Asymptote

Démarrage « rapide »

Version 1.2 – 25 Août 2014

Christophe
GROSPELLIER

cgmaths.fr/Atelier/Asymptote/Asymptote.html

Table des matières

Introduction

Cette documentation, très incomplète, est avant tout destinée à ceux qui voudraient débuter avec `Asymptote`. Si vous utilisez déjà ce logiciel depuis quelque temps, vous ne devriez pas apprendre grand chose... Mieux vaut vous concentrer sur la documentation officielle et les galeries d'exemples que l'on peut trouver sur internet (voir le paragraphe suivant). Certaines figures 3D et les animations peuvent être manipulées ou visualisées à condition de visionner ce document avec `Acrobat Reader`.

I – Où trouver de l'aide

Dans la documentation et le forum officiels (en anglais) : asymptote.sourceforge.net

Dans la galerie d'exemples de Philippe Ivalvi : www.piprime.fr/asymptote
Dans la galerie d'exemples de Gaétan Marris : www.marris.org/asymptote
Ces deux galeries seront un très bon complément à ce démarrage rapide, notamment pour comprendre quelques subtilités et pour aller (beaucoup) plus loin.

Sur le forum et le wiki de MathemaTex : mathematex.net
Sur le forum de Gaétan Marris : asy.gmaths.net/forum

Pour la 3D, il y a l'excellente documentation de Bruno Colombel : www.mathco.tuxfamily.org/asy3d-index.xml et ASY_3d.pdf

Et enfin, en lisant directement les fichiers `.asy` dans le répertoire d'installation d'`Asymptote`. Quelques connaissances en programmation et en anglais seront nécessaires.

II – Installation

Tout d'abord, une distribution LaTeX minimale doit être installée. `Asymptote` fait partie des principales distributions LaTeX.

Pour savoir si `Asymptote` est installé, taper `asy` dans un terminal, vous verrez apparaitre le numéro de version si c'est le cas. Taper `quit` pour sortir du mode interactif.

Sinon, il faut l'installer.

Je ne traiterai pas l'installation sur Microsoft Windows ou MacOS X, ne connaissant pas (ou plus) ces systèmes. Vous trouverez tous les renseignements dans la documentation officielle, ou sur asymptote.sourceforge.net.

Sur Unix, il existe un paquet `Asymptote` sur la plupart des distributions, à installer par votre gestionnaire de paquet, mais cette version est parfois loin d'être à jour...
Pour avoir une version récente du logiciel, une compilation des sources pour installer la version **svn** est une bonne alternative.

Voici comment faire sur **(X)Ubuntu 14.04**.
Attention, il faut désinstaller toute version préalablement installée.
Si `Asymptote` a été installé en même temps que Texlive, par le script `install-tl` par exemple, il faut le supprimer à l'aide de `tlmgr` :

```
sudo tlmgr remove --force asymptote
```

1. Les paquets nécessaires

```
sudo apt-get install build-essential subversion flex texinfo autoconf zlib1g-dev bison
sudo apt-get install freeglut3-dev cdbs libfftw3-dev libreadline6-dev libncurses5-dev
sudo apt-get install libgsl0-dev libsigsegv-dev imagemagick libosmesa6-dev
```

2. Récupérer les sources

```
mkdir asymptote_svn
cd asymptote_svn
svn co http://svn.code.sf.net/p/asymptote/code/trunk/asymptote
```

3. Compiler

```
cd asymptote
./autogen.sh
wget http://hboehm.info/gc/gc_source/gc-7.4.0.tar.gz
wget http://hboehm.info/gc/gc_source/libatomic_ops-7.4.0.tar.gz
./configure
make all
sudo make install
```

4. Pour les mises à jour, c'est

```
cd asymptote_svn/asymptote
svn update
sudo make install
```

Remarques :
Si vous êtes sous Gnome, vous avez l'éditeur de texte `gedit` déjà installé. Vous trouverez tout ce qu'il faut pour l'utiliser avec `Asymptote` à cette adresse : cgmaths.fr/Atelier/Asymptote/OutilsGedit.html

Un équivalent sur Windows est `Notepad++`, pour son utilisation, vous pouvez vous rendre dans la rubrique « Installation d'`Asymptote` » sur le forum asy.gmaths.net/forum.

III – Les différentes compilations

1) Options de compilation

Les options de compilation sont très nombreuses (voir la documentation officielle). Nous ne nous intéresserons ici qu'à quelques-unes.
Créez un fichier texte, nommez le `exemple.asy` et collez-y un des codes d'exemple ci-après (pas le premier !). Enregistrez le document.
La compilation de ce code se fait en ligne de commande, avec les options par défaut, par

```
asy exemple.asy
```

Quelques options intéressantes :

`-V` : utile notamment pour les figures 3D, la figure s'ouvre dans une fenêtre de rendu OpenGL. La figure est alors manipulable, et un double clique-droit permet d'accéder à différents paramètres.

`-noV` : pour que le `pdf` (ou `eps`, ou autre) soit créé directement.

`-f` : pour préciser le format de sortie (`pdf`, ...).
Équivalent à `settings.outformat="pdf";` (ou autre) dans le code de la figure.

`-prc` : pour les figures 3D : format `prc` d'adobe dans le `pdf`. Le `pdf` ne sera visualisable qu'avec une version suffisamment récente d'`Acrobat Reader`. La figure sera manipulable et on aura accès à un menu 3D.

`-noprc` : pas de format `prc`. La figure peut être visualisée dans n'importe quel lecteur `pdf`.
Équivalent à `settings.prc=false;` dans le code de la figure.

`-render n` : rendu des figures 3D (n pixels/bp).
Équivalent à `settings.render=n;` dans le code de la figure.

Exemple de compilation

```
asy -noV -noprc -f pdf exemple.asy
```

La figure produite peut être incluse dans un document `.tex` par un `\includegraphics`.

2) Compilation à l'intérieur d'un fichier .tex

Il sera vu le minimum ici. Pour davantage d'explications, voir la documentation officielle.

- **Méthode « classique »**
  Il faut ajouter dans le préambule du document la ligne :

  ```
  \usepackage{asymptote}
  ```

  Le code de la figure doit être compris entre les balises `\begin{asy}` et `\end{asy}`.

  ```
  \begin{asy}[width=\the\linewidth,inline=true]
  ...
  \end{asy}
  ```

  Ci-dessus sont précisées en options la largeur et `inline=true` qui permet d'utiliser des symboles LaTeX définis en dehors de l'environnement `asy`.

  La compilation se fait en trois temps :

  1. Une compilation `latex` (ou `pdflatex`). Un fichier `.asy` par figure est créé.
  2. Une compilation `asy`. Pour compiler toutes les figures, si le fichier `tex` s'appelle `exemple.tex` :

     ```
     asy exemple-*.asy
     ```

     Les figures sont créées au format `eps` pour une compilation `latex`, ou au format `pdf` pour une compilation `pdflatex`.
  3. Une nouvelle compilation `latex` (ou `pdflatex`).

- **latexmk**
  Il est possible de tout compiler en une fois grâce au script `latexmk`. Ce script est peut-être déjà installé sur votre distribution ou alors il est téléchargeable sur le CTAN :
  www.ctan.org/tex-archive/support/latexmk/.
  Une fois installé, il faut créer un fichier nommé `latexmkrc` contenant :

  ```
  sub asy {return system("asy '$_[0]'");}
  add_cus_dep("asy","eps",0,"asy");
  add_cus_dep("asy","pdf",0,"asy");
  add_cus_dep("asy","tex",0,"asy");
  ```

  Ce fichier peut être placé dans le même répertoire que le fichier `tex`.
  Note : sur linux, placer ce fichier une fois pour toutes dans le répertoire personnel en le nommant « `.latexmkrc` ».
  Attention, `perl` doit être installé sur l'ordinateur.

  Ensuite une seule compilation suffit :

  ```
  latexmk -pdf exemple.tex
  ```

  créera un `pdf` (seules les figures ayant été modifiées depuis la dernière compilation seront compilées).

Remarque :
Pour que les figures soient créées dans un sous-dossier `figures`, il faut rajouter dans le préambule du document `tex` :

```
\def\asydir{figures}
```

et remplacer la première ligne du fichier `latexmkrc` par :
`sub asy {return system("asy -o figures/ '$_[0]'");}`

- **Package asypictureB**
  Nouveau package, très bonne alternative à `latexmk`, mais ne fonctionne pas (encore) pour la 3D. Il suffit de l'importer dans le préambule à la place du package `asymptote`. Une seule compilation suffit : www.ctan.org/pkg/asypictureb.

IV – Quelques généralités

Une bonne partie de la documentation officielle sera reprise ici.
Les exemples de cette partie pourront être compilés simplement par : `asy mafigure.asy` (pour de l'eps) ou `asy -f pdf mafigure.asy` (pour du pdf).

1) Unités, dimensions de la figure

Contrairement à des logiciels comme GeoGebra, on ne peut pas placer de points « au hasard », tout se passe dans un repère : les points sont placés par leurs coordonnées.
Ces coordonnées, appelées *pairs* sont en "big points" PostScript (1 bp = 1/72 inch).
Ainsi, avec le code 1, on ne verra rien ! la figure étant beaucoup trop petite...
Les codes 2 et 3 produiront la même figure, à savoir un segment de longueur $2\sqrt{2}$ cm.

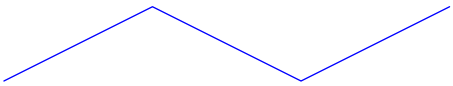
CODE 1
```
pair A=(0,0), B=(2,2);
draw(A--B);
```

CODE 2
```
pair A=(0,0), B=(2cm,2cm);
draw(A--B);
```

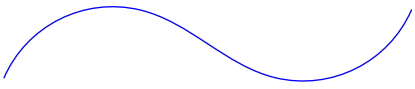
CODE 3
```
unitsize(1cm);
pair A=(0,0), B=(2,2);
draw(A--B);
```

Remarques :

- L'« unité » peut être différente pour les abscisses et les ordonnées : `unitsize(1cm,1.5cm);`
- On peut aussi spécifier la taille finale de la figure produite :
  Pour une figure de 3cm de largeur, on précisera : `size(3cm,0);` (0 pour ne pas avoir à spécifier la hauteur, la mise à l'échelle étant automatique.)
  La taille peut être spécifiée en `bp`, `pt`, `cm`, `mm` ou `inches`.
- `--` est un connecteur qui joint les points par un segment. Les autres connecteurs sont `..`, `::` et `---` (voir la documentation officielle).



CODE 4
```
size(0,1cm);

draw((0,0)--(1,.5)--(2,0)--(3,.5),blue);
```



CODE 5
```
size(0,1cm);

draw((0,0)..(1,.5)..(2,0)..(3,.5),blue);
```

2) Quelques commandes de dessins

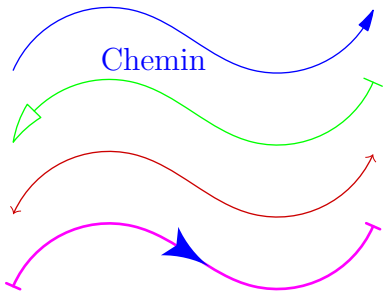
a) draw

```
void draw(picture pic=currentpicture, Label L="", path g,
          align align=NoAlign, pen p=currentpen,
          arrowbar arrow=None, arrowbar bar=None, margin margin=NoMargin,
          Label legend="", marker marker=nomarker)
```

Voyons en détail quelques options (pour les autres, voir la documentation officielle) :

- `picture pic=currentpicture` : dessine dans l'image `pic`, par défaut `currentpicture`.
- `Label L=""` : permet de placer un texte.
- `path g` : le *chemin* à dessiner.
- `pen p=currentpen` : le *stylo* à utiliser.
- `arrowbar arrow=None` : Les *flèches*. Les valeurs possibles sont : `None`, `Blank`, `BeginArrow`, `MidArrow`, `Arrow`, et `Arrows` (flèches aux deux extrémités). On peut de plus préciser le type de tête (`DefaultHead`, `SimpleHead`, `HookHead`, `TeXHead`), la taille, l'angle de la tête, le type de remplissage (`FillDraw`, `Fill`, `NoFill`, `UnFill`, `Draw`) et la position relative.

  Il y a aussi certaines versions modifiées (taille et angle) : `BeginArcArrow`, `MidArcArrow`, `ArcArrow`, et `ArcArrows`.
- `arrowbar bar=None` : Les *barres* aux extrémités. Les valeurs possibles sont : `None`, `BeginBar`, `Bar` et `Bars`. On peut préciser la taille.



CODE 6

```
size(5cm,0);

path chemin=(0,0)..(1,.5)..(2,0)..(3,.5);

draw("Chemin",chemin,blue,Arrow);
draw(shift(0,-.6)*chemin,green,BeginArrow(5mm,Draw),Bar);
draw(shift(0,-1.2)*chemin,.8*red,Arrows(TeXHead));
draw(shift(0,-1.8)*chemin,bp+magenta,MidArrow(HookHead,Fill(blue)),Bars(2mm));
```

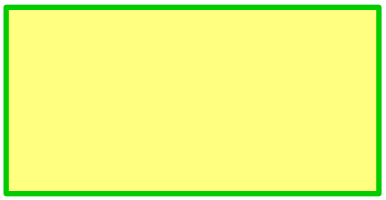
b) fill

```
void fill(picture pic=currentpicture, path g, pen p=currentpen)
void filldraw(picture pic=currentpicture, path g,
              pen fillpen=currentpen,pen drawpen=currentpen)
```

Ces deux fonctions servent à remplir les chemins cycliques :

CODE 7

```
size(5cm,0);

path rec=(0,0)--(1,0)--(1,.5)--(0,.5)--cycle;

fill(rec,lightyellow);
```



CODE 8

```
size(5cm,0);

path r=(0,0)--(1,0)--(1,.5)--(0,.5)--cycle;

filldraw(r,fillpen=lightyellow,
         drawpen=2bp+.8green);
```

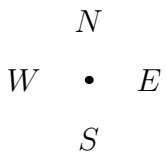
c) label

```
void label(picture pic=currentpicture, Label L, pair position,
           align align=NoAlign, pen p=nullpen, filltype filltype=NoFill)
```

Cette fonction permet de placer un texte au point de coordonnées `position`. Le texte peut être une simple chaîne de caractères (respectant la syntaxe LaTeX) ou une structure plus complexe obtenue par une fonction du type (il y en a d'autres, voir la documentation officielle) :

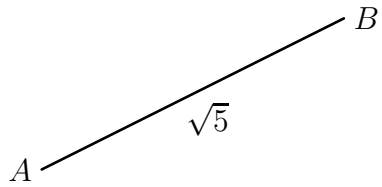
```
Label Label(string s="", pair position, align align=NoAlign,
            pen p=nullpen, embed embed=Rotate, filltype filltype=NoFill)
```

`align` peut prendre n'importe quel `pair` comme valeurs. Certaines sont prédéfinies : `N`, `S`, `E`, `W`, `NE`, `NW`, `SE`, `SW`, `LeftSide`, `RightSide`, `Center`.
Voir les exemples qui suivent pour les différents paramètres.



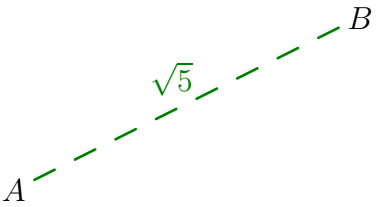
CODE 9

```
size(5cm,0);
pair A=(0,0);
dot(A);
label("$N$",A,5*N);
label("$S$",A,5*S);
label("$E$",A,5*E);
label("$W$",A,5*W);
```



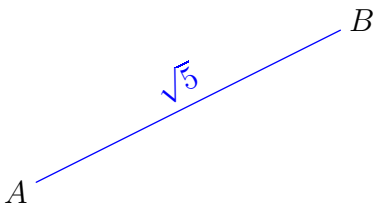
CODE 10

```
size(5cm);
pair A=(0,0), B=(2,1);
path seg=A--B;
label("$A$",A,W);
label("$B$",B,E);
draw("$\sqrt{5}$",seg,
     linewidth(bp));
```



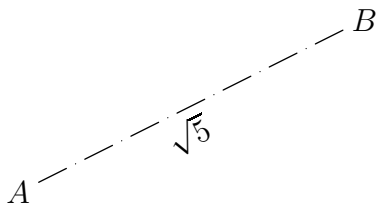
CODE 11

```
size(5cm);
pair A=(0,0), B=(2,1);
path seg=A--B;
label("$A$",A,dir(A-B));
label("$B$",B,dir(B-A));
draw(Label("$\sqrt{5}$",
     align=LeftSide),
     seg,bp+deepgreen+dashed);
```



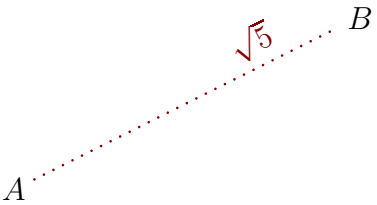
CODE 12

```
size(5cm);

pair A=(0,0), B=(2,1);
path seg=A--B;

label("$A$",A,dir(A-B));
label("$B$",B,dir(B-A));

draw(Label("$\sqrt{5}$",
       Rotate(dir(seg)),
       align=LeftSide),
     seg,blue);
```



CODE 13

```
size(5cm);

pair A=(0,0), B=(2,1);
path seg=A--B;

label("$A$",A,dir(A-B));
label("$B$",B,dir(B-A));

draw(Label("$\sqrt{5}$",
       Rotate(dir(seg)),
       align=S),
     seg,longdashdotted);
```



CODE 14

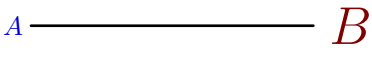
```
size(5cm);

pair A=(0,0), B=(2,1);
path seg=A--B;

label("$A$",A,dir(A-B));
label("$B$",B,dir(B-A));

draw(Label("$\sqrt{5}$",
       Rotate(dir(seg)),
       align=LeftSide,
       position=Relative(.75)),
     seg,bp+brown+dotted);
```

CODE 15

```
size(5cm);

pair A=(0,0), B=(1,0);
path seg=A--B;

label("$x'$",A,N);
label("$x$",B,N);

draw(Label("abscisses",
       filltype=Fill(orange)),
     seg,bp+deepcyan);
```

CODE 16

```
unitsize(1cm);

pair A=(0,0), B=(4,0);
path seg=A--B;

label("$A$",A,W);
label("$B$",B,E);

draw(Label("$4$~cm",
       align=Center,
       filltype=UnFill),
     seg,bp+fuchsia,
     Arrows(SimpleHead),Bars);
```



CODE 17

```
size(5cm,0);
import fontsize;
defaultpen(fontsize(10pt));

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

draw(A--B,linewidth(bp));

label("$A$",A,W,.8blue);

pen f20=fontsize(20pt);
label("$B$",B,E,f20+brown);
```

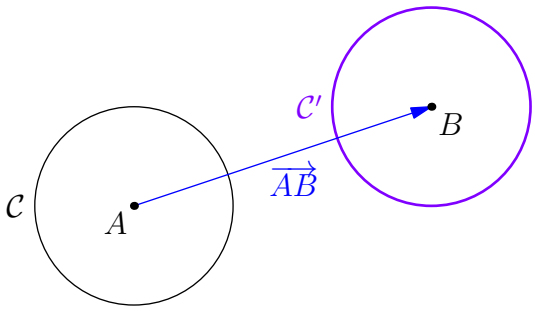
Remarque :
Dans les exemples, `B-A` peut être interprété comme le vecteur $\overrightarrow{AB}$.
On peut réaliser différentes opérations sur les `pairs`. Par exemple `pair I=(A+B)/2;` donnera le milieu I de [AB] (équivalent à `pair I=midpoint(A--B);`).

V – Transformations

Présentation rapide des transformations d'`Asymptote`.

