Le modèle draw2d.mod 0.6

Pour TeXgraph 1.96

25 janvier 2012

Résumé

Ce modèle permet l'utilisation de marqueurs le long d'un chemin, ainsi que des remplissages avec un gradient.

Table des matières

1 Utilisation du modèle draw2d.mod **2**

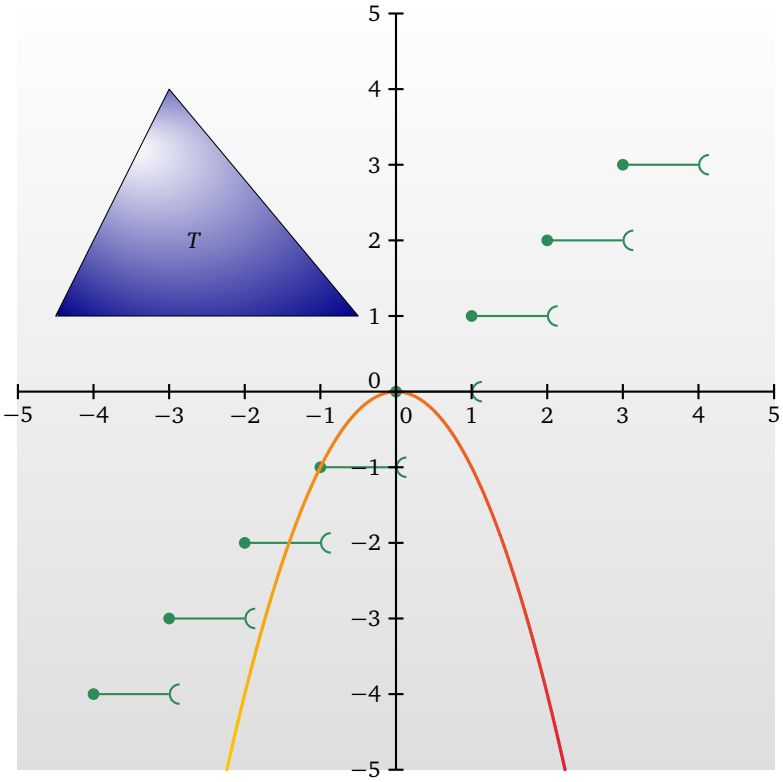
1.1 Présentation 2
1.2 Options pour des lignes en dégradé 2
1.3 Options pour des remplissages avec gradient . 2
1.4 Options pour les marqueurs 3
1.5 Valeur des options par défaut 3

2 Commandes de dessin **4**

2.1 Le type *dot* 4
2.2 Le type *label* 5
2.3 Le type *path* 5
2.4 Le type *line* 6
2.5 Le type *cartesian* 6
2.6 Le type *polar* 7
2.7 Le type *parametric* 7
2.8 Le type *seg* 8
2.9 Le type *interval* 9
2.10 Le type *straightL* 10
2.11 Le type *halfPlane* 10

3 Créer ses propres types **11**

3.1 Définition d'un type *circle* 11
3.2 Définition d'un type *domaine1* 12



1 Utilisation du modèle draw2d.mod

1.1 Présentation

Ce modèle commence par charger les trois fichiers de macros : *markers.mac*, *gradients.mac*, et *draw2d.mac* dans cet ordre (ces fichiers sont présents dans le dossier *../TeXgraph/macros*). Le premier fichier gère tout ce qui a attrait aux marqueurs de lignes, le deuxième gère les remplissages avec gradient ainsi que leurs différents exports (eps, pstricks, pgf, tikz, svg), le troisième fichier utilise les deux précédents et définit des macros de dessin qui sont toutes du type :

draw("type", <données>, [options])

Les types proposés pour le moment sont :

- *dot* : pour dessiner une liste de points. Un nouveau style de point est défini à l'occasion, c'est le style *pix* (pour pixel) ainsi que les exports correspondants,
- *label* : pour afficher un label, ce type est utilisé par les autres mis à part *dot*,
- *path* : pour dessiner un chemin, les éventuels marqueurs et gradients sont pris en compte dans les options, l'attribut **Arrows** est également pris en compte.
- *line* : pour dessiner une ligne polygonale, ce type hérite du type *path*,
- *cartesian* : pour dessiner une courbe cartésienne, ce type hérite du type *line*,
- *polar* : pour dessiner une courbe polaire, ce type hérite du type *line*,
- *parametric* : pour dessiner une courbe paramétrique, ce type hérite du type *line*,
- *seg* : pour dessiner un segment, ce type hérite du type *line*,
- *straightL* : pour dessiner une droite à partir de son équation ou de deux points, ce type hérite du type *line*,
- *halfPlane* : pour dessiner un demi-plan à partir d'une inéquation, ce type hérite du type *line*,

Les options sont une liste d'affectations <paramètre> := < valeur >, ces affectations sont **locales**. Les paramètres peuvent être des attributs prédéfinis comme **FillStyle**, **LineStyle**, ..., ou bien des paramètres définis par le modèle. La valeur par défaut des attributs prédéfinis est leur valeur courante au moment de l'appel à la commande **draw**. La valeur par défaut des paramètres définis par le modèle sera vue dans les sections suivantes.

