom2d, il ne faut donc pas mettre d'unité).

⟨***type***⟩ **:** est le type de tracé. Si ⟨*type*⟩= 0, les lignes seront en trait pleins, si ⟨*type*⟩= 1, les lignes seront en traits pointillés (dashed evenly), si ⟨*type*⟩= 2, les lignes seront en traits d'axes, si ⟨*type*⟩= 3, les lignes seront en traits pointillés avec des points (withdots). Enfin, si ⟨*type*⟩= 4, vous pouvez définir le *schéma* de trait avec la command SchemaHachure.

Attention, la macro hachure retourne une image, il faudra donc l'utiliser conjointement à la macro trace.

Pour utiliser le quatrième type de trait pour les hachures, il faudra définir son schéma (*pattern*) à l'aide de la commande suivante.

SchemaHachure(⟨*schéma*⟩)

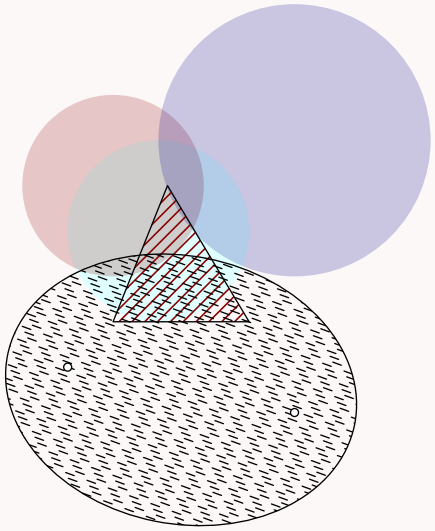
⟨***schéma***⟩ **:** est le *pattern* que l'on fournit classiquement à la commande dashpattern. Nous renvoyons à la documentation de METAPOST. Notez cependant que les dimensions à fournir aux commandes on et off seront mis à l'échelle à l'unité gddU de mp-geom2d. Il faut donc les consevoir comme exprimées dans l'unité de mp-geom2d.

Nous illustrons ci-dessous quelques commandes de tracé, de coloriage et de hachurage.

3. Le code est repris et adapté de l'ensemble de macros geometriesyr16 de Christophe Poulain https://melusine.eu.org/syracuse/poulecl/geometriesyr16/.

Exemple 1

```
input geom2d;

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

colorie C1 withcolor LightCyan;
colorieAvecTransparence(C2,
    DarkRed,0.2);
colorieAvecTransparence(C3,
    DarkBlue,0.2);

T1 = Triangle((-1,-2),(2,-2)
    ,(0.2,1));
trace hachure(T1,45,0.2,0)
    avecCrayon(0.6,DarkRed);
trace T1;
pE1 = Point(-2,-3);
pE2 = Point(3,-4);
E = EllipseF(pE1,pE2,3.9);
SchemaHachure(on0.2 off0.3);
trace hachure(E,-20,0.1,4);
trace E;
pointe pE1; pointe pE2;
endfig;
```



Les macros ci-dessus, font appel à une macro plus bas niveau qui permet de retourner le `path` à tracer ou colorier pour tous les objets `mp-geom2d`.

`gddTraceObjet(`⟨*objet*⟩`)` → `path`

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

Si l'objet n'est pas fermé, `mp-geom2d` fournit la macro suivante qui ajoute un `--cycle` à la suite du chemin :

`fermeture(`⟨*objet*⟩`)` → `path` (*fermé* `--cycle`)

⟨***objet***⟩ : — *du côté de METAPOST,* `path`*;*
— *du côté de* `mp-geom2d`*,* `triangle`*,* `segment`*,* `vecteur`*,* `cercle`*,* `chemin`*, ou* `courbe`*.*

On pourra spécifier le *crayon* utilisé pour les tracés avec la commande suivante :

`avecCrayon(`⟨*taille*⟩`,`⟨*couleur*⟩`)`

⟨***taille***⟩ : facteur d'échelle `numeric` du `pen` `pencircle` de METAPOST utilisé ;

⟨***couleur***⟩ : couleur `color` à utiliser.

Cette macro est un raccourci pour le *classique* `withpen pencircle scaled ... withcolor ...`. On peut d'ailleurs évidemment toujours utiliser les commandes METAPOST pour le dessin.

On peut représenter les objets de type `point`. Pour cela, on dispose d'une commande qui trace un petit cercle coloré à l'endroit du point. La commande est la suivante :

`pointe(`⟨*point*⟩`)`

⟨***point***⟩ : `point` ou `pair`

Cette macro utilise trois paramètres internes de `mp-geom2d` pour la taille, la couleur et le type de marque utilisés (`gddTaillePoint`, `gddCouleurPoint` et `gddPointeType`). Ces paramètres peuvent être modifiés avec les commandes suivantes :

`ParametreTaillePoint(`⟨*n*⟩`)`

⟨***n***⟩ : `numeric`, valeur par défaut 3.

`ParametreCouleurPoint(`⟨*c*⟩`)`

⟨***c***⟩ : `color`, valeur par défaut `white`.

Il existe deux autres types de pointage de point : la simple croix et le simple disque. Pour choisir un autre type de pointage (en utilisant la même commande `pointe`), on utilisera la commande suivante :

`ParametrePointeType(`⟨*s*⟩`)`

⟨***s***⟩ : `string`, peut valoir `pointe` (fonctionnement par défaut avec un cercle noir rempli d'une autre couleur, blanc par défaut), `"croix"` (pour une simple croix) ou `"disque"` (pour un disque plein).

Il arrive régulièrement que l'on veuille pointer une liste de points[4]. Pour cela, on dispose de la commande suivante :

4. Sur une idée d'Alain Matthes.

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

⟨***pointA***⟩, ⟨***pointB***⟩, **etc. :** liste de `point` ou `pair` séparés par des virgules.

4.4 Contenir les tracés

Il est souvent nécessaire de restreindre les tracés à une zone du plan $\mathbf{R}^2$. Pour cela, `mp-geom2d` fournit la commande `Fenetre` :

`Fenetre(`⟨*Xmin*⟩`,`⟨*Ymin*⟩`,`⟨*Xmax*⟩`,`⟨*Ymin*⟩`)`

Cette fonction trace un rectangle dont deux sommets opposés sont les points `(Xmin,Ymin)` et `(Xmax,Ymax)` avec la couleur METAPOST `background` et découpe l'image courante autour de ce cadre (avec la commande METAPOST `clip currentpicture`).

On pourra aussi restreindre les tracés à l'intérieur d'un chemin fermé issu d'un objet `mp-geom2d`. Pour cela, on utilisera la commande suivante :

`gddClip(`⟨*Objet*⟩`)`

⟨***Objet***⟩ **:** tout objet `mp-geom2d` « fermé ».

En interne, la macro `Fenetre` modifie la variable MetaPost `extra_endfig`. Pour vider cette chaîne de caractère, on pourra utiliser la commande suivante :

`FinFenetre`

5 Les types

`mp-geom2d`, fournit plusieurs types d'objets. Le type d'objet est stocké dans la table `gddT[]`, et les tables `gddA[]` à `gddF[]`, ainsi que la table `gddX` contiennent les propriétés des objets.

Nous allons ici décrire chaque type de l'extension `mp-geom2d`, leurs propriétés respectives, ainsi que des méthodes associées directement à l'objet.

5.1 Le type `point`

Ce type correspond au point de l'espace euclidien.

5.1.1 Constructeur

Pour être plus clair, voici la fonction principale pour créer un tel objet :

```
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`.

Cette fonction « retourne » le compteur `gdd0` et crée dans la table de type une entrée `point` et les attributs (coordonnées) correspondants `a` et `b` dans les tables `gddA` et `gddB`.

Avec un tel type de fonctionnement, la plupart des manipulations se font sur des `numeric`s. En effet, pour déclarer un `point`, il suffit d'écrire

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

`A` prend alors la valeur courante de `gdd0`. C'est l'identifiant du point.

On peut aussi définir un `point` par ses coordonnées *polaires* avec la commande suivante.

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

⟨***r***⟩ : `numeric`, module du point ;

⟨***a***⟩ : `numeric`, angle par rapport à l'axe des abscisses.

On peut définir un point dans un repère défini lui par un point d'origine et deux vecteurs avec la fonction suivante :

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

⟨***x***⟩ : `numeric`;

⟨***y***⟩ : `numeric`;

⟨***o***⟩ : `point` d'origine du repère ;

⟨***i***⟩ : `vecteur` (ou `point`) définissant l'axe des abscisses ;

⟨***j***⟩ : `vecteur` (ou `point`) définissant l'axe des ordonnées.

Le code suivant illustre l'utilisation de cette commande (avec des `point`s au lieu de `vecteur`s, type décrit plus tard).

```
O = Point(0,0);
I = Point(2,0);
```

```
J = Point(0,3);
A = PointDansRepere(2,3,O,I,J);
```

5.1.2 Macros associées

On peut récupérer l'abscisse et l'ordonnée d'un point avec les commandes suivantes :

`Abscisse(`⟨*a*⟩`)` → `numeric`

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

`Ordonnee(`⟨*a*⟩`)` → `numeric`

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

On pourra ajouter les abscisses, les ordonnées ou bien, à la manière des vecteurs, des points entre eux avec les commandes suivantes.

`AdditionAbscisses(`⟨*a*⟩`,`⟨*b*⟩`)` → `numeric`

⟨***a***⟩ : `point` ou `pair`;

⟨***b***⟩ : `point` ou `pair`.

`AdditionOrdonnee(`⟨*a*⟩`,`⟨*b*⟩`)` → `numeric`

⟨***a***⟩ : `point` ou `pair`;

⟨***b***⟩ : `point` ou `pair`.

et

`Addition(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩ : `point` ou `pair`;

⟨***b***⟩ : `point` ou `pair`.

On peut aussi calculer la longueur entre deux points grâce à la commande suivante :

`Longueur(`⟨*a*⟩`,`⟨*b*⟩`)` → `numeric`

⟨***a***⟩ : `point` ou `pair`;

⟨***b***⟩ : `point` ou `pair`.

On peut calculer le point équidistant de deux points, le milieu :

`Milieu(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩ : `point` ou `pair` ;

⟨***b***⟩ : `point` ou `pair`.

On peut réaliser la rotation d'un point autour de l'origine $(0,0)$ avec la commande suivante :

`Rotation(`⟨*a*⟩`,`⟨*b*⟩`)` → `point`

⟨***a***⟩ : `point` ou `pair` ;

⟨***b***⟩ : `numeric`, l'angle de rotation en radian.

Si on veut réaliser la rotation d'un point autour d'un autre, on utilisera la commande suivante :

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

⟨***a***⟩ : `point` ou `pair` ;

⟨***b***⟩ : `point` ou `pair`, centre de rotation ;

⟨***c***⟩ : `numeric`, l'angle de rotation en radian.

On peut calculer l'isobarycentre d'une liste de points (ou de `pair` de METAPOST) avec la commande suivante.

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

⟨***a***⟩ : `point` ou `pair` ;

⟨***b***⟩ : `point` ou `pair` ;

etc.

On peut calculer le barycentre d'une liste de points associés à des poids (mais ici, il n'est pas possible d'utiliser des `pair` de METAPOST).

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

⟨***A***⟩ : `point` ;

⟨***a***⟩ : `numeric`, poids associé à ⟨*A*⟩ ;

⟨***B***⟩ : `point` ;

⟨***b***⟩ : `numeric`, poids associé à ⟨*B*⟩ ;

etc.

On peut déterminer la bissectrice d'un secteur angulaire défini par trois points. La fonction retourne une `droite`.

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

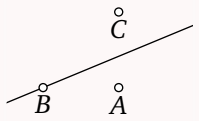
⟨***A***⟩ : `point`;

⟨***B***⟩ : `point`;

⟨***C***⟩ : `point`.

Exemple 2

```
input geom2d;
beginfig(1);
A = Point(1,0);
B = Point(0,0);
C = Point(1,1);
D = Bissectrice(A,B,C);
trace D;
marque.bot "A";
marque.bot "B";
marque.bot "C";
Fenetre(-0.5,-0.5,2,2);
endfig;
```



Le type `point` est évidemment lié au type METAPOST `pair`.

`mp-geom2d` fournit des macros qui permettent de passer de `point` à `pair` et réciproquement.

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

⟨***a***⟩ : `numeric`, abscisse ;

⟨***b***⟩ : `numeric`, ordonnée.

`PairTOPoint(`⟨*p*⟩`)` → `point`

⟨***p***⟩ : `pair`.

Ces deux commandes sont complétées par deux autres qui assurent qu'un élément est d'un type donné :

`PairImp(`⟨*p*⟩`)` → `pair`

⟨***p***⟩ : `point` ou `pair`.

`PointImp(`⟨*p*⟩`)` → `point`

⟨***p***⟩ : `point` ou `pair`.

On peut aussi récupérer un point le long d'un objet `mp-geom2d` (décrit dans les sections suivantes). La macro suivante retourne un `point` le long du chemin (cyclique ou non) de l'objet le paramétrant entre 0 et 1.

`PointDe(`⟨*o*⟩`,`⟨*t*⟩`)` → `point`

⟨***o***⟩ : n'importe quel objet `mp-geom2d` (même un `point` pour lequel lui-même est retourné) ;

⟨***t***⟩ : `numeric` compris entre 0 et 1 (qui paramètre le chemin de l'objet entre 0 et 1).

5.2 Le type `vecteur`

Ce type correspond aux vecteurs définis à l'aide de deux coordonnées de l'espace euclidien.

5.2.1 Constructeurs

La fonction créatrice d'un tel objet est celle-ci

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

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

⟨***a***⟩ : `numeric` ;

⟨***b***⟩ : `numeric`.

Cette fonction a la même architecture que celle correspondante au `point` : elle retourne la valeur courante de `gdd0` après incrémentation, puis affecte le type `vecteur` à l'entrée correspondante dans la table `gddT`.

On peut aussi définir un vecteur à partir d'un `pair` METAPOST.

```
vardef VecteurP(expr a) =
  save n; n = incr gdd0;
```

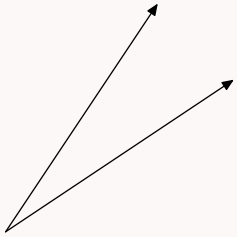
```
  gddT[n] = "vecteur"; gddA[n] = xpart a; gddB[n] = ypart a; n
enddef;
```

VecteurP(⟨a⟩) → vecteur
⟨a⟩ : pair.

Un vecteur peut se définir alors comme ceci :

Exemple 3

```
input geom2d;
beginfig(1);
AB = Vecteur(2,3);
pair a;
a := (3,2);
A = VecteurP(a);
trace AB;
trace A;
endfig;
```



5.2.2 Macros associées

Comme les objets de mp-geom2d ne sont pas des numerics, les opérations classiques de l'espace vectoriel ne peuvent pas s'écrire avec les simples caractères +, -, et *.

AdditionVecteur(⟨a⟩,⟨b⟩) → vecteur
⟨a⟩ : vecteur;
⟨b⟩ : vecteur.

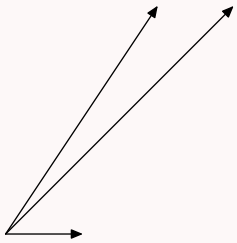
SoustractionVecteur(⟨a⟩,⟨b⟩) → vecteur
⟨a⟩ : vecteur;
⟨b⟩ : vecteur.

ScalaireVecteur(⟨k⟩,⟨b⟩) → vecteur
⟨k⟩ : numeric;
⟨b⟩ : vecteur.

On pourra alors faire les opérations classiques sur les vecteurs.

Exemple 4

```
input geom2d;
beginfig(1);
AB = Vecteur(2,3);
BC = Vecteur(1,0);
AC = AdditionVecteur(AB,BC);
kAC = ScalaireVecteur(3.0,AC);
trace AB;
trace BC;
trace AC;
endfig;
```



On peut aussi calculer le produit scalaire de deux vecteurs avec la commande suivante.

`ProduitScalaire(`$\langle a \rangle$`,`$\langle b \rangle$`)` → `numeric`

$\langle \boldsymbol{a} \rangle$: `vecteur`;

$\langle \boldsymbol{b} \rangle$: `vecteur`.

On peut aussi calculer la norme euclidienne d'un vecteur.

`Norme(`$\langle a \rangle$`)` → `numeric`

$\langle \boldsymbol{a} \rangle$: `vecteur`.

La commande suivante permet d'obtenir l'angle, en radian, entre $[0,\pi]$ formé entre deux vecteurs. Si on note u et v les deux vecteurs de $\mathbf{R}^2$, l'angle calculé par la commande suivante est obtenu avec la formule suivante :

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

`VecteursAngle(`$\langle a \rangle$`,`$\langle b \rangle$`)` → `numeric` (dans $[0,\pi]$)

$\langle \boldsymbol{a} \rangle$: `vecteur`;

$\langle \boldsymbol{b} \rangle$: `vecteur`.

5.3 Le type `segment`

Les segments sont définis par deux points de $\mathbf{R}^2$.

5.3.1 Constructeur

La fonction créatrice de cet objet est celle-ci

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

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

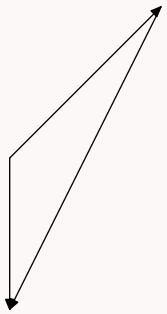
⟨***a***⟩ : `point`;

⟨***b***⟩ : `point`.

Un segment se définit alors comme ceci :

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);
fleche AB;
fleche BC;
fleche AC;
endfig;
```



5.3.2 Macros associées

On peut calculer la longueur d'un segment avec la macro suivante.

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

⟨***a***⟩ : `segment`.

On peut aussi convertir un segment en vecteur avec la macro suivante qui fait la différence des coordonnées des deux points définissant le segment.

`SegmentTOVecteur(`⟨*a*⟩`)` → `vecteur`

⟨***a***⟩ : `segment`.

5.4 Le type droite

Une droite est simplement définie par deux points.

5.4.1 Constructeur

Ainsi la fonction créatrice de cet objet est la suivante

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

`Droite(`⟨*a*⟩`,`⟨*b*⟩`)` → `droite`

⟨***a***⟩ : `point`;

⟨***b***⟩ : `point`.

Lors de la représentation des droites (voir section 4.3), le caractère *infini* de la droite est géré par la variable globale `gddExtensionDroite` qui vaut 10 par défaut.

On pourra la modifier avec la commande suivante :

`Set(`⟨*d*⟩`)`

⟨***d***⟩ : `numeric` (valeur par défaut 10).