Pour la première, pas besoin de figure... : `transform identity();`
Ensuite :

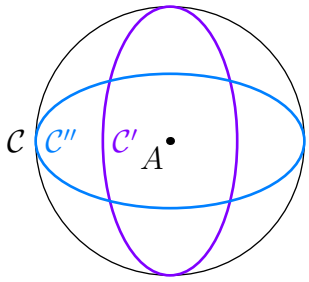
```
transform shift(pair z);
transform shift(real x,real y);
```



CODE 18

```
size(7cm);
pair A=(0,0), B=(3,1);

path c=unitcircle;
path c1=shift(3,1)*c;

draw("$\mathcal{C}$",c);
draw("$\mathcal{C'}$",c1,bp+purple);
draw("$\overrightarrow{AB}$",
     A--B,blue,Arrow);
dot("$A$",A,SW);
dot("$B$",B,SE);
```

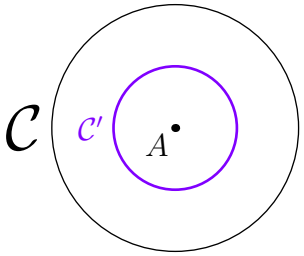
```
transform xscale(real x);
transform yscale(real y);
```



CODE 19

```
size(4cm);
pair A=(0,0);

path c=unitcircle;
path c1=xscale(.5)*c;
path c2=yscale(.5)*c;

draw("$\mathcal{C}$",c);
draw("$\mathcal{C'}$",c1,E,bp+purple);
draw("$\mathcal{C''}$",c2,E,bp+royalblue);

dot("$A$",A,SW);
```

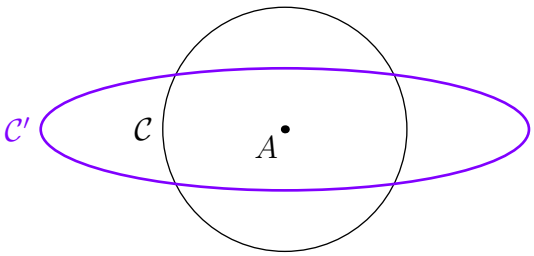
```
transform scale(real s);
```



CODE 20

```
size(4cm);
pair A=(0,0);

path c=unitcircle;
path c1=scale(.5)*c;

draw(scale(2)*"$\mathcal{C}$",c);
draw("$\mathcal{C'}$",c1,bp+purple);

dot("$A$",A,SW);
```

```
transform scale(real x,real y);
```

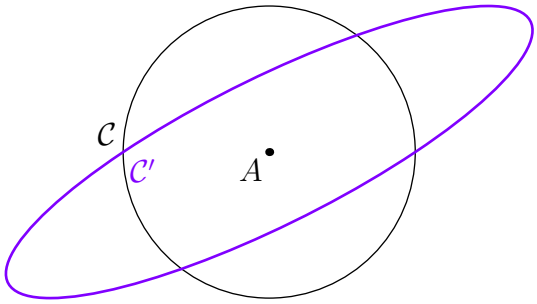


CODE 21

```
size(7cm);
pair A=(0,0);

path c=unitcircle;
path c1=scale(2,.5)*c;

draw("$\mathcal{C}$",c);
draw("$\mathcal{C'}$",c1,bp+purple);

dot("$A$",A,SW);
```

`transform slant(real s);`

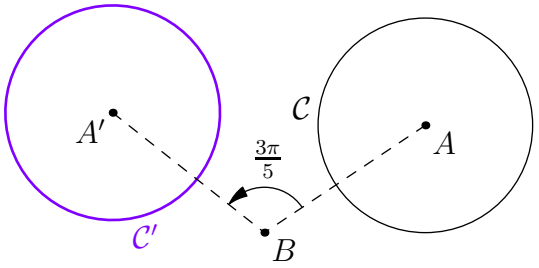
envoie le pair `(x,y)` sur `(x+s*y,y)`



CODE 22

```
size(7cm);
pair A=(0,0);

path c=unitcircle;
path c1=slant(1.5)*c;

draw("$\mathcal{C}$",c,NW);
draw("$\mathcal{C'}$",c1,SE,bp+purple);

dot("$A$",A,SW);
```

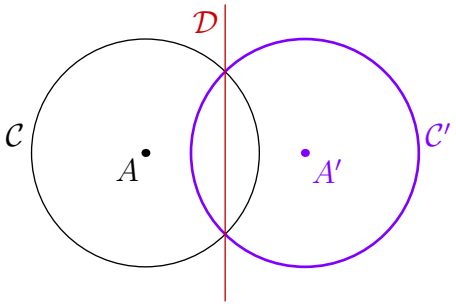
`transform rotate(real angle,pair z=(0,0));`



CODE 23

```
import markers; // pour utiliser markangle
size(7cm);
pair A=(0,0), B=(-1.5,-1);
transform rot=rotate(108,B);
path c=unitcircle;

pair A1=rot*A;
path c1=rot*c;

draw("$\mathcal{C}$",c,NW);
draw("$\mathcal{C'}$",c1,S,bp+purple);

draw(A--B--A1,dashed);
markangle("$\frac{3\pi}{5}$",A,B,A1,
          radius=6mm,Arrow);
dot("$A$",A,SE);dot("$A'$",A1,SW);
dot("$B$",B,SE);
```

`transform reflect(pair a, pair b);`

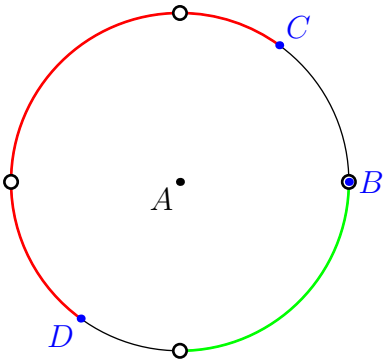


CODE 24

```
size(6cm);

pair A=(0,0), B=(.7,-1.3), C=(.7,1.3);

path c=unitcircle;
path c1=reflect(B,C)*c;
pair A1=reflect(B,C)*A;

draw("$\mathcal{C}$",c,NW);
draw("$\mathcal{C'}$",c1,NE,bp+purple);
draw(Label("$\mathcal{D}$",EndPoint),B--C,SW,.8red);
dot("$A$",A,SW);dot("$A'$",A1,SE,purple);
```

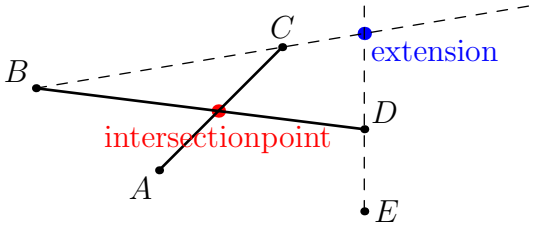
VI – Quelques figures basiques

Voici quelques figures permettant de parcourir une partie des outils de base d'Asymptote.



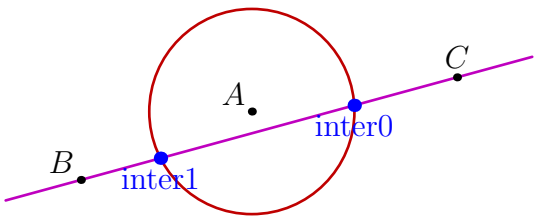
CODE 25

```
size(5cm,0);

pair A=(0,0);

path c=circle(A,1.5);
pair B=relpoint(c,0), C=relpoint(c,.15),
     D=relpoint(c,.65);
path a=subpath(c,reltime(c,.15),reltime(c,.65));
path b=subpath(c,3,4);
draw(c);
draw(a,bp+red);
draw(b,bp+green);
dot(c,linewidth(bp),UnFill);
dot("$A$",A,SW);dot("$B$",B,E,blue);
dot("$C$",C,NE,blue);dot("$D$",D,SW,blue);
```



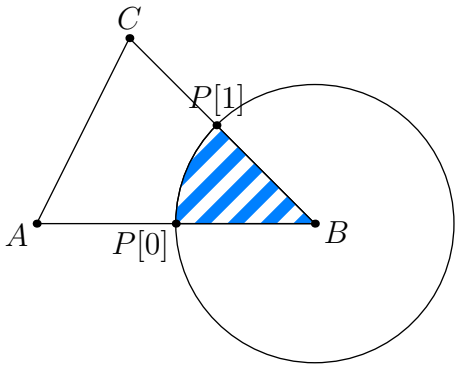
CODE 26

```
size(7cm,0);

pair A=(0,0), B=(-1.5,1), C=(1.5,1.5),
     D=(2.5,.5), pE=(2.5,-.5);
path seg1=A--C, seg2=B--D;

pair inter=intersectionpoint(seg1,seg2);
pair ext=extension(B,C,D,pE);

dot("intersectionpoint",inter,1.5*S,5bp+red);
dot("extension",ext,SE,5bp+blue);

draw(seg1,linewidth(bp));
draw(seg2,linewidth(bp));

draw(B--interp(B,C,2),dashed);
draw(pE--interp(pE,D,2.5),dashed);

dot("$A$",A,SW);dot("$B$",B,NW);
dot("$C$",C,N);dot("$D$",D,NE);
dot("$E$",pE,E);
```



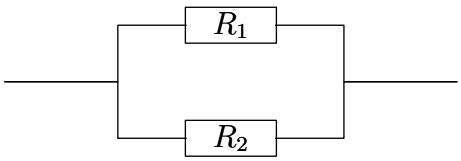
CODE 27

```
size(7cm,0);

pair A=(0,0), B=(-2.5,-1), C=(3,.5);
path c=circle(A,1.5);
path d=interp(B,C,-.2)--interp(B,C,1.2);
draw(c,bp+heavyred);draw(d,bp+heavymagenta);

// tableau de pairs
pair[] inter=intersectionpoints(c,d);

for(int i=0; i<inter.length; ++i) {
    dot("inter"+string(i),inter[i],S,5bp+blue);
}

dot("$A$",A,NW);dot("$B$",B,NW);dot("$C$",C,N);
```



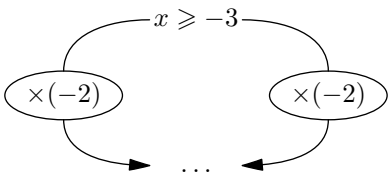
CODE 28

```
import patterns; // hachures
size(6cm,0);

pair A=(0,0), B=(3,0), C=(1,2);

path tri=A--B--C--cycle;
path c=circle(B,1.5);

pair[] P=intersectionpoints(tri,c);
path intersec=B--P[0]--arc(B,P[0],P[1],CW)--cycle;
// CW : ClockWise, CCW : Counter-ClockWise (défaut)
draw(tri);
draw(c);
// Définition des hachures :
add("hachure",hatch(3mm,NE,1.5mm+royalblue));

filldraw(intersec,pattern("hachure"));

dot("$P[0]$",P[0],SW);dot("$P[1]$",P[1],N);
dot("$A$",A,dir(C--A,B--A)); //direction de la bissectrice
dot("$B$",B,dir(C--B,A--B));dot("$C$",C,N);
```



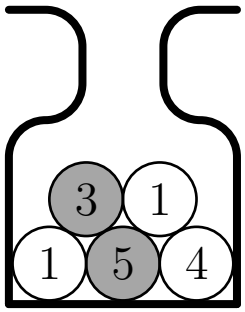
CODE 29

```
size(6cm);
pair A=(0,0), B=(1,0), C=(1,.5), D=(1,-.5),
     pE=(3,.5), F=(3,-.5), G=(3,0), H=(4,0);

draw(A--B--C--pE--G--H);
draw(A--B--D--F--G--H);

draw("$R_1$",box,(2,.5),xmargin=3mm,ymargin=0,UnFill);
draw("$R_1$",box,(2,.5),xmargin=3mm,ymargin=0);

draw("$R_2$",box,(2,-.5),xmargin=3mm,ymargin=0,UnFill);
draw("$R_2$",box,(2,-.5),xmargin=3mm,ymargin=0);
```



CODE 30

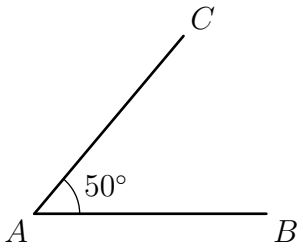
```
unitsize(1cm);
defaultpen(fontsize(10pt));
usepackage("amsmath");
usepackage("amsfonts");
usepackage("amssymb");

pair z0=(-1.75,0), z1=(1.75,0), z2=(0,1), z3=(0,-1);

object boite1=draw("$x \geqslant -3$",box,z2,invisible);
object boite2=draw("$\times (-2)$",ellipse,z0);
object boite3=draw("$\times (-2)$",ellipse,z1);
object boite4=draw("$\quad\dots\quad$",box,z3,invisible);

add(new void(picture pic, transform t) {
    draw(pic,point(boite1,W,t){W}..{S}point(boite2,N,t));
    draw(pic,point(boite2,S,t){S}..{E}point(boite4,NW,t),Arrow);
    draw(pic,point(boite1,E,t){E}..{S}point(boite3,N,t));
    draw(pic,point(boite3,S,t){S}..{W}point(boite4,NE,t),Arrow);
});
```



CODE 31

```
size(4cm);
transform r=reflect((0,0),(0,1));
// Fonction pour tracer les boules
void boule(Label L="", pair a, pen p=currentpen, pen fillpen=nullpen) {
       filldraw(shift(a)*unitcircle,fillpen,p);
       label(scale(1.5)*L,a);
}
// Urne
pair A=(3.1,0), B=(3.1,4), C=(2.1,5), D=(1.1,6), pE=(1.1,7), F=(2.1,8), G=(3.1,8);
pair H=r*A;
path demi=G--F{dir(180)}..{dir(270)}pE--D{dir(270)}..{dir(0)}C{dir(0)}..{dir(-90)}B--A;
path demi2=r*demi;
draw(demi^^demi2,linewidth(3bp));draw(A--H,linewidth(3bp));
// Boules
boule("$5$",(0,1.1),linewidth(bp),gray+opacity(.7));
boule("$1$",(-2,1.1),linewidth(bp));
boule("$4$",(2,1.1),linewidth(bp));
boule("$3$",(-1,2.85),linewidth(bp),gray+opacity(.7));
boule("$1$",(1,2.85),linewidth(bp));
```

Exemple provenant de la galerie de Philippe Ivaldi (Voir I)



CODE 32

```
import markers;
size(4cm);

real ang=50;
pair A=(0,0), B=(3,0);

transform r=rotate(ang,A);
pair C=r*B;

path seg1=A--B;
path seg2=r*seg1;

draw(seg1^^seg2,linewidth(bp));

// fonction format, voir ci-contre
markangle(format("$%f^\circ$",ang),
          B,A,C,radius=6mm);

label("$A$",A,SW);
label("$B$",B,SE);
label("$C$",C,NE);
```

format(6.66666)	6,667
format("$x=%f$", 6.66666)	$x = 6{,}66666$
format("$x=%.1f$", 6.66666)	$x = 6{,}7$
format("$x=%.2f$", 6.66666)	$x = 6{,}67$
format("$x=%.0f$", 6.6666)	$x = 7$
format("$x=%07.3f$", 6.6666)	$x = 006{,}667$
format("$x=%7.3f$", 6.6666)	$x =\ \ 6{,}667$
format("$x=%g$", 66.66666)	$x = 66{,}666$
format("$x=%g$", 666666.666)	$x = 666667$
format("$x=%g$", 666666666.666)	$x = 6{,}66667\times10^{8}$
format("$x=%e$", 666666.666)	$x = 6{,}666667\times10^{5}$
format("$x=%.2e$", 666666.666)	$x = 6{,}67\times10^{5}$
format("$x=%i$", 6)	$x = 6$
format("$x=%f$", 6.0)	$x = 6$
format("$x=%+.2f$", 6.66666)	$x = +6{,}67$
format("$x=%+.2f$", -6.66666)	$x = -6{,}67$
format("$x=% .2f$", 6.666666)	$x =\ \ 6{,}67$

CODE 33

```
import labelpath;
size(0,3.5cm);
path p=(0,0)..(2,-1)..(2.5,-.5)..(3,-.75);
labelpath("\Large Fin de la pr\'esentation des outils de base",p,purple);
draw(p,heavygreen);
```

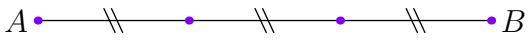
VII – Quelques modules

Je ne présenterai ici que les modules que j'utilise et donc que je connais un peu...
Pour les autres (`graph3`, `grid3`, `interpolate`, etc.), voir la documentation officielle, et les galeries d'exemples (I).

1) markers

Le module markers fournit tout ce qu'il faut pour marquer segments et angles.
Les marqueurs pour les segments sont :

```
marker StickIntervalMarker(int i=2, int n=1, real size=0, real space=0,
                           real angle=0, pair offset=0, bool rotated=true,
                           pen p=currentpen, frame uniform=newframe,
                           bool above=true)
```



CODE 34

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,StickIntervalMarker(3,2,
          angle=25,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

CODE 35

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,StickIntervalMarker(4,1,size=5mm,
     angle=-45,bp+.8green,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

```
marker CrossIntervalMarker(int i=2, int n=3, real size=0, real space=0,
                           real angle=0, pair offset=0, bool rotated=true,
                           pen p=currentpen, frame uniform=newframe,
                           bool above=true)
```

CODE 36

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CrossIntervalMarker(2,4,angle=0,
          bp+red,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

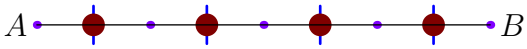
CODE 37

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CrossIntervalMarker(3,6,size=2mm,
          angle=45,bp+blue,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

```
marker CircleBarIntervalMarker(int i=2, int n=1, real barsize=0, real radius=0,
                               real angle=0, pair offset=0, bool rotated=true,
                               pen p=currentpen, filltype filltype=NoFill,
                               bool circleabove=false, frame uniform=newframe,
                               bool above=true)
```

CODE 38

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CircleBarIntervalMarker(3,2,
          barsize=5mm,radius=1.5mm,
          angle=-45,darkgreen,
          filltype=NoFill,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```



CODE 39

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CircleBarIntervalMarker(4,1,
          barsize=5mm,radius=1.5mm,bp+blue,
          filltype=Fill(brown),
          circleabove=true,
          dotframe(purple),above=false));
label("$A$",A,W);label("$B$",B,E);
```

CODE 40

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CircleBarIntervalMarker(4,1,
          barsize=5mm,radius=1.5mm,
          bp+blue,filltype=Fill(brown),
          dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

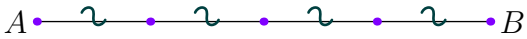
CODE 41

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,CircleBarIntervalMarker(3,0,
     radius=1.2mm,bp+fuchsia,
     filltype=NoFill,dotframe(purple)));
label("$A$",A,W);label("$B$",B,E);
```

```
marker TildeIntervalMarker(int i=2, int n=1, real size=0, real space=0,
                           real angle=0, pair offset=0, bool rotated=true,
                           pen p=currentpen, frame uniform=newframe,
                           bool above=true)
```

CODE 42

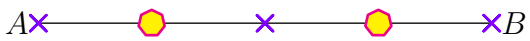
```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,TildeIntervalMarker(3,2,size=1mm,
          space=4mm,bp+orange,
          dotframe(purple),above=false));
label("$A$",A,W);label("$B$",B,E);
```



CODE 43

```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
draw(A--B,TildeIntervalMarker(4,1,size=1mm,
          angle=-40,offset=(0,2),bp+darkcyan,
          dotframe(purple),above=true));
label("$A$",A,W);label("$B$",B,E);
```

Il est bien sûr possible de définir ses propres « markers » :



CODE 44

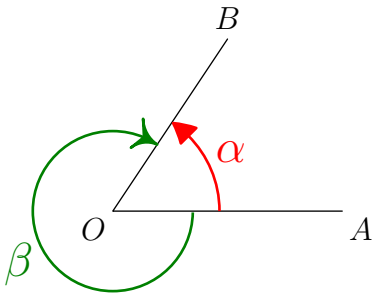
```
import markers;
size(7cm,0);
pair A=(0,0), B=(3,0);
frame cg;
filldraw(cg,scale(5)*polygon(7),Yellow,bp+Magenta);
marker markcg = marker(crossframe(n=4,size=1.5mm,bp+purple),
                       markinterval(2,cg,true));
draw(A--B,markcg);
label("$A$",A,W);label("$B$",B,E);
```

Les marques prédéfinies sont :

```
frame stickframe(int n=1, real size=0, pair space=0, real angle=0,
                 pair offset=0, pen p=currentpen)

frame circlebarframe(int n=1, real barsize=0,
                     real radius=0,real angle=0,
                     pair offset=0, pen p=currentpen,
                     filltype filltype=NoFill, bool above=false)

frame crossframe(int n=3, real size=0, pair space=0,
                 real angle=0, pair offset=0, pen p=currentpen)

frame tildeframe(int n=1, real size=0, pair space=0,
                 real angle=0, pair offset=0, pen p=currentpen)
```

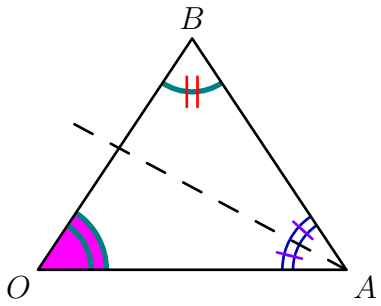
Pour marquer l'angle $\widehat{AOB}$ (dans le sens direct), on dispose de :

```
void markangle(picture pic=currentpicture, Label L="",
               int n=1, real radius=0, real space=0,
               pair A, pair O, pair B, arrowbar arrow=None,
               pen p=currentpen, filltype filltype=NoFill,
               margin margin=NoMargin, marker marker=nomarker)
```



CODE 45

```
import markers;
size(5cm,0);
pair O=(0,0), A=(2,0), B=(1,1.5);

draw(A--O--B);

markangle(scale(1.5)*"$\alpha$",radius=40,
          A,O,B,ArcArrow,bp+red);
markangle(scale(1.5)*"$\beta$",radius=30,
          B,O,A,BeginArcArrow(HookHead,2mm),
          bp+deepgreen);
label("$O$",O,SW);
label("$A$",A,SE);label("$B$",B,N);
```



CODE 46

```
import markers;
size(5cm,0);
pair O=(0,0), A=(2,0), B=(1,1.5),
     C=A+dir(A--O,A--B);

markangle(n=2,radius=20,
          A,O,B,p=2bp+deepcyan,
          filltype=Fill(magenta));
markangle(n=1,radius=20,
          O,B,A,p=2bp+deepcyan,
          marker(markinterval(stickframe(n=2,
          4mm,bp+red),true)));
markangle(n=2,radius=20,
          B,A,O,p=bp+heavyblue,
          marker(markinterval(2,stickframe(n=1,
          2mm,bp+purple),true)));
draw(A--interp(A,C,2),bp+dashed);

draw(A--O--B--cycle,linewidth(bp));
label("$O$",O,SW);
label("$A$",A,SE);label("$B$",B,N);
```

2) geometry

Ce module apporte un grand nombre d'outils permettant de faciliter la construction de figures géométriques dans le plan.
Une fois n'est pas coutume, il y a une **vraie** documentation en français, faite par l'auteur du module : Philippe Ivaldi. On la trouve ici : www.piprime.fr/files/res/geometry_fr.pdf
Elle est extrêmement complète. Il existe de plus un index à cette adresse :
www.piprime.fr/files/asymptote/geometry/modules/geometry.asy.index.type.html

Ce qui suit n'est ni plus ni moins qu'un copié-collé d'une toute petite partie de cette documentation dont la lecture est **essentielle**.

1. De nouveaux types d'objets sont définis :

   - `coordsys` : permet d'utliser n'importe quel repère cartésien.
   - `point` : équivalent au type `pair` dans le repère par défaut.
   - `vector` : aussi équivalent au type `pair` dans le repère par défaut.
   - `mass` : pour les barycentres.
   - `line` : quelques objets de type `line` :

     `line line(point A, bool extendA=true, point B, bool extendB=true)`
     Pour les droites, demi-droites (`extendA=false` ou `extendB=false`) et segments de droites (`extendA=false` et `extendB=false`).

     `line line(coordsys R=currentcoordsys, real a, real b, real c)`
     Renvoie la droite d'équation $ax+by+c=0$ dans le repère `R`.

     `line line(coordsys R=currentcoordsys, real slope, real origin)`
     Renvoie la droite de pente `slope` et d'ordonnée à l'origine `origin` donnés relativement au repère `R`.

     `line parallel(point M, line l)`
     Renvoie la droite parallèle à `l` passant par `M`.

     `line parallel(point M, explicit vector dir)`
     Renvoie la droite de vecteur directeur `dir` et passant par `M`.

     `line parallel(point M, explicit pair dir)`
     Renvoie la droite de vecteur directeur `dir` donné dans le repère courant `currentcoordsys` et passant par `M`.

     `line line(real a, point A=point(currentcoordsys,(0,0)))`
     Renvoie la droite passant par A et faisant un angle de a degrés avec l'axe des abscisses du repère dans lequel est défini A.

     `line bisector(line l1, line l2, real angle=0, bool sharp=true)`
     Renvoie l'image de la bissectrice de l'angle formé par les droites orientées `l1` et `l2` par la rotation de centre « l'intersection de `l1` et `l2` » et d'angle `angle`.
     Si le paramètre `sharp` vaut `true`, renvoie la bissectrice de l'angle aigu.

     `line sector(int n=2, int p=1, line l1, line l2, real angle=0, bool sharp=true)`
     Renvoie l'image de la p-iéme droite qui partage l'angle formé par les droites orientées `l1` et `l2` en `n` parties égales par la rotation de centre « l'intersection de `l1` et `l2` » et d'angle `angle`.
     Si le paramètre `sharp` vaut `true`, considère l'angle aigu.