1.2 Options pour des lignes en dégradé

Pour avoir un dégradé dans le tracé d'un contour, les options sont :

- LineStyle:=gradient annonce un tracé en dégradé, cette option pouvant être globale car par défaut c'est la valeur courante de **LineStyle** qui est prise en compte,
- LineColorA := < couleur > définit la couleur de départ, *white* par défaut,
- LineColorB := < couleur > définit la couleur d'arrivée, *red* par défaut,
- GradLineStep := < longueur en cm > définit la longueur des morceaux de couleur unie, 0.35 cm par défaut

1.3 Options pour des remplissages avec gradient

Pour avoir un dégradé dans le remplissage (gradient), les options sont :

- FillStyle:=gradient annonce un remplissage en dégradé, cette option pouvant être **globale** car par défaut c'est la valeur courante de **FillStyle** qui est prise en compte,
- FrenchBabel := < 0/1 > est une option **globale** qui vaut 0 au chargement du modèle, elle ne concerne que les exports *tikz* et *pgf* car il y a un conflit avec les caractères actifs de l'option *frenchb* paquet *babel* lorsque celui-ci est utilisé. La valeur 1 permet d'ajouter l'instruction `\shorthandoff{;:!?}` avant la déclaration du « shading » et l'instruction `\shorthandon{;:!?}` après cette même déclaration,
- FillColorA := < couleur > définit la couleur de départ, *white* par défaut,
- FillColorB := < couleur > définit la couleur d'arrivée, *red* par défaut,
- GradStyle := < linear/radial > définit le style de gradient, *linear* par défaut,
- GradAngle := < angle en degrés > définit l'angle par rapport à l'axe orienté (Ox) pour le style de gradient *linear*, il indique la direction du remplissage, cet angle vaut 0 par défaut c'est à dire un remplissage de la gauche vers la droite,
- RadialCenter := < point de $[0;1] \times [0;1]$ > localise le centre pour le style de gradient *radial*, c'est le point de départ du remplissage, c'est un complexe qui doit être dans le carré $[0;1] \times [0;1]$, par défaut c'est le point $0.25 + 0.75i$.

1.4 Options pour les marqueurs

Pour ajouter des marqueurs le long d'un chemin, il y a deux options et celles-ci seront appliquées à chaque **composante connexe de la ligne** :

- marker := < [pos1, mark1, pos2, mark2, ...] > définit la liste des marqueurs et leur position. Les positions (*pos1*, *pos2*, ...) sont des nombres compris entre 0 et 1, 0 indiquant le début de la composante et 1 la fin. Les valeurs *mark1*, *mark2*, ..., sont des valeurs représentant des marqueurs, ceux-ci sont énumérés dans la figure suivante.
- scale := < entier positif > permet de modifier la taille des marqueurs, sa valeur est de 1 par défaut.

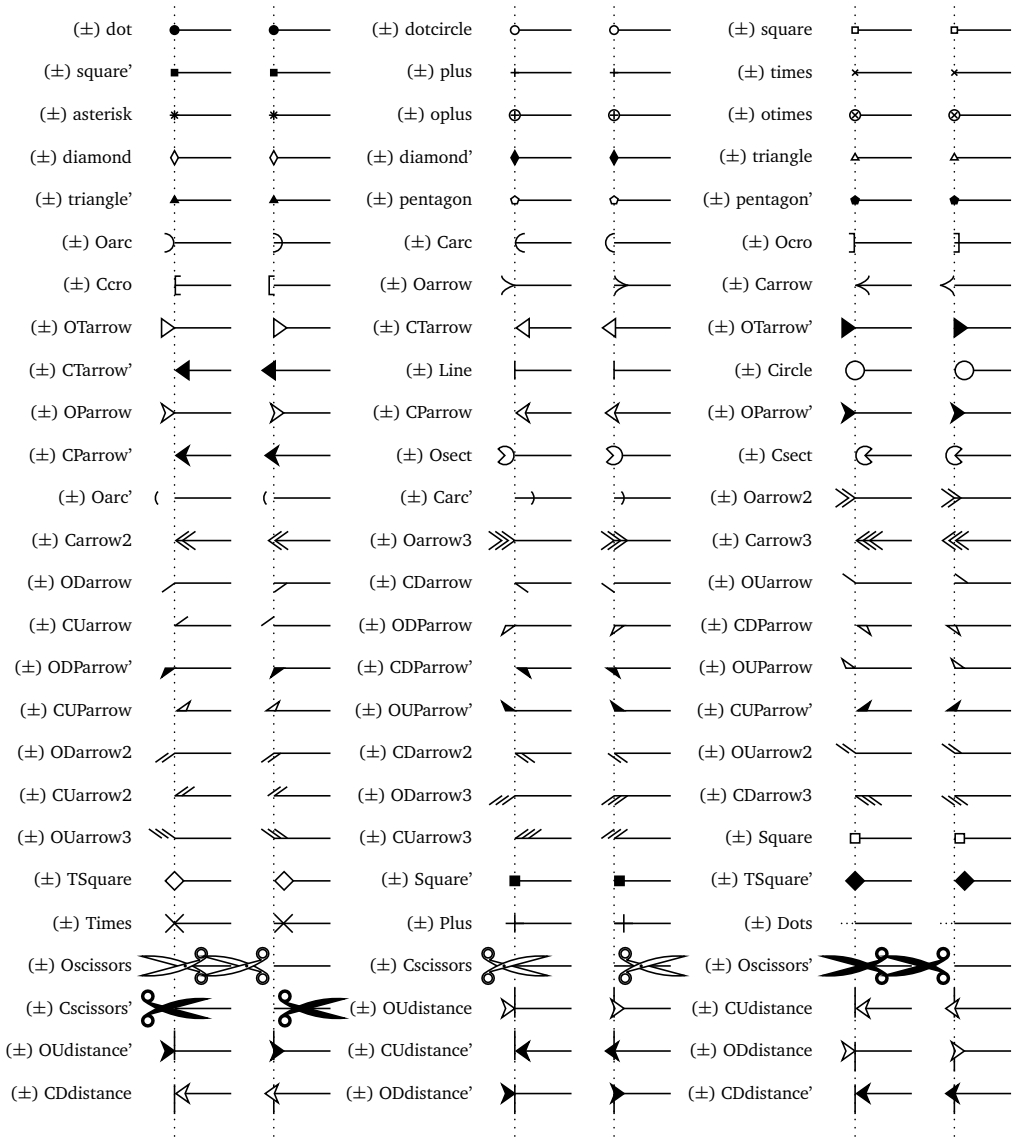


FIGURE 1: Listes des marqueurs

NB : par défaut un marqueur fermé se termine au même point que le segment, un marqueur ouvert est ajouté au bout du segment, pour inverser la situation il suffit de remplacer le marqueur par son opposé (*-Orc*, *-Carc*, ...).