5.4.2 Macros associées

La macro suivante permet, sous forme d'un triplet, et donc d'une `color`, d'obtenir les coefficients d'une droite donnée. Pour une droite (D), la macro donne le triplet $(u,v,w) \in \mathbf{R}^3$ tel que $\forall (x,y) \in (D)$, $ux+vy+w=0$.

`EquationDroite(`⟨*a*⟩`)` → `color`

⟨***a***⟩ : `droite`.

On peut calculer la projection d'un point sur une droite avec la commande suivante.

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

⟨***p***⟩ : `point`;

⟨***a***⟩ : `droite`.

On peut obtenir l'ordonnée relative d'un point sur une droite avec la macro suivante.

```
OrdonneeRelativePointDroite(⟨p⟩,⟨a⟩) → numeric
        ⟨p⟩ : point;
        ⟨a⟩ : droite.
```

Le calcul de la distance d'un point à une droite se fait avec la macro suivante.

```
DistancePointDroite(⟨p⟩,⟨a⟩) → numeric
        ⟨p⟩ : point;
        ⟨a⟩ : droite.
```

La macro suivante permet d'obtenir la droite perpendiculaire à un droite passant par un point.

```
DroitePerpendiculaire(⟨a⟩,⟨p⟩) → droite
        ⟨a⟩ : droite;
        ⟨p⟩ : point.
```

Sur le même modèle, la macro suivante permet d'obtenir la droite parallèle à une autre passant par un point.

```
DroiteParallele(⟨a⟩,⟨p⟩) → droite
        ⟨a⟩ : droite;
        ⟨p⟩ : point.
```

On pourra obtenir le `point` d'intersection de deux droites avec la macro suivante.

```
IntersectionDroites(⟨a⟩,⟨b⟩) → point
        ⟨a⟩ : droite;
        ⟨b⟩ : droite.
```

La macro suivante permet de reporter un point sur une droite avec une certaine longueur (signée), à la manière d'un compas.

```
ReportSurDroite(⟨P⟩,⟨D⟩,⟨l⟩) → point
```

⟨*P*⟩ : point ;

⟨*D*⟩ : droite ;

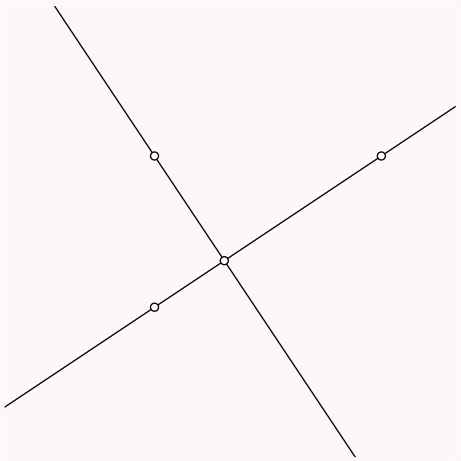
⟨*l*⟩ : numeric (peut être négatif).

Le côté du report par rapport au point ⟨*P*⟩ dépend de la définition de la droite : pour une droite définie par deux points *A* et *B*, alors la direction du report sera suivant le vecteur $\overrightarrow{AB}$. Pour changer de direction, il suffira de multiplier le paramètre de distance ⟨*l*⟩ par −1.

Voici un exemple permettant d'illustrer l'utilisation de quelques commandes relatives aux droites.

Exemple 6

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Droite(A,B);
trace AB;
pointe A;
pointe B;
C = Point(0,2);
DC = DroitePerpendiculaire(AB,C);
trace DC;
pointe C;
D = ProjectionPointSurDroite(C,AB
    );
pointe D;
Fenetre(-2,-2,4,4);
endfig;
```



5.5 Le type cercle

Un cercle peut être défini par un point et un rayon.

5.5.1 Constructeurs

La fonction créatrice de base de cet objet est la suivante

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

Cercle(⟨*a*⟩,⟨*b*⟩) → cercle

⟨*a*⟩ : point ou pair, centre du cercle ;

⟨*b*⟩ : numeric, rayon du cercle.

Deux autres fonctions permettent de définir des cercles. La fonction `CercleCP` définit le cercle par un centre et un point par lequel passe le cercle.

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

⟨***a***⟩ : `point` ou `pair`, centre du cercle ;

⟨***b***⟩ : `point`, tel que ⟨*b*⟩−⟨*a*⟩ est le rayon.

On peut aussi définir un cercle par deux points définissant son diamètre. La fonction correspondante est la suivante

`CercleD(`⟨*a*⟩`,`⟨*b*⟩`)` → `cercle`

⟨***a***⟩ : `point` ou `pair` ;

⟨***b***⟩ : `point` ou `pair`, tel que (⟨*a*⟩−⟨*b*⟩) est le diamètre du cercle.

On peut aussi définir un cercle comme le cercle qui passe par trois points avec la macro suivante.

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

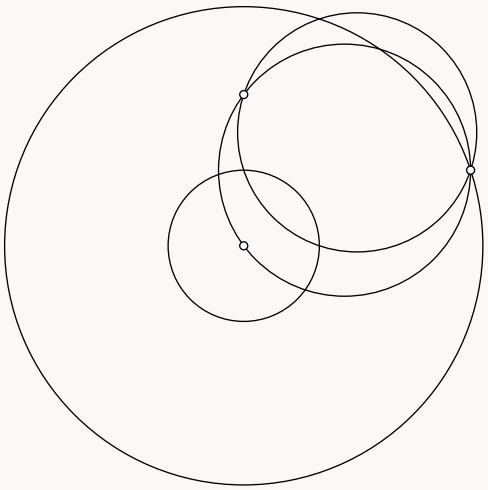
⟨***a***⟩ : `point` ou `pair` ;

⟨***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 = Cercle(O,r);
COA = CercleCP(O,A);
CAB = CercleD(A,B);
COAB = CercleTroisPoints(O,A,B);
trace COr;
trace COA;
trace CAB;
trace COAB;
pointe A; pointe B; pointe O;
endfig;
```



5.5.2 Macros associées

De nombreuses macros sont associées aux cercles.
On pourra récupérer le rayon d'un cercle avec la commande suivante.

`Rayon(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩ : `cercle`.

On pourra obtenir le centre (`point`) d'un cercle avec la commande :

`Centre(`⟨*a*⟩`)` → `point`

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

On peut calculer l'intersection entre deux cercles grâce à la macro suivante.

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

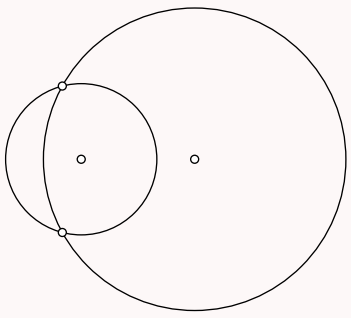
⟨***a***⟩ : `cercle`;

⟨***b***⟩ : `cercle`.

Cette macro ne donnera qu'un seul point d'intersection. Pour obtenir les deux points d'intersection, il faudra inverser l'ordre d'appel sur les cercles comme l'exemple suivant l'illustre.

Exemple 8

```
input geom2d;
beginfig(1);
A = Point(0,0);
CA = Cercle(A,1);
B = Point(1.5,0);
CB = Cercle(B,2);
D1 = IntersectionCercles(CA,CB);
D2 = IntersectionCercles(CB,CA);
trace CA; trace CB;
pointe A; pointe B; pointe D1;
    pointe D2;
endfig;
```



Si il n'existe pas d'intersection, alors la compilation échouera avec l'erreur `! Pythagorean subtraction X.XXXX+-+X.XXXX has been replaced by 0`. Si les deux cercles sont confondus, alors l'erreur sera `! Division by zero.`.

`mp-geom2d` fournit aussi une macro permettant d'obtenir les intersections entre une droite et un cercle.

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

⟨***d***⟩ : `droite`;

⟨***c***⟩ : `cercle`;

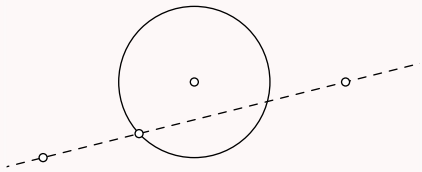
⟨***n***⟩ : `numeric` qui vaut 1 ou 2 suivant le point d'intersection que l'on souhaite (s'il n'existe qu'un point d'intersection alors les deux valeurs renvoient le même point).

S'il n'existe pas d'intersection entre la droite et le cercle, vous obtiendrez l'erreur ! `Intersection between line and circle does not exist`.

L'exemple suivant permet d'illuster cette macro.

Exemple 9

```
input geom2d;
beginfig(1);
A = Point(2,2);
C_A = Cercle(A,1);
B = Point(0,1);
C = Point(4,2);
BC = Droite(B,C);
E1 = IntersectionDroiteCercle(BC,
    C_A,1);
trace C_A;
trace BC dashed evenly;
pointe A; pointe B; pointe C;
    pointe E1;
Fenetre(-0.5,-0.5,5,4);
endfig;
```



On peut obtenir les tangentes intérieures et extérieures communes à deux cercles avec les deux macros suivantes. Là encore, comme il existe deux tangentes, pour obtenir les deux, on inversera l'ordre d'appel des deux cercles.

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

⟨***a***⟩ : `cercle`;

⟨***b***⟩ : `cercle`.

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

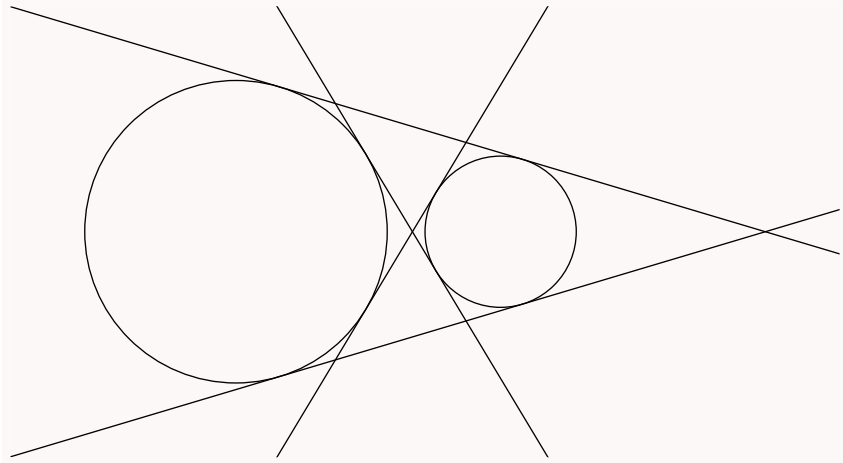
⟨***a***⟩ : `cercle`;

⟨***b***⟩ : `cercle`.

L'exemple suivant illustre l'utilisation de ces deux macros.

Exemple 10

```
input geom2d;
beginfig(1);
C1 = Cercle((-2,0),2);
C2 = Cercle((1.5,0),1);
T1 = TangenteCommuneExterieure(C1,C2);
T2 = TangenteCommuneExterieure(C2,C1);
T3 = TangenteCommuneInterieure(C1,C2);
T4 = TangenteCommuneInterieure(C2,C1);
trace C1; trace C2;
trace T1; trace T2;
trace T3; trace T4;

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



La macro suivante permet de calculer l'axe radical (`droite`) de deux cercles[5].

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

⟨***a***⟩ : `cercle`;

⟨***b***⟩ : `cercle`.

La macro suivante permet d'obtenir le centre radical de trois cercles.

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

5. La méthode de construction a été largement inspirée du code de T. Thurston dans son document *Drawing with METAPOST* [6].

⟨*a*⟩ : cercle;
⟨*b*⟩ : cercle;
⟨*c*⟩ : cercle.

La macro suivante permet de calculer l'axe de similitude de trois cercles.

AxeDeSimilitude(⟨*a*⟩,⟨*b*⟩,⟨*c*⟩) → droite
⟨*a*⟩ : cercle;
⟨*b*⟩ : cercle;
⟨*c*⟩ : cercle.

5.6 Le type ellipse

5.6.1 Constructeurs

Les ellipses peuvent être définies de plusieurs façons. Tout d'abord, on peut la définir avec son centre, un des points de l'ellipse sur le grand axe (appelé vertex), et un des points de l'ellipse sur le petit axe (appelé co-vertex). Cependant, lors de la création d'une ellipse de nombreux attributs sont calculés.

Le code du constructeur est le suivant :

```
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` ou `pair`, le centre ;

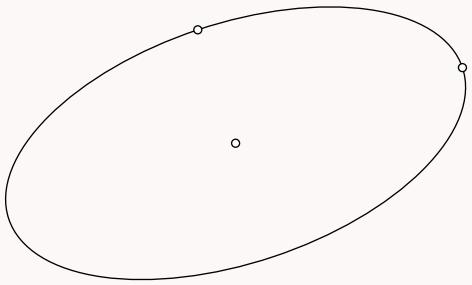
⟨***a***⟩ : `point` ou `pair`, le vertex ;

⟨***b***⟩ : `point` ou `pair`, le co-vertex.

Cette macro s'utilise comme le montre l'exemple suivant.

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);
trace E;
pointe A; pointe B; pointe C;
endfig;
```



On pourra définir une ellipse avec la donnée de ses deux foyers et du demi-grand axe avec la commande suivante :

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

⟨***A***⟩ : `point` ou `pair`, premier foyer ;

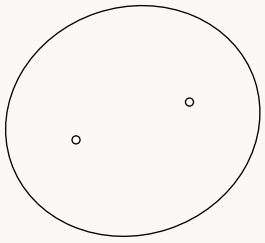
⟨***B***⟩ : `point` ou `pair`, deuxième foyer ;

⟨***a***⟩ : `numeric`, le demi-grand axe.

Cette macro s'utilise comme le montre l'exemple suivant.

Exemple 12

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



On pourra aussi définir une ellipse à partir d'un foyer, de sa directrice et de l'excentricité avec la commande suivante :

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

⟨***F***⟩ : `point` ou `pair`, un foyer ;

⟨***D***⟩ : `droite`, la directrice associée ;

⟨***e***⟩ : `numeric`, l'excentricité (< 1).

5.6.2 Macros associées

On pourra obtenir le centre (`point`) d'une ellipse avec la commande :

`Centre(`⟨*a*⟩`)` → `point`

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

On obtient le vertex et le co-vertex avec les commandes suivantes :

`Vertex(`⟨*a*⟩`)` → `point`

⟨***a***⟩ : `ellipse`.

`CoVertex(`⟨*a*⟩`)` → `point`

⟨***a***⟩ : `ellipse`.

La commande suivante permet d'obtenir les deux foyers selon l'entier passé en argument.

`Foyer(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`

⟨***a***⟩ : `ellipse` ;

⟨***n***⟩ : `numeric`, entier qui vaut 1 ou 2 pour obtenir chaque foyer.

On peut obtenir le demi-grand axe et le demi-petit axe avec les commandes suivantes.

`DemiGrandAxe(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩ : `ellipse`.

`DemiPetitAxe(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩ : `ellipse`.

On obtient l'excentricité, souvent notée e, avec la commande suivante.

`Excentricite(`⟨*a*⟩`)` → `numeric`

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

Pour obtenir l'inclinaison (coefficient directeur de la droite passant par les foyers de l'ellipse), on utilisera la commande suivante.

`Inclinaison(`⟨*a*⟩`)` → `numeric`

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

On peut obtenir les directrices de l'ellipse avec la commande suivante :

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

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

⟨***n***⟩ : 1 ou 2 (`numeric`), pour choisir l'une des deux directrices.

Pour obtenir la tangente (`droite`) à un point de l'ellipse, on utilisera la commande suivante.

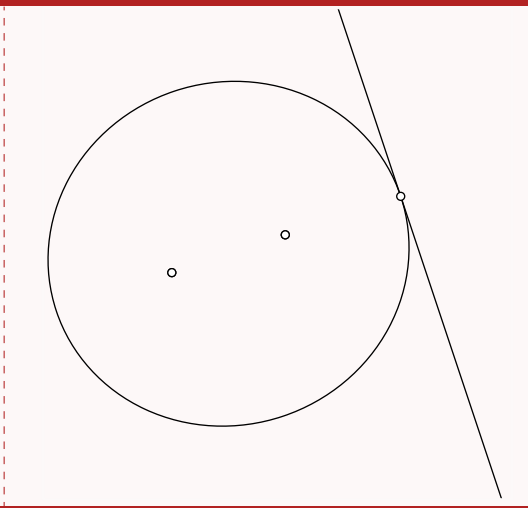
`TangenteEllipse(`⟨*e*⟩`,`⟨*p*⟩`)` → `droite`

⟨***e***⟩ : `ellipse`;

⟨***p***⟩ : `point` ou `pair`.

Exemple 13

```
input geom2d;
beginfig(1);
F1 = Point(3,1);
F2 = Point(1.5,0.5);
E = EllipseF(F1,F2,2.4);
pointe F1; pointe F2;
trace E;
M' = PointDe(E,0.5);
D = TangenteEllipse(E,M');
trace D;
pointe M';
pointe Foyer(E,1);
pointe Foyer(E,2);
Fenetre(-0.2,-2.5,6,4);
endfig;
```



Pour obtenir les tangentes (`droite`) passant par un point extérieur à l'ellipse, on utilisera la commande suivante. Si le point choisi n'est pas extérieur à l'ellipse, alors il y aura une erreur de compilation.

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

⟨***e***⟩ : `ellipse`;

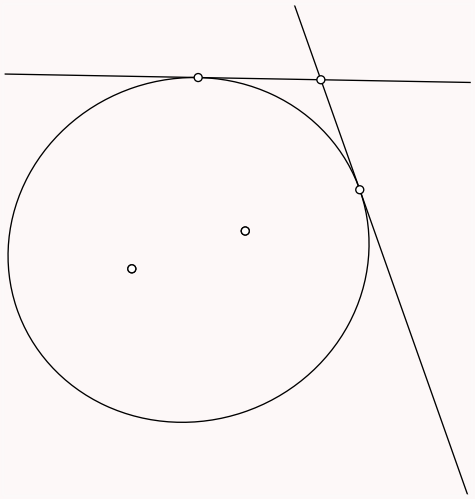
⟨***p***⟩ : `point` ou `pair`;

⟨***n***⟩ : `numeric` qui vaut 1 ou 2 pour choisir la tangente parmi les deux possibles.

Les points de tangence sont accessibles comme deuxième point de la définition de la `droite` grâce à la table `gddB[]`.

Exemple 14