     `line perpendicular(point M, line l)`
     Renvoie la droite perpendiculaire à `l` passant par `M`.

     `line perpendicular(point M, explicit vector normal)`
     Renvoie la droite passant par `M` et de vecteur normal `normal`.

`line perpendicular(point M, explicit pair normal)`
Renvoie la droite passant par `M` et de vecteur normal `normal` donné dans le repère courant `currentcoordsys`.

`line reverse(line l)`
Renvoie la droite représentée par `l` avec une orientation contraire à celle de `l`.

`line extend(line l)`
Renvoie la droite portée par `l` qui peut être une demi-droite ou un segment de droite.

`line complementary(explicit line l)`
Renvoie la demi-droite complémentaire de `l`. Ne fonctionne que si `l` représente effectivement une demi-droite.

- `segment` :

  `segment segment(point A, point B)`

  On notera notamment la routine :

  `line bisector(segment s, real angle=0)`
  qui renvoie l'image de la médiatrice de `s` par la rotation de centre « le milieu de `s` » et d'angle `angle`.

  Ainsi que :

  `line[] complementary(explicit segment s)`
  qui renvoie sous forme de tableau les deux demi-droites de support `s` et d'extrémités respectives `s.A` et `s.B`.

- `conic` : pour une conique quelconque non dégénérée. Les types dérivés sont : `circle`, `ellipse`, `parabola` et `hyperbola`.

  De nombreuses routines permettent de définir un cercle. Entre autres :

  `circle circle(explicit point C, real r)`
  Renvoie le cercle de centre `C` et de rayon `r`.

  `circle circle(point A, point B)`
  Renvoie le cercle de diamètre `AB`.

  `circle circle(point A, point B, point C)`
  Renvoie le cercle passant par les points distincts `A`, `B` et `C`.

  Etc. (Voir la documentation)

- `arc` : pour les arcs orientés d'ellipses.

  ```
  arc arc(ellipse el, real angle1, real angle2,
          polarconicroutine polarconicroutine=polarconicroutine(el), bool direction=CCW)
  ```

- `abscissa` : pour instancier une abscisse sur un objet de type `line`, `segment`, `conic` et `arc`.

- `triangle` : structure assez complexe bénéficiant de nombreuses routines. Encore une fois, le minimum sera vu ici, la lecture de la documentation est **essentielle**.

```
struct triangle {
  restricted point A, B, C;       // points marquant les sommets du triangle

  struct vertex {                 // sommet de triangle
    int n;
    triangle t; }

  restricted vertex VA, VB, VC;   // les sommets du triangle

  struct side {                   // côté de triangle
    int n;
    triangle t; }

  side AB, BC, CA, BA, AC, CB; } // les côtés du triangle
```

```
void label(picture pic=currentpicture, Label LA="$A$",
           Label LB="$B$", Label LC="$C$",
           triangle t,
           real alignAngle=0,
           real alignFactor=1,
           pen p=nullpen, filltype filltype=NoFill)
```

Place les labels LA, LB et LC aux sommets du triangle t, alignés suivant la première bissectrice du sommet correspondant.

Les paramètres alignAngle et alignFactor permettent de modifier la direction et la longueur de l'alignement.

```
void show(picture pic=currentpicture,
          Label LA="$A$", Label LB="$B$", Label LC="$C$",
          Label La="$a$", Label Lb="$b$", Label Lc="$c$",
          triangle t, pen p=currentpen, filltype filltype=NoFill)
```

Trace le triangle t et affiche les labels aux sommets du triangle ainsi que les longueurs de ses côtés. Cette routine est surtout utile pour localiser les sommets t.A, t.B et t.C en cours de codage.

```
void draw(picture pic=currentpicture, triangle t,
          pen p=currentpen, marker marker=nomarker)
```

Trace le triangle t ; les côtés sont tracés comme des segments.

```
void drawline(picture pic=currentpicture, triangle t, pen p=currentpen)
```

Trace le triangle t ; les côtés sont tracés comme des droites.

```
triangle triangle(point A, point B, point C)
```

Renvoie le triangle dont les sommets sont A, B et C.

```
triangle triangleabc(real a, real b, real c, real angle=0, point A=(0,0))
```

Retourne le triangle ABC tel que BC=a, AC=b, AB=c et $\left(\vec{\imath}\,;\overrightarrow{AB}\right)$=angle.

```
triangle triangleAbc(real alpha, real b, real c, real angle=0, point A=(0,0))
```

Retourne le triangle ABC tel que $\left(\overrightarrow{AB};\overrightarrow{AC}\right)$=alpha, AC=b, AB=c et $\left(\vec{\imath}\,;\overrightarrow{AB}\right)$=angle.

```
triangle triangle(line l1, line l2, line l3)
```

Renvoie le triangle dont les côtés sont l1, l2 et l3.

Et de nombreuses autres routines permettant de tracer l'orthocentre, les hauteurs, le centre de gravité, les médianes, etc.

- trilinear : instancie des coordonnées trilinéaires relatives à un triangle (*sic*).
- inversion : permet d'instancier l'inversion de pôle C et de puissance k.

```
struct inversion
{
  point C;
  real k;
}
```

2. De nouvelles transformations sont définies :

```
transform scale(real k, point M)
```

Homothétie de centre M et de rapport k.

```
transform scale(real k, line l1, line l2, bool safe=false)
```

Renvoie l'affinité de rapport k, d'axe l1 et de direction l2.

Si safe vaut true et l1 parallèle à l2, la routine renvoie l'identité.

```
transform projection(point A, point B)
```

Projection orthogonale sur la droite (AB).

`transform projection(line l)`

Renvoie la projection orthogonale sur `l`.

`transform projection(point A, point B, point C, point D, bool safe=false)`

Projection sur la droite (AB) parallèlement à (CD).
Si `safe` vaut `true` et (AB) est parallèle à (CD), l'identité est renvoyée.
Si `safe` vaut `false` et (AB) est parallèle à (CD), l'homothétie de centre `O` et de rapport infini est renvoyée.

`transform projection(line l1, line l2, bool safe=false)`

Renvoie la projection sur `l1` parallèlement à `l2`.
Si `safe` vaut `true` et `l1` parallèle à `l2`, la routine renvoie l'identité.

`transform vprojection(line l, bool safe=false)`

Renvoie la projection sur `l` parallèlement à la verticale.
Si `safe` vaut `true` et `l` est une droite verticale, la routine renvoie l'identité.

`transform hprojection(line l, bool safe=false)`

Renvoie la projection sur `l` parallèlement à l'horizontale.
Si `safe` vaut `true` et `l` est une droite horizontale, la routine renvoie l'identité.

`transform scale(real k, point A, point B, point C, point D, bool safe=false)`

Affinité de rapport `k`, d'axe (AB) et de direction (CD). Si `safe` vaut `true` et (AB) est parallèle à (CD), l'identité est renvoyée.
Si `safe` vaut `false` et (AB) est parallèle à (CD), l'homothétie de centre `O` et de rapport infini est renvoyée.

`transform xscale(real k, point M)`

Affinité de rapport `k`, d'axe « l'axe passant par `M` et parallèle à `(Oy)` » et de direction `(Ox)`.

`transform yscale(real k, point M)`

Affinité de rapport `k`, d'axe « l'axe passant par `M` et parallèle à `(Ox)` » et de direction `(Oy)`.

`transform scale0(real x)`

Homothétie de rapport `x` et de centre « l'origine du repère courant ». Cette transformation est identique à `scale(x, origin())`.

`transform xscale0(real x)`

Identique à `xscale(x, origin())`.

`transform yscale0(real x)`

Identique à `yscale(x, origin())`.

`transform rotate0(real angle)`

Identique à `rotate(angle, origin())`.

`transform reflect(line l)`

Renvoie la réflexion par rapport à `l`.

3. `geometry` introduit aussi des fonctions de marquage :

```
void distance(picture pic=currentpicture, Label L="", point A, point B,
              bool rotated=true, real offset=3mm,
              pen p=currentpen, pen joinpen=invisible,
              arrowbar arrow=Arrows(NoFill))

void markrightangle(picture pic=currentpicture, point A, point O,
                    point B, real size=0, pen p=currentpen,
                    margin margin=NoMargin,
                    filltype filltype=NoFill)

void perpendicularmark(picture pic=currentpicture, point z,
                       explicit pair align,
                       explicit pair dir=E, real size=0,
                       pen p=currentpen,
                       margin margin=NoMargin,
                       filltype filltype=NoFill)

void perpendicularmark(picture pic=currentpicture, point z,
                       vector align,
                       vector dir=E, real size=0,
                       pen p=currentpen,
                       margin margin=NoMargin,
                       filltype filltype=NoFill)

void perpendicularmark(picture pic=currentpicture, point z, explicit pair align, path g,
                       real size=0, pen p=currentpen,
                       margin margin=NoMargin,
                       filltype filltype=NoFill)

void perpendicularmark(picture pic=currentpicture, point z, vector align, path g,
                       real size=0, pen p=currentpen,
                       margin margin=NoMargin,
                       filltype filltype=NoFill)

path compassmark(pair O, pair A, real position, real angle=10)
```

Pour `compassmark`, voir aussi page 52.

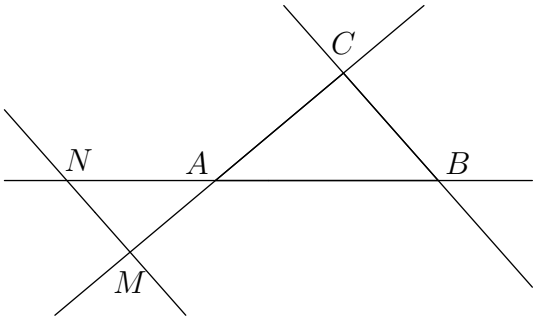
À noter aussi la routine :

```
void addMargins(picture pic=currentpicture,
                real lmargin=0, real bmargin=0,
                real rmargin=lmargin, real tmargin=bmargin,
                bool rigid=true, bool allObject=true)
```

Pour le tracé des droites, ajoute des marges à l'image (remplace avantageusement `interp`, voir page 12).

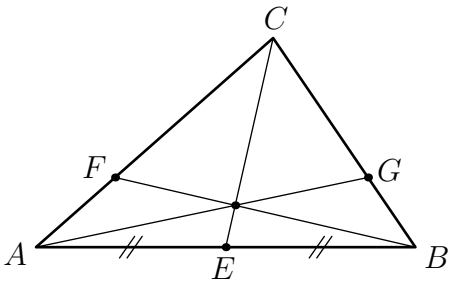
Si `rigid` vaut `false`, les marges sont ajoutées si et seulement si une courbe infinie se prolonge jusqu'à la marge.
Si `allObject` vaut `false`, les objets de taille fixe (comme les labels et pointes de flèches) seront ignorés.

Quelques exemples :



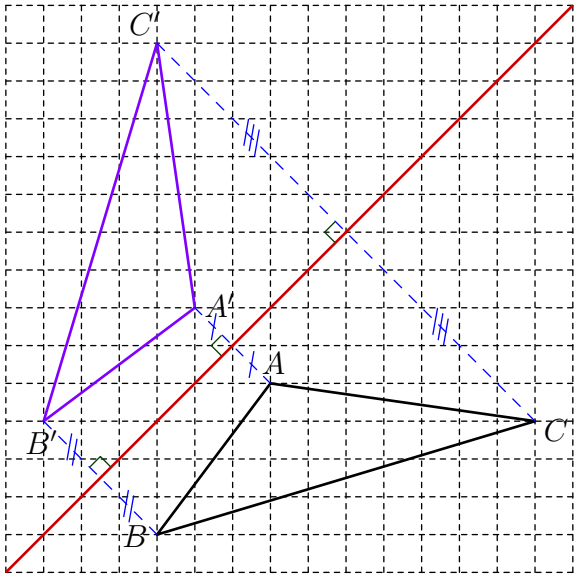
CODE 47

```
import geometry;
size(7cm);

real ac=3, am=2, coef=am/ac;
triangle t=triangleAbc(40,ac,4);
drawline(t); label("$A$",t.A,NW);
label("$B$",t.B,NE); label("$C$",t.C,2N);
point M=relpoint(line(t.AC),-coef);
line par=parallel(M,t.BC);
draw(par);
point pN=intersectionpoint(par,t.AB);
label("$M$",M,2*S);
label("$N$",pN,N+.5*E);

addMargins(1cm,.5cm);
```



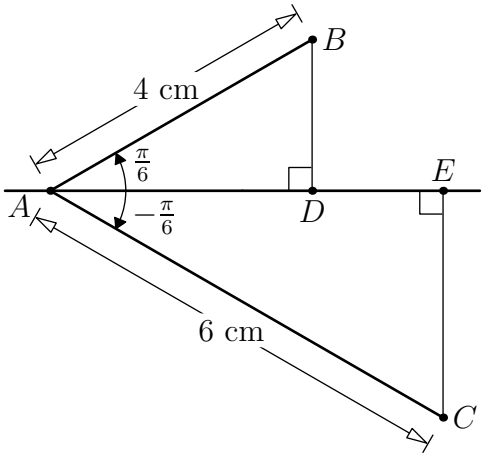
CODE 48

```
import geometry;
size(6cm);

triangle t=triangleabc(4,5,6);
label(t); draw(t,linewidth(bp));

point pE=midpoint(t.AB),
      pF=t.A+(1/3)*(t.C-t.A),
      pG=t.B+(1/3)*(t.C-t.B);
dot("$E$",pE,-dir(t.VC));
dot("$F$",pF,-dir(t.VB));
dot("$G$",pG,-dir(t.VA));

draw(segment(t.AB),
     StickIntervalMarker(2,2,angle=-35));
draw(t.C--pE^^t.B--pF^^t.A--pG);

line CE=line(t.C,pE);
line AG=line(t.A,pG);
dot(intersectionpoint(CE,AG));
```



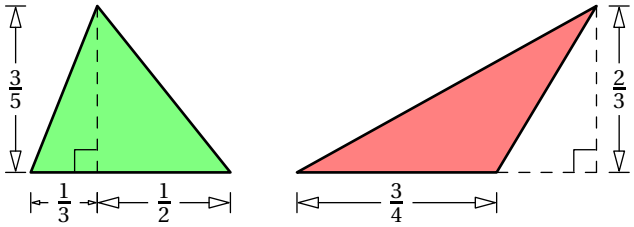
CODE 49

```
import geometry;
unitsize(.5cm);
add(shift(-6,-6)*grid(15,15,linetype("4 4")));
point A=(1,-1), B=(-2,-5), C=(8,-2);
line axe=line(origin(),45);
triangle t=triangle(A,B,C);
triangle t2=reflect(axe)*t;

draw(axe,bp+.8*red);
draw(t,linewidth(bp));
draw(t2,bp+purple);
label(t,alignAngle=-30);
label(LA="$A'$",LB="$B'$",LC="$C'$",
      t2,alignAngle=30);

pen db=dashed+blue;
draw(t.A--t2.A,db,
     StickIntervalMarker(2,1,angle=35,blue));
draw(t.B--t2.B,db,
     StickIntervalMarker(2,2,angle=35,blue));
draw(t.C--t2.C,db,
     StickIntervalMarker(2,3,angle=35,blue));

transform paxe=projection(axe);
point v=relpoint(axe,-1);
markrightangle(t2.C,paxe*t2.C,v,2mm,darkgreen);
markrightangle(t2.A,paxe*t2.A,v,2mm,darkgreen);
markrightangle(t2.B,paxe*t2.B,v,2mm,darkgreen);
shipout(bbox(1mm,invisible));
```



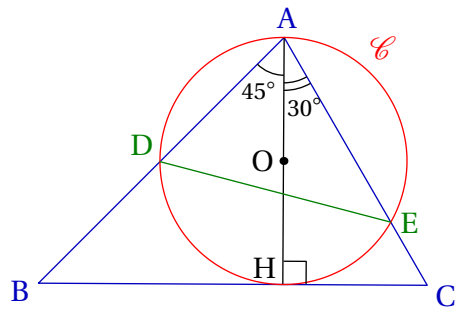
CODE 50

```
import geometry;
unitsize(1cm);

point A=(0,0);
point B=relpoint(line(30,A),4);
point C=relpoint(line(-30,A),6);
point D=(B.x,0), pE=(C.x,0);

draw(A--B^^A--C,linewidth(bp));
draw(B--D^^pE--C);
draw(Ox,linewidth(bp));

markrightangle(B,D,A);
markrightangle(A,pE,C);
markangle("$\frac\pi6$",D,A,B,ArcArrow);
markangle("$-\frac\pi6$",C,A,D,BeginArcArrow);
distance("$4$~cm",A,B,rotated=false,-4mm);
distance("$6$~cm",A,C,rotated=false,4mm);

dot("$A$",A,S+2W); dot("$B$",B); dot("$C$",C);
dot("$D$",D,S); dot("$E$",pE,N);
```



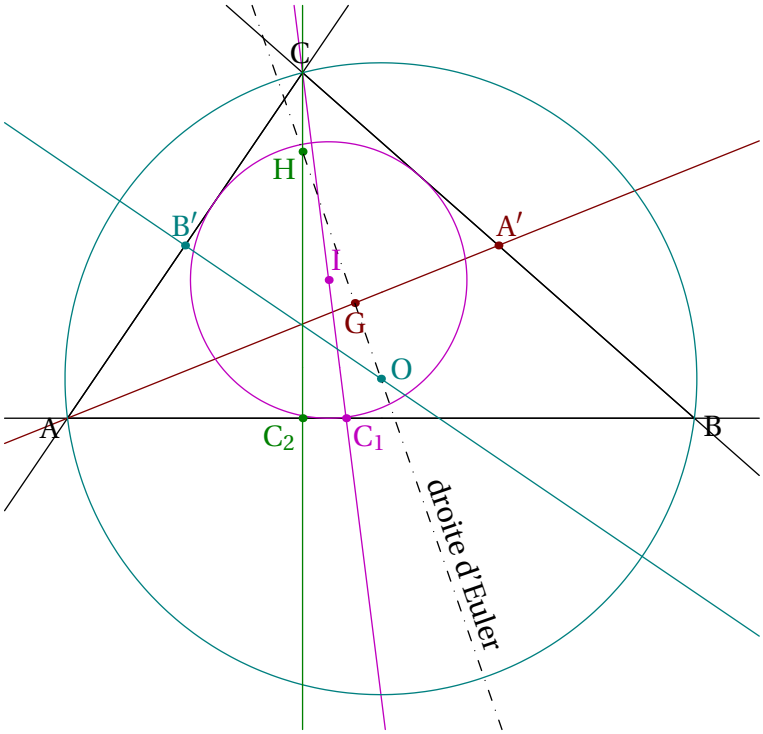
CODE 51

```
size(8cm);
import geometry;
usepackage("fourier");

void filldrawtri(triangle t,
     pen fpen=currentpen, pen dpen=currentpen)
{
filldraw(t.A--t.B--t.C--cycle,fpen,dpen);
}

point A=(0,0),B=(3,0),C=(1,2.5),pI=(A.x,C.y);
point D=(4,0),pE=(7,0),G=(8.5,2.5),F=(G.x,0);
point H=projection(A,B)*C;
triangle t1=triangle(A,B,C);
triangle t2=triangle(D,pE,G);

filldrawtri(t1,lightgreen,linewidth(bp));
filldrawtri(t2,lightred,linewidth(bp));

draw(C--H^^pE--F^^G--F,dashed);
markrightangle(C,H,A,3mm);
markrightangle(G,F,D,3mm);
distance("$\frac{1}{3}$",A,H,4mm,
         Arrows(1mm,NoFill));
distance("$\frac{1}{2}$",H,B,4mm);
distance("$\frac{3}{5}$",A,pI,
         rotated=false,-2mm);
distance("$\frac{3}{4}$",D,pE,4mm);
distance("$\frac{2}{3}$",F,G,rotated=false);
```



CODE 52

```
import geometry;
usepackage("fourier","upright");
usepackage("mathrsfs");
size(6cm);

triangle t=triangleAbc(75,7,8.5,225);
point H=foot(t.VA);
circle c=circle(t.A,H);
point[] ct=intersectionpoints(t,c);

draw(t,heavyblue);
draw(t.A--H);
draw(c,red);
draw(ct[0]--ct[3],deepgreen);

label(t,heavyblue);dot("$O$",c.C,W);
label("$\mathscr{C}$",angpoint(c,50),NE,red);
label("$H$",H,NW);
label("$D$",ct[0],NW,deepgreen);
label("$E$",ct[3],E,deepgreen);
markrightangle(t.A,H,t.C);
markangle(scale(.8)*"$45^\circ$",5mm,
          t.B,t.A,H);
markangle(scale(.8)*"$30^\circ$",n=2,6mm,
          H,t.A,t.C);
```



CODE 53

```
import geometry;
usepackage("fourier","upright"); usepackage("mathrsfs");
size(10cm);

pen mediane=brown, bissec=heavymagenta, mediat=deepcyan, ortho=deepgreen;
// ------------------------------------------------------------------------
triangle t=triangleabc(5,4,6); drawline(t); label(t);
// ------------------------------------------------------------------------
line med=median(t.VA); point A1=midpoint(t.BC); point G=centroid(t);
draw(med,mediane); dot("$A'$",A1,N+.5*E,mediane); dot("$G$",G,S,mediane);
// ------------------------------------------------------------------------
line biss=bisector(t.VC); point Bi=bisectorpoint(t.AB);
point Ci=incenter(t); circle ci=incircle(t);
draw(biss,bissec); draw(ci,bissec);
dot("$C_1$",Bi,SE,bissec); dot("$I$",Ci,N+.5*E,bissec);
// ------------------------------------------------------------------------
line mediatrice=bisector(t.AC); point B1=midpoint(t.AC);
point Cc=circumcenter(t); circle cc=circle(t);
draw(mediatrice,mediat); draw(cc,mediat);
dot("$B'$",B1,N,mediat); dot("$O$",Cc,E+.5*N,mediat);
// ------------------------------------------------------------------------
line haut=altitude(t.VC); point C2=foot(t.VC);
point orth=orthocentercenter(t);
draw(haut,ortho); dot("$C_2$",C2,SW,ortho); dot("$H$",orth,SW,ortho);
// ------------------------------------------------------------------------
draw(line(orth,Cc),dashdotted);
label(Label("droite d'Euler",Rotate(dir(orth--Cc)),align=RightSide),
      relpoint(line(orth,Cc),1.7));
// ------------------------------------------------------------------------
addMargins(.5cm,.5cm);
```

3) trembling

Ce module permet de faire des figures « à main levée ». Plusieurs paramètres permettent de modifier l'effet :

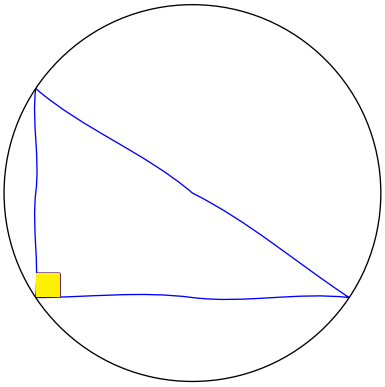
```
real magneticRadius=1;
real trembleFuzz(){return min(1e-3,magneticRadius/10);}
real trembleAngle=4, trembleFrequency=0.5, trembleRandom=2;
```

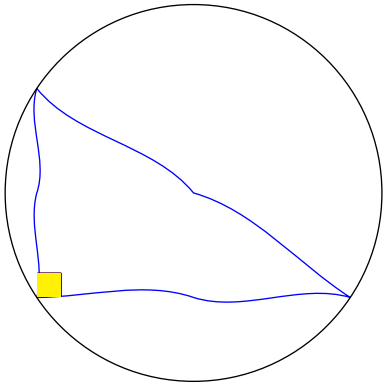
Pour saisir leur rôle, le plus simple est de faire des essais. Des explications sont données directement dans le fichier `trembling.asy`.