Les types *path* et *line* (et donc tous les types qui utilisent ces deux là) testent l'attribut **Arrows** et ajoutent des marqueurs lorsque cet attribut vaut 1 ou 2. Par défaut la flèche ajoutée est *Carrow*, mais il est possible de changer la flèche par défaut avec le paramètre global :

CurrentArrow := < marker >

1.5 Valeur des options par défaut

Lors du chargement du modèle dans l'interface graphique, deux macros sont crées : *defaultGradLineOptions()* et *defaultShadePathOptions()*, celles-ci sont appelées par les macros de dessin et ont pour objectif de mettre les différentes options à leur valeur par défaut. En modifiant ces macros on peut changer les valeurs par défaut. Voici leur contenu :

Macro *defaultGradLineOptions()* :

```
[
LineColorA:=white {début},
LineColorB:=red {fin},
close:=0 {fermée ou non},
GradLineStep:=0.35 {pas}
]
```

Macro *defaultShadePathOptions()* :

```
[
FillColorA:=white {début},
FillColorB:=red {fin},
GradStyle:=linear {ou radial},
GradAngle:=0 {degrés},
RadialCenter:=0.25+0.75*i {dans [0;1]x[0;1]}
]
```

2 Commandes de dessin

2.1 Le type *dot*

draw("dot", <liste de points 2d>, [options])

Lorsque l'attribut **DotStyle** n'a pas la valeur *pix*, la commande a le même effet que la commande **Point<liste de points 2d>)** à la différence près que la modification des attributs prédéfinis dans les options est locale.

Lorsque l'attribut **DotStyle** a la valeur *pix*, les points sont dessinés sous forme de pixels, ceux-ci ne seront visibles que si l'élément graphique contient l'instruction : **NewBitmap()**. Un export vectoriel a été prévu en *eps* et en *tikz/pgf*, sinon un export bitmap est toujours possible.

Exemple

```
\begin{texgraph}[name=typedot, file]
Graph image = [
 view(2.5, 4, 0, 1), Marges(0.75, 0.5, 0.5, 0.75), size(12),
 Axes(Xmin+i*Ymin, (Xmax-Xmin)/3+i*(Ymax-Ymin)/5, 1+i),
 m :=Re(MaxPixels()), NewBitmap(lightgray),
 draw("dot",
        for r from 0 to m  do
           a :=Re(Pixel2Scr(r)),
           u :=0.5,
           for n from 1 to 50 do u :=a*u*(1-u) od,
           for n from 1 to 75 do u :=a*u*(1-u), a+i*u od
        od,
    [DotStyle :=pix, Color :=red])
];
\end{texgraph}
```

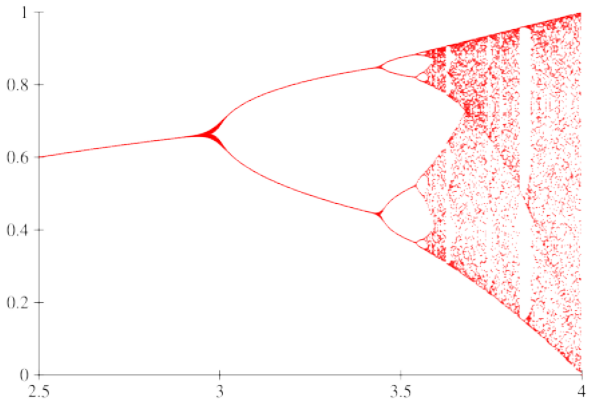


FIGURE 2: *Diagramme de bifurcation de la suite* $u_{n+1} = au_n(1-u_n)$.

L'exemple ci-dessus est un export *eps* converti en *png* dans l'interface graphique (bouton *Snapshot*). Il est à noter que l'export *tikz/pgf* sature très vite la mémoire de TEX lorsque le nombre de pixels augmente, l'export *eps* est donc à privilégier.

NB : le style de point *pix* n'est utilisable que dans l'interface graphique de TeXgraph.

2.2 Le type *label*

draw("label", "texte1", [options1], "texte2", [options2], ...)

Cette commande permet de placer un ou plusieurs labels, les options sont :

- anchor := < affixe > définit le point d'ancrage du label (valeur **Nil** par défaut),
- labeldir := < North/NE/East/SE/South/SW/West/NW > indique la position du label par rapport au point d'ancrage. La distance est de 0.25 cm par défaut et peut être modifiée avec le paramètre *labelsep*. On peut personnaliser le positionnement avec la syntaxe labeldir := < [distance, direction] > où *direction* est un vecteur non nul. Par défaut la valeur de ce paramètre est **Nil**.
- labelsep := < distance > pour modifier la distance du label au point d'ancrage lorsque le paramètre *labeldir* est une des huit constantes citées ci-dessus. La valeur est de 0.25 cm par défaut.
- showdot := < 0/1 > permet d'afficher ou non le point d'ancrage, la valeur est 0 par défaut.

La mise des options à leur valeur par défaut est faite une seule fois (avant l'évaluation de *[options1]*). Cette commande est utilisée par tous les types qui vont suivre. Afin d'éviter d'éventuelles incohérences dans les attributs lorsque le style *framed* est utilisé, la commande exécute la macro *framestyle()*. Cette macro n'existe pas par défaut et peut être créée ainsi :

```
setframestyle([FillStyle:=full, LineStyle:=noline])
```

L'option globale *TeXifyLabels := 0/1* permet d'avoir les labels compilés par TEX dans les exports *eps* et *svg*.