```
input geom2d;
beginfig(1);
F1 = Point(3,1);
F2 = Point(1.5,0.5);
E = EllipseF(F1,F2,2.4);
pointe F1; pointe F2;
trace E;
M = Point(4,3);
D1 = TangenteExterieureEllipse(E,
    M,1);
trace D1;
pointe gddB[D1];
D2 = TangenteExterieureEllipse(E,
    M,2);
trace D2;
pointe gddB[D2];
pointe M;
pointe Foyer(E,1);
pointe Foyer(E,2);
Fenetre(-0.2,-2.5,6,4);
endfig;
```



5.7 Le type parabole

Le constructeur de base de la parabole définit cet objet à partir du foyer et de la directrice. Le code du constructeur est le suivant :

```
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;
```

`ParaboleFD(`⟨*f*⟩`,`⟨*d*⟩`)` → `chemin`

⟨***f***⟩ : est le foyer (`point`) de la parabole ;

⟨***d***⟩ : est la directrice (`droite`) de la parabole.

Lors de la représentation d'une parabole, le caractère *infini* est géré par la variable globale `gddC2Dparam` qui vaut 15 par défaut.

Pour la modifier, on utilisera la commande suivante :

`ParametreConiqueCoef(`⟨*n*⟩`)`

⟨***n***⟩ : `numeric`, paramètre pour le tracé des hyperboles et paraboles (valeur par défaut 15).

5.7.1 Fonctions associées

On obtient l'excentricité, souvent notée *e*, avec la commande suivante.

`Excentricite(`⟨*a*⟩`)` → `numeric`

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

Pour obtenir l'inclinaison (coefficient directeur de la droite axe de symétrie de la parabole), on utilisera la commande suivante :

`Inclinaison(`⟨*a*⟩`)` → `numeric`

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

On peut obtenir le sommet de la parabole avec la commande suivante :

`Sommet(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`

⟨***a***⟩ : `parabole` ou `hyperbole` ;

⟨***n***⟩ : 1 ou 2 (`numeric`), argument utile pour l'hyperbole qui possède deux sommets. Ici, peu importe la valeur de ⟨*n*⟩, l'unique sommet sera donné.

On peut obtenir la directrice de la parabole avec la commande suivante :

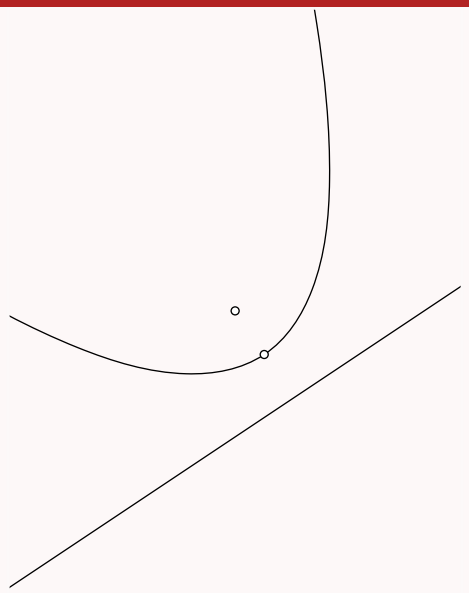
`Directrice(`⟨*a*⟩`,`⟨*n*⟩`)` → `droite`

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

⟨***n***⟩ : 1 ou 2 (`numeric`), argument utile pour l'hyperbole qui possède deux directrices. Ici, peu importe la valeur de ⟨*n*⟩, l'unique directrice sera donnée.

Exemple 15

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Droite(A,B);
F = Point(-1,1);
Para = ParaboleFD(F,AB);
trace AB;
trace Para;
pointe Sommet(Para,1);
pointe F;
Fenetre(-4,-5,2,5);
endfig;
```



5.8 Le type hyperbole

Le constructeur de base de l'hyperbole définit cet objet à partir du foyer et de la directrice. Le code du constructeur est le suivant :

```
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;
```

On peut voir dans ce constructeur l'utilisation de la table de `path` étendu `gddPX[][]` qui permet d'associer plusieurs `path`s à un seul objet.

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

⟨***f***⟩ : est le foyer (`point`) de l'hyperbole ;

⟨***d***⟩ : est la directrice (`droite`) de l'hyperbole ;

⟨***e***⟩ : est l'excentricité de l'hyperbole.

On pourra aussi définir une hyperbole avec ses deux foyers et la distance entre un sommet et le centre de l'hyperbole (définition bifocale).

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

⟨***f1***⟩ : est un foyer (`point`) de l'hyperbole ;

⟨***f2***⟩ : est le deuxième foyer (`point`) de l'hyperbole ;

⟨***a***⟩ : est la moitié de la distance entre les deux sommets de l'hyperbole.

5.8.1 Fonctions associées

On pourra obtenir le centre (`point`) d'une hyperbole avec la commande :

`Centre(`⟨*a*⟩`)` → `point`

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

On obtient l'excentricité, souvent notée e, avec la commande suivante.

`Excentricite(`⟨*a*⟩`)` → `numeric`
⟨***a***⟩ : `ellipse`, `parabole` ou `hyperbole`.

Pour obtenir l'inclinaison (coefficient directeur de la droite passant par les foyers de l'hyperbole), on utilisera la commande suivante.

`Inclinaison(`⟨*a*⟩`)` → `numeric`
⟨***a***⟩ : `ellipse`, `parabole` ou `hyperbole`.

On peut obtenir les sommets de l'hyperbole avec la commande suivante :

`Sommet(`⟨*a*⟩`,`⟨*n*⟩`)` → `point`
⟨***a***⟩ : `parabole` ou `hyperbole`;
⟨***n***⟩ : 1 ou 2 (`numeric`), pour choisir l'un des deux sommets.

On peut obtenir les directrices de l'hyperbole avec la commande suivante :

`Directrice(`⟨*a*⟩`,`⟨*n*⟩`)` → `droite`
⟨***a***⟩ : `ellipse`, `parabole` ou `hyperbole`;
⟨***n***⟩ : 1 ou 2 (`numeric`), pour choisir l'une des deux directrices.

Pour obtenir le *cercle principal* de l'hyperbole, on pourra utiliser la commande suivante :

`CerclePrincipale(`⟨*h*⟩`)` → `cercle`
⟨***h***⟩ : `hyperbole`.

On pourra aussi obtenir les deux asymptotes de l'hyperbole avec la commande suivante :

`AsymptoteHyperbole(`⟨*h*⟩`,`⟨*n*⟩`)` → `droite`
⟨***h***⟩ : `hyperbole`;
⟨***n***⟩ : 1 ou 2 (`numeric`), pour choisir l'une des deux asymptotes.

Parce qu'une hyperbole est constituée de deux parties disjointes, on ne peut pas, comme pour les autres objets `mp-geom2d`, utiliser directement la commande `trace` directement sur l'objet. Il faudra utiliser la commande suivante pour obtenir le `path` associé à une des deux parties de l'hyperbole.

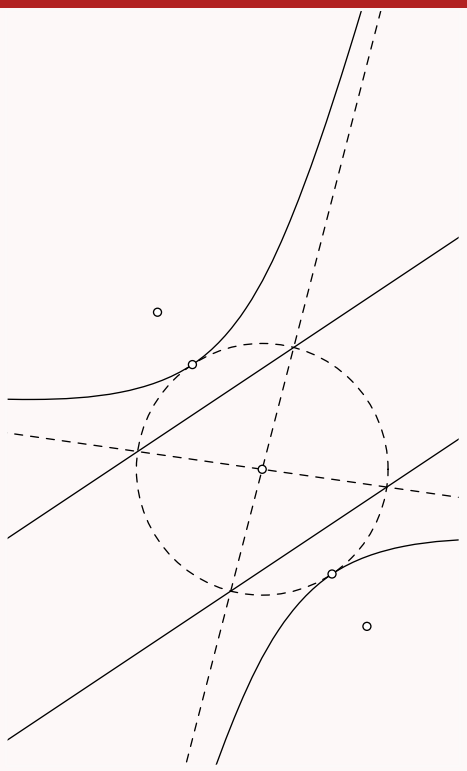
`DemiHyperbole(`⟨*h*⟩`,`⟨*n*⟩`)` → `path`

⟨***h***⟩ : `hyperbole`;

⟨***n***⟩ : 1 ou 2 (`numeric`), pour choisir l'une des deux parties.

Exemple 16

```
input geom2d;
beginfig(1);
A = Point(0,0);
B = Point(3,2);
AB = Droite(A,B);
F = Point(-1,1);
Hyper = HyperboleFD(F,AB,1.5);
trace AB;
trace DemiHyperbole(Hyper,1);
trace DemiHyperbole(Hyper,2);
trace Directrice(Hyper,2);
trace CerclePrincipale(Hyper)
    dashed evenly;
trace AsymptoteHyperbole(Hyper,1)
     dashed evenly;
trace AsymptoteHyperbole(Hyper,2)
     dashed evenly;
pointe Centre(Hyper);
pointe Foyer(Hyper,1);
pointe Foyer(Hyper,2);
pointe Sommet(Hyper,1);
pointe Sommet(Hyper,2);
Fenetre(-3,-5,3,5);
endfig;
```



5.9 Le type `triangle`

Les triangles sont définis par trois points de $\mathbf{R}^2$.

5.9.1 Constructeur

La fonction créatrice de cet objet est la suivante

```
vardef Triangle (expr a,b,c) =
  save n; n = incr gdd0; gddT[n] = "triangle";
```

```
  gddA[n] = PointImp(a); gddB[n] = PointImp(b); gddC[n] =
      PointImp(c); n
enddef;
```

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

⟨***a***⟩ : `point`;

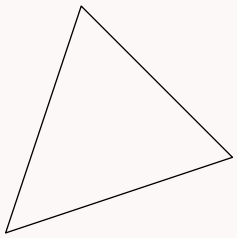
⟨***b***⟩ : `point`;

⟨***c***⟩ : `point`.

Ici, on voit l'appel à la troisième table `gddC` pour stocker le troisième point.
Un triangle se définit alors comme ceci :

Exemple 17

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



5.9.2 Macros associées

La macro suivante permet de calculer l'aire d'un triangle.

`AireTriangle(`⟨*a*⟩`)` → `numeric`

⟨***a***⟩ : `triangle`.

On peut calculer aussi l'orthocentre d'un triangle avec la commande suivante.

`Orthocentre(`⟨*a*⟩`)` → `point`

⟨***a***⟩ : `triangle`.

La macro suivante calcule le cercle inscrit d'un triangle.

`CercleInscrit(`⟨*a*⟩`)` → `cercle`

⟨***a***⟩ : `triangle`.

On peut aussi obtenir le cercle circonscrit avec la macro suivante.

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

⟨***a***⟩ : `triangle`.

On peut aussi calculer les cercles exinscrits. Soient A, B et C trois points définissant un triangle ABC[6]. La commande suivante permet d'obtenir, au choix, un des trois cercles exinscrits.

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

⟨***a***⟩ : `triangle`;

⟨***n***⟩ : `numeric` qui vaut 1, 2 ou 3. Si ⟨*n*⟩= 1, c'est le cercle exinscrit tangent au côté $[BC]$ du triangle, si ⟨*n*⟩= 2, c'est celui tangent au côté $[AC]$ et si ⟨*n*⟩= 3, c'est celui tangent au côté $[AB]$.

On peut aussi obtenir le cercle d'Euler d'un triangle avec la commande suivante.

`CercleEuler(`⟨*a*⟩`)` → `cercle`

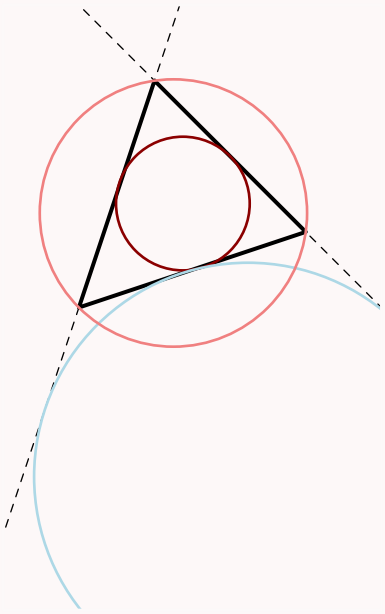
⟨***a***⟩ : `triangle`.

Voici un exemple d'illustration des quelques-unes des macros relatives aux triangles.

6. Évidemment, les points peuvent s'appeler autrement.

Exemple 18

```
input geom2d;
beginfig(1);
A = Point(3,1);
B = Point(1,3);
C = Point(0,0);
AB = Droite(A,B);
BC = Droite(B,C);
ABC = Triangle(A,B,C);
trace AB dashed evenly;
trace BC dashed evenly;
trace ABC avecCrayon(1.5,black);
Euler = CercleEuler(ABC);
trace Euler avecCrayon(1,DarkRed)
    ;
C_E1 = CercleExinscrit(ABC,2);
trace C_E1 avecCrayon(1,LightBlue
    );
C_C = CercleCirconscrit(ABC);
trace C_C avecCrayon(1,LightCoral
    );
Fenetre(-1,-4,4,4);
endfig;
```



5.10 Le type polygone

Les polygones sont définis par N points de $\mathbf{R}^2$.

5.10.1 Constructeurs

La fonction créatrice de cet objet est la suivante

```
vardef Polygone (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; % nombre de côté
  gddB[n] = IsoBarycentre(plist); % centre
  n
enddef;
```

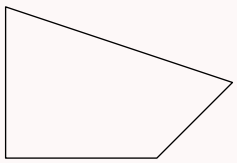
`Polygone(`⟨*liste*⟩`)` → `polygone`

⟨***liste***⟩ : est une liste de `point` ou de `pair`.

Ici, on voit l'appel à la *double* table `gddX` pour stocker les N points.
Un polygone se définit alors comme ceci :

Exemple 19

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



On pourra aussi construire des polygones réguliers avec la commande suivante.

`PolygoneRegulier(`⟨*N*⟩`,`⟨*rayon*⟩`,`⟨*rotation*⟩`,`⟨*translation*⟩`)` → `polygone`

⟨***N***⟩ : entier (`numeric`) indiquant le nombre de points ;

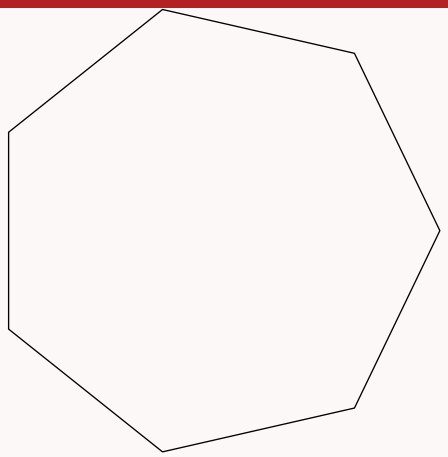
⟨***rayon***⟩ : `numeric`, rayon du cercle circonscrit ;

⟨***rotation***⟩ : `numeric`, rotation du polygone ;

⟨***translation***⟩ : `(`point) ou `vecteur` (ou même `pair`), déplacement du centre du polygone.

Exemple 20

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



5.10.2 Macros associées

On pourra obtenir le nombre de côtés d'un polygone avec la commande suivante.

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

⟨***a***⟩ : polygone.

On pourra avoir accès aux différents sommets du polygone, numérotés à partir de 1, avec la macro suivante.

PointPolygone(⟨*a*⟩,⟨*i*⟩) → point

⟨***a***⟩ : polygone ;

⟨***i***⟩ : numeric, entier plus grand que 1, permettant d'accéder au i^e point du polygone.

5.11 Le type chemin

Pour ce type particulier, mp-geom2d stocke le path dans la table gddP réservée. Ainsi la fonction créatrice de ce type d'objet est la suivante

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

Chemin(⟨*p*⟩) → chemin

⟨***p***⟩ : un path de METAPOST

5.11.1 Macros associées

La macro suivante contruit, à partir d'une liste de points, une ligne brisée de type chemin.

LigneBrisee(⟨*liste*⟩) → chemin

⟨***liste***⟩ : est une liste de point ou de pair.

5.12 Le type courbe

Pour ce type particulier, mp-geom2d stocke une chaine de caractère (un nom de fichier à exploiter avec l'extension graph.mp) dans la table gddS réservée.

5.12.1 Constructeur

Ainsi la fonction créatrice de ce type d'objet est la suivante

```
vardef CourbeDat (expr s) =
  gddT[incr gdd0] = "courbe"; gddS[gdd0] = s; gdd0
enddef;
```

`CourbeDat(`⟨*s*⟩`)` → `courbe`

⟨***s***⟩ : un nom de fichier sous forme de `string` de METAPOST

6 Arc de cercle

Avec `mp-geom2d`, quelques commandes permettent de travailler avec les arcs de cercle. Tout d'abord, on peut décider de ne représenter qu'un arc de cercle d'un `cercle` précédemment défini. Ceci se fait avec la commande suivante qui prend en argument un `cercle` et deux angles en radian, et retourne un `path` de METAPOST correspondant à l'arc de cercle compris entre les deux angles donnés (l'angle 0 étant parallèle à l'axe Ox).

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

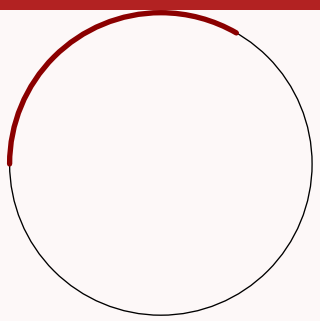
⟨***C***⟩ : `cercle` dont on souhaite représenter un arc ;

⟨***a***⟩ : premier angle en radian (`numeric`) ;

⟨***b***⟩ : deuxième angle en radian (`numeric`).

Exemple 21

```
input geom2d;
beginfig(1);
C = Cercle(origine,2);
trace C;
trace gddTraceArcDeCercle(C,Pi/3,
    Pi) avecCrayon(2,DarkRed);
endfig;
```



On peut aussi définir un objet `chemin`. Pour cela on pourra utiliser les deux commandes suivantes. La première permet de définir un arc de cercle à partir d'un point, d'un rayon et de deux angles.

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

⟨***P***⟩ : `point` ou `pair` centre de l'arc de cercle ;

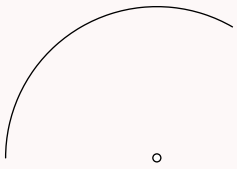
⟨***r***⟩ : rayon de l'arc de cercle (`numeric`) ;

⟨***a***⟩ : premier angle en radian (numeric) ;

⟨***b***⟩ : premier angle en radian (numeric).