Il faut utiliser la structure `tremble`, ainsi que la *méthode* `deform` :

```
tremble tremble(real angle=trembleAngle, real frequency=trembleFrequency,
                real random=trembleRandom, real fuzz=trembleFuzz())

path deform(path g...pair[] magneticPoints)
```

Il faudra donc impérativement un type `path` ...





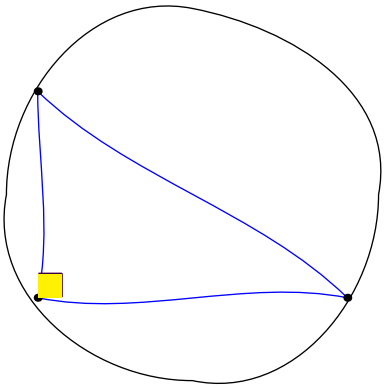
CODE 54

```
import trembling;
import geometry;
size(5cm,0);
pair A=(0,0), B=(3,0), C=(0,2),
O=midpoint(B--C);
path t=B--A--C--cycle;
// paramètres par défaut
tremble tr=tremble(angle=4,
                   frequency=.5,
                   random=2);
draw(tr.deform(t),blue);
path c=circle(O,sqrt(13)/2);
draw(c);
markrightangle(B,A,C,bp+darkmagenta,
               filltype=Fill(Yellow));
```

CODE 55

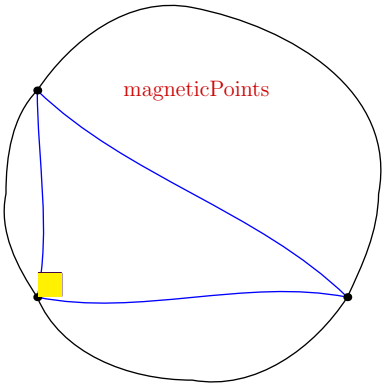
```
import trembling;
import geometry;
size(5cm,0);
pair A=(0,0), B=(3,0), C=(0,2),
O=midpoint(B--C);
path t=B--A--C--cycle;
tremble tr=tremble(angle=10,
                   frequency=.5,
                   random=2);
draw(tr.deform(t),blue);
path c=circle(O,sqrt(13)/2);
draw(c);
markrightangle(B,A,C,bp+darkmagenta,
               filltype=Fill(Yellow));
```

Les points qui doivent être « magnétisés » sont à donner en paramètre à `deform` :



CODE 56

```
import trembling;
import geometry;
tremble tr=tremble(angle=6,
                   frequency=1,
                   random=5);
size(5cm,0);
pair A=(0,0), B=(3,0), C=(0,2),
     O=midpoint(B--C);
path t=B--A--C--cycle;
path c=circle(O,sqrt(13)/2);
draw(tr.deform(t),blue);
dot(A^^B^^C);
draw(tr.deform(c));
markrightangle(B,A,C,bp+darkmagenta,
               filltype=Fill(Yellow));
```



CODE 57

```
import trembling;
import geometry;
tremble tr=tremble(angle=6,
                   frequency=1,
                   random=5);
size(5cm,0);
pair A=(0,0), B=(3,0), C=(0,2),
     O=midpoint(B--C);
path t=B--A--C--cycle;
path c=circle(O,sqrt(13)/2);
draw(tr.deform(t),blue);
dot(A^^B^^C);
// A, B et C "magnétisés" :
draw(tr.deform(c,A,B,C));
markrightangle(B,A,C,bp+darkmagenta,
               filltype=Fill(Yellow));
label(scale(.7)*"magneticPoints",
      C,9E,.8red);
```

4) graph

La documentation officielle est très complète sur ce module. Ce qui suit est un résumé de ce qui sera le plus utile dans un premier temps.

a) Axes et repères

graph fournit les routines suivantes :

```
void xaxis(picture pic=currentpicture, Label L="", axis axis=YZero,
           real xmin=-infinity, real xmax=infinity, pen p=currentpen,
           ticks ticks=NoTicks, arrowbar arrow=None, bool above=false);
```

Dessine l'axe des abscisses. Certains paramètres méritent de s'y attarder :

- `axis` : détermine la position de l'axe. Les différents types sont :
  - `YZero(bool extend=true)` : l'axe a pour équation $y = 0$ (ou $y = 1$ pour une échelle logarithmique). `extend=true` (valeur par défaut) pour que l'axe soit aux dimensions complètes de la figure.
  - `YEquals(real Y, bool extend=true)` : l'axe a pour équation $y = Y$. (Voir plutôt yequals.)
  - `Bottom(bool extend=false)` : l'axe est au bas de la figure.
  - `Top(bool extend=false)` : l'axe est en haut.
  - `BottomTop(bool extend=false)` : axes en bas et en haut.
- `xmin` et `xmax` : automatiquement déterminés par les dimensions de la figure si non spécifiés.
- `ticks` : les traits de graduation. Les valeurs sont `NoTicks` (par défaut, aucune graduation), `LeftTicks` (graduations uniquement à gauche de l'axe), `RightTicks` (graduations uniquement à droite de l'axe) et `Ticks` (graduations des deux cotés de l'axe). Ces trois dernières routines prennent elles-mêmes de nombreux arguments en option :

```
ticks Ticks(Label format="", ticklabel ticklabel=null,
            bool beginlabel=true, bool endlabel=true,
            int N=0, int n=0, real Step=0, real step=0,
            bool begin=true, bool end=true, tickmodifier modify=None,
            real Size=0, real size=0, bool extend=false,
            pen pTick=nullpen, pen ptick=nullpen);
```

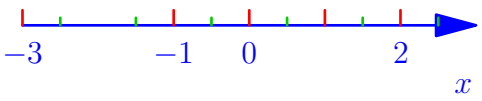
  - `format` : format des nombres sur les axes (par défaut `"$%.4g$"`, voir page 15). Pour n'avoir que les traits, sans les labels, la valeur est `"%"`.
  - `ticklabel` : fonction `string(real x)` qui retourne le label (par défaut `format(format.s,x)`) de chaque graduation principale d'abscisse x. Voir par exemple labelfrac page 41 et le code 73.
    le `ticklabel OmitFormat(string s=defaultformat ... real[] x)` permet d'enlever certains labels, mais pas la graduation correspondante.
    `NoZeroFormat` est une abréviation pour `OmitFormat(0)`.
  - `beginlabel` et `endlabel` : inclut le premier et le dernier label.
  - `N` : Quand la mise à l'échelle est activée (par défaut), l'axe est divisé en `N` intervalles séparés par les graduations principales.
  - `n` : Chaque intervalle principal est divisé en `n` intervalles secondaires séparés par les graduations secondaires.
  - `Step` : la valeur entre chaque trait de graduation principale (si `N=0`).
  - `step` : la valeur entre chaque trait de graduation secondaire (si `n=0`).
  - `begin` et `end` : inclut le premier et le dernier trait de graduation principale.

— `tickmodifier modify` : fonction permettant de modifier des graduations « manuellement ».
Certains `tickmodifier` sont prédéfinis :
`OmitTick(... real[] x)`, `OmitTickInterval(real a, real b)` et
`OmitTickIntervals(real[] a, real[] b)` peuvent servir à enlever des graduations, ainsi que leurs labels.
`NoZero` est une abréviation pour `OmitTick(0)`.

— `Size` et `size` : taille des graduations principales et secondaires.

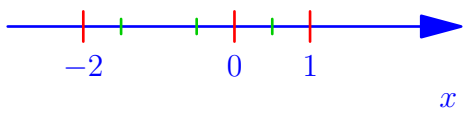
— `extend` : étend les graduations. Permet de tracer des grilles.

— `pen pTick` et `ptick` : stylos pour le tracé des traits de graduation.



CODE 58

```
import graph;
unitsize(1cm);
xaxis(L="$x$", axis=YZero,
      xmin=-3, xmax=3, p=bp+blue,
      ticks=LeftTicks(ticklabel=OmitFormat(1),
                endlabel=false,Step=1,step=.5,
                end=false,modify=OmitTick(-2),
                pTick=bp+red,ptick=bp+.8green),
      arrow=Arrow);
```



CODE 59

```
import graph;
unitsize(1cm);
real[] tabT={-2,0,1}, tabt={-1.5,-.5,.5};
xaxis(L="$x$", axis=YZero(extend=true),
      xmin=-3, xmax=3, p=bp+blue,
      ticks=Ticks(Ticks=tabT, ticks=tabt,
              pTick=bp+red,ptick=bp+.8green),
      arrow=Arrow, above=false);
```

Il est aussi possible de spécifier l'emplacement des graduations à l'aide d'un tableau de réels à l'aide de la routine (voir le code 59, valable aussi pour `LeftTicks` et `RightTicks`) :

```
ticks Ticks(Label format="", ticklabel ticklabel=null,
            bool beginlabel=true, bool endlabel=true,
            real[] Ticks, real[] ticks=new real[],
            real Size=0, real size=0, bool extend=false,
            pen pTick=nullpen, pen ptick=nullpen)
```

De la même façon qu'`xaxis` est défini :

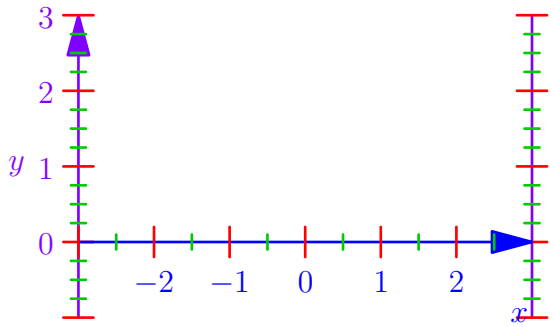
```
void yaxis(picture pic=currentpicture, Label L="", axis axis=XZero,
           real ymin=-infinity, real ymax=infinity, pen p=currentpen,
           ticks ticks=NoTicks, arrowbar arrow=None, bool above=false,
           bool autorotate=true);
```

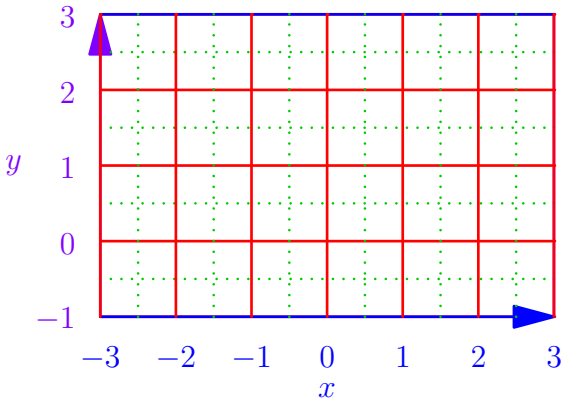
Les paramètres sont similaires à ceux d'`xaxis`. On notera pour le type `axis` les valeurs `Left`, `Right` et `LeftRight`.

Il est possible de fixer les valeurs `xmin`, `xmax`, `ymin` et `ymax` par la routine :

```
xlimits(picture pic=currentpicture, real min=-infinity,
        real max=infinity, bool crop=NoCrop);
```

et la routine `ylimits`.
Le booléen `crop=Crop` sert à couper les parties de la figure qui dépassent les limites.





CODE 60

```
import graph;
unitsize(1cm);
xaxis(L="$x$", axis=YZero,
      xmin=-3, xmax=3, p=bp+blue,
      ticks=Ticks(endlabel=false,
                  beginlabel=false,
                  Step=1,step=.5,
                  end=false,pTick=bp+red,
                  ptick=bp+.8green),
      arrow=Arrow);

yaxis(L="$y$", axis=LeftRight,
      ymin=-1, ymax=3, p=bp+purple,
      ticks=Ticks(beginlabel=false,
                  Step=1,step=.25,
                  pTick=bp+red,
                  ptick=bp+.8green),
      arrow=Arrow);
```

CODE 61

```
import graph;
unitsize(1cm);

xlimits(-3, 3);
ylimits(-1, 3);
xaxis(L="$x$", axis=BottomTop,p=bp+blue,
      ticks=Ticks(Step=1,step=.5,extend=true,
                  pTick=bp+red,
                  ptick=bp+dotted+.8green),
      arrow=Arrow);

yaxis(L="$y$", axis=LeftRight,p=bp+purple,
      ticks=Ticks(Step=1,step=.5,extend=true,
                  pTick=bp+red,
                  ptick=bp+dotted+.8green),
      arrow=Arrow);
```

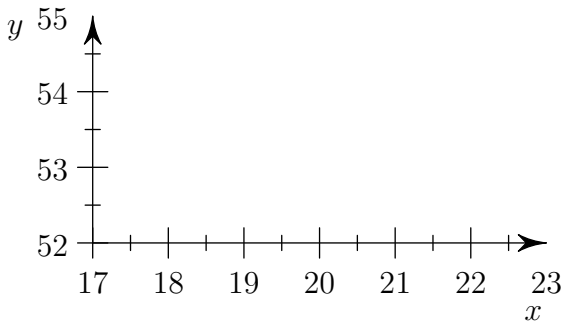
Pour des axes déportés, on peut utiliser les routines suivantes :

```
void xequals(picture pic=currentpicture, Label L="", real x,
             bool extend=false, real ymin=-infinity, real ymax=infinity,
             pen p=currentpen, ticks ticks=NoTicks, bool above=true,
             arrowbar arrow=None);


void yequals(picture pic=currentpicture, Label L="", real y,
             bool extend=false, real xmin=-infinity, real xmax=infinity,
             pen p=currentpen, ticks ticks=NoTicks, bool above=true,
             arrowbar arrow=None);
```

et

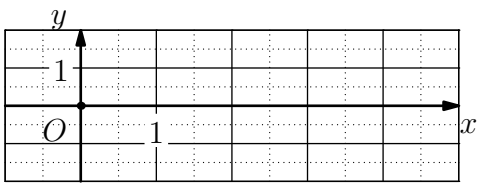
```
void axes(picture pic=currentpicture, Label xlabel="", Label ylabel="",
          pair min=(-infinity,-infinity), pair max=(infinity,infinity),
          pen p=currentpen, arrowbar arrow=None, bool above=false);
```

Voir le code 65. Dans cet exemple, les axes sont tracés dans une image séparée, qui est ajoutée à `currentpicture` à la fin.



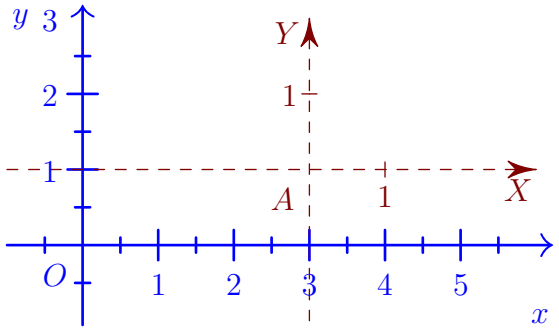
CODE 62

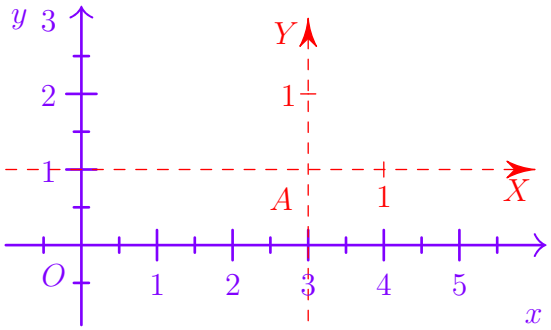
```
import graph;
unitsize(1cm);
xlimits(17,23); ylimits(52,55);
xequals(pic=currentpicture,L="$y$",x=17,
        ticks=Ticks(Step=1,step=.5,end=false),
        arrow=Arrow(HookHead));
yequals(pic=currentpicture,L="$x$",y=52,
        ticks=Ticks(Step=1,step=.5,end=false),
        arrow=Arrow(HookHead));
```



CODE 63

```
import graph;
unitsize(x=1cm, y=.5cm);
xlimits(-1, 5); ylimits(-2, 2);
xaxis(BottomTop, Ticks("%",extend=true,
      pTick=linewidth(.5pt), ptick=dotted));
yaxis(LeftRight, Ticks("%",extend=true,
      pTick=linewidth(.5pt), ptick=dotted));
xequals(Label("$y$",align=2NW), 0,
        p=linewidth(bp), Arrow(2mm));
yequals(Label("$x$",align=2SE), 0,
        p=linewidth(bp), Arrow(2mm));
labelx(Label("$1$",UnFill), 1);
labely(Label("$1$",UnFill), 1);
dot("$O$",(0,0),2SW);
```





CODE 64

```
import graph;
unitsize(1cm);
xlimits(-1,6); ylimits(-1,3);
xaxis(L="$x$", p=bp+blue,
      ticks=Ticks(NoZero,endlabel=false,
                  beginlabel=false,Step=1,step=.5,
                  begin=false,end=false),
      arrow=Arrow(TeXHead));
yaxis(L="$y$", p=bp+blue,
      ticks=Ticks(NoZero,beginlabel=false,
                  Step=1,step=.5,
                  begin=false,end=false),
      arrow=Arrow(TeXHead));
labelx("$O$",0,2*SW,blue);

xequals(pic=currentpicture,L="$Y$",x=3,
        p=brown+dashed,ticks=NoTicks,
        arrow=Arrow(HookHead));
yequals(pic=currentpicture,L="$X$",y=1,
        p=brown+dashed,ticks=NoTicks,
        arrow=Arrow(HookHead));
labelx("$A$",(3,1),2*SW,brown);
xtick("$1$",(4,1),size=1mm,brown);
xtick(dir=S,(4,1),size=1mm,brown);
ytick("$1$",(3,2),size=1mm,brown);
ytick(dir=W,(3,2),size=1mm,brown);
```

CODE 65

```
import graph;
unitsize(1cm);
xlimits(-1,6); ylimits(-1,3);
xaxis(L="$x$", p=bp+purple,
      ticks=Ticks(NoZero,endlabel=false,
                  beginlabel=false,Step=1,step=.5,
                  begin=false,end=false),
      arrow=Arrow(TeXHead));
yaxis(L="$y$", p=bp+purple,
      ticks=Ticks(NoZero,beginlabel=false,
                  Step=1,step=.5,
                  begin=false,end=false),
      arrow=Arrow(TeXHead));
labelx("$O$",0,2*SW,purple);

picture pic;
unitsize(pic,1cm);
axes(pic, xlabel="$X$", ylabel="$Y$",
     min=(-4,-2), max=(3,2),p=dashed+red,
     arrow=Arrow(HookHead));
labelx(pic,"$A$",(0,0),2*SW,red);
xtick(pic,"$1$",1,size=1mm,red);
xtick(pic,dir=S,1,size=1mm,red);
ytick(pic,"$1$",1,size=1mm,red);
ytick(pic,dir=W,1,size=1mm,red);
add(pic.fit(),(3,1));
```

Les routines suivantes permettent de placer manuellement des graduations (voir les exemples page précédente) :

```
void xtick(picture pic=currentpicture, Label L="", explicit pair z,
           pair dir=N, string format="",
           real size=Ticksize, pen p=currentpen);

void xtick(picture pic=currentpicture, Label L="", real x,
           pair dir=N, string format="",
           real size=Ticksize, pen p=currentpen);
```

De la même façon que `xtick` est défini `ytick`.

```
void tick(picture pic=currentpicture, pair z,
          pair dir, real size=Ticksize, pen p=currentpen);

void labelx(picture pic=currentpicture, Label L="", explicit pair z,
            align align=S, string format="", pen p=nullpen);

void labelx(picture pic=currentpicture, Label L="", real x,
            align align=S, string format="", pen p=nullpen);

void labelx(picture pic=currentpicture, Label L,
            string format="", explicit pen p=currentpen);
```

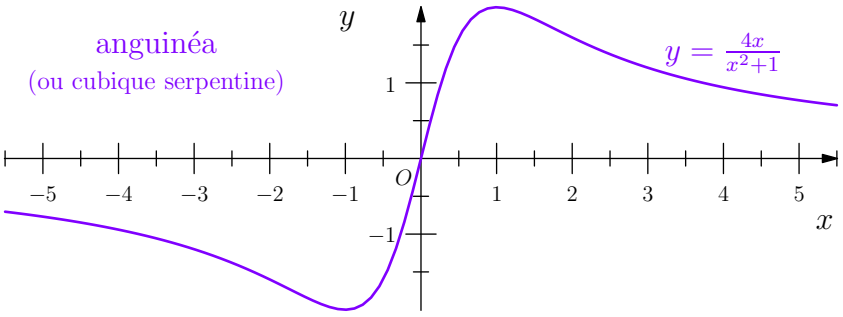
De la même façon que `labelx` est défini `labely`.

b) Représentations de fonctions

Plusieurs routines permettent de définir un graphe. Seules les plus utiles pour débuter seront vues (voir la documentation officielle pour les autres).

```
guide graph(picture pic=currentpicture, real f(real), real a, real b,
            int n=ngraph, real T(real)=identity,
            interpolate join=operator --);
```

Retourne le graphe de la fonction `f` sur l'intervalle [T(a);T(b)], basé sur `n` points (100 par défaut) régulièrement espacés dans [a;b].