2.3 Le type *path*

draw("path", <commande path>, [options])

C'est la commande de base pour le dessin de chemins, elle utilise pour cela la commande **Path(<commande path>)**, mais elle permet également un remplissage avec un gradient linéaire ou radial, l'utilisation de marqueurs le long du chemin, ce chemin pouvant être lui-même dessiné avec un dégradé entre deux couleurs. On peut donc utiliser les options de marqueurs (page 3), les options de remplissage avec gradient (page 2) et les options de tracé de lignes en dégradé (page 2), en plus des attributs prédéfinis. À cela s'ajoute des options pour gérer un label :

- legend := < "texte" > permet d'ajouter un label, lorsque les paramètres *anchor* et *labelpos* sont à **Nil** (valeur par défaut), le texte est placé centre de gravité du chemin.
- labelcolor := < couleur > si le label doit être d'une autre couleur que le chemin.
- anchor := < affixe > définit le point d'ancrage du label (**Nil** par défaut), lorsque la valeur n'est pas donnée, c'est le paramètre *labelpos* qui est pris en compte.
- labelpos := < nombre dans [0 ;1] > permet de placer le label le long du chemin, la valeur 0 représente le premier point et la valeur 1 représente le dernier. La valeur par défaut est **Nil**.
- labeldir := < North/NE/East/SE/South/SW/West/NW > indique la position du label par rapport au point d'ancrage, mais lorsque les paramètres *anchor* et *labelpos* sont à **Nil** (valeur par défaut), il s'agit de la position par rapport au rectangle qui englobe le chemin. La distance au point d'ancrage est de 0.25 cm par défaut et peut être modifiée avec le paramètre *labelsep*. On peut personnaliser le positionnement par rapport au point d'ancrage avec la syntaxe labeldir := < [distance, direction] > où *direction* est un vecteur non nul. Par défaut la valeur de ce paramètre est **Nil**.
- labelsep := < distance > pour modifier la distance du label au point d'ancrage lorsque le paramètre *labeldir* est une des huit constantes citées ci-dessus. La valeur est de 0.25 cm par défaut.
- showdot := < 0/1 > permet d'afficher ou non le point d'ancrage, la valeur est 0 par défaut.

Et des options enrichissant le type de tracé de ligne :

- lineborder := < épaisseur >, ajoute une bordure de part et d'autre de la ligne de l'épaisseur voulue. Cette épaisseur est nulle par défaut (c'et à dire pas de bordure).
- bordercolor := < couleur >, précise la couleur de la bordure lorsqu'il y en a une, *white* par défaut.
- doubleline := < 0/1 >, permet de tracer ou non une double ligne.
- doublesep := < épaisseur >, épaisseur du trait central lorsqu'il y a double ligne. Par défaut, celle-ci est 1.25 fois l'épaisseur courante.
- doublecolor := < couleur >, précise la couleur du trait central lorsqu'il y a double ligne, *white* par défaut.

Exemple

```
\begin{texgraph}[name=typepath, file]
Include "draw2d.mod";
Graph image = [
 view(-5, 5, -5, 5), Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
 draw("path", [carre(Xmin+i*Ymin, Xmax+i*Ymin, 1), line, closepath],
          [FillStyle :=gradient, GradAngle :=-90, FillColorB :=lightgray,
           LineStyle :=noline]),
 draw("path", [-4.5+2*i, -2+2*i, circle], [doubleline :=1, doublecolor :=red, Width :=12,
          FillStyle :=gradient, GradStyle :=radial, FillColorB :=darkblue,
           legend :="$S$"]),
 draw("path", [-4-4*i, -1, 3-3*i, 4+i, bezier],
          [LineStyle :=gradient, LineColorA :=yellow, Width :=12, Arrows :=1,
           marker :=[0, Line, 0.5, Carrow], scale :=1.5,
           legend :="$L$", labelpos :=0.55, labeldir :=North]),
 draw("path", [carre(-1-0.6*i, 1.7-0.6*i, -1), line, closepath],
          [lineborder :=20, Color :=darkgreen, FillStyle :=full, FillOpacity :=0.3,
           FillColor :=cyan])
];
\end{texgraph}
```

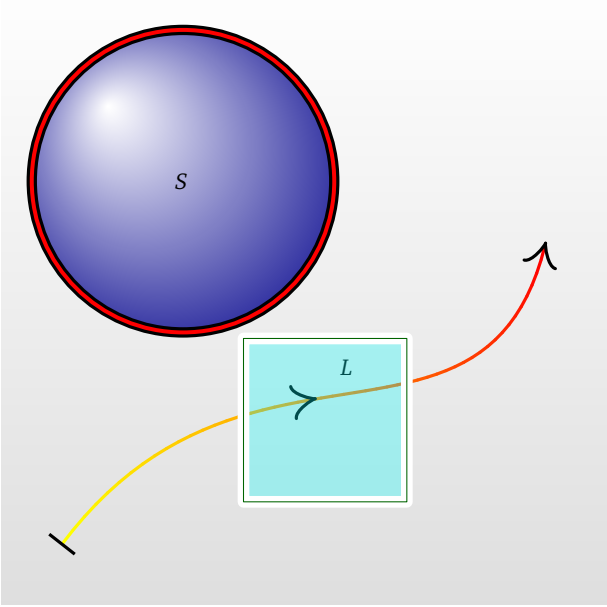


FIGURE 3: *Exemple avec le type* path.

Le type *path* suffit en théorie pour tout dessin de traits, cependant pour des raisons de simplicité d'utilisation, d'autres types ont été introduits.