Exemple 22

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



La deuxième commande définissant un chemin est la suivante. Elle prend comme arguments trois points, notons les C, A et B : le centre de l'arc de cercle C, et les points A et B. Elle prend ensuite comme argument le rayon de l'arc défini entre les segments $[C,A]$ et $[C,B]$.

ArcEntrePoints(⟨*P*⟩,⟨*r*⟩,⟨*A*⟩,⟨*B*⟩,⟨*s*⟩) → chemin

⟨***P***⟩ : point ou pair centre de l'arc de cercle ;

⟨***r***⟩ : rayon de l'arc de cercle (numeric) ;

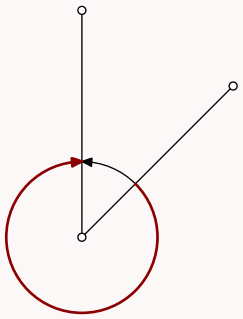
⟨***A***⟩ : premier point ou pair ;

⟨***B***⟩ : deuxième point ou pair ;

⟨***s***⟩ : numeric, -1 ou 1 suivant le sens choisi.

Exemple 23

```
input geom2d;
beginfig(1);
C = Point(0,0);
A = Point(2,2);
B = Point(0,3);
Cab = ArcEntrePoints(C,1,A,B,1);
Cba = ArcEntrePoints(C,1,A,B,-1);
trace Segment(C,A);
trace Segment(C,B);
pointe A; pointe B; pointe C;
fleche Cab;
fleche Cba avecCrayon(1,DarkRed);
endfig;
```



7 Quelques macros

Cette section regroupe quelques macros qui ne sont pas spécifiques à un type.

La macro suivante permet de calculer un point d'intersection entre deux objets `mp-geom2d`. C'est en réalité une surcouche à la primitive METAPOST `intersectionpoint` appliquée sur les objets *tracés* (avec `gddTraceObjet`).

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

⟨***o***⟩ : un objet `mp-geom2d` ;

⟨***b***⟩ : un objet `mp-geom2d`.

8 Quelques transformations

8.1 Homothétie

On peut réaliser une homothétie sur n'importe quel objet `mp-geom2d` avec la commande suivante.

`Homothetie(`⟨*p*⟩`,`⟨*o*⟩`,`⟨*k*⟩`)` → du même type que ⟨*p*⟩

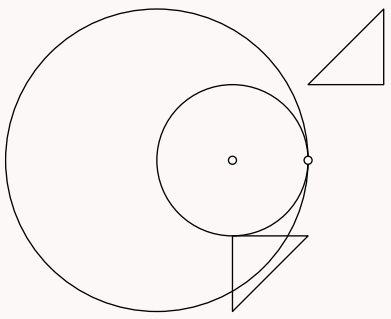
⟨***p***⟩ : un objet `mp-geom2d` ;

⟨***o***⟩ : `point` ou `pair`, centre de l'homothétie ;

⟨***k***⟩ : `numeric`, facteur de l'homothétie.

Exemple 24

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Cercle(B,1);
T = Triangle((1,1),(0,0),(0,1));
C_H = Homothetie(C_B, A, 2);
T_H = Homothetie(T, A ,-1);
trace C_B; trace C_H;
trace T; trace T_H;
pointe A; pointe B;
endfig;
```



8.2 Symétrie axiale

On peut réaliser une symétrie axiale sur n'importe quel objet `mp-geom2d` avec la commande suivante.

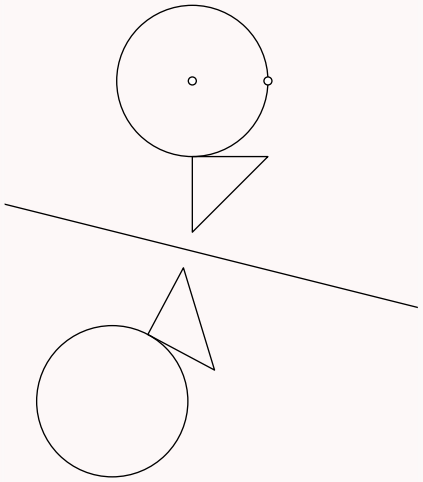
`SymetrieAxiale(`⟨*p*⟩`,`⟨*d*⟩`)` → du même type que ⟨*p*⟩

⟨***p***⟩ : un objet `mp-geom2d` ;

⟨***d***⟩ : `droite` définissant l'axe de symétrie.

Exemple 25

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Cercle(B,1);
T = Triangle((1,1),(0,0),(0,1));
D = Droite((-1,0),(3,-1));
C_H = SymetrieAxiale(C_B, D);
T_H = SymetrieAxiale(T, D);
trace C_B; trace C_H;
trace T; trace T_H;
trace D;
pointe A; pointe B;
Fenetre(-2.5,-3.5,3,3.2);
endfig;
```



8.3 Symétrie centrale

On peut réaliser une symétrie centrale sur n'importe quel objet `mp-geom2d` avec la commande suivante.

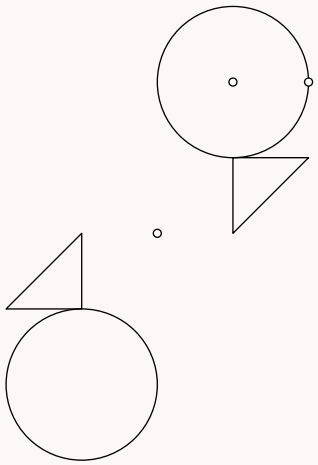
`SymetrieCentrale(`⟨*p*⟩`,`⟨*d*⟩`)` → du même type que ⟨*p*⟩

⟨***p***⟩ : un objet `mp-geom2d` ;

⟨***d***⟩ : `point` définissant le centre de symétrie.

Exemple 26

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Cercle(B,1);
T = Triangle((1,1),(0,0),(0,1));
D = Point(-1,0);
C_H = SymetrieCentrale(C_B, D);
T_H = SymetrieCentrale(T, D);
trace C_B; trace C_H;
trace T; trace T_H;
pointe D;
pointe A; pointe B;
Fenetre(-3.5,-3.5,3,3.2);
endfig;
```



8.4 Inversion

On peut calculer l'inversion d'un point, d'un cercle ou d'une droite par rapport à un cercle avec la macro suivante.

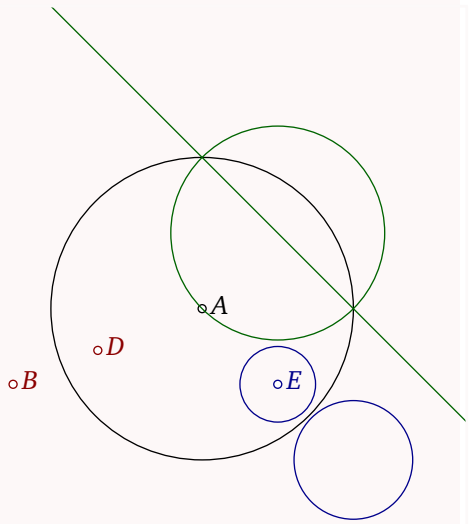
`Inversion(`⟨*p*⟩`,`⟨*c*⟩`)` → `point`

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

⟨***c***⟩ : `cercle`.

Exemple 27

```
input geom2d;
beginfig(1);
  A = Point(2,2);
  C = Cercle(A,2);
  trace C;
  marque.rt "A";
  B = Point(-0.5,1);
  D = Inversion(B,C);
  drawoptions(withcolor DarkRed);
  marque.rt "B"; marque.rt "D";

  drawoptions(withcolor DarkBlue)
      ;
  E = Point(3,1);
  CE = Cercle(E,0.5);
  trace CE;
  marque.rt "E";
  iCE = Inversion(CE,C);
  trace iCE;


  drawoptions(withcolor DarkGreen
      );
  d = Droite((3,3),(4,2));
  trace d;
  Cd = Inversion(d,C);
  trace Cd;
  Fenetre(-1,-1,5.5,6);
endfig;
```



8.5 Similitude à centre (directe)

On peut calculer la similitude à centre d'un point de rapport donnée. Cela se fera avec la macro suivante.

`SimilitudeACentre(`⟨*p*⟩`,`⟨*d*⟩`,`⟨*a*⟩`,`⟨*k*⟩`)` → du même type que ⟨*p*⟩

⟨***p***⟩ : un objet `mp-geom2d` ;

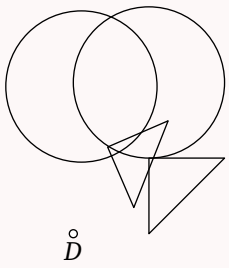
⟨***d***⟩ : `point` définissant le centre de la similitude ;

⟨***a***⟩ : angle en radian (`numeric`) de la similitude ;

⟨***k***⟩ : facteur (`numeric`) de la similitude.

Exemple 28

```
input geom2d;
beginfig(1);
A = Point(1,2);
B = Point(0,2);
C_B = Cercle(B,1);
facteurSimi = 0.4;
angleSimi = Pi/3;
T = Triangle((1,1),(0,0),(0,1));
D = Point(-1,0);
C_H = SimilitudeACentre(C_B, D,
    angleSimi,facteurSimi);
T_H = SimilitudeACentre(T, D,
    angleSimi,facteurSimi);
trace C_B; trace C_H;
trace T; trace T_H;
marque.bot "D";
Fenetre(-3.5,-3.5,3,3.2);
endfig;
```



9 Annotations et labels

9.1 Signes

mp-geom2d fournit la macro suivante pour marquer un angle droit formé par trois points :

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

⟨***a***⟩,⟨***b***⟩,⟨***c***⟩ : sont les trois `point`s formant l'angle droit $\widehat{ABC}$;

⟨***x***⟩ : est la « taille » (`numeric`) du signe d'orthogonalité.

On peut aussi réaliser des marques entre deux points ou sur un segment, un vecteur, ou tout type d'objet. Pour cela, la première macro est la suivante :

`Marque(`⟨*a*⟩`,`⟨*b*⟩`,`⟨*n*⟩`)`

⟨***a***⟩ : premier `point` formant le segment à marquer ;

⟨***b***⟩ : deuxième `point` formant le segment à marquer ;

⟨***n***⟩ : est le type (`numeric`) de marque, il y en a quatre ⟨*n*⟩= 1, 2, 3 ou 4.

On peut aussi marquer n'importe quel type de trait avec la macro suivante :

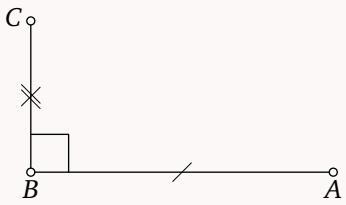
`MarqueTrait(`⟨*a*⟩`,`⟨*n*⟩`)`

⟨***a***⟩ : est n'importe quel objet `mp-geom2d` ou même un `path` METAPOST ;

⟨***n***⟩ : est le type (`numeric`) de marque, il y en a quatre ⟨*n*⟩= 1, 2, 3 ou 4.

Exemple 29

```
input geom2d;
beginfig(1);
A = Point(4,0);
B = Point(0,0);
C = Point(0,2);
trace Segment(B,A);
trace Segment(B,C);
marque.bot "A";
marque.bot "B";
marque.lft "C";
trace SigneOrtho(A,B,C,0.5);
trace Marque(A,B,1);
trace MarqueTrait(Segment(B,C),3)
    ;
endfig;
```



9.2 Labels

Comme le code produit avec `mp-geom2d` est un code METAPOST[5], on peut utiliser les outils classiques de labélisation. Cependant, pour permettre plus de flexibilité (et notamment une compatibilité avec LuaLaTeX et luamplib [1] ou minim-mp [4]), si le code est compilé avec METAPOST, alors le package latexmp [2] est chargé et fournit la commande `textext()`.

De plus, `mp-geom2d` fournit quelques commandes pour se faciliter la vie.

On peut marquer les points avec la commande suivante. Cette commande *pointe* le point (avec la commande `pointe`, voir section 4.3) et inscrit un label.

`marque.`⟨*place*⟩ `"`⟨*nom*⟩`"`

⟨***place***⟩ : peut être les classiques placements de METAPOST : `top`, `bot`, `rt`, `lft`,+`urt`, `ulft`, `lrt`, `llft`.

⟨***nom***⟩ : entre double quote, doit être un nom de variable de point. Le nom sera composé en mode mathématique (entre $). Si le nom contient un _, tout ce qui sera après sera mis en indice (ex. `A_be` deviendra A_{be}). Si le nom est `alpha`, `beta`, `gamma` ou `delta`, alors le résultat donnera la lettre grecque composée en mode mathématique.

`mp-geom2d` fournit aussi une adaptation au classique `label` de METAPOST.

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

⟨***place***⟩ : peut être les classiques placements de METAPOST : `top`, `bot`, `rt`, `lft`,+`urt`, `ulft`, `lrt`, `llft`.

⟨***materiel***⟩ : classiquement ce qu'on donne à `label`, une chaîne de caractères, ou une `picture` (qui peut-être produite par exemple avec `btex ... etex` ou, puisque latexmp est chargé par `mp-geom2d`, `textext()`).

⟨***point***⟩ : doit être un `point` de `mp-geom2d`.

On peut aussi étiquetter un `chemin`, une `courbe` ou un `path` avec la macro suivante :

`EtiquetteChemin.`⟨*place*⟩`(`⟨*materiel*⟩`,`⟨*chemin*⟩`,`⟨*position*⟩`)`

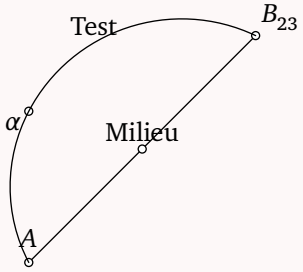
⟨***place***⟩ : peut être les classiques placements de METAPOST : `top`, `bot`, `rt`, `lft`,+`urt`, `ulft`, `lrt`, `llft`.

⟨***materiel***⟩ : classiquement ce qu'on donne à `label`, une chaîne de caractères, ou une `picture` (qui peut-être produite par exemple avec `btex ... etex` ou, puisque latexmp est chargé par `mp-geom2d`, `textext()`).

⟨***chemin***⟩ : doit être un `chemin`, une `courbe`, ou un `path`.

Exemple 30

```
input geom2d;
beginfig(1);
A = Point(0,0);
B_23 = Point(3,3);
alpha = Point(0,2);
trace Segment(A,B_23);
C = Milieu(A,B_23);
pointe C;
gddLabel.top(textext("Milieu"),C)
    ;
marque.top "A";
marque.urt "B_23";
marque.llft "alpha";
P = Chemin(Pt(A)..Pt(alpha)..Pt(
    B_23));
trace P;
EtiquetteChemin.top("Test",P,0.6)
    ;
endfig;
```



10 Repère

`mp-geom2d` fournit un ensemble de commandes permettant de faciliter le dessin de repère.

La commande principale est la définition du repère, c'est-à-dire la boîte dans laquelle le dessin sera représenté.

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

⟨***l***⟩ : `numeric`, largeur du repère (en unité `gddU`) ;

⟨***h***⟩ : `numeric`, hauteur du repère (en unité `gddU`) ;

⟨***ox***⟩ : `numeric`, distance (en unité `gddU`) de l'origine (`point` (0,0)) par rapport au bord gauche ;

⟨***oy***⟩ : `numeric`, distance (en unité `gddU`) de l'origine (`point` (0,0)) par rapport au bord bas ;

⟨***ux***⟩ : `numeric`, unité de l'axe (Ox) (en unité `gddU`) ;

⟨***uy***⟩ : `numeric`, unité de l'axe (Oy) (en unité `gddU`).

Cette commande ne retourne, ni ne trace rien. Elle sert à spécifier quelques variables internes de définition du dessin. Elle modifie d'ailleurs le comportement de la macro `gddEnPlace` (voir page 5). Noter que cette commande impose le fait que l'origine (c'est-à-dire le `point` (0,0)) soit à l'intérieur du repère.

Cette macro s'accompagne de deux autres, elles aussi *silencieuses*, ne servant qu'à :
— sauvegarder la `picture` courante ;
— construire une `picture` avec le contenu se trouvant entre `Debut` et `Fin` ;
— rogner (avec `clip`) la `picture` aux cadre du repère construit ;
— ajouter la `picture` courant à celle sauvegarder ;
— enfin rétablir le fonctionnement de `gddEnPlace` comme avant l'utilisation de `Repere` .

Ces deux commandes sont appelées `Debut` et `Fin`.

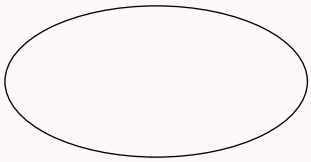
`Debut`

`Fin`

Ainsi, il est très simple d'illustrer le fait qu'à un changement de base orthogonale prêt, un cercle est une ellipse.

Exemple 31

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
Repere(5,5,1,1,1,0.5);
Debut;
trace CA;
Fin;
endfig;
```



On dispose de deux commandes pour tracer les axes du repère. La première trace les axes passant par l'origine.

`Axes` *(trace un ensemble d'éléments sur le repère)*

Cette commande inscrit aussi les labels des axes des abscisses et des ordonnées qui sont stockées dans les variables globales dédiées suivantes :

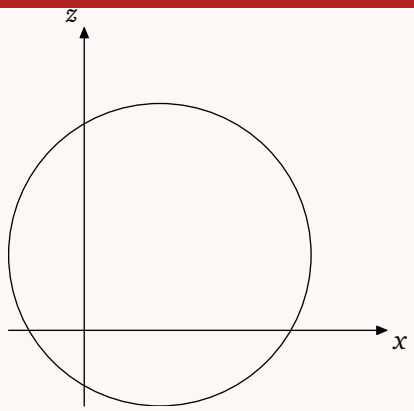
`gddXlabel` (*string, valeur par défaut* `"$x$"`)

`gddYlabel` (*string, valeur par défaut* `"$y$"`)

Attention, ces commandes doivent s'utiliser avant l'appel à `Debut`.
L'exemple suivant permet d'illustrer le tracé des axes.