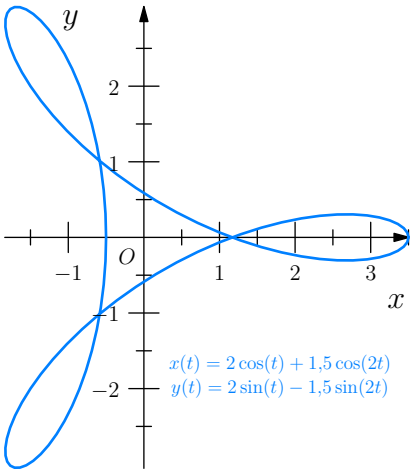


CODE 66

```
import graph;
unitsize(1cm);
xaxis("$x$", Ticks(scale(.7)*Label(), NoZero),Arrow(2mm));
yaxis("$y$", Ticks(scale(.7)*Label(), NoZero, beginlabel=false, endlabel=false,
                   begin=false, end=false), Arrow(2mm));
labelx(scale(.7)*"$0$",0,SW);
real f(real x) {return 4*x/(x^2+1);}
draw(graph(f,-5.5,5.5),bp+purple);
label("$y=\frac{4x}{x^2+1}$",(4,f(4)),1.5N,purple);
label("anguin\'ea",(-3.5,1.5),purple);
label(scale(.8)*"(ou cubique serpentine)",(-3.5,1),purple);
```

```
guide graph(picture pic=currentpicture, real x(real), real y(real),
            real a, real b, int n=ngraph, real T(real)=identity,
            interpolate join=operator --);
```

Retourne la courbe paramétrée $\big(x(t)\,;y(t)\big)$ pour t dans l'intervalle [T(a);T(b)].

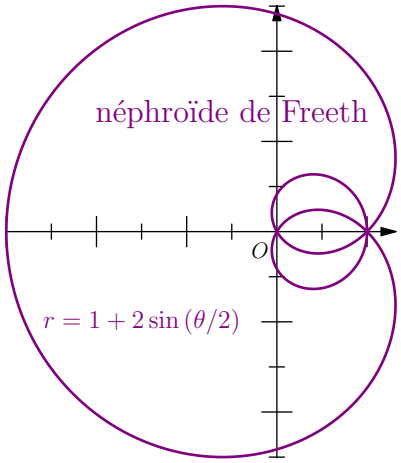


CODE 67

```
import graph;
unitsize(1cm);
usepackage("inputenc","utf8"); usepackage("icomma");
xaxis("$x$", Ticks(scale(.7)*Label(), NoZero),Arrow(2mm));
yaxis("$y$", Ticks(scale(.7)*Label(), NoZero, beginlabel=false, endlabel=false,
                   begin=false, end=false), Arrow(2mm));
labelx(scale(.7)*"$0$",0,SW);
real x(real t) {return 2*cos(t)+1.5*cos(2*t);}
real y(real t) {return 2*sin(t)-1.5*sin(2*t);}
pen pg=royalblue;
draw(graph(x,y,0,2*pi),bp+pg);
label(scale(.6)*"$x(t)=2\cos(t)+1,5\cos(2t)$",(1.8,-2),1.5N,pg);
label(scale(.6)*"$y(t)=2\sin(t)-1,5\sin(2t)$",(1.8,-2),pg);
```

```
guide polargraph(picture pic=currentpicture, real f(real), real a,
                 real b, int n=ngraph, interpolate join=operator --);
```

Retourne le graphe de la fonction `f` en coordonnées polaires sur l'intervalle [a;b].

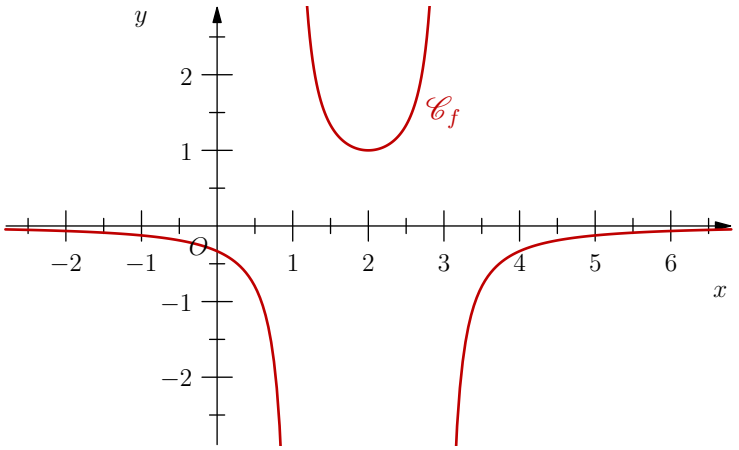


CODE 68

```
import graph;
size(6cm);
usepackage("inputenc","utf8");
xaxis(Ticks("%", NoZero, Step=1, step=.5), Arrow(2mm));
yaxis(Ticks("%", NoZero, Step=1, step=.5), Arrow(2mm));
labelx(scale(.7)*"$0$",0,SW);
real r(real t) {return 1+2*sin(t/2);}
pen pg=deepmagenta;
draw(polargraph(r,0,4*pi,n=400),bp+pg);
label(scale(.8)*"$r=1+2\sin\left(\theta/2\right)$",(-1.5,-1),pg);
label("néphroïde de Freeth",(-.5,1.3),pg);
```

Remarque : Fonctions non définies en certaines valeurs
Deux possibilités ici :

1. « enlever » les valeurs interdites : pour l'exemple, considérons la fonction $f : x \longmapsto \dfrac{-1}{x^2-4x+3}$.
   Cette fonction est définie sur $\mathbb{R} \setminus \{1;3\}$. La courbe sera découpée en trois graphes, `c1`, `c2` et `c3` qui éviteront soigneusement les valeurs 1 et 3.



CODE 69

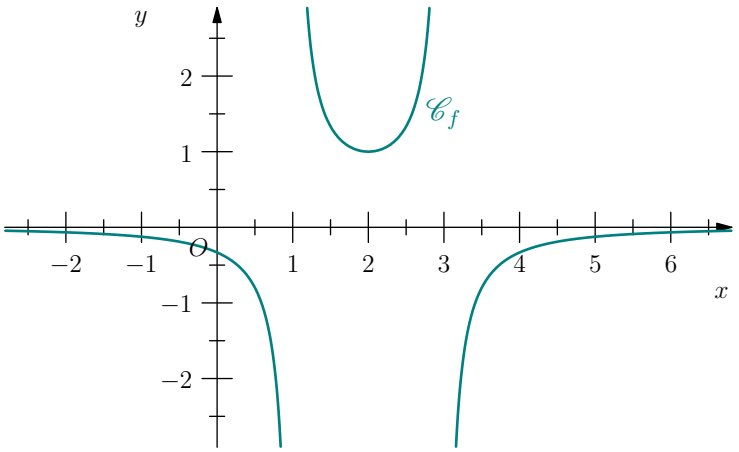
```
import graph;
import contour;
usepackage("mathrsfs");
unitsize(x=1cm,y=1cm);

transform ec=scale(.8);
xaxis(ec*"$x$", Ticks(ec*Label(), NoZero), Arrow(2mm));
yaxis(ec*"$y$", Ticks(ec*Label(), NoZero), Arrow(2mm));
labelx(ec*"$O$",0,SW);

real f(real x) {return -1/(x^2-4*x+3);}
path c1=graph(f,-2.8,.9); // on évite les valeurs
path c2=graph(f,1.1,2.9); // interdites
path c3=graph(f,3.1,6.8); //
draw(c1^^c2^^c3,bp+heavyred);
ylimits(-2.9,2.9,Crop); // on coupe ce qui dépasse
label("$\mathscr{C}_f$",(3,1.5),heavyred);
```

2. utiliser le module `contour` et tracer la courbe comme une ligne de niveau.
   On utilise la routine suivante :

```
guide[][] contour(real f(real, real), pair a, pair b,
                  real[] c, int nx=ngraph, int ny=nx,
                  interpolate join=operator--, int subsample=1)
```

   - `f` : fonction de deux variables réelles.
   - `a` et `b` : les sommets de la diagonale du domaine rectangulaire.
   - `c` : tableau contenant les valeurs des lignes de niveau.
   - `nx` et `ny` : nombre de subdivisions dans les directions x et y (détermine la précision).
   - `join` : l'opérateur d'interpolation (`--` ou `..`).
   - `subsample` : nombre de points intérieurs à inclure dans chaque carré de la grille (en plus des points sur le bord).

   Pour notre exemple, la courbe $\mathscr{C}_f$ a pour équation $y = \dfrac{-1}{x^2-4x+3}$. D'où l'on tire : $y \times \left(x^2-4x+3\right) = -1$.
   On considérera donc la fonction $\mathrm{F}(x,y) = y\left(x^2-4x+3\right)$ et on tracera la ligne de niveau -1.



CODE 70

```
import graph;
import contour;
usepackage("mathrsfs");
unitsize(x=1cm,y=1cm);

transform ec=scale(.8);
xaxis(ec*"$x$", Ticks(ec*Label(), NoZero), Arrow(2mm));
yaxis(ec*"$y$", Ticks(ec*Label(), NoZero), Arrow(2mm));
labelx(ec*"$O$",0,SW);

real F(real x, real y) {return y*(x^2-4*x+3);}
draw(contour(F,(-2.8,-2.9),(6.8,2.9),new real[] {-1}),bp+deepcyan);
label("$\mathscr{C}_f$",(3,1.5),deepcyan);
```

Voir aussi le code 71

c) Extension graph_pi

Cette extension de Philippe Ivaldi apporte bon nombre d'outils intéressants.
Elle peut être téléchargée à cette adresse : git.piprime.fr/?p=asymptote/pi-packages.git;a=tree, lien `snapshot`, et placée dans le dossier `$HOME/.asy`.
Pour une utilisation avec `git`, voir :
forum.mathematex.net/asymptote-f34/version-2-10-et-graph-pi-t13226.html#p127606.
Les modules `graph`, `markers` et `base_pi` (voir à l'adresse ci-dessus) sont chargés par `graph_pi`, ainsi que le package LaTeX `mathrsfs`.

Axes et repères :

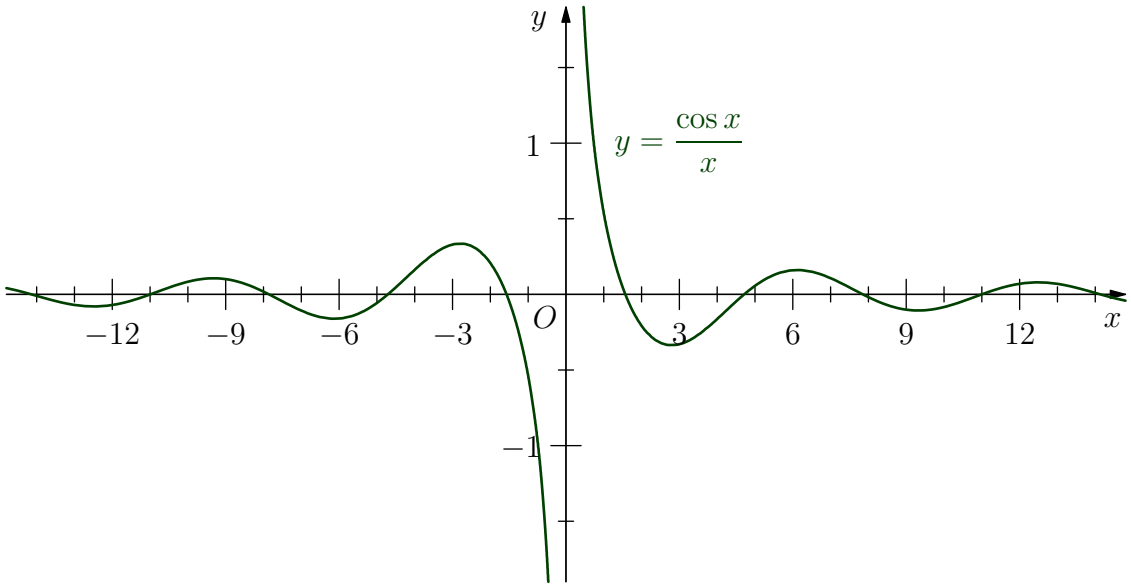
```
void graphicrules(picture pic=currentpicture, real unit=1cm,
                  real xunit=unit != 0 ? unit : 0,
                  real yunit=unit != 0 ? unit : 0,
                  real xmin=-infinity, real xmax=infinity,
                  real ymin=-infinity, real ymax=infinity,
                  bool crop=NoCrop, bool xcrop=crop, bool ycrop=crop)
```

Passe les dimensions du graphique à `cartesianaxis`, `grid`, et `millimeterpaper` (voir ci-après).

```
void cartesianaxis(picture pic=currentpicture,
                   Label Lx=Label("$x$",align=2S),
                   Label Ly=Label("$y$",align=2W),
                   real xmin=-infinity, real xmax=infinity,
```

```
real ymin=-infinity, real ymax=infinity,
real extrawidth=1, real extraheight=extrawidth,
pen p=currentpen,
ticks xticks=Ticks("%",pTick=nullpen, ptick=grey),
ticks yticks=Ticks("%",pTick=nullpen, ptick=grey),
bool viewxaxis=true,
bool viewyaxis=true,
bool above=true,
arrowbar arrow=Arrow)
```

Trace l'axe des abscisses, des ordonnées ou les deux (`viewxaxis` et `viewyaxis` qui prennent les valeurs `true` ou `false`). `extrawidth` est la longueur rajoutée à l'axe des abscisses par rapport à `xmin` et `xmax`. On a la même chose avec `extraheight` pour l'axe des ordonnées.



CODE 71

```
import graph_pi;
import contour;

graphicrules(xunit=.5cm, yunit=2cm, xmin=-14.8, xmax=14.8, ymin=-1.9, ymax=1.9);
cartesianaxis(extrawidth=0,xticks=Ticks(NoZero), yticks=Ticks(NoZero), Arrow(2mm));
labelx("$0$",0,SW);
real F(real x, real y) {return x*y-cos(x);}
draw(contour(F,(-14.8,-1.9),(14.8,1.9),new real[] {0}),bp+darkgreen);
label("$y=\displaystyle\frac{\cos x}{x}$",(3,1),darkgreen);
```

```
void labeloij(picture pic=currentpicture,
              Label Lo=Label("$O$",NoFill),
              Label Li=Label("$\overrightarrow{\imath}$",NoFill),
              Label Lj=Label("$\overrightarrow{\jmath}$",NoFill),
              pen p=scale(2)*currentpen,
              pair diro=SW, pair diri=labelijmargin*S, pair dirj=labelijmargin*1.5*W,
              filltype filltype=NoFill, arrowbar arrow=Arrow(2mm),
              marker marker=dot)
```

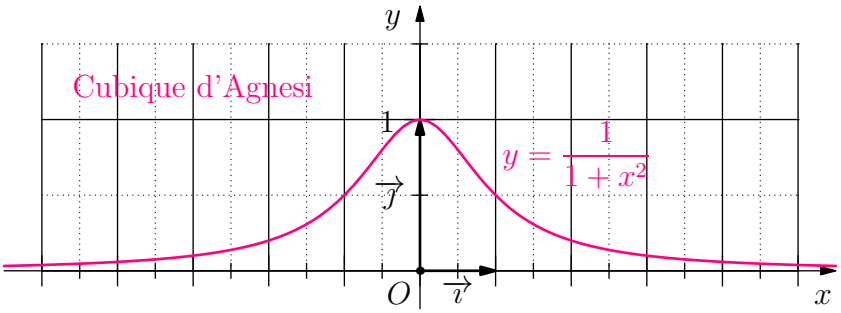
Trace le repère $\left(O\,;\vec{\imath},\vec{\jmath}\right)$.

```
void labeloIJ(picture pic=currentpicture,
              Label Lo=Label("$O$",NoFill),
              Label LI=Label("$I$",NoFill),
              Label LJ=Label("$J$",NoFill),
              pair diro=SW, pair dirI=labelIJmargin*S, pair dirJ=labelIJmargin*W,
              pen p=currentpen,
              filltype filltype=NoFill,
              marker marker=dot)
```

Trace le repère $(O\,;I,J)$.

```
void grid(picture pic=currentpicture,
          real xmin=pic.userMin.x, real xmax=pic.userMax.x,
          real ymin=pic.userMin.y, real ymax=pic.userMax.y,
          real xStep=1, real xstep=.5,
          real yStep=1, real ystep=.5,
          pen pTick=currentpen, pen ptick=grey, bool above=false)
```

Trace une grille.



CODE 72

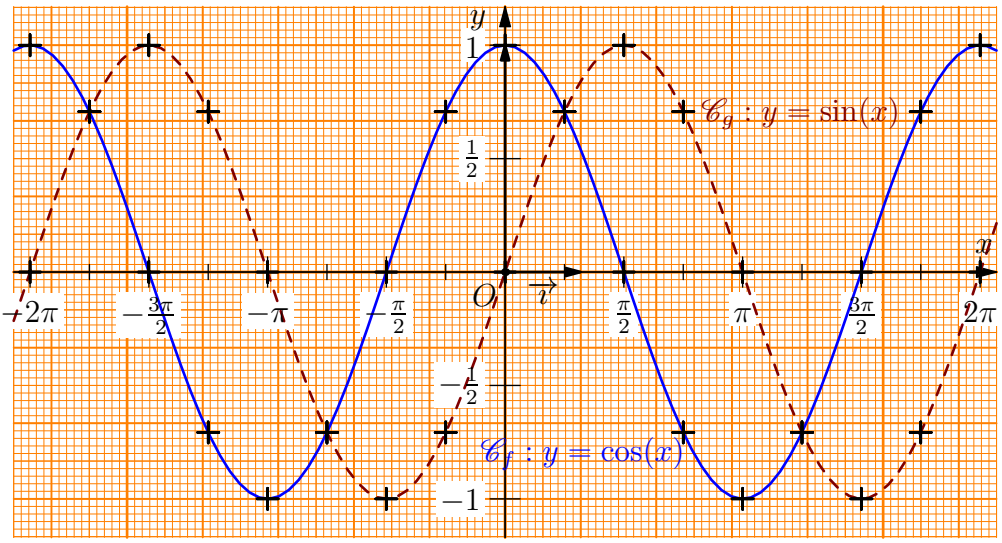
```
import graph_pi;

graphicrules(xunit=1cm, yunit=2cm, xmin=-5, xmax=5, ymin=0, ymax=1.5);
grid(pTick=currentpen,ptick=dotted);
cartesianaxis(xticks=Ticks("%"), yticks=Ticks(NoZero,Step=1,step=.5), Arrow(2mm));
labeloij();
real f(real x) {return 1/(1+x^2);}
draw(graph(f), bp+fuchsia);
label("$y=\displaystyle\frac{1}{1+x^2}$",(1,f(1)),NE,fuchsia);
label("Cubique d'Agnesi",(-3,1.2),fuchsia);
```

```
picture millimeterpaper(picture pic=currentpicture, pair O=(0,0),
                        real xmin=infinity, real xmax=infinity,
                        real ymin=infinity, real ymax=infinity,
                        pen p=.5bp+orange)
```

Grille sous forme de papier millimétré.

```
ticklabel labelfrac(real ep=1/10^5, real factor=1,
                    string symbol="",
                    bool signin=false, bool symbolin=true,
                    bool displaystyle=false,
                    bool zero=true)
```

Ce `ticklabel` permet d'écrire les graduations sous forme de fractions. Attention, `base_pi` doit être dans le dossier `$HOME/.asy`.



CODE 73

```
import graph_pi;

marker croix=marker(scale(4)*rotate(45)*cross(4),linewidth(bp));
pen bpd=bp+brown+linetype("4 4");

graphicrules(xunit=1cm, yunit=3cm, xmin=-6.5, xmax=6.5, ymin=-1.17, ymax=1.17);
add(millimeterpaper(p=1bp+orange),(0,0));
labeloij(Lj=Label(""));
cartesianaxis(Lx=Label("$x$",align=2N),extrawidth=0,
              xticks=Ticks(Label(Fill(white)),
                           labelfrac(factor=pi,symbol="\pi",symbolin=true,
                                     zero=false),Step=pi/2, step=pi/4),
              yticks=Ticks(Label(Fill(white)),
                           labelfrac(zero=false),Step=.5));
// Définition des fonctions
real f(real x) {return cos(x);}
real g(real x) {return sin(x);}
// Tracé de courbe
draw(graph(f),bp+blue);
draw(graph(g),bpd);
// Points sur les courbes
real l=-2pi;
for(int i=0; i<17; ++i) {
  draw((l,f(l)),croix);
  draw((l,g(l)),croix);
  l=l+pi/4;
}
label("$\mathscr{C}_f : y=\cos(x)$",(2.5,f(2.5)),W,blue);
label("$\mathscr{C}_g : y=\sin(x)$",(2.5,g(2.5)),NE,brown);
```

Représentations graphiques de suites :

```
recursiveroutime recursiveoption(Label L="u",
                                 bool labelbegin=true,
                                 bool labelend=true,
                                 bool labelinner=true,
```

```
                                    bool labelalternate=false,
                                    string format="",
                                    int labelplace=onX,
                                    pen px=nullpen,
                                    pen py=nullpen,
                                    bool startonyaxis=false,
                                    arrowbar circuitarrow=None,
                                    marker automarker=marker(cross(4)),
                                    marker xaxismarker=nomarker,
                                    marker yaxismarker=nomarker,
                                    marker xmarker=nomarker,
                                    marker fmarker=nomarker)
```

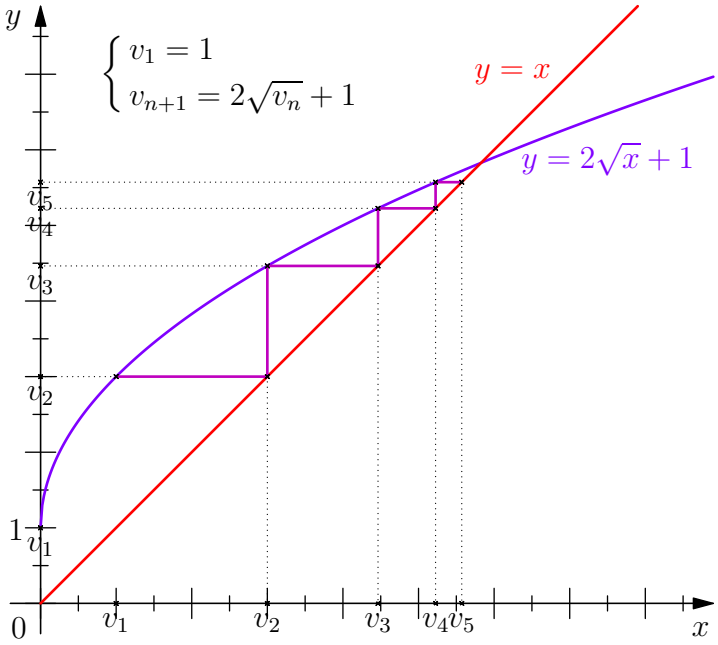
Les options du graphique :

- `L` : le nom de la suite (à taper sans $).
- `labelbegin` : si la valeur est `true`, le premier terme est placé.
- `labelend` : même chose pour le dernier terme.
- `labelinner` : si la valeur est `true`, les termes compris entre le premier et le dernier sont placés.
- `labelalternate` : si la valeur est `true`, les termes sont alternés par rapport à l'axe.
- `format` : le format d'affichage de la valeur. Par exemple `"%.2f"` (2 chiffres après la virgule).
- `labelplace` : emplacement des termes : `onX`, `onY`, ou `onXY`.
- `px` et `py` : stylos pour les tracés des traits de lecture.
- `startonyaxis` : comme son nom l'indique...
- `circuitarrow` : flèches sur le « circuit ».
- Pour tous les `marker`, voir le code 75.