2.4 Le type *line*

draw("line", <liste de points 2d>, [options])

Cette commande permet le tracé d'une ligne polygonale, comme la commande **Ligne()**, mais elle permet l'utilisation des options de marqueurs (page 3), des options de remplissage avec gradient (page 2), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Deux options supplémentaires sont disponibles pour le type *line* :

- close := < 0/1 > qui permet d'indiquer si la ligne doit être refermée ou non, si c'est le cas, chaque composante connexe sera alors refermée. La valeur par défaut est 0.
- radius := < longueur > permet d'arrondir les « angles » de la ligne polygonale. La valeur par défaut est 0.

2.5 Le type *cartesian*

draw("cartesian", <f(x)>, [options])

Cette commande trace une courbe cartésienne $y = f(x)$ en utilisant la commande **draw("line",...)** ce qui permet l'utilisation des options de marqueurs (page 3), des options de remplissage avec gradient (page 2), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Trois options supplémentaires sont disponibles pour le type *cartesian* :

- x := < [xmin,xmax] > qui définit l'intervalle de tracé, par défaut c'est l'intervalle [*tMin*, *tMax*], sauf quand l'attribut **ForMinToMax** vaut 1, auquel cas l'intervalle par défaut est [*Xmin*, *Xmax*],
- discont := < 0/1 > qui indique s'il faut prendre en compte ou non des discontinuités, la valeur par défaut est 0,
- nbdiv := < entier positif > indique le nombre de niveaux de dichotomie possibles pour le tracé, la valeur par défaut est de 5.

2.6 Le type *polar*

draw("polar", <r(t)>, [options])

Cette commande trace une courbe polaire $\rho = r(t)$ en utilisant la commande **draw("line",...)** ce qui permet l'utilisation des options de marqueurs (page 3), des options de remplissage avec gradient (page 2), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Trois options supplémentaires sont disponibles pour le type *polar* :

- t := < [tmin,tmax] > qui définit l'intervalle de tracé, par défaut c'est l'intervalle [*tMin*, *tMax*], sauf quand l'attribut **ForMinToMax** vaut 1, auquel cas l'intervalle par défaut est [*Xmin*, *Xmax*],
- discont := < 0/1 > qui indique s'il faut prendre en compte ou non des discontinuités, la valeur par défaut est 0,
- nbdiv := < entier positif > indique le nombre de niveau de dichotomies possibles pour le tracé, la valeur par défaut est de 5.

2.7 Le type *parametric*

draw("parametric", <f(t)>, [options])

Cette commande trace une courbe paramétrée par la fonction complexe $f(t)$ en utilisant la commande **draw("line",...)**, ce qui permet l'utilisation des options de marqueurs (page 3), des options de remplissage avec gradient (page 2), des options de tracé de lignes en dégradé (page2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Trois options supplémentaires sont disponibles pour le type *parametric* :

- t := < [tmin,tmax] > qui définit l'intervalle de tracé, par défaut c'est l'intervalle [*tMin*, *tMax*], sauf quand l'attribut **ForMinToMax** vaut 1, auquel cas l'intervalle par défaut est [*Xmin*, *Xmax*],
- discont := < 0/1 > qui indique s'il faut prendre en compte ou non des discontinuités, la valeur par défaut est 0,
- nbdiv := < entier positif > indique le nombre de niveau de dichotomies possibles pour le tracé, la valeur par défaut est de 5.

Exemple

```
\begin{texgraph}[name=courbes, file]
Include "draw2d.mod";
Graph image = [
 Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
 draw("line", carre(Xmin+i*Ymin, Xmax+i*Ymin, 1),
          [close :=1, LineStyle :=noline, FillStyle :=gradient, GradAngle :=-45,
           FillColorB :=darkgray]),
 draw("polar", 2*(1+cos(t)), [t :=[-pi, pi], Width :=12, FillStyle :=gradient, Width :=8,
           FillColorB :=darkblue, GradStyle :=radial, RadialCenter :=0.5+0.5*i,
           legend :="$\rho(t)=2(1+\cos(t))$", labeldir :=South]),
 draw("parametric", 2*(cos(2*t)+i*sin(3*t))-2.5+2.5*i,
              [t :=[pi/2, pi+pi/2], Width :=24, LineStyle :=gradient, LineColorA :=yellow,
              LineColorB :=red, GradLineStep :=0.25, Width :=8,
              legend :="$\cal C$", labelpos :=0.25, labeldir :=West] ),
 Axes(0, 1+i),
 draw("cartesian", Ent(x),
         [x :=[-4, 4], discont :=1, Color :=seagreen, marker :=[0, dot, 1, Oarc], Width :=8])
];
\end{texgraph}
```

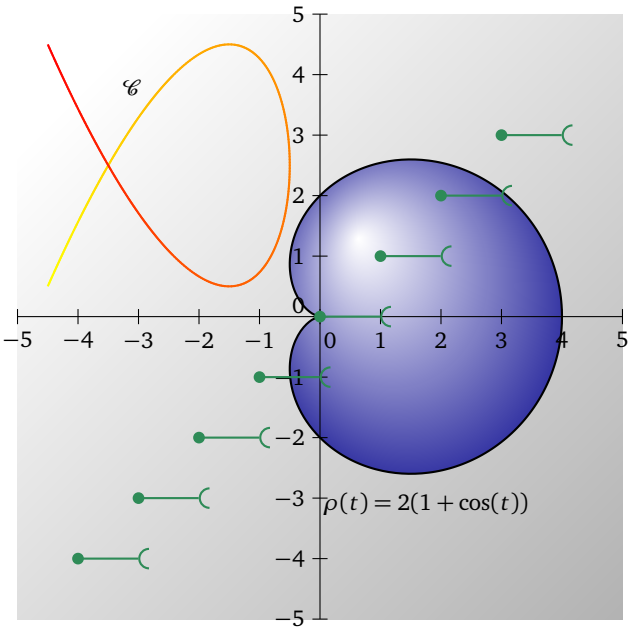


FIGURE 4: *Exemple de tracés de courbes.*

NB : on voit sur cet exemple que le départ en blanc du gradient de la cardioïde n'est pas au centre alors qu'il devrait l'être, ce problème se produit uniquement avec le code TeXgraph dans le source LaTeX, et manifestement c'est le `\pgfdeclareradialshading` de la figure précédente qui a été pris en compte. Il semblerait que la nouvelle déclaration n'ait pas effacé pas l'ancienne... Bizarrerie de pgf ?