Exemple 32

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
Repere(5,5,1,1,1,1);
gddYlabel:="$z$";
Axes;
Debut;
trace CA;
Fin;
endfig;
```



On pourra aussi définir un repère en utilisant une syntaxe permettant de spécifier les abscisses et les ordonnées extrémales.

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

⟨***xmin***⟩ : `numeric`, abscisse minimum (en unité `gddU`).

⟨***xmax***⟩ : `numeric`, abscisse maximum (en unité `gddU`).

⟨***ymin***⟩ : `numeric`, ordonnée minimum (en unité `gddU`).

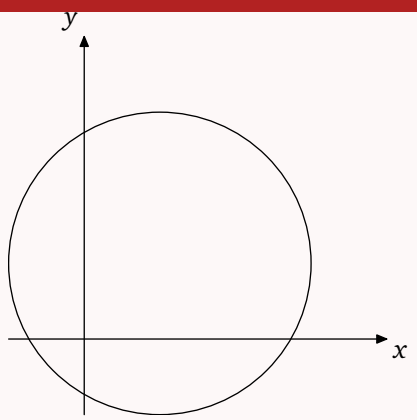
⟨***ymax***⟩ : `numeric`, ordonnée maximum (en unité `gddU`).

⟨***ux***⟩ : `numeric`, unité de l'axe (Ox) (en unité `gddU`).

⟨***uy***⟩ : `numeric`, unité de l'axe (Oy) (en unité `gddU`).

Exemple 33

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
Axes;
Debut;
trace CA;
Fin;
endfig;
```

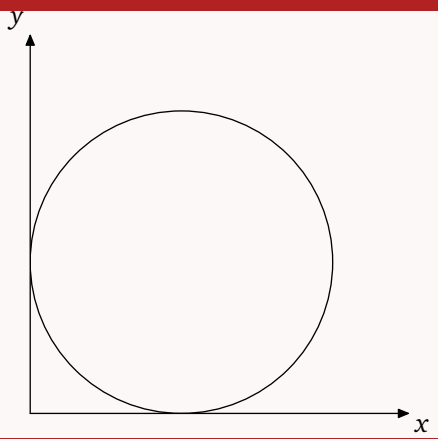


`mp-geom2d` fournit aussi de quoi tracer les axes sur le bord du cadre avec la commande suivante.

`AxesBords` *(trace un ensemble d'éléments sur le repère)*

Exemple 34

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
AxesBords;
Debut;
trace CA;
Fin;
endfig;
```



Les commandes suivantes permettent de graduer les axes (classiques ou sur le bord).

Là encore, ce sont des commandes qui n'ont pas d'arguments ni ne retournent rien et qui tracent.

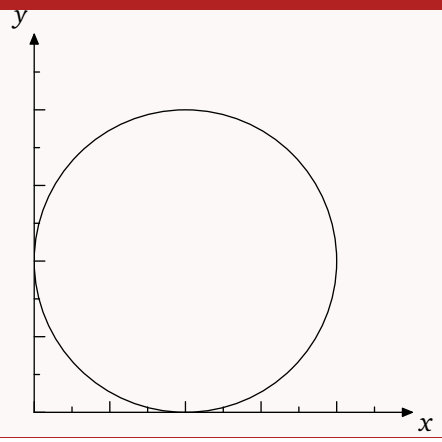
`Graduations` *(trace un ensemble d'éléments sur le repère)*

`GraduationsBords` *(trace un ensemble d'éléments sur le repère)*

On utilisera la version en cohérence avec les axes choisis.

Exemple 35

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
AxesBords;
GraduationsBords;
Debut;
trace CA;
Fin;
endfig;
```



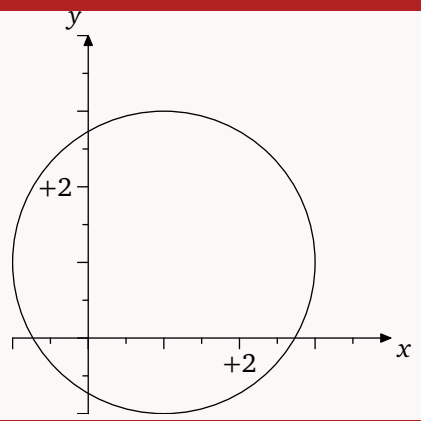
Si on désire marquer les unités, `mp-geom2d` propose la macro suivante (qui n'est utilisable que si l'on ne place pas les axes sur le bord).

`Unites(`⟨*unité*⟩`)` *(trace un ensemble d'éléments sur le repère)*

⟨***unité***⟩ : *numeric, valeur à inscrire sur les deux axes.*

Exemple 36

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
Axes;
Debut;
  Graduations;
  Unites(2);
  trace CA;
Fin;
endfig;
```



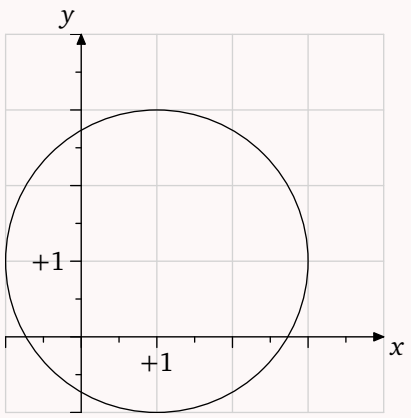
On peut aussi ajouter une grille sur notre repère avec la macro suivante.

`GrilleRepere` *(trace un ensemble d'éléments sur le repère)*

Dans l'exemple suivant, on règle la couleur de la grille avec la commande MetaPost `drawoptions`.

Exemple 37

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
drawoptions(withcolor LightGrey);
GrilleRepere;
drawoptions();
Axes;
Debut;
  Graduations;
  Unites(1);
  trace CA;
Fin;
endfig;
```



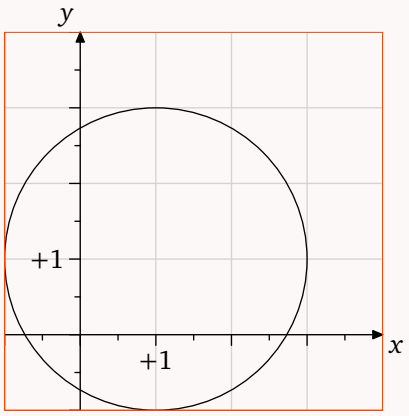
On pourra aussi ajouter un cadre au repère avec la macro suivante.

`CadreRepere` → `path`

Dans l'exemple suivant, on règle la couleur de la grille avec la commande MetaPost `drawoptions`.

Exemple 38

```
input geom2d;
beginfig(1);
A = Point(1,1);
CA = Cercle(A,2);
RepereMinMax(-1,4,-1,4,1,1);
drawoptions(withcolor LightGrey);
GrilleRepere;
drawoptions();
Axes;
Debut;
  Graduations;
  Unites(1);
  trace CA;
  trace CadreRepere withcolor
      OrangeRed;
Fin;
endfig;
```



11 Quelques constantes et fonctions mathématiques

mp-geom2d définit deux constantes mathématiques Pi et _E pour les constantes $\pi \simeq 3.14159265$ et $e = 2.71828183$.

De plus, le package définit quelques fonctions mathématiques de la variable réelle :

sin(⟨x⟩)

cos(⟨x⟩)

tan(⟨x⟩)

exp(⟨x⟩)

ln(⟨x⟩)

ch(⟨x⟩)

sh(⟨x⟩)

th(⟨x⟩)

arcsin(⟨x⟩)

arccos(⟨x⟩)

arctan(⟨x⟩)

12 Représentation de courbes et de fonctions

`mp-geom2d` fournit aussi quelques macros facilitant la représentation simple de courbes et de fonctions mathématiques.

12.1 Fonction de la variable réelle

Pour représenter une fonction de la variable réelle, on utilisera la macro suivante :

`Representation(`⟨*fonction*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***fonction***⟩ : est une macro METAPOST qui définit la fonction mathématique d'une variable réelle que l'on souhaite représenter ;

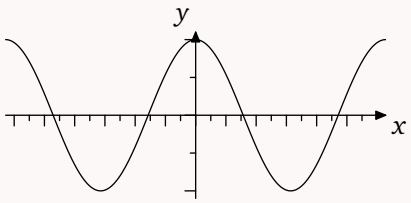
⟨***ti***⟩ : est la valeur (`numeric`) initiale de la variable à partir de laquelle on souhaite construire la représentation de la fonction ;

⟨***tf***⟩ : est la valeur (`numeric`) finale de la variable jusqu'à laquelle on souhaite construire la représentation de la fonction ;

⟨***n***⟩ : est le nombre de pas (`numeric`) utilisé pour la discrétisation de la représentation.

Exemple 39

```
input geom2d;
beginfig(1);
RepereMinMax(-2Pi,2Pi
    ,-1.1,1.1,0.4,1);
Axes;
Debut;
  Graduations;
  trace Representation(cos,-2Pi,2
      Pi,100);
Fin;
endfig;
```



12.2 Courbe paramétrique

Pour représenter une courbe plane définie par deux fonctions décrivant l'abscisse et l'ordonnée en fonction d'un paramètre, on utilisera la macro suivante :

`Courbe(`⟨*fct_abscisse*⟩`)(`⟨*fct_ordonnee*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***fct_abscisse***⟩ : est une macro METAPOST qui définit la fonction mathématique d'une variable réelle décrivant l'évolution de l'abscisse des points de la courbe que l'on souhaite représenter ;

⟨***fct_ordonnee***⟩ : est une macro METAPOST qui définit la fonction mathématique d'une variable réelle décrivant l'évolution de l'ordonnée des points de la courbe que l'on souhaite représenter ;

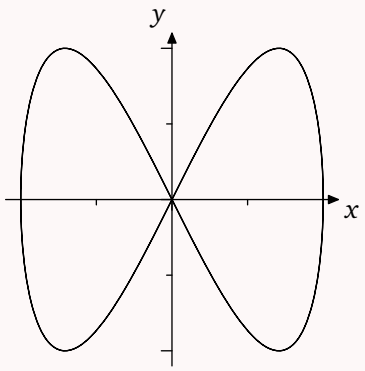
⟨***ti***⟩ : est la valeur (numeric) initiale de la variable à partir de laquelle on souhaite construire la représentation de la fonction ;

⟨***tf***⟩ : est la valeur (numeric) finale de la variable jusqu'à laquelle on souhaite construire la représentation de la fonction ;

⟨***n***⟩ : est le nombre de pas (numeric) utilisé pour la discrétisation de la représentation.

Exemple 40

```
input geom2d;
vardef f_a(expr t)= cos(t) enddef
    ;
vardef f_o(expr t)= sin(2*t)
    enddef;

beginfig(1);
RepereMinMax
    (-1.1,1.1,-1.1,1.1,2,2);
Axes;
Debut;
 Graduations;
 trace Courbe(f_a,f_o,-2Pi,2Pi
     ,100);
Fin;
endfig;
```



Pour représenter une courbe plane définie par une fonction décrivant les coordonées polaires en fonction d'un paramètre, on utilisera la macro suivante :

`CourbeEnPolaires(`⟨*fonction*⟩`)(`⟨*ti*⟩`,`⟨*tf*⟩`,`⟨*n*⟩`)` → `path`

⟨***fonction***⟩ : est une macro METAPOST qui définit la fonction mathématique d'une variable réelle décrivant l'évolution des coordonnées polaires des points de la courbe que l'on souhaite représenter ;

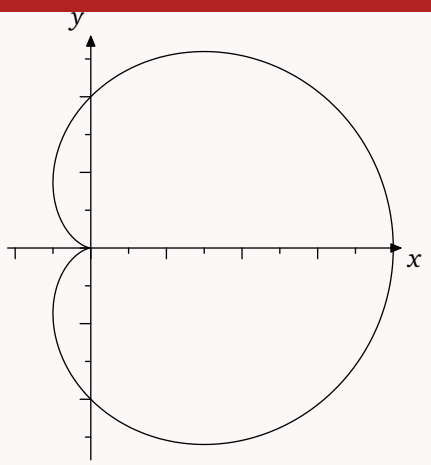
⟨***ti***⟩ : est la valeur (numeric) initiale de la variable à partir de laquelle on souhaite construire la représentation de la fonction ;

⟨***tf***⟩ : est la valeur (numeric) finale de la variable jusqu'à laquelle on souhaite construire la représentation de la fonction ;

⟨***n***⟩ : est le nombre de pas (numeric) utilisé pour la discrétisation de la représentation.

Exemple 41

```
input geom2d;
a := 2;
vardef cardioide(expr t)= a*(1+
    cos(t)) enddef;
beginfig(1);
RepereMinMax
    (-1.1,4.1,-2.8,2.8,1,1);
Axes;
Debut;
  Graduations;
  trace CourbeEnPolaires(
      cardioide,0,2Pi,100);
Fin;
endfig;
```



12.3 Champs de vecteurs

`mp-geom2d` fournit des macros pour la représentation des champs de vecteurs.

Tout d'abord, on peut tracer des champs de vecteurs associés à une équation différentielle du premier ordre pour une fonction y de la variable x :

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

La macro associée est la suivante :

`ChampVecteurs(`⟨*fonction*⟩`)(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*px*⟩`,`⟨*py*⟩`,`⟨*dx*⟩`,`⟨*couleur*⟩`)` → `path`

⟨***fonction***⟩ : est une macro METAPOST qui définit une fonction mathématique de $\mathbf{R}^2$ dans $\mathbf{R}$;

⟨***x***⟩ : est une valeur (`numeric`), en unité `gddU`, qui permet de décaler la grille des vecteurs suivant la direction x, les points étant tous les ⟨*x*⟩+*i*⟨*px*⟩ pour i entier ;

⟨***y***⟩ : est une valeur (`numeric`), en unité `gddU`, qui permet de décaler la grille des vecteurs suivant la direction y, les points étant tous les ⟨*y*⟩+*i*⟨*py*⟩ pour i entier ;

⟨***px***⟩ : est la valeur (`numeric`), en unité `gddU`, du pas de la grille suivant l'axe x pour la représentation des vecteurs ;

⟨***py***⟩ : est la valeur (`numeric`), en unité `gddU`, du pas de la grille suivant l'axe y pour la représentation des vecteurs ;

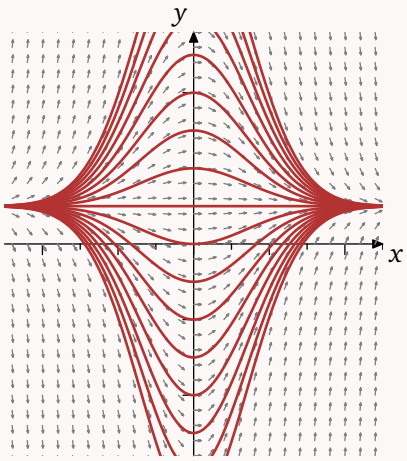
⟨***dx***⟩ : est la norme (`numeric`), en unité `gddU`, des vecteurs du champ de vecteur ;

⟨***couleur***⟩ : est la couleur (`color`) utilisée pour tracer les vecteurs.

Comme cette macro utilise la macro `drawarrow` de METAPOST, il faudra jouer avec le paramètre `ahlength` pour régler la taille du triangle des flèches (voir [5]).

Exemple 42

```
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);
RepereMinMax
    (-2.5,2.5,-2.8,2.8,1,1);
Axes;
Debut;
Graduations;
ahlength := 1;
ChampVecteurs(F
    ,0,0,0.2,0.2,0.1,0.5white);
% Courbes intégrales
for n = 0 upto 16:
   a := (n/2) - 4;
   trace Representation(f
       ,-2.5,2.5,50)
   avecCrayon(1,(0.7,0.2,0.2));
endfor
Fin;
endfig;
```



Sur le même modèle, on peut tracer des champs de vecteurs d'une fonction de $\mathbf{R}^2$ dans $\mathbf{R}^2$. La macro associée est la suivante :

`ChampVecteursDD(`⟨*fonction*⟩`)(`⟨*x*⟩`,`⟨*y*⟩`,`⟨*px*⟩`,`⟨*py*⟩`,`⟨*dx*⟩`,`⟨*couleur*⟩`)` → `path`

⟨***fonction***⟩ : est une macro METAPOST qui définit une fonction mathématique de $\mathbf{R}^2$ dans $\mathbf{R}^2$ (donc qui retourne une `pair`) ;

⟨***x***⟩ : est une valeur (`numeric`), en unité `gddU`, qui permet de décaler la grille des vecteurs suivant la direction x, les points étant tous les ⟨*x*⟩+*i*⟨*px*⟩ pour i entier ;

⟨***y***⟩ : est une valeur (`numeric`), en unité `gddU`, qui permet de décaler la grille des vecteurs suivant la direction y, les points étant tous les ⟨*y*⟩+*i*⟨*py*⟩ pour i entier ;

⟨***px***⟩ : est la valeur (`numeric`), en unité `gddU`, du pas de la grille suivant l'axe x pour la représentation des vecteurs ;

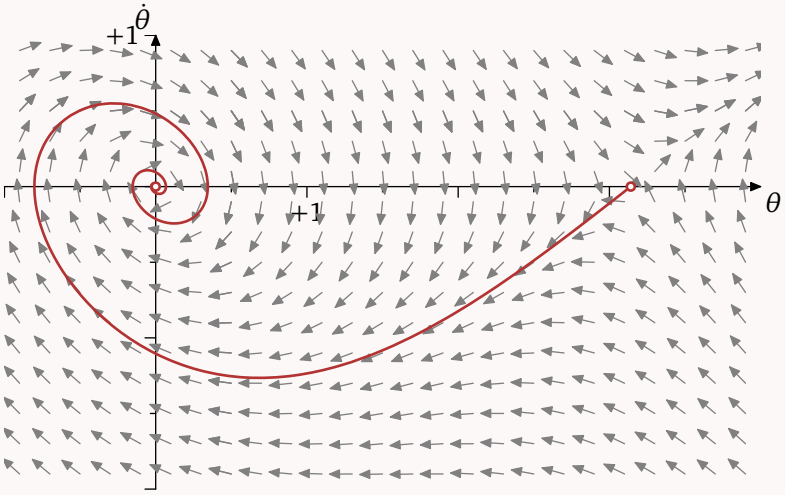
⟨***py***⟩ : est la valeur (`numeric`), en unité `gddU`, du pas de la grille suivant l'axe y pour la représentation des vecteurs ;

⟨***dx***⟩ : est la norme (`numeric`), en unité `gddU`, des vecteurs du champ de vecteur ;

⟨***couleur***⟩ : est la couleur (`color`) utilisée pour tracer les vecteurs.