`recursivegraph recursivegraph(real F(real), real u0, int n0=0, int n)`

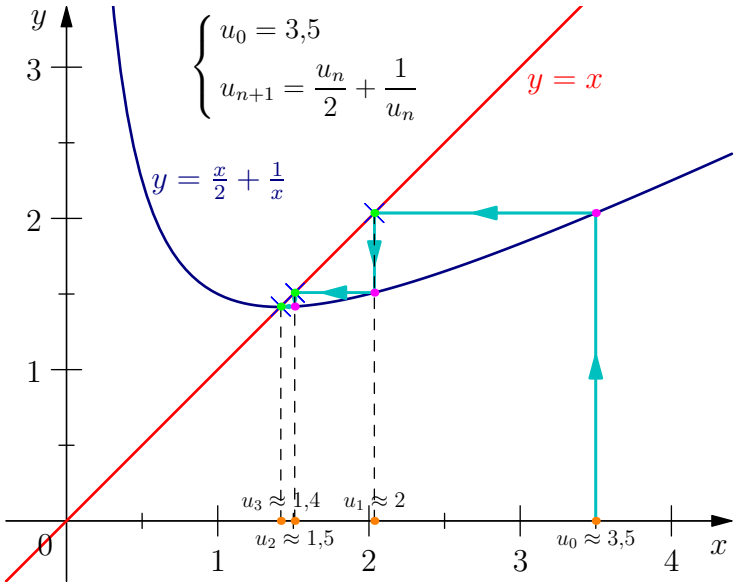
« Graphe » de la suite : `u0` est le premier terme, `n0` est l'indice du premier terme et `n` pour spécifier le nombre de termes placés sur le graphique (le dernier terme placé est celui d'indice $n-1$).

```
void draw(picture pic=currentpicture, Label L="", recursivegraph g,
          recursiveroutime lr=DefaultRecursiveOption, align align=NoAlign,
          pen p=currentpen, arrowbar arrow=None, arrowbar bar=None,
          margin margin=NoMargin, Label legend="", marker marker=nomarker)
```

CODE 74

```
import graph_pi;
usepackage("amsmath");
graphicrules(xunit=1cm, yunit=1cm, xmin=-.4, xmax=8.9, ymin=-.4, ymax=7.9);
cartesianaxis(extrawidth=0,xticks=Ticks("%"),yticks=Ticks("%"));
label("$0$",(0,0),2*SW);labely("$1$",1,1.5W);
real f(real x){return 2*sqrt(x)+1;}
draw(graph(f,0,8.9,n=400),bp+purple);
draw(graph(new real(real x){return x;},0,7.9),bp+red);
draw(recursivegraph(f,u0=1,n0=1,n=6),
     recursiveoption(L="v", labelplace=onXY),
     bp+heavymagenta);
label("$y=2\sqrt{x}+1$",(7.5,f(7.5)),3S,purple);
label("$y=x$",(7,7),2W,red);
label("$\left\{\begin{aligned}&v_1=1\\&v_{n+1}=2\sqrt{v_n}+1
               \end{aligned}\right.$",(2.5,7));
```

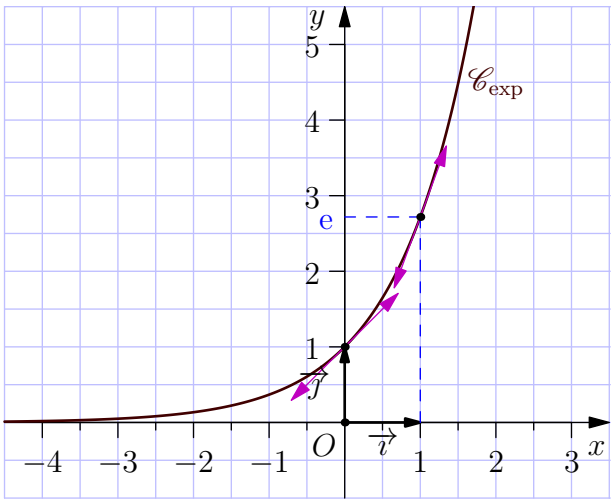


CODE 75

```
import graph_pi;
usepackage("amsmath");usepackage("icomma");
graphicrules(xunit=2cm, yunit=2cm, xmin=-.4, xmax=4.4, ymin=-.4, ymax=3.4);
cartesianaxis(extrawidth=0,xticks=Ticks(NoZero),yticks=Ticks(NoZero));
label("$0$",(0,0),2*SW);
real f(real x){return x/2+1/x;}
draw(graph(f,.1,4.4),bp+deepblue);
draw(graph(new real(real x){return x;}),bp+red);
ylimits(-.4,3.4,Crop);
draw(recursivegraph(f,3.5,n=4), recursiveoption(L=scale(.7)*"u",
                                labelbegin=true, labelend=true,
                                labelinner=true, labelalternate=true,
                                format="\approx %.1f", labelplace=onX,
                                px=dashed, py=nullpen,
                                startonyaxis=false, circuitarrow=MidArrow(3mm),
                                automarker=marker(scale(5)*cross(4),blue),
                                xaxismarker=dot(orange), yaxismarker=nomarker,
                                xmarker=dot(green), fmarker=dot(magenta)),
     bp+heavycyan);
label("$y=\frac{x}{2}+\frac{1}{x}$",(.5,f(.5)),E,deepblue);
label("$y=x$",(3,3),SE,red);
label(scale(.9)*"$\left\{\begin{aligned}&u_0=3,5\\
                &u_{n+1}=\frac{u_n}{2}+\frac{1}{u_n}
                \end{aligned}\right.$",(.75,3),E);
```

```
void graphpoint(picture pic=currentpicture,
                Label L="",
                real f(real), real xCoordinate,
                real xmin=0, real ymin=0,
                int draw=onXY,
                pen px=nullpen, pen py=px,
                arrowbar arrow=None, arrowbar bar=None,
                margin marginy=NoMargin, margin marginx=NoMargin,
                bool extend=false, bool extendx=extend, bool extendy=extend,
                Label legendx="", Label legendy="",
                marker markerx=nomarker, marker markery=nomarker)
```

```
path tangent(path g, real x, path b=box(userMin(currentpicture),userMax(currentpicture)))


void addtangent(picture pic=currentpicture,
                path g,
                pair pt,//Point on the path g // real x (x-Coordinate) est aussi défini
                real size=infinity,//ABSOLUTE size of the tangent line
                bool drawright=true,//Draw the tangent at the right
                bool drawleft=true,//... left
                pair v=(infinity,infinity),//A finite value forces the value of the derivative
                pair vr=v,//A finite value forces the value of the derivative at right
                pair vl=v,//A finite value forces the value of the derivative at left
                arrowbar arrow=null,//null=automatic determination
                margin margin=NoMargin,//Useful with size=infinity
                Label legend="",
                pen p=currentpen,
                real dt=2,
                bool differentiable=true)
```



CODE 76

```
import graph_pi;
graphicrules(xunit=1cm, yunit=1cm, xmin=-4, xmax=3, ymin=-.5, ymax=5);
cartesianaxis(xticks=RightTicks(NoZero), yticks=LeftTicks(NoZero,Step=1));
grid(pTick=paleblue,ptick=paleblue);
labeloij();

real f(real x){return exp(x);}
path Cf=graph(f,n=200);
draw(Cf, bp+darkbrown);

addtangent(Cf,x=0,size=1cm,heavymagenta);
addtangent(Cf,x=1,size=1cm,heavymagenta);
graphpoint(f,1,px=dashed+blue);

dot((1,f(1))); dot((0,f(0)));
labely("e",f(1),blue);
ylimits(-1,5.5,Crop);
label("$\mathscr{C}_\mathrm{exp}$",(1.5,f(1.5)),E,darkbrown);
```

5) animation

Ce module permet de faire des animations aux formats `gif` et `mpeg` en utilisant `convert` d'ImageMagick (qui doit donc être installé sur l'ordinateur), ainsi qu'au format `pdf` (animations visibles avec `Acrobat Reader`).
Pour obtenir un `gif` par exemple, il suffit de compiler avec l'option `-f gif` :

```
asy -f gif
```

ou de rajouter dans le fichier `asy`

```
settings.outformat="gif";
```

En pratique, le module `animate.asy` importe le module `animation.asy` et le package `animate.sty`. Pour plus d'éclaircissements il faudra donc se référer au fichier `animation.asy`, ainsi qu'à la documentation du package LaTeX `animate.sty` (qui est donc requis).

a) Création d'animations autonomes

Les principales routines utilisées ici sont :

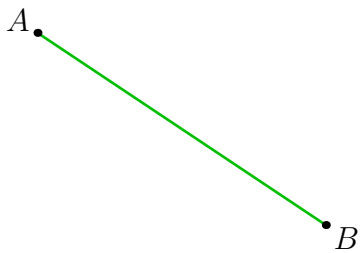
```
void erase(picture pic=currentpicture);
```
qui efface l'image,

```
void save()
```
qui sauvegarde l'image à l'endroit où elle est utilisée,

```
void restore()
```
qui restaure (c'est facile l'anglais !) l'image au point où elle a été sauvegardée par `save()`, et enfin

```
void add(picture pic=currentpicture, enclosure enclosure=NoBox)
```
qui ajoute l'image à l'animation.

Un exemple : construction de la médiatrice d'un segment. (Pour voir l'animation, il faut utiliser `Acrobat Reader`.)



CODE 77

```
import geometry;
import animate;
settings.tex="pdflatex";
settings.outformat="pdf";
size(15cm,0);
point A=(0,0), B=(3,-2), lab=(6,1);
circle cA=circle(A,2.5), cB=circle(B,2.5);
point[] int=intersectionpoints(cA,cB);
segment seg=segment(A,B);
line med=line(int[0],int[1]);
animation Anim;
// ----------------------------------------
// Etape 0
draw(seg,bp+heavygreen);
dot("$A$",A,unit(A-B)); dot("$B$",B,unit(B-A));
Anim.add();
// Etape 1
draw(cA,bp+blue);
draw(cB,bp+blue);
Anim.add();
// Etape 2
save(); // tout ce qu'il y a avant sera conservé dans la suite
label("\begin{minipage}{4cm}
On trace deux cercles\\
de m\^eme rayon.\end{minipage}",lab,E);
draw(lab..controls (4,2) and (3,2)..angpoint(cA,45),Arrow());
draw(lab..controls (5.5,1) and (4.5,1)..angpoint(cB,60),Arrow());
Anim.add();
restore();
// Etape 3
save();
label("\begin{minipage}{4cm}
Deux points\\
\'equidistants\\
de $A$ et $B$.\end{minipage}",lab,E);
draw(lab..controls (3,.5) and (2,-.5)..int[0],Arrow());
draw(lab..controls (5,1) and (4.5,1)..int[1],Arrow());
dot(int[0],heavyred); dot(int[1],heavyred);
Anim.add();
restore();
// Etape 4
save();
draw(med,bp+heavyred);
label("\begin{minipage}{4cm}
M\'ediatrice de $\left[AB\right]$\end{minipage}",lab,E,heavyred);
draw(lab..controls (5,1.5) and (4,1.6)..relpoint(med,1.4),Arrow());
dot(int[0],heavyred); dot(int[1],heavyred);
Anim.add();
restore();
// ----------------------------------------
erase(); // permet d'éviter les effets de "décalage"
label(Anim.pdf("controls,step"));
```

Quelques sorties :

`label(Anim.pdf("<options>",keep=true,multipage=false));`
pour obtenir un pdf par image.

`label(Anim.pdf("<options>",keep=true,multipage=true));`

pour obtenir l'animation et le pdf multipage (pdf contenant une page par image).

`label(Anim.pdf("<options>"));`
pour l'animation uniquement.

`Anim.movie();`
pour obtenir un pdf multipage.

`Anim.glmovie();`
pour une sortie dans la fenêtre OpenGL, à compiler avec `asy -V`

Pour les options entre guillemets, ce sont les options du package `animate.sty` (il faut consulter sa documentation pour davantage de précisions). On trouve entre autres :

- `autoplay` : démarre l'animation automatiquement.
- `loop` : l'animation recommence en boucle.
- `controls` : pour les boutons de contrôle (sinon, il faut cliquer sur l'animation pour la lancer).
- `buttonsize=<size>`, `buttonbg=<colour>`, `buttonfg=<colour>` : mise en forme des boutons.
- `palindrome` : l'animation s'exécute d'avant en arrière.
- `step` : pas à pas.
- `width=...`, `height=...`, `depth=...` : redimensionne l'animation.
- `scale=...` : mise à l'échelle de l'animation.

Il est aussi possible de préciser le temps (en millisecondes) entre chaque image, par exemple :

`label(Anim.pdf("controls",delay=50));`
pour une image toutes les 50 millisecondes.

`movie()` peut aussi prendre des arguments, utile par exemple pour la création d'un `gif` (puisqu'il n'y aura pas de boutons de contrôle...). L'exemple ci-contre peut être compilé directement par : `asy mongif.asy`.

CODE 78

```
import animation;
settings.outformat="gif";
size(100,100);
path c1=circle((0,0),1);
path c2=circle((0,.6),.4);
draw(c1,bp+purple);
animation Anim;
// ------------------------------
for(int i=0; i < 360; i+=10) {
save();
draw(rotate(i)*c2,bp+heavymagenta);
Anim.add();
restore();}
// ------------------------------
Anim.movie(loops=3,delay=50);
```

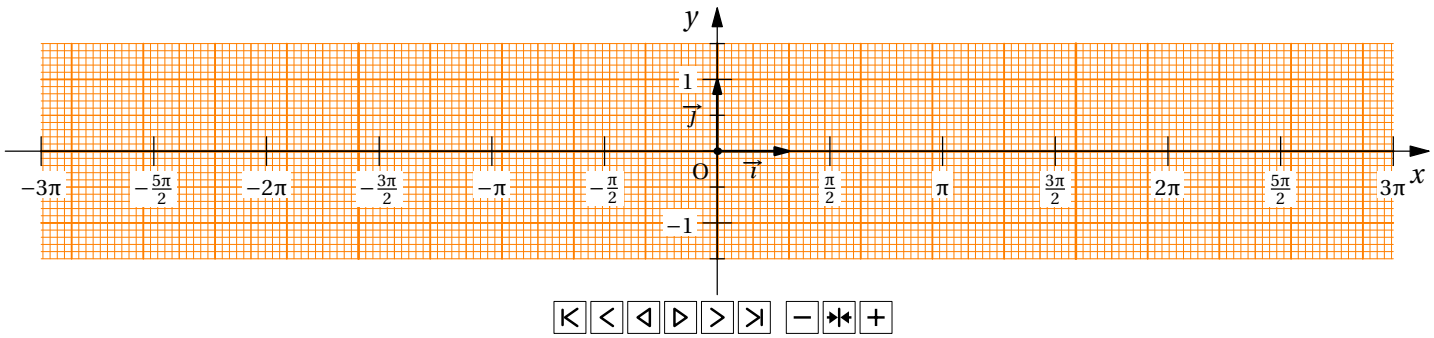
b) Inclure une animation externe dans un fichier .tex

L'animation doit être créée avec la sortie `Anim.movie()` (pdf multipage). Le package `animate.sty` va se charger de reconstituer l'animation à l'aide de la commande :
`\animategraphics[<options>]{<frame rate>}{<file basename>}{<first>}{<last>}`
`<file basename>` est le nom de l'animation sans extension et `<frame rate>` est le nombre d'images par seconde. `<first>` et `<last>`, la première et dernière image.
Exemple :

```
\documentclass{article}
\usepackage{animate}
\begin{document}
\animategraphics[width=\linewidth,controls]{10}{animcos}{}{}
\end{document}
```

CODE 79

```
// Animation réalisée avec l'aide de Gaétan Marris et Olivier Guibé
import graph_pi;
import animate;
settings.tex="pdflatex";
settings.outformat="pdf";
usepackage("fourier","upright");

real f(real x) {return cos(x);}
path Cf1=graph(f,0,pi,n=20,Hermite);
path Cf2=graph(f,-pi,pi,n=20,Hermite);
path Cf3=graph(f,pi,3pi,n=20,Hermite);
transform sc=scale(.8);
pen pinit=1.2bp+magenta, pf=1.2bp+.8blue, pvec=bp+deepgreen;

animation A;
// ------------------------------------------------------------------------------
void trace(path f, pen p,
        Label L="", real r=0, real per=2pi, pair NS=(0,0),
        pen pv=nullpen, arrowbar arrow=Arrow(HookHead,2mm)){
        draw(f,p);
        if (NS!=(0,0))
        draw(Label(sc*L,Fill(white),align=NS),(r,f(r))--(r+per,f(r+per)),pv,arrow);
}
graphicrules(xunit=1cm, yunit=1cm, xmin=-3pi, xmax=3pi, ymin=-1.5, ymax=1.5);
add(millimeterpaper(p=1bp+orange),(0,0));
labeloij(Lo=sc*"$O$",Li=sc*"$\overrightarrow{\imath}$",Lj=sc*"$\overrightarrow{\jmath}$");
cartesianaxis(xticks=Ticks(sc*Label(Fill(white)),
                           labelfrac(factor=pi,symbol="\pi",symbolin=true,
                                     zero=false),Step=pi/2, ptick=black),
              yticks=Ticks(sc*Label(Fill(white)),
                           labelfrac(zero=false),Step=1,step=.5));
A.add();
// ------------------------------------------------------------------------------
for (real p=-1; p<=1; p+=.1) {
  save();
  trace(Cf1,pinit);
  draw(xscale(-p)*Cf1,pinit);
  A.add();
  restore();
}
for (real t=0; t<=2pi; t+=pi/25) {
  save();
  trace(Cf2,pinit,"$2\pi \vec{\imath}$",.75,N,pvec);
  draw(shift(t,0)*Cf2,pf);
  A.add();
  restore();
}
for (real t=0; t<=2pi+pi/25; t+=pi/25) {
  save();
  trace(Cf2,pinit,"$2\pi \vec{\imath}$",.75,N,pvec);
  trace(Cf3,pf,"$-2\pi \vec{\imath}$",-pi,-2pi,S,pvec);
  draw(shift(-t,0)*Cf2,pf);
  A.add();
  restore();
}
erase();
A.movie();
```



Pour voir l'animation, il faut utiliser `Acrobat Reader`.

c) Inclure le code dans un fichier .tex

On se contentera ici de copier le code du fichier `inlinemovie.tex` de la galerie d'exemples du site officiel : asymptote.sourceforge.net/gallery/animations

```
\documentclass{article}
\usepackage[inline]{asymptote}
%\usepackage{asymptote}
\usepackage{animate}
\begin{document}

Here is an inline PDF movie, generated with the commands
\begin{verbatim}
pdflatex inlinemovie
asy inlinemovie-*.asy
pdflatex inlinemovie
\end{verbatim}
or equivalently,
\begin{verbatim}
latexmk -pdf inlinemovie
\end{verbatim}

\begin{center}
\begin{asy}
import animate;
animation A=animation("movie1");
real h=2pi/10;

picture pic;
unitsize(pic,2cm);
for(int i=0; i < 10; ++i) {
  draw(pic,expi(i*h)--expi((i+1)*h));
  A.add(pic);
}
label(A.pdf("controls",delay=50,keep=!settings.inlinetex));
\end{asy}
%Uncomment the following line when not using the [inline] package option:
%\ASYanimategraphics[controls]{50}{movie1}{}{}
\end{center}
```

```
And here is another one, clickable but without the control panel:
\begin{center}
\begin{asy}
import animate;
animation A=animation("movie2");
real h=2pi/10;

picture pic;
unitsize(pic,2cm);
for(int i=0; i < 10; ++i) {
  draw(pic,expi(-i*h)--expi(-(i+1)*h),red);
  A.add(pic);
}
label(A.pdf(keep=!settings.inlinetex));
\end{asy}
%Uncomment the following line when not using the [inline] package option:
%\ASYanimategraphics[controls]{10}{movie2}{}{}
\end{center}
\end{document}
```

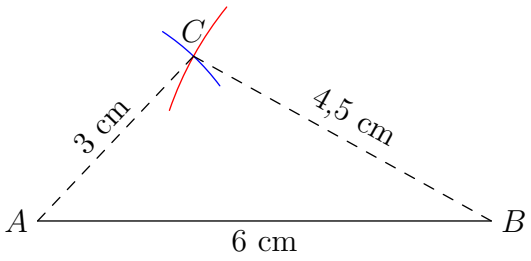

ANNEXE A – STRUCTURES - EXEMPLE

`geometry` ajoute une fonction `compassmark` qui ne convient pas forcément à tous les usages (il faut connaître le point par lequel passe le « trait de compas »).
Il est cependant possible de l'utiliser pour créer une structure `compass` permettant de dessiner des traits de compas à partir de deux points et de récupérer les points d'intersections de ces traits.