2.8 Le type *seg*

draw("seg", <[A,B]>, [options])

Cette commande permet le tracé d'un segment avec l'utilisation des options de marqueurs (page 3), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Options disponibles pour la gestion automatique d'un éventuel label lorsque le paramètre *anchor* vaut **Nil** :

- rotation := < 0/1 > pour que le texte soit parallèle ou non au segment (1 par défaut),
- labelpos := < nombre dans [0; 1] > indique la position du point d'ancrage le long du segment. La valeur 0 correspond au premier point, et la valeur 1 au deuxième. Par défaut la valeur est de 0.5, ce qui correspond au milieu,
- labelsep := < distance > indique la distance entre le label et le segment, cette distance peut être négative (pour placer le label sous le segment), et sa valeur par défaut est de 0.25 cm,

Exemple

```
\begin{texgraph}[name=seg, file]
Include "draw2d.mod";
Graph image = [
view(-4, 4, -4, 4), Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
v :=-1.5*i, seg :=shift([-3+4*i, 3+5*i], v),
draw("seg", seg, [legend :="segment", Arrows :=1] ),
seg :=shift(seg, v),
draw("seg", seg, [legend :="segment", labelsep :=0, marker :=[0, Oarc, 1, Oarc] ] ),
seg :=shift(seg, v), setframestyle([FillStyle :=full, LineStyle :=noline, FillColor :=white]),
draw("seg", seg, [legend :="segment", labelsep :=0, Arrows :=2 ] ),
seg :=shift(seg, v),
draw("seg", seg, [legend :="segment", labelpos :=0.25, labelsep :=-0.25, LineStyle :=dashed,
            LabelStyle :=left, rotation :=0, marker :=[0, dot, 0.75, Cscissors, 1, dot]] ),
seg :=shift(seg, v),
draw("seg", seg, [legend :="segment", LabelStyle :=left, labelsep :=-0.25, showdot :=1,
            rotation :=0, LabelAngle :=-90, marker :=[0, Line, 1, CParrow]] )
];
\end{texgraph}
```

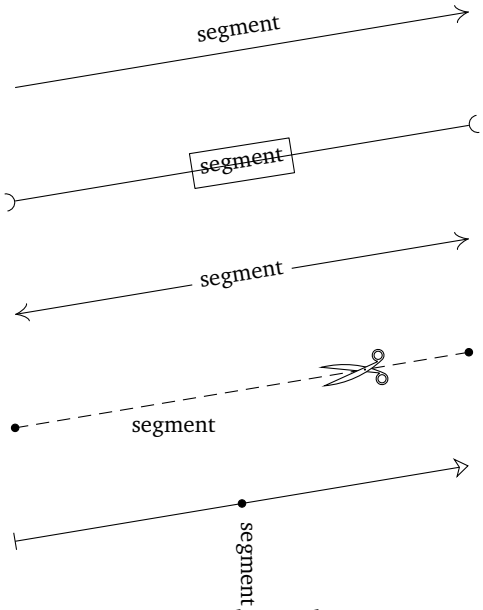


FIGURE 5: *Exemple avec le type* seg.

2.9 Le type *interval*

draw("interval", <[A,B]>, [options])

Cette commande permet le tracé d'un segment délimité par deux crochets ouverts ou fermés, symbolisant un intervalle, il est possible de hachurer cet intervalle. Les options sont :

- hatch := < 0/1 > pour hachurer l'intervalle (1 par défaut),
- hatchangle := < nombre en degrès > indique l'inclinaison des hachures par rapport au segment. Par défaut la valeur est de −45 degrés,
- marker := < "[]" ou "[[" ou "]]" ou "][" > indique le type d'intervalle, la valeur par défaut est "[]".

Ce type utlise le type *seg*, on peut donc utiliser les options de *seg*.

Exemple

```
\begin{texgraph}[name=interval, file]
Include "draw2d.mod";
Graph image = [
view(-4, 4, -1.5, 1.5), Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
dollar :=1, LabelSize :=footnotesize,
v :=i, axeX([v, 1], 1),
draw("interval", [-1+v, 3+v], [legend :="$(x+1)(x-3)<0$", labelsep :=0.5,
            marker :="][", Width :=8, Color :=crimson]
  ),
v :=-i, dollar :=1, axeX([v, 1], 1),
draw("interval", [-4.5+v, -1+v], [legend :="$(x+1)(x-3)\geqslant0$", labeldir :=North,
          anchor :=1+v, marker :="]]", Width :=8, Color :=darkblue]
  ),
draw("interval", [3+v, 4.5+v], [ marker :="[[", Width :=8, Color :=darkblue]
  )
];
\end{texgraph}
```

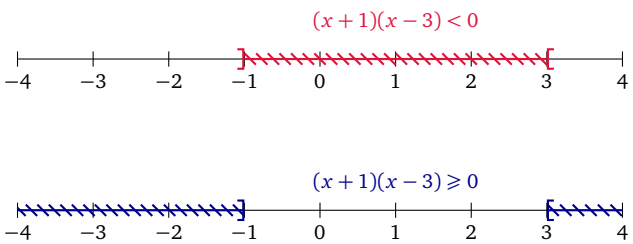


FIGURE 6: *Exemple avec le type* interval.