Exemple 43

```
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);
Repere(10,6,2,4,2,2);
Axes;
Unites(1);
Debut;
Graduations;
trajectoire := CourbeDat("solution0",0);
ChampVecteursDD(F,0.5,0.5,0.2,0.2,0.15,0.5white);
trace trajectoire avecCrayon(1,(0.7,0.2,0.2));
pointe Point(0,0) avecCrayon(1,(0.7,0.2,0.2));
pointe Point(3.1415,0) avecCrayon(1,(0.7,0.2,0.2));
Fin;
endfig;
```



13 Couleurs `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

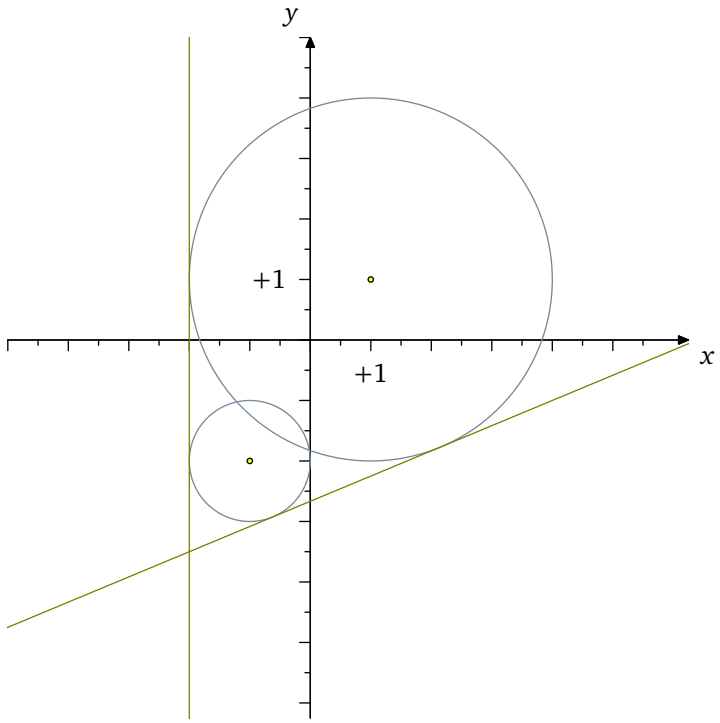
AntiqueWhite
Beige
Blue
CadetBlue
CornflowerBlue
DarkBlue
DarkGreen
DarkOliveGreen
DarkSalmon
DarkSlateGrey
DeepSkyBlue
FireBrick
Gainsboro
Gray
Honeydew
Ivory
LawnGreen
LightCyan
LightGreen
LightSeaGreen
LightSlateGrey
LimeGreen
MediumAquamarine
MediumSeaGreen
MediumVioletRed
Moccasin
OldLace
OrangeRed
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

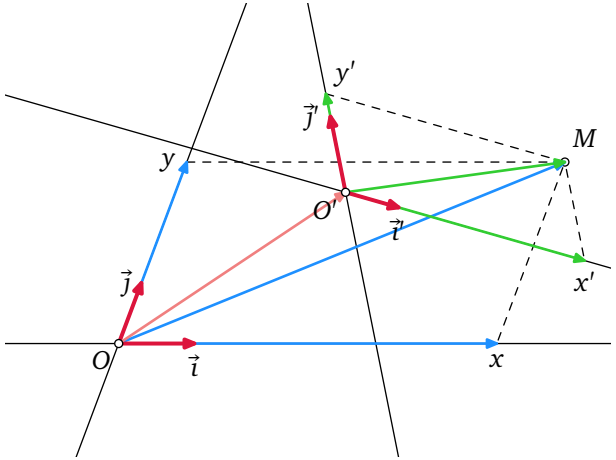
14 Galerie

14.1 Repère et tangentes extérieures




```
input geom2d;

labeloffset := 6;
gddTaillePoint := 2;
gddCouleurPoint := Yellow;

beginfig(1);
 Repere(9,9,4,5,0.8,0.8);
 Axes;
 Debut;
  Axes;
  Graduations; Unites(1);

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

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

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

  drawoptions(withcolor Olive);
  trace T1;
  trace T2;

  drawoptions();
  pointe Point(1,1);
  pointe Point(-1,-2);

 Fin;
endfig;
end
```

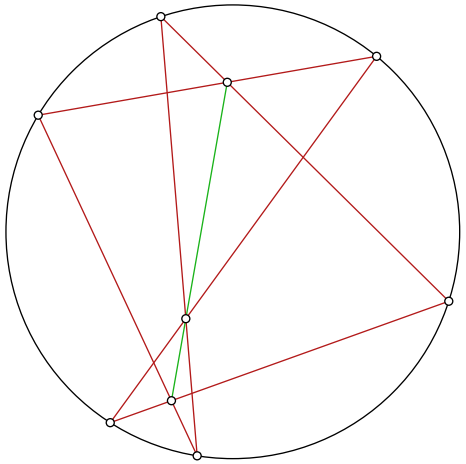
14.2 Vecteur dans un repère




```
input geom2d;

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

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

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

  Repere(8,6,1.5,1.5,1,1);
  Debut;
    trace Droite(O,I);
    trace Droite(O,J);
    trace Droite(O',I');
    trace Droite(O',J');

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

    marque.urt "M";

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

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

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

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

    drawoptions();
    marque.llft "O";
    marque.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));
```

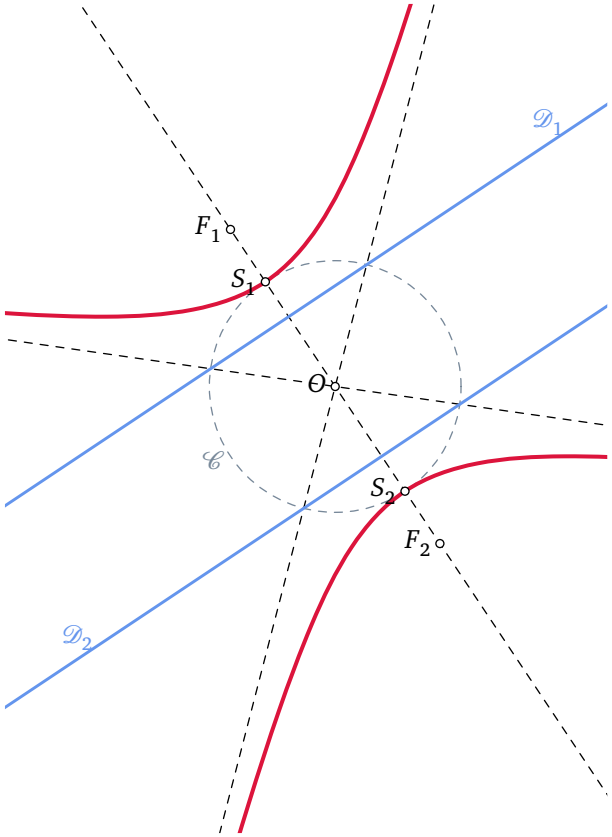
```
61
62    label.bot(textext("\(x'\)"), PtR(H'));
63    label.urt(textext("\(y'\)"), PtR(K'));
64    label.bot(textext("\(\vec\imath'\)"), PtR(I'));
65    label.lft(textext("\(\vec\jmath'\)"), PtR(J'));
66
67    Fin;
68  endfig;
69  end
```

14.3 Théorème de Pascal



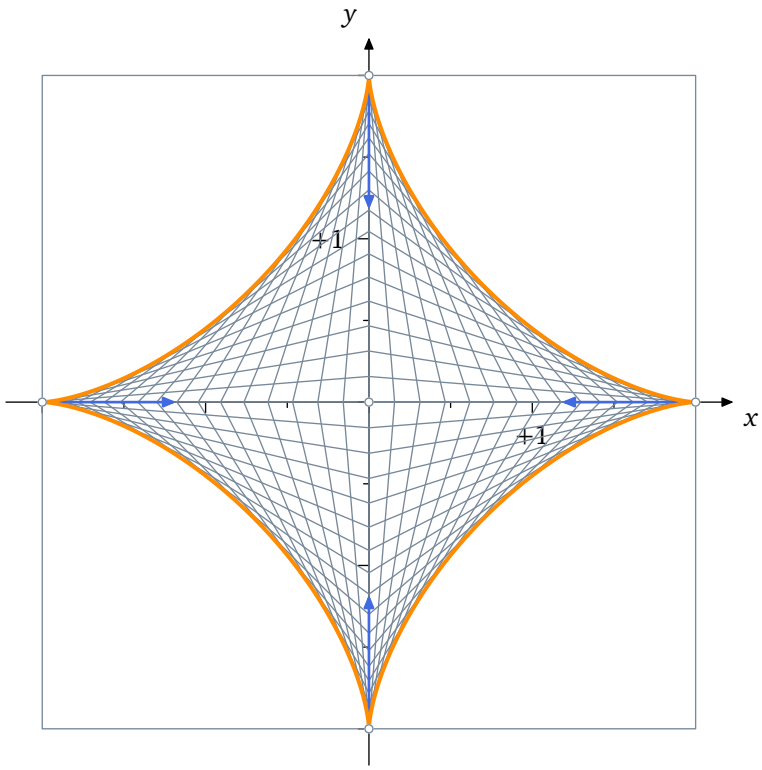
```
input geom2d;

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

14.4 Hyperbole



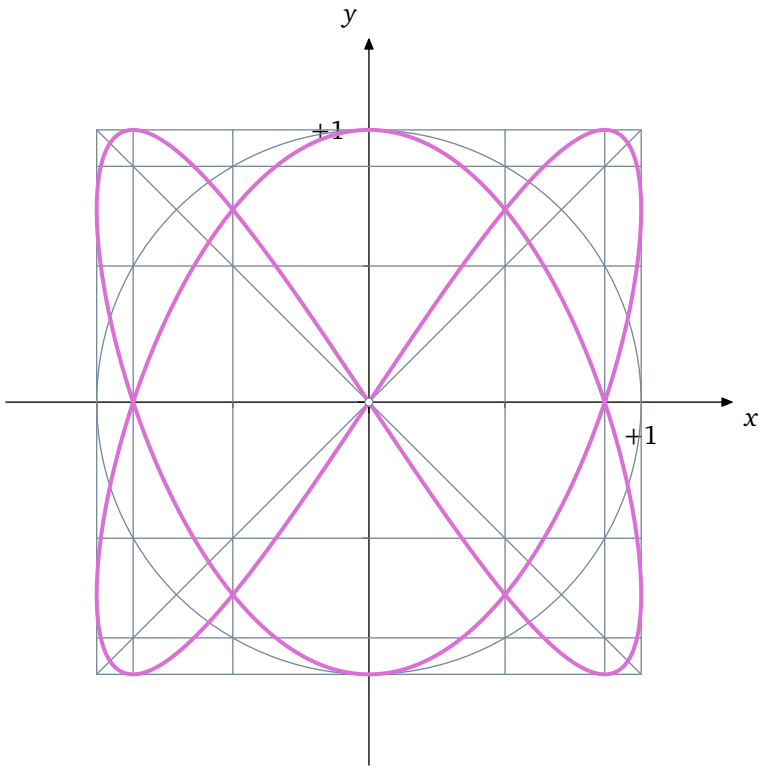

```
input geom2d;

beginfig(1);
  A = Point(0,0);
  B = Point(3,2);
  AB = Droite(A,B);
  F_1 = Point(-1,1);
  Hyper = HyperboleFD(F_1,AB,1.5);
  O  = Centre(Hyper);
  F_2 = Foyer(Hyper,2);
  Axe = Droite(F_1,F_2);
  S_1 = Sommet(Hyper,1);
  S_2 = Sommet(Hyper,2);

  trace Axe dashed evenly;
  C = CerclePrincipale(Hyper) ;
  trace C avecCrayon(0.5,LightSlateGray) dashed evenly;
  A_1 = AsymptoteHyperbole(Hyper,1);
  A_2 = AsymptoteHyperbole(Hyper,2);
  trace A_1 dashed evenly;
  trace A_2 dashed evenly;

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

  trace D_1 avecCrayon(1.1,CornflowerBlue);
  trace D_2 avecCrayon(1.1,CornflowerBlue);
  trace DemiHyperbole(Hyper,1) avecCrayon(1.5,Crimson);
  trace DemiHyperbole(Hyper,2) avecCrayon(1.5,Crimson);

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


end.
```

14.5 Astroïde comme enveloppe de droites



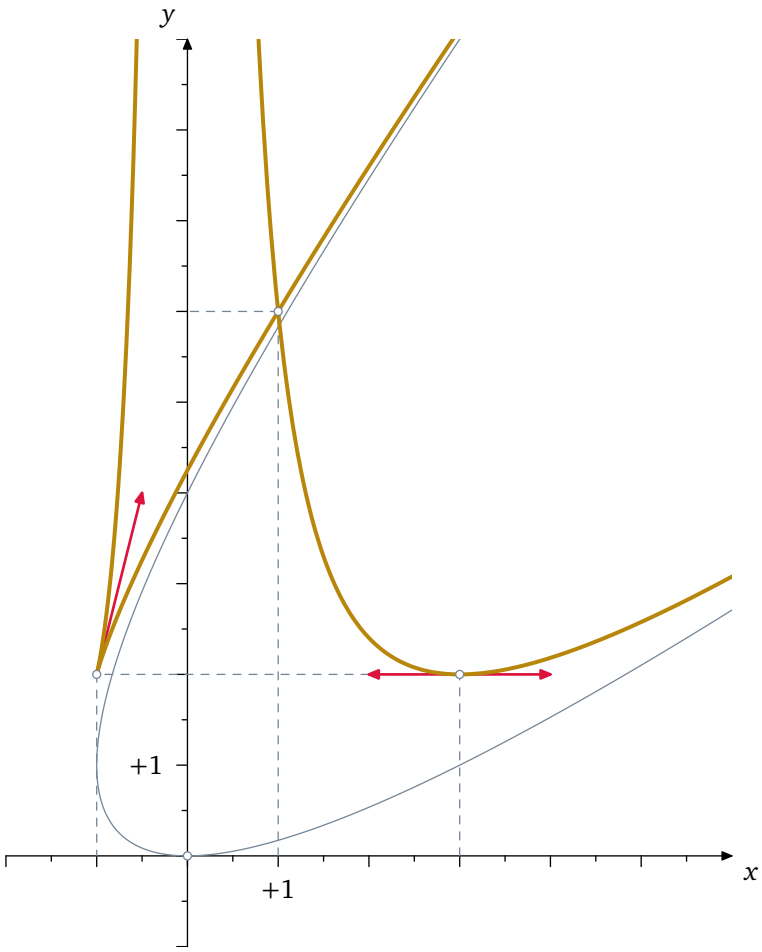

```
input geom2d;

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);
 Repere(8,8,4,4,1.8,1.8);
 Axes;
 Debut;
  Graduations; Unites(1);
  drawoptions(withpen pencircle scaled 0.5 withcolor LightSlateGrey);
  trace ((-2,-2)--(-2,2)--(2,2)--(2,-2)--cycle);

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

  fleche Segment((2,0),(1.2,0)) avecCrayon(1,RoyalBlue);
  fleche Segment((-2,0),(-1.2,0)) avecCrayon(1,RoyalBlue);
  fleche Segment((0,2),(0,1.2)) avecCrayon(1,RoyalBlue);
  fleche Segment((0,-2),(0,-1.2)) avecCrayon(1,RoyalBlue);

  trace Courbe(f,g,0,2*Pi,500) avecCrayon(1.5,DarkOrange);

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

 Fin;
endfig;
end
```

14.6 Courbe de Lissajous



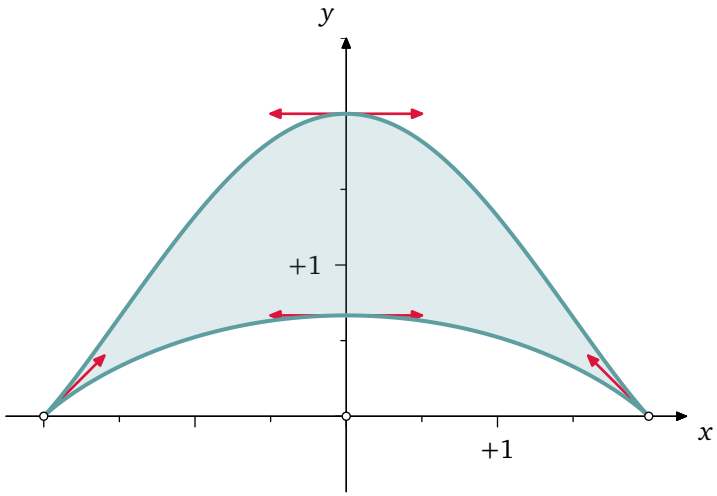

```
input geom2d;

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);
 Repere(8,8,4,4,3,3);
 Axes;
 Debut;
  Graduations; Unites(1);

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

  trace Courbe(f,g,0,2*Pi,500) avecCrayon(1.5,Orchid);

  pointe Point(0,0);

 Fin;
endfig;
end
```

14.7 Étude de fonctions



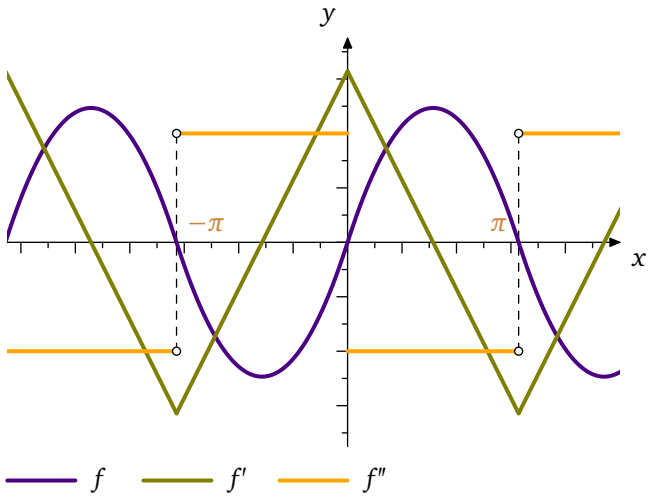

```
input geom2d;

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);

 Repere(8,10,2,1,1,1);
 Axes;
 Debut;
  Graduations; Unites(1);

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

  fleche Segment((-1,2),(-1,2)+(0.5,2)) avecCrayon(1,Crimson);
  fleche Segment((3,2),(4,2)) avecCrayon(1,Crimson);
  fleche Segment((3,2),(2,2)) avecCrayon(1,Crimson);

  trace Courbe(f,g,-10,-0.05,300) avecCrayon(1.5,DarkGoldenrod);
  trace Courbe(f,g,0.05,10,300) avecCrayon(1.5,DarkGoldenrod);