CODE

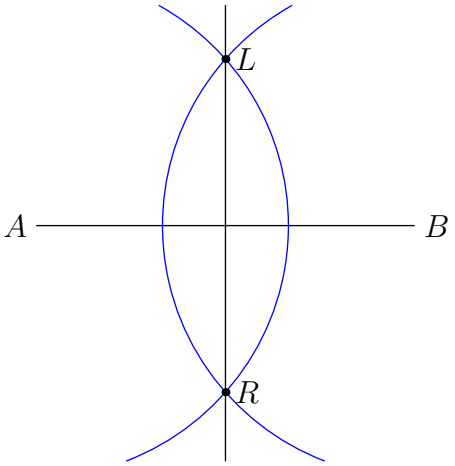
```
import geometry;

struct compass
{
  restricted point A, B; // extrémités du segment
  restricted real radiusA, radiusB; // rayons des arcs
  restricted real positionA, positionB; // positions des "traits de compas"
  restricted real angleA, angleB; // tailles des "traits de compas"
  restricted circle circleA, circleB; // cercles de centres A et B, de rayon radiusA et radiusB
  restricted point L, R; // points d'intersection des traits : L pour LeftSide, R pour RightSide
  restricted path markAL, markAR, markBL, markBR; // les "traits de compas"

  void init(point A, point B, real radiusA, real radiusB,
            real positionA, real positionB, real angleA, real angleB)
  {
    this.A=A; this.B=B;
    this.radiusA=radiusA; this.radiusB=radiusB;
    this.positionA=positionA; this.positionB=positionB;
    this.angleA=angleA; this.angleB=angleB;
    // points d'intersection des traits
    this.circleA=circle(this.A,this.radiusA); this.circleB=circle(this.B,this.radiusB);
    point[] inter=intersectionpoints(this.circleA,this.circleB);
    this.L=inter[1]; this.R=inter[0];
    // traits de compas de centre A
    this.markAL=compassmark(this.A,this.L,position=this.positionA,this.angleA);
    this.markAR=compassmark(this.A,this.R,position=1-this.positionA,this.angleA);
    // traits de compas de centre B
    this.markBL=compassmark(this.B,this.L,position=this.positionB,this.angleB);;
    this.markBR=compassmark(this.B,this.R,position=1-this.positionB,this.angleB);;
  }
}
compass compass(point A, real radiusA, real positionA=.5, real angleA=20,
                point B, real radiusB=radiusA, real positionB=1-positionA, real angleB=angleA)
{
  real lmax=max(radiusA,radiusB,length(B-A));
  if (lmax > radiusA+radiusB+length(B-A)-lmax)
     abort("compass: the longest side of a triangle must be strictly smaller
                                than the sum of two other !");
  compass c;
  c.init(A,B,radiusA,radiusB,positionA,positionB,angleA,angleB);
  return c;
}
// Fonction pour tracer tous les traits de compas
void markcompass(compass c, pen penAL=currentpen, pen penAR=penAL, pen penBL=penAL, pen penBR=penAL)
{draw(c.markAL,penAL);draw(c.markAR,penAR);
draw(c.markBL,penBL);draw(c.markBR,penBR);}
```

Ce code peut être sauvegardé dans un fichier `compass.asy`, à mettre dans le répertoire des codes de figures, ou bien dans le dossier `.asy` du répertoire personnel. Il suffira ensuite de l'importer par un `import compass;`.



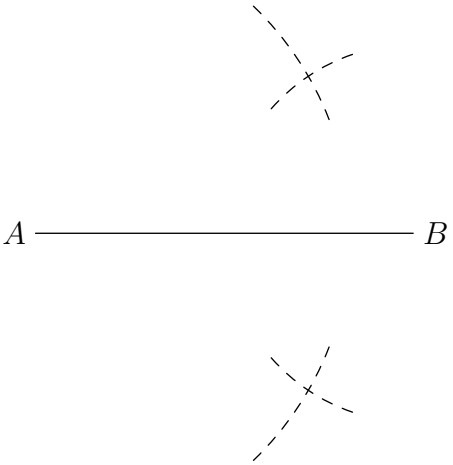
CODE 80

```
import compass;
usepackage("icomma");
unitsize(1cm);
point A=(0,0), B=(6,0);
draw("$6$~cm",A--B);
compass comp=compass(A,radiusA=3,
                     B,radiusB=4.5);
draw(comp.markAL,blue);
draw(comp.markBL,red);
draw(Label("$3$~cm",Rotate(-dir(comp.L--A))),
     comp.L--A,dashed);
draw(Label("$4,5$~cm",Rotate(-dir(B--comp.L))),
     B--comp.L,dashed);
label("$C$",comp.L,1.5N);
label("$A$",A,W); label("$B$",B,E);
```



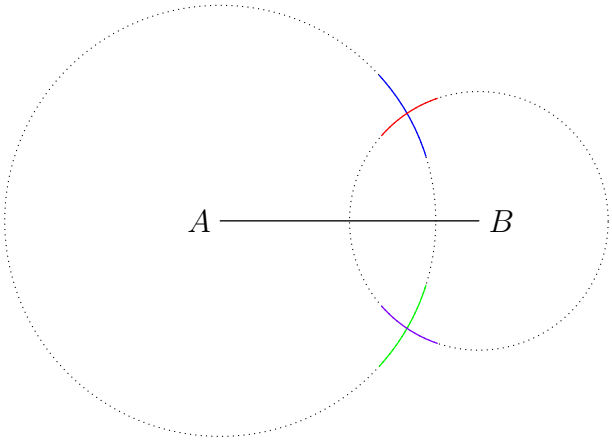
CODE 81

```
import compass;
size(6cm,0);
point A=(0,0), B=(3,0);
draw(A--B);
compass comp=compass(A,radiusA=2,
        positionA=.85,angleA=130,B);
draw(comp.markAL^^comp.markBL,blue);
dot("$L$",comp.L); dot("$R$",comp.R);
drawline(comp.L,comp.R);
label("$A$",A,W); label("$B$",B,E);
```



CODE 82

```
import compass;
size(6cm,0);
point A=(0,0), B=(3,0);
draw(A--B);
compass comp=compass(A,radiusA=2.5,
                     positionA=.35,angleA=25,
                     B,radiusB=1.5,
                     positionB=.5,angleB=30);
markcompass(comp,dashed);
label("$A$",A,W); label("$B$",B,E);
```



CODE 83

```
import compass;
size(8cm,0);
point A=(0,0), B=(3,0);
draw(A--B);
compass comp=compass(A,radiusA=2.5,angleA=25,
                     B,radiusB=1.5,angleB=30);
draw(comp.circleA^^comp.circleB,dotted);
markcompass(comp,penAL=blue,penAR=green,
            penBL=red,penBR=purple);
label("$A$",A,W); label("$B$",B,E);
```

Remarque :
Pour comprendre davantage encore les structures, voir cet exemple très bien documenté de Philippe Ivaldi :
www.piprime.fr/1274/various_asymptote-fig0300/

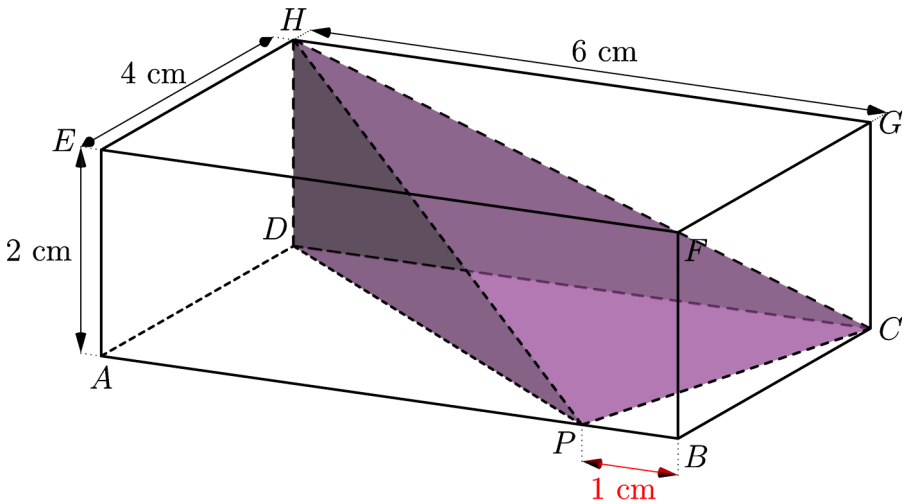
ANNEXE B – QUELQUES MACROS DE MARQUAGE « 3D »

Les routines suivantes pourront être mises dans un fichier `macros3D.asy` (Voir dernière annexe).

- Cette première routine permet de mettre des cotes sur les figures 3D.
  La longueur AB est marquée par une double flèche tracée avec le stylo `p`, à une distance `d` de [AB] (si `cc=false`, le label change de côté). `v` est un vecteur normal à la flèche (permet de mettre la flèche dans la direction voulue).
  Si vous visualisez ce document à l'aide d'`Acrobat Reader`, vous pouvez activer la figure et la bouger avec la souris.

CODE

```
void cote3D(picture pic=currentpicture,
                Label L="", triple A, triple B,
                real d=5mm, triple v, bool cc=true,
                pen p=currentpen, pen joinpen=dotted,
                arrowbar3 arrow=Arrows3)
{
        transform3 T=shift(d*unit(v));
        triple A=A, B=B;
        pic.add(new void(picture f, transform3 t) {
                picture opic;
                path3 dist;
                triple Ap=t*A, Bp=t*B;
                triple a=T*Ap, b=T*Bp;
                if (cc) {dist=a--b;}
                else {dist=b--a;}
                draw(opic,L,dist,p,arrow);
                draw(opic,a--Ap^^b--Bp,joinpen);
                add(f,opic);
                }, true);
}
```



CODE 84

```
import three;
import macros3D;
defaultrender.merge=true;
settings.render=4;
size(12cm);
real a=6, b=4, c=2, d=1;
currentprojection=orthographic(120*a/3,-120*b,120*2c/3);
// facteur de 120 pour que le prc soit à la "bonne" taille
// à cause d'un bogue d'Acrobat Reader sous linux,
// inutile sous Windows, une fois n'est pas coutume...
transform3 T=shift(0,0,c);
triple A=(0,0,0), B=(a,0,0), C=(a,b,0), D=(0,b,0),
       pE=T*A, F=T*B, G=T*C, H=T*D, P=(a-d,0,0);
pen ps=lightmagenta+opacity(.5), pT=bp+linetype("4 4");

draw(A--B--C--G--H--pE--F--G^^pE--A^^F--B,linewidth(bp));
draw(A--D--C^^H--D,pT);
draw(surface(H--D--P--cycle),ps,pT);
draw(surface(H--P--C--cycle),ps,pT);
draw(surface(H--D--C--cycle),ps,pT);
draw(surface(D--P--C--cycle),ps,pT);

label("$A$",A,S); label("$B$",B,SE); label("$C$",C,E);
label("$D$",D,NW); label("$E$",pE,NW+2W); label("$F$",F,SE);
label("$G$",G,E); label("$H$",H,N); label("$P$",P,SW);

cote3D(format("$%f$~cm",a),G,H,unit(H-pE));
cote3D(format("$%f$~cm",b),H,pE,3mm,unit(pE-F));
cote3D(format("$%f$~cm",c),pE,A,3mm,unit(pE-F));
cote3D(format("$%f$~cm",d),P,B,unit(B-F),red);
```

- Cette routine due à Philippe Ivaldi permet de marquer les angle droits.
  Elle marque l'angle droit $(\overrightarrow{MA};\overrightarrow{MB})$.

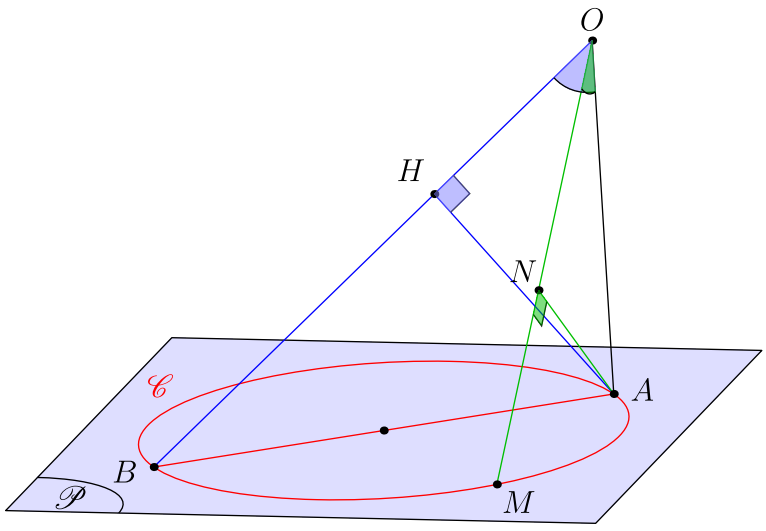
CODE

```
void drawrightangle(picture pic=currentpicture,
                    triple M, triple A, triple B,
                    real radius=0,
                    pen p=currentpen,
                    pen fillpen=nullpen,
                    projection P=currentprojection)
{
  p=linejoin(0)+linecap(0)+p;
  if (radius==0) radius=arrowfactor*sqrt(2);
  transform3 T=shift(-M);
  triple OA=radius/sqrt(2)*unit(T*A),
    OB=radius/sqrt(2)*unit(T*B),
    OC=OA+OB;
  path3 _p=OA--OC--OB;
  picture pic_;
    draw(pic_, _p, p=p);
    if (fillpen!=nullpen) draw(pic_, surface(O--_p--cycle), fillpen);
  add(pic,pic_,M);
}
```

- Une adaptation de la routine précédente, pour marquer un angle quelconque.
  Si `cc=false`, elle trace « l'autre » arc.
  Si vous visualisez ce document à l'aide d'`Acrobat Reader`, vous pouvez activer la figure et la bouger avec la souris.

CODE

```
void markangle3D(picture pic=currentpicture,
                 Label L="",
                 triple M, triple A, triple B,
                 bool cc=true,
                 real radius=0,
                 pen p=currentpen,
                 pen fillpen=nullpen,
                 arrowbar3 arrow=None,
                 projection P=currentprojection)
{
  p=linejoin(0)+linecap(0)+p;
  if (radius==0) radius=arrowfactor*sqrt(2);
  transform3 T=shift(-M);
  triple OA=radius/sqrt(2)*unit(T*A),
    OB=radius/sqrt(2)*unit(T*B);
  path3 pl=O--OA--OB;
  triple V=normal(pl);
  path3 _p; real k;
  if (cc) k=1;
  else k=-1;
  picture pic_;
  _p=arc(O,OA,OB,k*V);
    if (fillpen!=nullpen) draw(pic_, surface(O--_p--cycle), fillpen);
  draw(pic_, L, _p, p=p, arrow);
  add(pic,pic_,M);
}
```

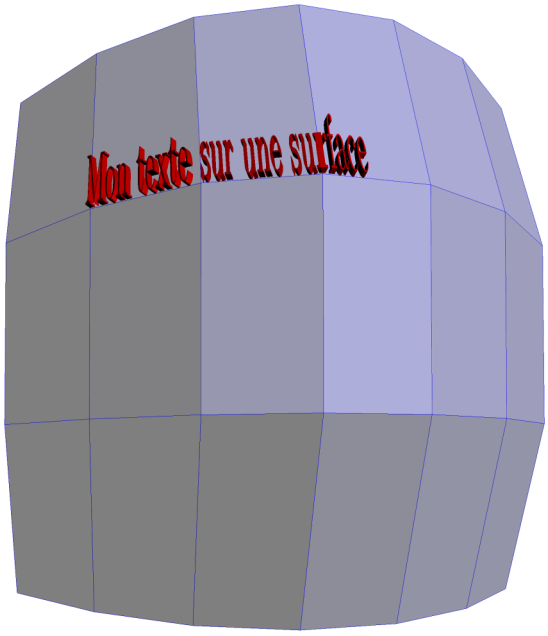


CODE 85

```
size(10cm,0);
import three;
import macros3D; // voir derniere annexe
usepackage("icomma");
usepackage("mathrsfs");
settings.outformat="pdf";
settings.render=0;
real coef=25; // pour un bogue d'Acrobat sous linux, inutile sous Windows
currentprojection = currentprojection=orthographic(
camera=(coef*20,coef*6,coef*6), up=(-0.004142,0.000122,0.021101),
target=(-1.71e-15,-4.28e-16,-4.28e-16), zoom=0.7);
currentlight=nolight;

triple p1=(5,0,0), p2=(5,5,0), p3=(0,5,0);
path3 pl=plane(p1,p3);
path3 c3=circle(p2/2,2,Z);

triple A=relpoint(c3,.35), B=relpoint(c3,.85), M=relpoint(c3,.12);
triple pO=(A.x,A.y,3), H1=rotate(180,pO,B)*A, N1=rotate(180,pO,M)*A;
triple H=intersectionpoint(pO--B,A--H1), pN=intersectionpoint(pO--M,A--N1);

draw(surface(pl),paleblue+opacity(.5),black);
draw(c3^^A--B,red);
draw(pO--A);draw(pO--B,blue);draw(pO--M,heavygreen);
draw(A--H,blue);draw(A--pN,heavygreen);

dot("$A$",A,2*unit(A-B));dot("$B$",B,2*unit(B-A));dot(p2/2);
dot("$O$",pO,N);dot("$M$",M,SE);dot("$H$",H,2*unit(H-A));
dot("$N$",pN,2*unit(pN-A));
label("$\mathscr{C}$",relpoint(c3,.7),NW,red);
label("$\mathscr{P}$",p1,6*unit(p3-p1));
drawrightangle(H,A,pO,5mm,fillpen=lightblue+opacity(.5));
drawrightangle(pN,M,A,5mm,fillpen=heavygreen+opacity(.5));
markangle3D(pO,H,A,10mm,fillpen=lightblue+opacity(.5));
markangle3D(pO,pN,A,10mm,fillpen=heavygreen+opacity(.5));
markangle3D(p1,p2,O,22mm);
```

ANNEXE C – ÉCRIRE SUR UNE SURFACE

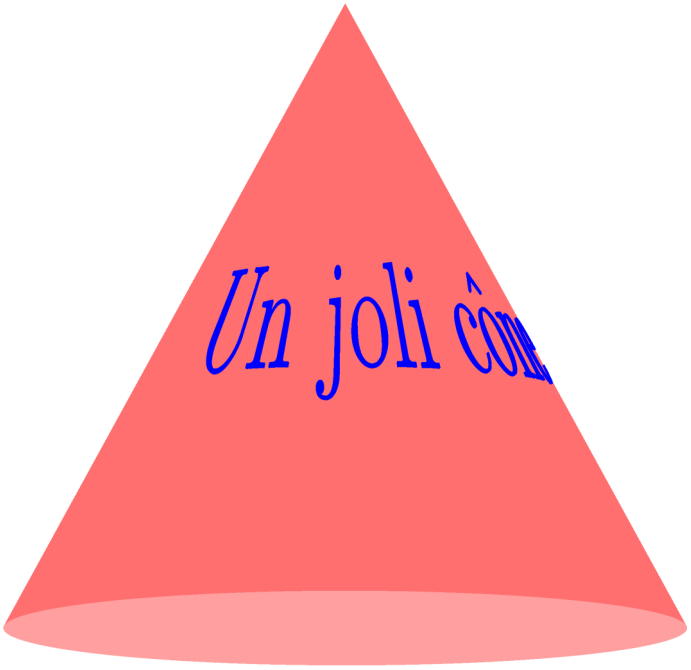
Exemple 1 : Pour essayer de comprendre comment sont placés les labels, remarquer les valeurs de `nx` et `ny`, ainsi que celles de `uoffset` et `voffset` pour comparer avec les lignes de la figure.
Si vous visualisez ce document à l'aide d'`Acrobat Reader`, vous pouvez activer la figure et la bouger avec la souris.



CODE 86

```
import graph3;
size(10cm,0);
settings.render=4;
currentprojection=perspective(
     camera=(-0.508471034714439,-0.453210476508603,5.8054086347958),
     up=(-4.06484402934237e-05,0.0135768552457447,0.000957626274221287),
     target=(0.0977992777408489,0.0442771771973709,-1.22203503249791),
     zoom=1,
     angle=23.1851359726969,
     autoadjust=false);

real f(pair z) {return -(2z.x^2+z.y^2);}

surface surf=surface(f,(-1,-1),(1,1),nx=6,ny=3);
draw(surf,lightblue+opacity(0.5),blue);
draw(surface(xscale(4)*scale(.1)*"Mon texte sur une surface",
     surf,uoffset=1,voffset=2,height=0.02),red);
```

Exemple 2 : Sur un solide de révolution, les valeurs de `uoffset` et `voffset` sont à chercher plus ou moins par tâtonnement.
Si vous visualisez ce document à l'aide d'`Acrobat Reader`, vous pouvez activer la figure et la bouger avec la souris.



CODE 87

```
size(9cm,0);
import solids;
settings.render=4;
currentlight=nolight;
revolution Cone=cone(0,3,6,axis=Z,n=1);
surface ConeSurf=surface(Cone);
draw(ConeSurf,mediumred+opacity(0.5));
draw(surface(xscale(1.5)*scale(.2)*"Un joli c\^one",ConeSurf,
             uoffset=6,voffset=.3,height=0.02),blue);
```

ANNEXE D – PERSPECTIVE CAVALIÈRE

La représentation en perspective cavalière n'est pas conforme à notre vision naturelle des objets, mais elle reste la plus utilisée en Mathématiques (notamment car, selon une direction donnée, les rapports de longueur sont conservés).

`Asymptote` ne prévoit pas (pour l'instant) ce type de projection. Celles s'en rapprochant le plus sont `obliqueX` et `obliqueY`, mais, si l'angle de fuite est réglable, le coefficient de réduction ne l'est pas.