2.10 Le type *straightL*

draw("straightL", <a*x+b*y=c/[A,B]>, [options])

Cette commande permet le tracé d'une droite définie par une équation cartésienne de la forme $a*x+b*y=c$ ou bien deux points distincts $[A,B]$. La droite est intersectée avec la fenêtre graphique ce qui donne une segment, celui-ci est tracé avec le type *line*, ce qui permet l'utilisation des options de marqueurs (page 3), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Options disponibles pour la gestion automatique d'un éventuel label lorsque le paramètre *anchor* vaut **Nil** :

- rotation := < 0/1 > pour que le texte soit parallèle ou non la droite (0 par défaut),
- labelpos := < nombre dans [0; 1] > indique la position du point d'ancrage le long de la droite. La valeur 0 correspond au premier point, et la valeur 1 au deuxième. Par défaut la valeur est de 0.5, ce qui correspond au milieu,
- labelsep := < distance > indique la distance entre le label et le segment, cette distance peut être négative (pour placer le label sous la droite), et sa valeur par défaut est de 0.25 cm,

Exemple

```
\begin{texgraph}[name=straightL, file]
Include "draw2d.mod";
Graph image = [
view(-4, 4, -4, 4), Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
SaveAttr(), Width :=2, Color :=gray, Grille(0, 0.5*(1+i)), Width :=4,
Arrows :=1, Color :=black, Axes(0, 1+i), RestoreAttr(),
Width :=8,
draw("straightL", x+2*y=2, [Color :=blue, legend :="$x+2y=2$", rotation :=1]),
draw("straightL", x=-3, [Color :=crimson, labelpos :=0.1, legend :="$x=-3$",
                LabelStyle :=left]),
draw("straightL", [-2-2*i, 1], [Color :=darkgreen, labelpos :=0.85, legend :="$\cal D$"]),
draw("label", "$A$", [anchor :=-2-2*i, labeldir :=NW, showdot :=1],
          "$B$", [anchor :=1])
];
\end{texgraph}
```

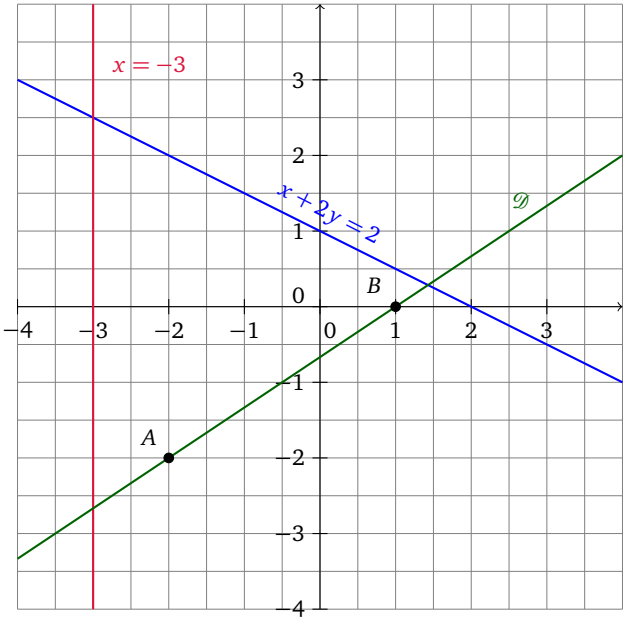


FIGURE 7: *Exemple avec le type* straightL.

2.11 Le type *halfPlane*

draw("halfPlane", a*x+b*y<=c, [options])

Cette commande permet le tracé d'une droite définie par une inéquation cartésienne de la forme $a*x+b*y<c$ ou $a*x+b*y<=c$ ou $a*x+b*y>c$ ou $a*x+b*y>=c$. Le demi-plan est intersecté avec la fenêtre graphique ce qui donne un polygone, celui-ci est tracé avec le type *line*, ce qui permet l'utilisation des options de marqueurs (page 3), des

options de remplissage avec gradient (page 2), des options de tracé de lignes en dégradé (page 2), des options pour tracer une ligne double ou bordée (page 5), et des options pour un éventuel label (page 5), en plus des attributs prédéfinis. Par défaut, le contour de ce polygone n'est pas visible (**LineStyle** réglé sur *noline*), et le remplissage est plein (**FillStyle** réglé sur *full*). La droite représentant la frontière est dessinée en trait plein dans le cas d'une inégalité large, en tirets sinon. Options disponibles pour la gestion automatique d'un éventuel label lorsque le paramètre *anchor* vaut **Nil** :

- rotation := < 0/1 > pour que le texte soit parallèle ou non au segment (1 par défaut),
- labelpos := < nombre dans [0; 1] > indique la position du point d'ancrage le long du bord. La valeur 0 correspond au premier point, et la valeur 1 au deuxième. Par défaut la valeur est de 0.5, ce qui correspond au milieu,
- labelsep := < distance > indique la distance entre le label et le segment, cette distance peut être négative (pour placer le label de l'autre côté), et sa valeur par défaut est de 0.25 cm,

Par défaut le label est placé dans le demi-plan, le long de la frontière, au milieu du segment visible.

Exemple

```
\begin{texgraph}[name=halfPlane, file]
Include "draw2d.mod";
Graph image = [
   view(-5, 5, -5, 5), Marges(0.5, 0, 0, 0.5), size(8), FrenchBabel :=1,
   SaveAttr(), Width :=2, Color :=gray, Grille(-5-5*i, 0.5+0.5*i),
   RestoreAttr(),
   Width :=8, FillOpacity :=0.6,
   draw("halfPlane" , x+2*y>=5, [FillColor :=navy, labelpos :=0.425,
                        legend :="$x+2y \leqslant 5$" ]),
   draw("halfPlane" , -5*x+2*y<2, [FillColor :=crimson, labelpos :=0.425,
                         legend :="$-5x+2y< 2$" ]),
   draw("halfPlane" , x<-2, [FillColor :=darkgreen, labelpos :=0.6,
                     legend :="$x<-2$" ]),
   draw("halfPlane" , y<=-1, [FillColor :=gold,
                      legend :="$y\leqslant -1$" ]),
   Width :=4, Arrows :=1, Axes(-5-5*i, 1+i, 1+i)
];
\end{texgraph}
```