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

 Fin;
endfig;
end
```

14.8 Le bicorne



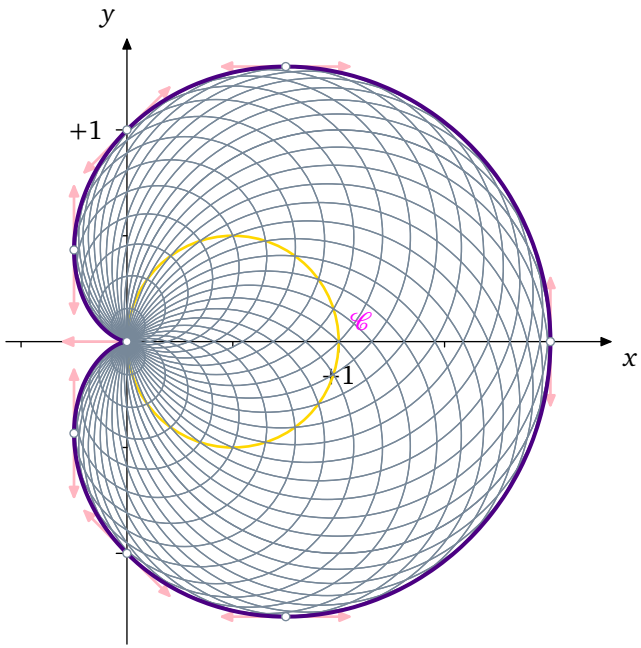

```
input geom2d;

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

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

beginfig(1);

 Repere(9,6,4.5,1,2,2);
 Axes;
 Debut;
  Axes;
  Graduations; Unites(1);

  colorieAvecTransparence(bicorne,CadetBlue,0.2);

  traceDoubleVecteur((0,2),(0.5,0)) avecCrayon(1,Crimson);
  traceDoubleVecteur((0,2/3),(0.5,0)) avecCrayon(1,Crimson);
  fleche Segment((2,0),(1.6,0.4)) avecCrayon(1,Crimson);
  fleche Segment((-2,0),(-1.6,0.4)) avecCrayon(1,Crimson);

  trace bicorne avecCrayon(1.5,CadetBlue);

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

 Fin;
endfig;
end
```

14.9 Une fonction et ses dérivées



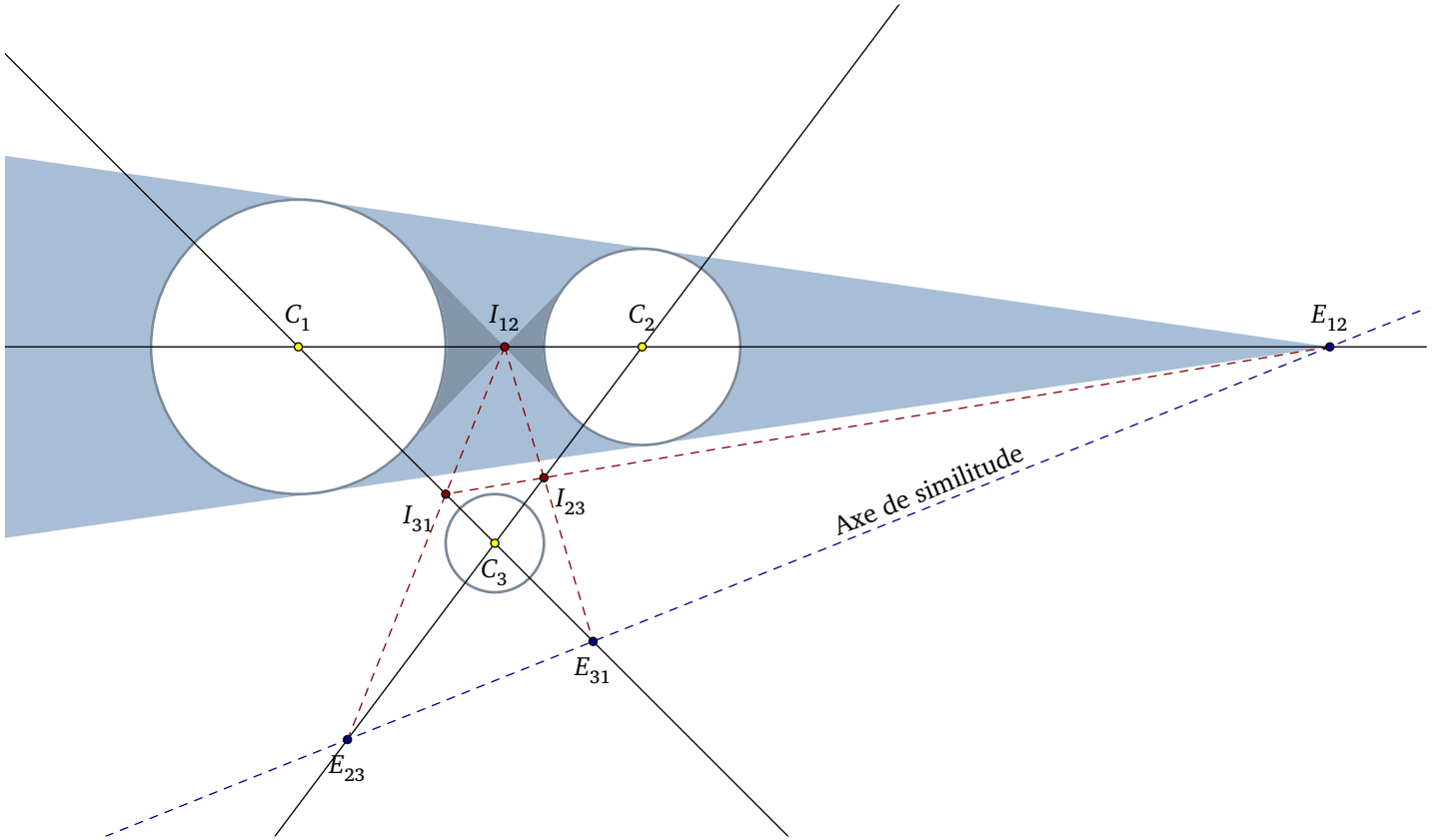

```
input geom2d;

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);

Repere(9,6,5,3,.8,0.8);
Axes;
 Debut;
  Graduations;

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

  trace Representation(f,0,Pi,100) avecCrayon(1.5,Indigo);
  trace (Representation(f,0,Pi,100) scaled -1) avecCrayon(1.5,Indigo)
      ;
  trace (Representation(f,0,Pi,100) shifted (-2*Pi,0)) avecCrayon
      (1.5,Indigo);
  trace (Representation(f,0,Pi,100) scaled -1 shifted (2*Pi,0))
      avecCrayon(1.5,Indigo);

  trace Representation(g,0,Pi,100) avecCrayon(1.5,Olive);
  trace (Representation(g,0,Pi,100) xscaled -1) avecCrayon(1.5,Olive)
      ;
  trace (Representation(g,0,Pi,100) shifted (-2*Pi,0)) avecCrayon
      (1.5,Olive);
  trace (Representation(g,0,Pi,100) xscaled -1 shifted (2*Pi,0))
      avecCrayon(1.5,Olive);

  trace Representation(h,0,Pi,100) avecCrayon(1.5,Orange);
  trace (Representation(h,0,Pi,100) scaled -1) avecCrayon(1.5,Orange)
      ;
  trace (Representation(h,0,Pi,100) shifted (-2*Pi,0)) avecCrayon
      (1.5,Orange);
  trace (Representation(h,0,Pi,100) scaled -1 shifted (2*Pi,0))
      avecCrayon(1.5,Orange);

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

  label.urt(textext("\(-\pi\)"), (-Pi,0) gddEnPlace) withcolor Peru;
  label.ulft(textext("\(\pi\)"), (Pi,0) gddEnPlace) withcolor Peru;
Fin;

trace (0,-0.5)--(1,-0.5) avecCrayon(1.5,Indigo);
label.rt(textext("\(f\)"), (1,-0.5) gddEnPlace);
trace (2,-0.5)--(3,-0.5) avecCrayon(1.5,Olive);
label.rt(textext("\(f'\)"), (3,-0.5) gddEnPlace);
trace (4,-0.5)--(5,-0.5) avecCrayon(1.5,Orange);
label.rt(textext("\(f''\)"), (5,-0.5) gddEnPlace);

endfig;

end
```

14.10 Cardioïde




```
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 traceDoubleVecteur(expr o,d)= drawdblarrow ((o-d)--(o+d))
     gddEnPlace enddef;

beginfig(1);
 Repere(10,10,2,5,3.5,3.5);
 Axes;
 Debut;
  Graduations; Unites(1);

  drawoptions(withcolor LightSlateGrey);
  draw fullcircle shifted (0.5,0) gddEnPlace avecCrayon(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)))
          gddEnPlace;
  endfor;


  traceDoubleVecteur(rp(0),(0,0.3)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(Pi/3),(0.3,0)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(Pi/2),(0.2,0.2)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(2Pi/3),(0,0.3)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(-Pi/3),(0.3,0)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(-Pi/2),(0.2,-0.2)) avecCrayon(1,LightPink);
  traceDoubleVecteur(rp(-2Pi/3),(0,0.3)) avecCrayon(1,LightPink);

  fleche Segment(origine,(-0.3,0)) avecCrayon(1,LightPink);
  trace CourbeEnPolaires(r,-Pi,Pi,100) avecCrayon(1.5,Indigo);

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

  label.urt(textext("\(\mathcal{C}\)"),PtR(Point(1,0))) withcolor
       Magenta;
 Fin;
endfig;
end
```

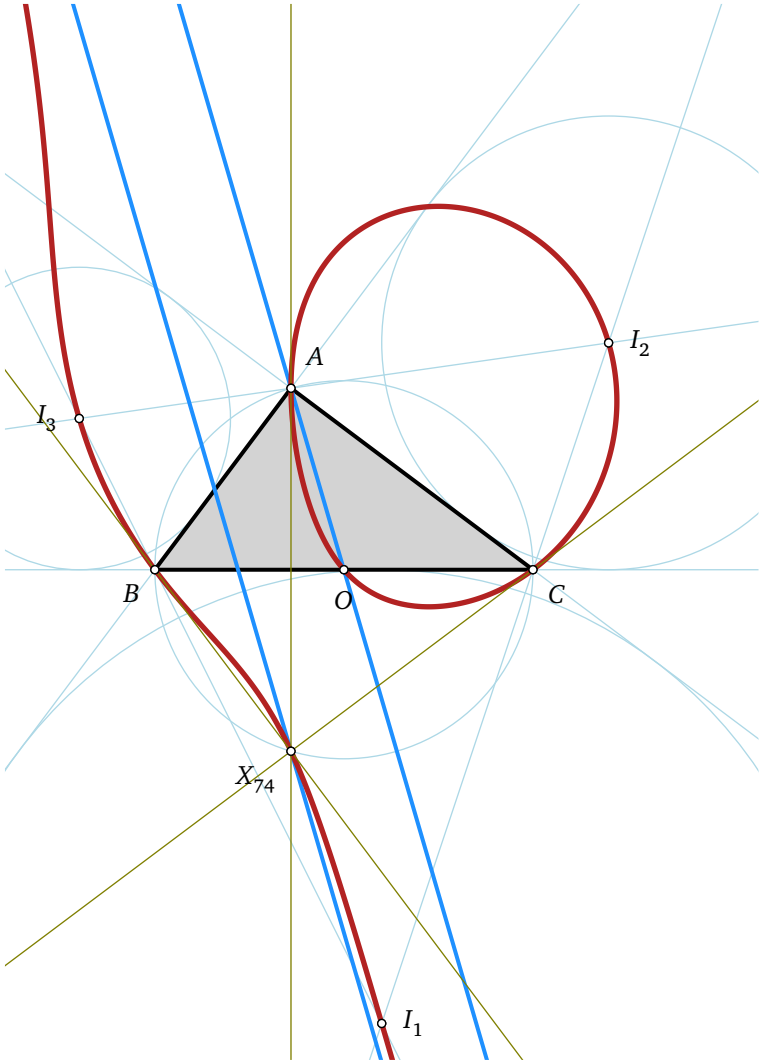
14.11 Axe de similitude



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

  T1 = TangenteCommuneExterieure(C1,C2);
  T2 = TangenteCommuneExterieure(C2,C1);
  T3 = TangenteCommuneInterieure(C1,C2);
  T4 = TangenteCommuneInterieure(C2,C1);

  T5 = TangenteCommuneExterieure(C2,C3);
  T6 = TangenteCommuneExterieure(C3,C2);
  T7 = TangenteCommuneInterieure(C2,C3);
  T8 = TangenteCommuneInterieure(C3,C2);

  T9 = TangenteCommuneExterieure(C1,C3);
  T10 = TangenteCommuneExterieure(C3,C1);
  T11 = TangenteCommuneInterieure(C1,C3);
  T12 = TangenteCommuneInterieure(C3,C1);

  E12 = IntersectionDroites(T1,T2);
  E23 = IntersectionDroites(T5,T6);
  E31 = IntersectionDroites(T9,T10);
  I12 = IntersectionDroites(T3,T4);
  I23 = IntersectionDroites(T7,T8);
  I31 = IntersectionDroites(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);
  colorie C1 withcolor white;
  colorie C2 withcolor white;
  colorie C3 withcolor white;
  trace C1; trace C2; trace C3;

  drawoptions();
  trace Droite(Centre(C1),Centre(C2));
  trace Droite(Centre(C3),Centre(C2));
  trace Droite(Centre(C1),Centre(C3));

  D_E = Droite(E12,E23);
  trace D_E dashed evenly withcolor DarkBlue;

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


  drawoptions();
  pointe Centre(C1);
  pointe Centre(C2);
  pointe Centre(C3);
  gddCouleurPoint := DarkBlue;
  pointe E12; pointe E31; pointe E23;
  gddCouleurPoint := DarkRed;
  pointe I12; pointe I31; pointe I23;

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

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


```

```
label.top(btex $C_{1}$ etex,PointTOPair(Centre(C1)) gddEnPlace);
label.top(btex $C_{2}$ etex,PointTOPair(Centre(C2)) gddEnPlace);
label.bot(btex $C_{3}$ etex,PointTOPair(Centre(C3)) gddEnPlace);

draw textext("Axe de similitude") rotated (22) shifted (0.5[Pt(E23)
      ,Pt(E12)]
  gddEnPlace +(0,3));
  Fenetre(-10,-10,19,7);
endfig;
end.
```

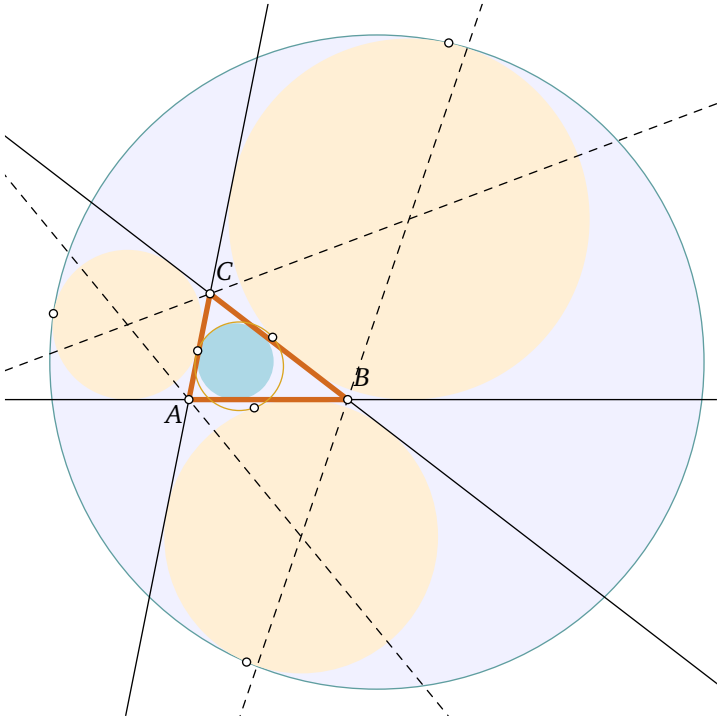
14.12 Tracé d'une cubique




```
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 := CourbeDat("K001-1",0);
  C2 := CourbeDat("K001-2",1);

  CE1 := CercleExinscrit(T,1);
  CE2 := CercleExinscrit(T,2);
  CE3 := CercleExinscrit(T,3);
  I_1 := Centre(CE1);
  I_2 := Centre(CE2);
  I_3 := Centre(CE3);
  CC := CercleCirconscrit(T);
  O  := Centre(CC);

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

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

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

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

end
```

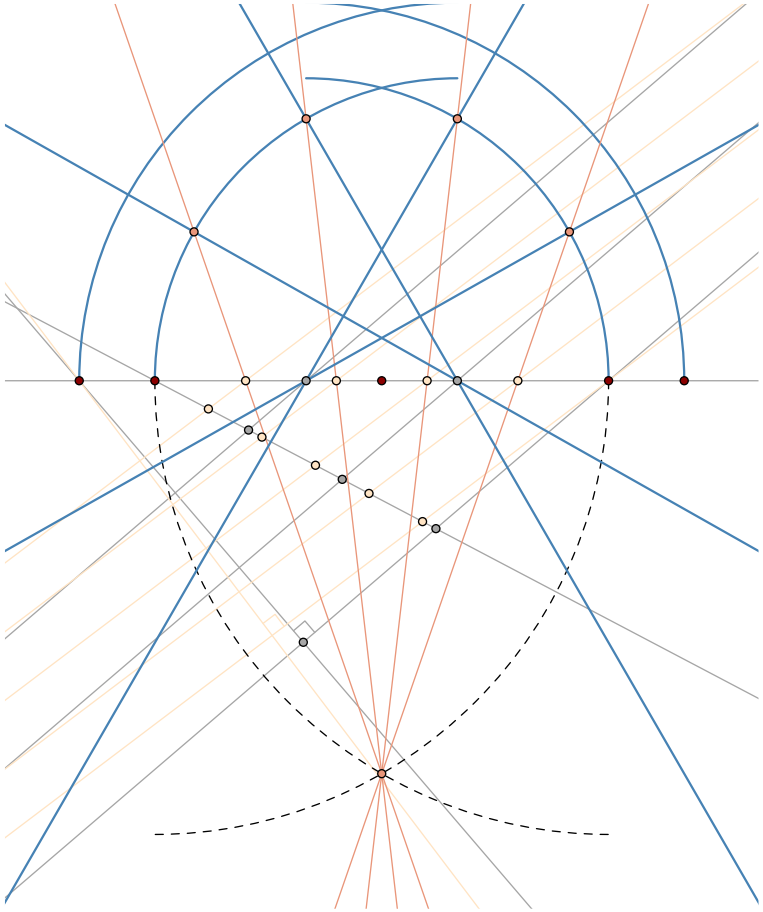
14.13 Apollonius




```
input geom2d;
gddU:=0.35cm;
beginfig(1);
% nos trois points
A = Point(0,0);
B = Point(6,0);
C = Point(0.8,4);
T_ABC = Triangle(A,B,C);

C_I = CercleInscrit(T_ABC);
C_A = CercleExinscrit(T_ABC,2);
C_B = CercleExinscrit(T_ABC,3);
C_C = CercleExinscrit(T_ABC,1);

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

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

A_S = AxeDeSimilitude(C_A,C_B,C_C);
P_CA = ProjectionPointSurDroite(I_C_A,A_S);
P_CB = ProjectionPointSurDroite(I_C_B,A_S);
P_CC = ProjectionPointSurDroite(I_C_C,A_S);

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

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

D2 = Droite(C_R,P_B);
P2 = IntersectionDroiteCercle(D2,C_B,1);
Q2 = IntersectionDroiteCercle(D2,C_B,2);

D3 = Droite(C_R,P_C);
P3 = IntersectionDroiteCercle(D3,C_C,2);
```