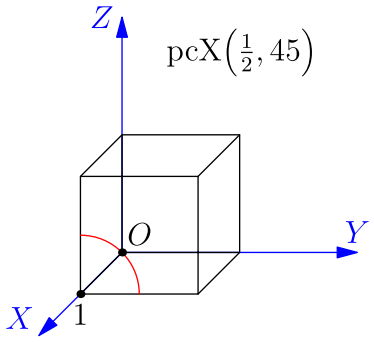
Il est cependant possible de modifier les définitions d'`obliqueX` et `obliqueY` pour obtenir deux perspectives cavalières : `pcX(real k, real angle)` et `pcY(real k, real angle)`. Pour les figures suivantes, le code des deux projections sera mis dans le fichier `macros3D.asy` (voir dernière annexe) :

CODE

```
projection pcX(real k=.5, real angle=45)
{
  transform3 t=identity(4);
  real c2=Cos(angle)^2;
  real s2=1-c2;
  t[0][0]=-k*Cos(angle);
  t[1][0]=-k*Sin(angle);
  t[1][1]=0;
  t[0][1]=1;
  t[1][2]=1;
  t[2][2]=0;
  t[2][0]=1;
  t[2][3]=-1;
  return projection((1,c2,s2),normal=(1,0,0),
    new transformation(triple,triple,triple) {
      return transformation(t);});
}
projection pcX=pcX();
```

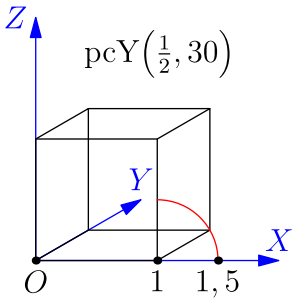
CODE

```
projection pcY(real k=.5, real angle=45)
{
  transform3 t=identity(4);
  real c2=Cos(angle)^2;
  real s2=1-c2;
  t[0][1]=k*Cos(angle);
  t[1][1]=k*Sin(angle);
  t[1][2]=1;
  t[2][1]=-1;
  t[2][2]=0;
  t[2][3]=-1;
  return projection((c2,-1,s2),normal=(0,-1,0),
    new transformation(triple,triple,triple) {
      return transformation(t);});
}
projection pcY=pcY();
```



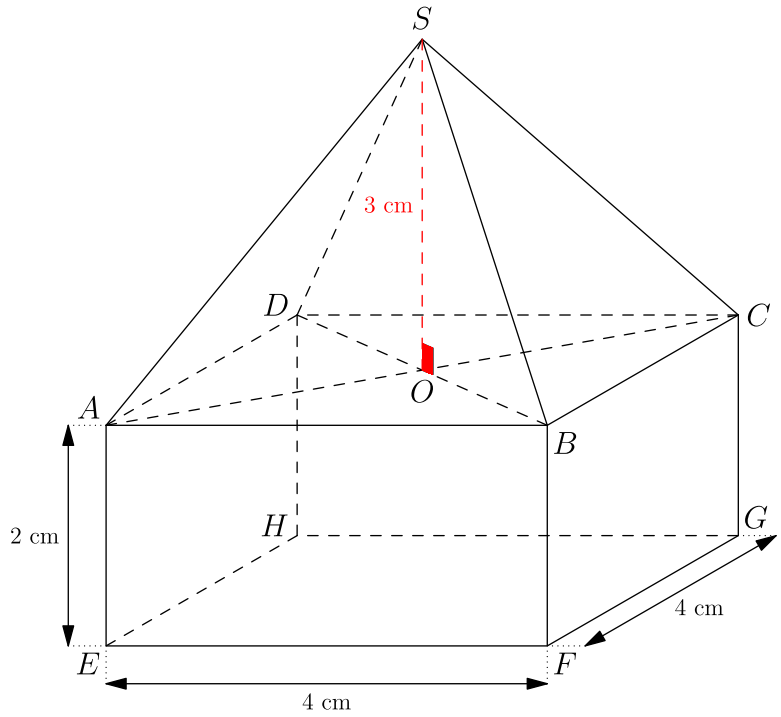
CODE 88

```
import three; import macros3D;
size(5cm);
settings.render=0; settings.prc=false;
currentprojection=pcX;
draw(Label("$X$",EndPoint,NW),
     O--2*X,blue,Arrow3);
draw(Label("$Y$",EndPoint,N),
     O--2*Y,blue,Arrow3);
draw(Label("$Z$",EndPoint,W),
     O--2*Z,blue,Arrow3);
draw(unitbox);
draw(arc(X,(1,.5,0),(1,0,.5)),red);
dot("$O$",O,NE); dot("$1$",(1,0,0),S);
label("pcX$\left(\frac12,45\right)$",
      (0,1,1.7));
```



CODE 89

```
import three; import macros3D;
size(4cm);
settings.render=0; settings.prc=false;
currentprojection=pcY(.5,30);
draw(Label("$X$",EndPoint,N),
     O--2*X,blue,Arrow3);
draw(Label("$Y$",EndPoint,N),
     O--2*Y,blue,Arrow3);
draw(Label("$Z$",EndPoint,W),
     O--2*Z,blue,Arrow3);
draw(unitbox);
draw(arc(X,(1.5,0,0),(1,0,.5)),red);
dot("$O$",O,S); dot("$1$",(1,0,0),S);
dot("$1,5$",(1.5,0,0),S);
label("pcY$\left(\frac12,30\right)$",
     (1,0,1.7));
```



CODE 90

```
import three;
import macros3D;
size(10cm);
settings.render=0;
settings.prc=false;

currentlight=nolight;
currentprojection=pcY(.5,30);

transform3 T=shift(-2*Z);
triple A=O, B=(4,0,0), C=(4,4,0), D=(0,4,0), pS=(2,2,3),
       pO=(2,2,0), pE=T*A, F=T*B, G=T*C, H=T*D;

draw(A--C^^B--D,dashed);
draw(A--B--C--pS--A^^pS--B);
draw(pS--D^^A--D--C,dashed);
draw(scale(.75)*"$3$~cm",pS--pO,dashed+red);
draw(A--pE--F--G--C^^B--F);
draw(pE--H--G^^H--D,dashed);

drawrightangle(pO,pS,B,5mm,fillpen=red);
cote3D(scale(.75)*"$2$~cm",A,pE,A-B);
cote3D(scale(.75)*"$4$~cm",pE,F,pE-A);
cote3D(scale(.75)*"$4$~cm",F,G,F-pE);

label("$A$",A,NW); label("$B$",B,SE);
label("$C$",C,E); label("$D$",D,W+.5N);
label("$S$",pS,N); label("$O$",pO,S);
label("$E$",pE,SW); label("$F$",F,SE);
label("$G$",G,NE); label("$H$",H,W+.5N);
```

ANNEXE E – INTERSECTIONS PLAN-POLYÈDRE

J'ai écris ces macros suite à une demande sur le forum de MathemaTex :
forum.mathematex.net/asymptote-f34/intersection-de-deux-plans-t13121.html#p126743

Il s'agit ici de tracer l'intersection de deux surfaces planes et par extension, celle d'un plan et d'un polyèdre. `Asymptote` ne propose pas encore d'outil pour le faire. il est cependant possible d'utiliser le module `bsp`, mais on perd la possibilité de manipuler la figure. Pour ceux que ça intéresse, voir le post suivant sur le forum de MathemaTex : forum.mathematex.net/asymptote-f34/modules-bsp-et-animate-t12770.html#p123610

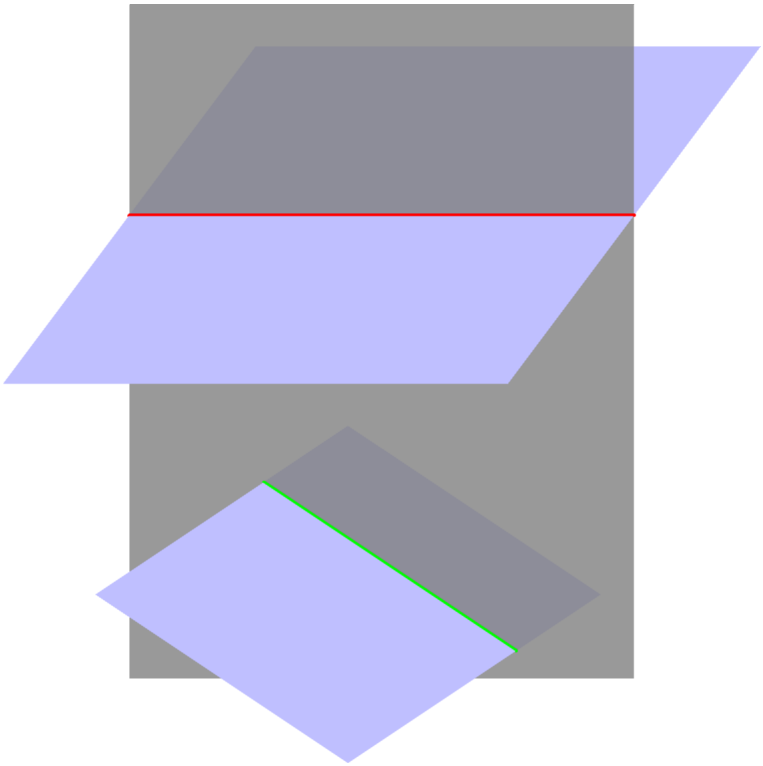
Les routines suivantes pourront être mises dans un fichier `macros3D.asy` (Voir dernière annexe).

CODE

```
bool inplan(triple p, path3 surfplane) {
  triple n=normal(surfplane);
  triple[] int=intersectionpoints(p--shift(n)*p,surface(surfplane));
  return (int.length!=0 && abs(int[0]-p)<.0001);
}

path3 interplane(path3 plan1, path3 surfplane) {
  path3 intersec;
  path3[] ligne;
  triple[][] iter;
  triple[] pinter;
  int n=size(plan1);
  for (int i=0; i<n; ++i) {
    ligne[i]=point(plan1,i)--point(plan1,(i+1)%n);
    if(inplan(point(plan1,i),surfplane) && inplan(point(plan1,(i+1)%n),surfplane)) {
      intersec=intersec & ligne[i];}
    else {
      iter[i]=intersectionpoints(ligne[i],surface(surfplane));
      if(iter[i].length==1) pinter.push(iter[i][0]);}}
  if(pinter.length==0 && intersec==nullpath3){
    warning("interplane","pas de point d'intersection");
    intersec=nullpath3;}
  else if(pinter.length==1 && intersec==nullpath3){
    warning("interplane","un seul point d'intersection");
    intersec=nullpath3;}
  else if(pinter.length>=2 && abs(pinter[0]-pinter[1])>=.0001 && intersec==nullpath3)
        intersec=pinter[0]--pinter[1];
  else if(pinter.length>2 && abs(pinter[0]-pinter[1])<.0001 && intersec==nullpath3)
        intersec=pinter[0]--pinter[2];
  return intersec;
}
```

Pour le tracé de l'intersection, il faudra juste s'assurer qu'elle existe ! (voir les deux dernières lignes du code 91)

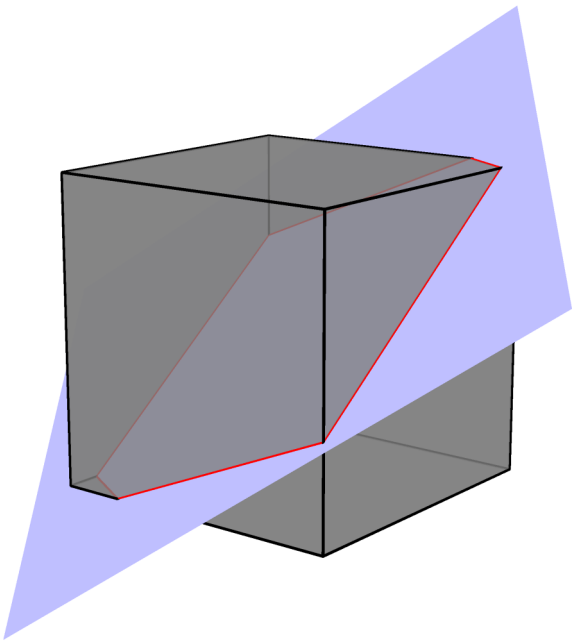
On pourra compiler les figures suivantes par la commande `asy -V` afin de pouvoir manipuler la figure (appuyez sur la touche `e` du clavier afin de produire le `pdf`, ou passez par le menu : double clic droit).



CODE 91

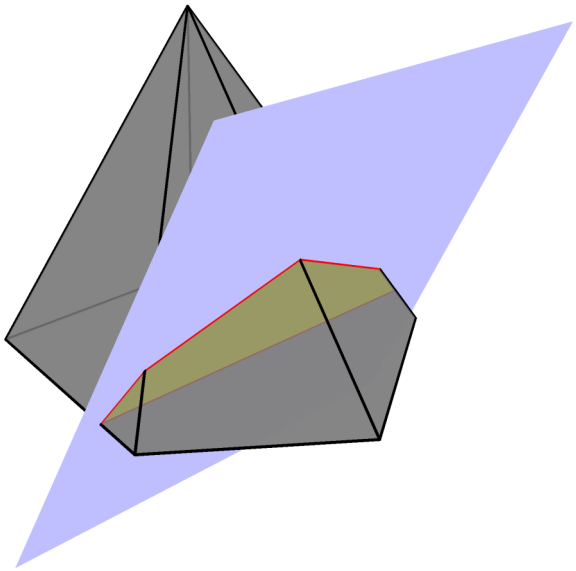
```
size(10cm,0);
import macros3D;
settings.prc=false;
settings.outformat="pdf";
currentprojection=obliqueX;
currentlight=nolight;

triple v1=(0,3,0), v2=(0,0,4), v3=(0,3,0),
       v4=(-3,0,.5), v5=(0,1.5,-1), v6=(-3,0,-.5);

path3 pl1=plane(v1,v2,0);
path3 pl2=plane(v3,v4,(1.5,0,2.5));
path3 pl3=plane(v5,v6,(2,.8,1.5));;

draw(surface(pl1),gray+opacity(.8));
draw(surface(pl2),paleblue);
draw(surface(pl3),paleblue);

if(interplane(pl2,pl1) != nullpath3) draw(interplane(pl2,pl1),1.2bp+red);
if(interplane(pl3,pl1) != nullpath3) draw(interplane(pl3,pl1),1.2bp+green);
```

Avec des polyèdres :



CODE 92

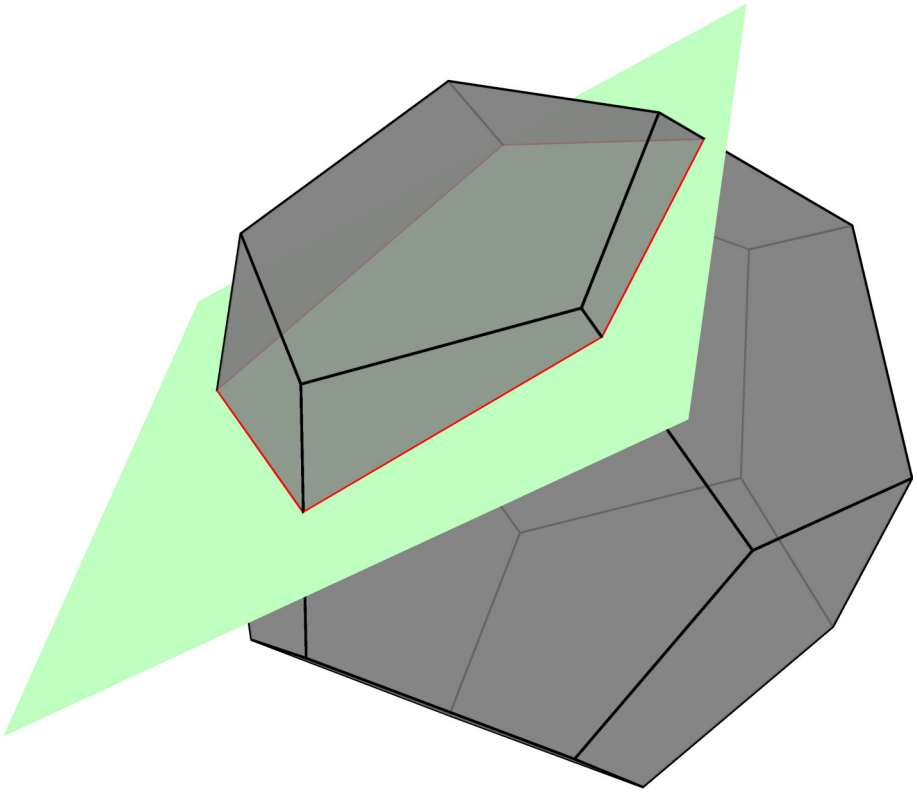
```
size(7.5cm,0);
import macros3D;
settings.prc=false;
settings.outformat="pdf";
settings.render=4;
currentlight=nolight;

triple A=(1,0,0), B=(1,1,0), C=(0,1,0),
       D=(1,0,1), pE=(1,1,1), F=(0,1,1),
       G=(0,0,1);

path3[] cube;
cube[0]=O--A--B--C--cycle;
cube[1]=O--A--D--G--cycle;
cube[2]=O--C--F--G--cycle;
cube[3]=pE--D--A--B--cycle;
cube[4]=pE--F--C--B--cycle;
cube[5]=pE--F--G--D--cycle;

triple v1=(2,0,0), v2=(0,2,0);
path3 plan=shift(-.1*X-Y+.3*Z)*
      rotate(50,X+Y+Z)*plane(v1,v2);
draw(surface(plan),paleblue);

for (int i=0; i<6; ++i){
  draw(surface(cube[i]),gray+opacity(.8),
       bp+black);
  if(interplane(cube[i],plan) != nullpath3)
    draw(interplane(cube[i],plan),.8bp+red);}
```



CODE 93

```
size(7.5cm,0);
import macros3D;
settings.prc=false;
settings.outformat="pdf";
settings.render=4;
currentlight=nolight;

triple A=(1,0,0), B=(1.5,1,0), C=(1,1.5,0),
       D=(0,1,0), pS=(0,0,1), pE=(-.5,.5,0);

path3[] pyramide={O--pE--D--C--B--A--cycle,
                  pS--O--A--cycle,
                  pS--A--B--cycle,
                  pS--B--C--cycle,
                  pS--C--D--cycle,
                  pS--D--pE--cycle,
                  pS--pE--O--cycle};

triple v1=(2,0,0), v2=(0,0.8,1.5);
path3 plan=plane(v1,v2,(-.5,.5,-.5));
path3 moninter;

for (int i=0; i<7; ++i){
  draw(surface(pyramide[i]),gray+opacity(.8));
  draw(pyramide[i],bp+black);
  if(interplane(pyramide[i],plan) != nullpath3)
    {
    moninter=moninter & interplane(pyramide[i],
                                   plan);}
}
moninter=moninter--cycle;
draw(moninter,.8bp+red);
draw(surface(reverse(moninter)^^plan,
     planar=true),paleblue);
draw(surface(moninter),yellow);
```


`interplane` est compatible avec polyhedron_js.asy (www.piprime.fr/asymptote/unofficial-asymptote-packages/) :



CODE 94

```
size(12cm,0);
import polyhedron_js;
import macros3D;
settings.prc=false;
settings.outformat="pdf";
settings.render=4;
currentlight=nolight;

polyhedron[] parr={dodecahedron};
filldraw(parr,new pen[]{gray},op=0.8);

triple v1=(2,0,0), v2=(0,2,0);
path3 plan=shift(-.1*X-1.5*Y+.1*Z)*rotate(45,X+Y+Z)*plane(v1,v2);
draw(surface(plan),palegreen);

for (int i=0; i<12; ++i){
  if(interplane(dodecahedron[i],plan) != nullpath3)
    draw(interplane(dodecahedron[i],plan),.8bp+red);}
```

ANNEXE F – FICHIER MACROS3D.ASY

CODE

```
/* Fichier de macros pour les figures3D
À placer dans le répertoire $HOME/.asy
Christophe Grospellier */

import three;

// -------- Définition de la perspective cavalière pcX -------

projection pcX(real k=.5, real angle=45)
{
  transform3 t=identity(4);
  real c2=Cos(angle)^2;
  real s2=1-c2;
  t[0][0]=-k*Cos(angle);
  t[1][0]=-k*Sin(angle);
  t[1][1]=0;
  t[0][1]=1;
  t[1][2]=1;
  t[2][2]=0;
  t[2][0]=1;
  t[2][3]=-1;
  return projection((1,c2,s2),normal=(1,0,0),
                    new transformation(triple,triple,triple) {
                      return transformation(t);});
}
projection pcX=pcX();

// -------- Définition de la perspective cavalière pcY -------

projection pcY(real k=.5, real angle=45)
{
  transform3 t=identity(4);
  real c2=Cos(angle)^2;
  real s2=1-c2;
  t[0][1]=k*Cos(angle);
  t[1][1]=k*Sin(angle);
  t[1][2]=1;
  t[2][1]=-1;
  t[2][2]=0;
  t[2][3]=-1;
  return projection((c2,-1,s2),normal=(0,-1,0),
                    new transformation(triple,triple,triple) {
                      return transformation(t);});
}

projection pcY=pcY();

// ------------------Suite page suivante-------------------------
```

```
void cote3D(picture pic=currentpicture,
                Label L="", triple A, triple B,
                real d=5mm, triple v, bool cc=true,
                pen p=currentpen, pen joinpen=dotted,
                arrowbar3 arrow=Arrows3)
{
        transform3 T=shift(d*unit(v));
        triple A=A, B=B;
        pic.add(new void(picture f, transform3 t) {
                picture opic;
                path3 dist;
                triple Ap=t*A, Bp=t*B;
                triple a=T*Ap, b=T*Bp;
                if (cc) {dist=a--b;}
                else {dist=b--a;}
                draw(opic,L,dist,p,arrow);
                draw(opic,a--Ap^^b--Bp,joinpen);
                add(f,opic);
                }, true);
}

// -------------drawrightangle de P. Ivaldi----------------------

void drawrightangle(picture pic=currentpicture,
                    triple M, triple A, triple B,
                    real radius=0,
                    pen p=currentpen,
                    pen fillpen=nullpen,
                    projection P=currentprojection)
{
  p=linejoin(0)+linecap(0)+p;
  if (radius==0) radius=arrowfactor*sqrt(2);
  transform3 T=shift(-M);
  triple OA=radius/sqrt(2)*unit(T*A),
    OB=radius/sqrt(2)*unit(T*B),
    OC=OA+OB;
  path3 _p=OA--OC--OB;
  picture pic_;
    draw(pic_, _p, p=p);
    if (fillpen!=nullpen) draw(pic_, surface(O--_p--cycle), fillpen);
  add(pic,pic_,M);
}

// -----------------Suite page suivante-----------------------
```


```
void markangle3D(picture pic=currentpicture,
                 Label L="",
                 triple M, triple A, triple B,
                 bool cc=true,
                 real radius=0,
                 pen p=currentpen,
                 pen fillpen=nullpen,
                 arrowbar3 arrow=None,
                 projection P=currentprojection)
{
  p=linejoin(0)+linecap(0)+p;
  if (radius==0) radius=arrowfactor*sqrt(2);
  transform3 T=shift(-M);
  triple OA=radius/sqrt(2)*unit(T*A),
    OB=radius/sqrt(2)*unit(T*B);
  path3 pl=O--OA--OB;
  triple V=normal(pl);
  path3 _p; real k;
  if (cc) k=1;
  else k=-1;
  picture pic_;
  _p=arc(O,OA,OB,k*V);
    if (fillpen!=nullpen) draw(pic_, surface(O--_p--cycle), fillpen);
  draw(pic_, L, _p, p=p, arrow);
  add(pic,pic_,M);
}

// --------- Intersections de plans ---------------------------

bool inplan(triple p, path3 surfplane) {
  triple n=normal(surfplane);
  triple[] int=intersectionpoints(p--shift(n)*p,surface(surfplane));
  return (int.length!=0 && abs(int[0]-p)<.0001);
}
path3 interplane(path3 plan1, path3 surfplane) {
  path3 intersec;
  path3[] ligne;
  triple[][] iter;
  triple[] pinter;
  int n=size(plan1);
  for (int i=0; i<n; ++i) {
    ligne[i]=point(plan1,i)--point(plan1,(i+1)%n);
    if(inplan(point(plan1,i),surfplane) && inplan(point(plan1,(i+1)%n),surfplane)) {
      intersec=intersec & ligne[i];}
    else {iter[i]=intersectionpoints(ligne[i],surface(surfplane));
          if(iter[i].length==1) pinter.push(iter[i][0]);}}
  if(pinter.length==0 && intersec==nullpath3){
    warning("interplane","pas de point d'intersection");
    intersec=nullpath3;}
  else if(pinter.length==1 && intersec==nullpath3){
    warning("interplane","un seul point d'intersection");
    intersec=nullpath3;}
  else if(pinter.length>=2 && abs(pinter[0]-pinter[1])>=.0001 && intersec==nullpath3)
         intersec=pinter[0]--pinter[1];
  else if(pinter.length>2 && abs(pinter[0]-pinter[1])<.0001 && intersec==nullpath3)
         intersec=pinter[0]--pinter[2];
  return intersec;
}
```

Fin

Index