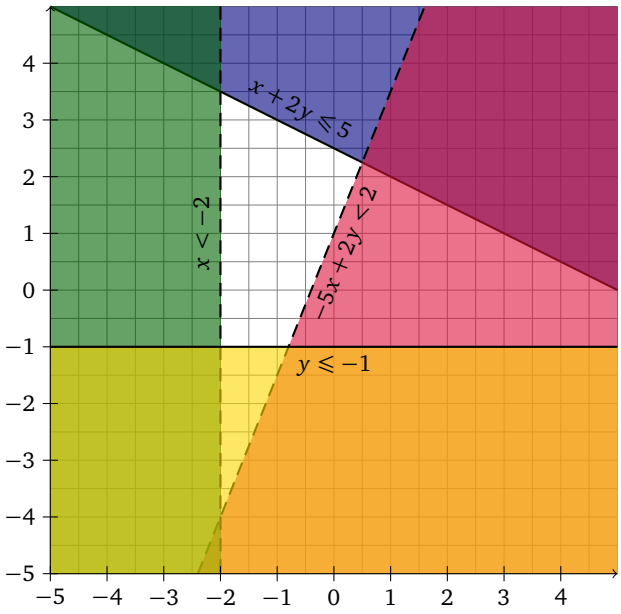


FIGURE 8: *Exemple avec le type* halfPlane.

3 Créer ses propres types

Donnons deux exemples :

3.1 Définition d'un type *circle*

Pour définir la commande **draw("circle", <données>, [options])**, il nous suffit de créer la macro :

Drawcircle(<données>, [options])

celle-ci cera automatiquement appelée par la commande **draw** pour effectuer le dessin, alors que pour un export c'est la macro **Exportcircle(<données>, [options])** qui sera appelée, si cette macro n'existe pas, c'est l'export par défaut qui aura lieu. Les données dont nous avons besoin pour dessiner un cercle sont : le centre et le rayon, ou bien trois points. Voici une définition possible de cette macro en utilisant le type *path* :

Drawcircle([données], [options])

```
[
$a:=%1[1], $b:=%1[2], $c:=%1[3],
if c=Nil then $x:=[a+b,a,circle] else x:=[%1,circle] fi,
draw("path", x, %2)
]
```

S'il n'y a que deux points dans les données, on convient qu'il s'agit du centre et du rayon, dans ce cas la commande pour définir le chemin est `[<point du cercle>, <centre>, circle]`, sinon on convient qu'il s'agit de trois point du cercle, auquel cas la commande est `[<point1>, <point2>, <point3>, circle]`. Toutes les options du type *path* peuvent être utilisées.

3.2 Définition d'un type *domaine1*

Nous allons reprendre l'idée de la macro prédéfinie *domaine1* pour dessiner la partie du plan comprise entre une courbe cartésienne d'équation $y = f(x)$, l'axe Ox et deux verticales $x = a$ et $x = b$, en utilsant le type *line*. On utilise les mêmes options que le type *cartesian* (page 7). Voici une définition possible de cette macro :

Exemple

```
\begin{texgraph}[name=domaine1, file]
Include "draw2d.mod";
Mac Drawdomaine1 = { Drawdomaine1( f(x), [options] ) }
      [SaveAttr(), x :=if ForMinToMax then [Xmin, Xmax] else [tMin, tMax] fi,
       discont :=0, nbdiv :=5, $options :=%2, tMin :=x[1], tMax :=x[2], x :=Nil,
       $L :=Get(Cartesienne(%1, nbdiv, discont), 0),
       draw("line", [tMin, for $z in L By jump do z od, tMax], [%2, radius :=0, close :=0]),
       RestoreAttr() ];
   f = sin(%1)^3*3;
Graph objet1 = [
       view(-4.5, 4.5, -3.5, 3.5), Marges(0, 0, 0, 0), size(8), FrenchBabel :=1,
       draw("domaine1", f(x), [ x :=[-4, 4], FillStyle :=gradient, GradAngle :=90]),
       draw("cartesian", f(x), [ x :=[-4, 4], Color :=darkblue, Width :=8]),
       Axes(0, 1+i) ];
\end{texgraph}
```

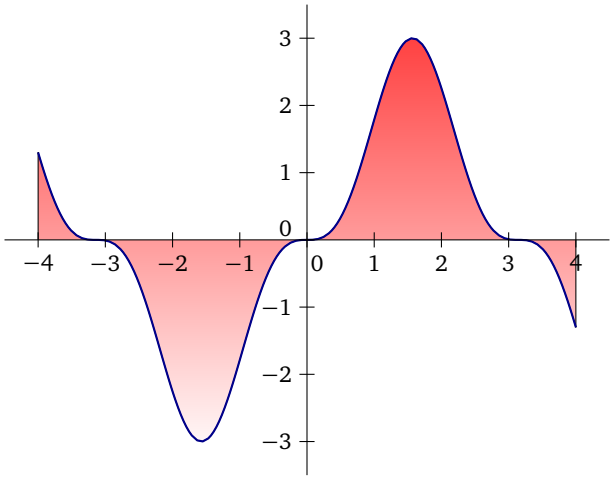


FIGURE 9: *Exemple d'ajout d'un nouveau type.*

Dans la macro *Drawdomaine1* : on sauvegarde les attributs prédéfinis. On initialise les paramètres *x*, *discont* et *nbdiv*. On évalue les options. On régle les attributs **tMin**, **tMax**. On calcule la liste des points de la courbe sans clipping, puis on passe

au tracé de la ligne, en ajoutant un point de l'axe Ox en début et en fin, en supprimant les éventuels *jump* de la liste, et en ajoutant `radius:=0,close:=0` aux options. On termine en restaurant les attributs prédéfinis.