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

% le cercle 'dapollonius
Apol = CercleTroisPoints(P1,P2,P3);
colorie Apol withcolor 1.05*Lavender;
trace Apol withcolor CadetBlue;

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

trace d_AB;
trace d_BC;
trace d_CA;

trace d_CAA dashed evenly;
trace d_CAB dashed evenly;
trace d_CAC dashed evenly;

trace T_ABC withpen pencircle scaled 2 withcolor Chocolate;

C_E = CercleEuler(T_ABC);
trace C_E withcolor Goldenrod;

pointe P1;
pointe P2;
pointe P3;

pointe Q1;
pointe Q2;
pointe Q3;


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

14.14 Épure d'ogive

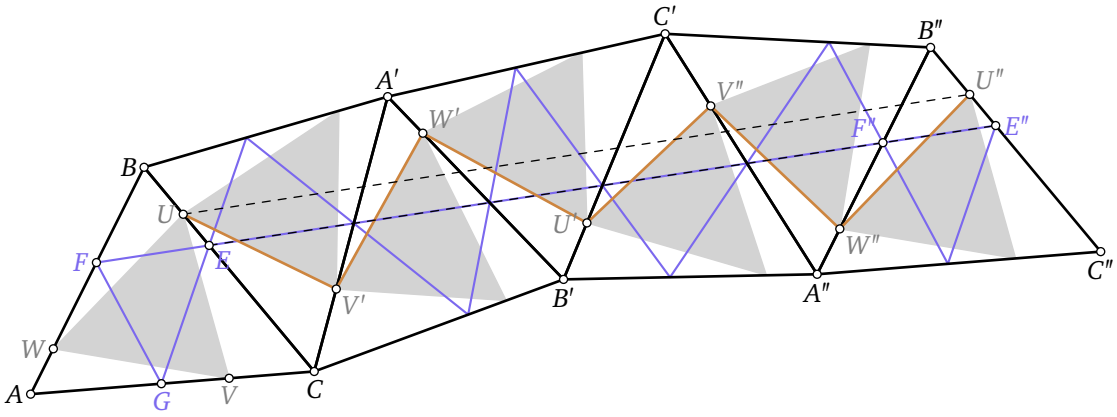


```
input geom2d;

beginfig(1);
   % les points de base pour l'ogive
   O = Point(0,0);
   A = Point(-3,0);
   A2 = Point(-4,0);
   B = Point(3,0);
   B2 = Point(4,0);
   % la droite de base de l'ogive
   AB = Droite(A,B);
   % on se prend une droite quelconque
   Ct = Point(0.8,-2); % point quelconque
   AC = Droite(A,Ct); % droite quelconque
   % on prend trois longueures égales sur la droite
   longueurTier := 1.4;
   C1 = ReportSurDroite(A,AC,longueurTier);
   C2 = ReportSurDroite(C1,AC,longueurTier);
   C3 = ReportSurDroite(C2,AC,longueurTier);
   % on définit la droite qui passe par B et le dernier point
   BC = Droite(B,C3);
   % on construit une droite perpendiculaire à BC
   D = Point(-4,0);
   perpD = DroitePerpendiculaire(BC,D);
   DD = IntersectionDroites(perpD,BC);
   % on projette C2 et C1 sar la droite AB pour diviser [AB] en 3
        parties
   % égales
   perpC2 = DroitePerpendiculaire(perpD,C2);
   perpC1 = DroitePerpendiculaire(perpD,C1);
   D1 = IntersectionDroites(AB,perpC1);
   D2 = IntersectionDroites(AB,perpC2);
   % on construit l'enveloppe de l'ogive
   C_DA = CercleCP(D2,A);
   C_DB = CercleCP(D1,B);
   C_DA2 = CercleCP(D2,A2);
   C_DB2 = CercleCP(D1,B2);
   % on construit le point P pour la projection finale
   C_AB = CercleCP(A,B);
   C_BA = CercleCP(B,A);
   P = IntersectionCercles(C_BA,C_AB);
```

```
    % on prend 5 longueures égales sur la droite quelconque du début
    longueurCinq :=0.8;
    F1 = ReportSurDroite(A,AC,longueurCinq);
    F2 = ReportSurDroite(F1,AC,longueurCinq);
    F3 = ReportSurDroite(F2,AC,longueurCinq);
    F4 = ReportSurDroite(F3,AC,longueurCinq);
    F5 = ReportSurDroite(F4,AC,longueurCinq);
    % on projette les points sur AB pour diviser [AB] en 5 parties
         égale
    BF5 = Droite(B,F5);
    perpF5 = DroitePerpendiculaire(BF5,A2);
    CC = IntersectionDroites(BF5,perpF5);
    perpF4 = DroitePerpendiculaire(perpF5,F4);
    perpF3 = DroitePerpendiculaire(perpF5,F3);
    perpF2 = DroitePerpendiculaire(perpF5,F2);
    perpF1 = DroitePerpendiculaire(perpF5,F1);
    G1 = IntersectionDroites(AB,perpF1);
    G2 = IntersectionDroites(AB,perpF2);
    G3 = IntersectionDroites(AB,perpF3);
    G4 = IntersectionDroites(AB,perpF4);
    % on projette les Gi sur l'ogive avec le point P
    PG1 = Droite(P,G1);
    PG2 = Droite(P,G2);
    PG3 = Droite(P,G3);
    PG4 = Droite(P,G4);
    I1 = IntersectionDroiteCercle(PG1,C_DA,1);
    I2 = IntersectionDroiteCercle(PG2,C_DA,2);
    I3 = IntersectionDroiteCercle(PG3,C_DB,2);
    I4 = IntersectionDroiteCercle(PG4,C_DB,2);
    % à partir des 2 points issus de la division en 3 de [AB]
    % on trace les séparations des pierres qui constituent l'ogive
    Dvoute1 = Droite(D2,I1);
    Dvoute2 = Droite(D2,I2);
    Dvoute3 = Droite(D1,I3);
    Dvoute4 = Droite(D1,I4);

    % les tracés
    drawoptions(withcolor 1.3*Grey);
    trace perpC2; trace perpC1;
    trace AB;
    trace AC;
    trace BC;
    trace perpD;
    trace SigneOrtho(B,DD,A2,0.2);

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

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

    drawoptions();
    trace gddTraceArcDeCercle(C_AB,0,-Pi/2) dashed evenly;
    trace gddTraceArcDeCercle(C_BA,Pi,3Pi/2) dashed evenly;

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

    drawoptions(withcolor SteelBlue withpen pencircle scaled 0.85pt);
    trace Dvoute1; trace Dvoute2; trace Dvoute3; trace Dvoute4;
    drawoptions();

    gddCouleurPoint := DarkRed;
    pointe O;
    pointe A;
    pointe B; pointe D;
    pointe B2;

    gddCouleurPoint := Bisque;
    pointe F1; pointe F2; pointe F3; pointe F4; pointe F5;
    pointe G1; pointe G2;pointe G3;pointe G4;
    gddCouleurPoint := 1.3*Grey;
    pointe DD;
    pointe C1; pointe C2; pointe C3;
    pointe D1; pointe D2;

    gddCouleurPoint := DarkSalmon;
    pointe P;
    pointe I1; pointe I2; pointe I3; pointe I4;
    Fenetre(-5,-7,5,5);
endfig;
end.
```

14.15 Triangle orthique et problème de Fagnano

Figure 21 de [3].

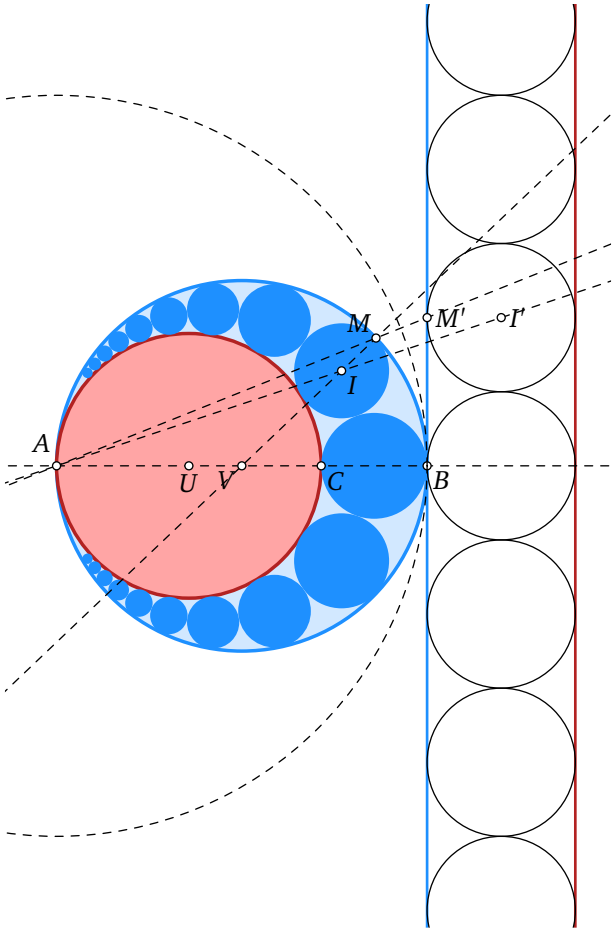



```
input geom2d;
gddU:=1.5cm;
beginfig(1);
numeric A[],B[],C[],E[],F[],G[],U[],W[],V[],T[],R[],P[];
A[1] = Point(0,0);
B1 = Point(1,2);
C1 = Point(2.5,0.2);
W1 = PointDe(Segment(A1,B1),0.2);
U1 = PointDe(Segment(B1,C1),0.23);
V1 = PointDe(Segment(C1,A1),0.3);
G1 = ProjectionPointSurDroite(B1,Droite(A1,C1));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


for i:=0 step 3 until 3:

D[1+i] = Droite(B[1+i],C[1+i]);
A[2+i] = SymetrieAxiale(A[1+i],D[1+i]);

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

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

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

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

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

D[3+i] = Droite(A[3+i],B[3+i]);
C[4+i] = SymetrieAxiale(C[3+i],D[3+i]);
A[4+i] = A[3+i];
B[4+i] = B[3+i];
U[4+i] = SymetrieAxiale(U[3+i],D[3+i]);
V[4+i] = SymetrieAxiale(V[3+i],D[3+i]);
W[4+i] = W[3+i];
E[4+i] = SymetrieAxiale(E[3+i],D[3+i]);
G[4+i] = SymetrieAxiale(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]);
   colorie Q[i] avecCrayon(0.5,LightGray);
   trace P[i] avecCrayon(0.8,MediumSlateBlue);
endfor;
for i:=1 upto 7:
   trace T[i] avecCrayon(1,black);
endfor;

trace LigneBrisee(U1,V2,W3,U4,V5,W6,U7) avecCrayon(1,Peru) ;
trace Segment(U1,U7) dashed evenly;
trace Segment(E1,E7) dashed evenly;
pointe A1; pointe B1; pointe C1;
pointe A2; pointe B3; pointe C4;
pointe A5; pointe B6; pointe 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);

pointe U1; pointe V1; pointe 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;
pointe U4; pointe V2; pointe W3;
pointe U7; pointe V5; pointe W7;
pointe E1; pointe F1; pointe 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;
pointe F7;
gddLabel.rt(textext("$E''$"),E7) withcolor MediumSlateBlue;
pointe E7;
endfig;
end.
```

14.16 Chaîne de Pappus

La chaîne de Pappus par inversion géométrique.

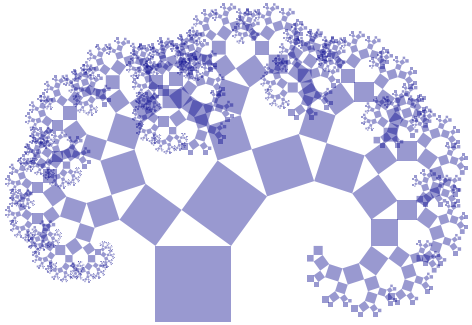



```
input geom2d;

gddU:=0.7cm;

beginfig(1);
   r = 2.5;
   R = 3.5;
   A = Point(0,0);
   C_A = Cercle(A,2*R);
   C = Point(2r,0);
   B = Point(2R,0);
   C_V = CercleD(A,B);
   C_U = CercleD(A,C);
   C_CB=CercleD(C,B);
   V = Centre(C_V);
   U = Centre(C_U);
   AB = Droite(A,B);
   I_CU = Inversion(C_U,C_A);
   I_CV = Inversion(C_V,C_A);

   L_AB = Droite(A,B);
   Up = IntersectionDroites(L_AB,I_CU);
   Vp = IntersectionDroites(L_AB,I_CV);
   Cp0 = CercleD(Vp,Up);
   Rp= Rayon(Cp0);


   colorieAvecTransparence(C_V,DodgerBlue,0.2);
   trace C_V avecCrayon(1.3,DodgerBlue);

   colorie C_U withcolor 4.9*FireBrick;
   trace C_U avecCrayon(1.3,FireBrick);
   colorie C_CB withcolor DodgerBlue;
   trace I_CV avecCrayon(1,DodgerBlue);
   trace I_CU avecCrayon(1,FireBrick);

   Ip[0] = Milieu(Vp,Up);
   N=9;
   trace 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] = Droite(A,Mp[i]);
   DVIp[i] = Droite(V,Ip[i]);
   DAIp[i] = Droite(A,Ip[i]);
   M[i] = IntersectionDroiteCercle(DAMp[i],C_V,2);
   DVM[i] = Droite(V,M[i]);
   I[i] = IntersectionDroites(DAIp[i],DVM[i]);
   colorie CercleCP(I[i],M[i]) withcolor DodgerBlue;
   trace CercleCP(Ip[i],Mp[i]);
   if(i=1):
      drawoptions(dashed evenly);
      trace DAIp[i];trace DVM[i];trace DAMp[i];
      drawoptions();
      pointe Ip[i]; pointe Mp[i];
      pointe M[i]; pointe 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;
trace AB dashed evenly;
marque.bot "U";
marque.llft "V";
marque.ulft "A"; marque.lrt "B"; marque.lrt "C";
trace C_A dashed evenly;

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

end.
```

14.17 L'arbre de Pythagore par similitude

La construction de l'arbre de Pythagore par similitude. Ce code est inspiré de la construction proposée par MicMath sur sa chaîne Twitch.



```
input geom2d;
warningcheck:=0;
angleArbre = Pi/5;

vardef ArbrePythagore(expr A,B,n)=
   save C,D,E;
   numeric C,D,E;
   D = SimilitudeACentre(A,B,-Pi/2,1);
   C = SimilitudeACentre(B,A,Pi/2,1);
   colorieAvecTransparence(Polygone(A,B,C,D), DarkBlue,0.4);
   if(n>0):
      E = SimilitudeACentre(C,D,angleArbre,cos(angleArbre));
      ArbrePythagore(D,E,n-1);
      ArbrePythagore(E,C,n-1);
   fi
enddef;

beginfig(1);

A = Point(0,0);
B = Point(1,0);

ArbrePythagore(A,B,10);

endfig;
end.
```

15 Historique

decembre 2025 : Ajout de la macro FinFenetre, ajout des macros de similitude à centre SimilitudeACentre, ajout de l'exemple « Arbre de Pythagore », correction de bugs, et amélioration de la documentation. Utilisation de withtransparency pour la commande colorieAvecTransparence quand mp-geom2d est utilisé avec LuaTEX et luamplib. Version 1.4.

mai 2025 : Ajout de la macro pointes pour le pointage d'une liste de points. Ajout des macros ParametrePointTaille, ParametrePointCouleur, ParametrePointeType et ParametreConiqueCoef pour modifier les valeurs des variables globales. Ajout de la commande gddClip. Traduction des macros, de la documentation et des exemples en anglais.

25 février 2025 : ajout de la macro PointIntersection. Ajout de la macro HyperboleF . Quelques bugs, et amélioration de la documentation.

février 2025 : ajout de la macro hachure ainsi que la macro SchemaHachure. Correction d'un bug dans la macro EllipseFD. Version 1.2.

décembre 2024 : ajout de la macro DroiteParallele.

novembre 2024 : ajout de la macro EllipseFD ainsi que le stockage des directrices dans le type ellipse. Correction de la commande gddLabel. Ajout de l'exemple de la chaîne de Pappus.

21 octobre 2024 : corrections de Quark67 (merci à lui) sur la documentation et modifications suite à ses remarques. DemieHyperbole devient DemiHyperbole (sans compatibilité). Création de deux nouveaux types de pointage de point : croix et disque (voir commande pointe).

13 octobre 2024 : publication de la version 1.0 sur le CTAN.

Références

[1] Hans Hagen et al. *The luamplib package. Use LuaTeX's built-in MetaPost interpreter*. Version 2.34.5. 3 août 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. 21 juin 2020. URL : https://ctan.org/pkg/latexmp.

[3] Hans Rademacher et 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. 6 avr. 2024. URL : https://ctan.org/pkg/minim-mp.

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

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

Index