

# Du nouveau côté *TikZ*

## Introduction

Après avoir publié la traduction du Tutoriel *TikZ* de Till Tantau (n° 48), les Cahiers GUTenberg ont publié le Petit manuel de prise en main de *TikZ* (n° 50) écrit pour la raison suivante : le texte de Till Tantau est excellent et donne envie d'aller plus loin ... ce qui signifie se plonger dans 350 pages de documentation où l'on ne trouve pas facilement ce que l'on cherche. Dans le petit manuel, les possibilités de *TikZ* ont été regroupées par chapitres de difficultés croissantes afin que ceux qui veulent s'initier à cet outil graphique puissent le faire même s'ils n'ont que peu de temps à y consacrer. Dans la conclusion de ce document, on regrettait l'impossibilité de réaliser certains tracés, tout en précisant que ce « manque » semblait n'être que provisoire.

Effectivement, la version 2.00 cvs de *TikZ* (juin 2010) permet, entre autres, le calcul des points d'intersection de deux courbes. Malgré la nécessité d'un calcul complémentaire, cette possibilité est utilisable pour quelques figures. On verra ce qui manque encore et comment y remédier. Il y a toujours les trois couches de langage :

- la couche *TikZ*, *frontend layer*, avec une syntaxe pas trop lourde et facilement utilisable ; il y a une version PlainTeX, une version LATEX et une version ConTeXt ;
- la couche de base, *basic layer* que l'on peut appeler couche pgf et qu'il faut parfois, hélas, utiliser en attendant l'achèvement de la prochaine version de *TikZ* stable ;
- la couche système, *system layer*, contenant le nécessaire pour générer du PDF, du PostScript, du HTML, ou encore un DVI portable.

Le présent document a la même présentation et la même motivation que le petit manuel ; il rendra d'autant plus service que la documentation (non terminée) a déjà 731 pages ! Il reprend d'abord les quatre premiers chapitres du petit manuel, section par section, et ajoute les nouvelles possibilités ainsi que les modifications de syntaxe (qu'il n'est pas nécessaire de respecter grâce à des modules de compatibilité disponibles dans la distribution). On trouvera ensuite deux chapitres concernant trois nouvelles possibilités importantes : calcul des points d'intersection de courbes, tracé de tronçons de courbe et visualisation de données un peu « à la MetaPost ».

Les nouvelles possibilités utilisent une syntaxe avancée de mots clés nécessitant le chargement du module spécifique `pgfkeys.tex` qui peut être utilisé par ailleurs. La version 2.00, et donc la version 2.00 cvs également, contiennent un très grand nombre de fichiers ; cela facilite le développement par de nombreux volontaires sans trop perturber l'utilisateur. Par contre, cela rend tout travail de synthèse particulièrement pénible.

Ce document est présenté comme le petit manuel avec de nombreux exemples, en particulier pour les nouvelles possibilités. Les références [x], [x.x] et [x.x.x] se rapportent aux sections, sous-sections et paragraphes de la doc de la version 2.00 cvs du 2 juin 2010. Seule, cette dernière version contient les commandes correspondant aux trois nouvelles possibilités qui font l'objet des deux derniers chapitres. Cette version cvs est en cours de développement et peut présenter des instabilités.

Ce travail, certainement très imparfait, est mis à disposition en espérant qu'il pourra apporter quelque chose à ceux qui apprécient *TikZ*. Si le petit manuel est fait pour les débutants, ce document s'adresse à ceux qui ont déjà une petite pratique.

Que les *TikZniciens* me pardonnent les fautes d'orthographe et les éventuelles erreurs.

Yves Soulet, rentrée 2010

## Avertissement

- L'ensemble des exemples présentés dans ce document exigent :
- la version *TikZ* 2.00 cvs du 2 juin 2010 ; bien qu'en développement, donc pouvant être instable, elle peut rendre des services telle qu'elle est ;
  - les chargements des fichiers suivants :

```
\usepackage{tikz}
\input{pgfkeys}
\usetikzlibrary{calc}
\usetikzlibrary{shapes}
\usetikzlibrary{backgrounds}
\usetikzlibrary{intersections}
\usetikzlibrary{through}
\usetikzlibrary{plotmarks}
\usetikzlibrary{datavisualization}
\usetikzlibrary{datavisualization.formats.functions}
```
  - le fichier de macros *suptikz.tex* (joint au présent document) qui contient les définitions des quatre macros de lettrage *\pose*, *\poseb*, *\pos* et *\posb* utilisées ainsi que trois macros commodes pour les axes et les grilles *\axes*, *\axesgrille* et *\axesgrillefine*.

Les nouveautés importantes exposées dans ce document concernent les points d'intersection des courbes, le tracé de tronçons de courbes et la visualisation des données mais cette version contient beaucoup d'autres choses nouvelles et utiles ; voilà une liste de petites possibilités pouvant intéresser des utilisateurs :

- positionnement absolu par rapport à la page : on peut relier un point donné d'une figure à un point donné d'une autre figure (si les deux figures sont sur la même page !) ;
- écriture sur une courbe quelconque ;
- placement de flèches sur des courbes aux endroits souhaités ;
- tracé d'un cercle ou d'un rectangle circonscrit à une partie de figure ;
- affichage des points de contrôle des courbes de Bézier ;
- loupe pour « zoomer » sur une partie de la figure exigeant un grossissement ;
- et toutes les autres que l'auteur de ces lignes n'a pas encore découvertes dans les actuelles 731 pages de documentation !

Enfin, il faut préciser que ce document n'a pas de conclusion ; ce qui pourrait en être une peut-être dit dès l'introduction : *TikZ* est en plein développement, il est puissant et agréable à utiliser ... mais aura-t-il un jour une documentation de référence de lecture aisée, bien ordonnée et pas trop volumineuse où sont bien différenciées les bases et les applications vers les disciplines ?

# Coordonnées

## 1.1 Coordonnées cartésiennes

Pas de modification pour les coordonnées cartésiennes [13.2.1], mais une option qui peut rendre service ; si on écrit :

```
\begin{tikzpicture}[x=1mm,y=1mm,scale=2]
```

toutes les dimensions affectées à l'intérieur de l'environnement, et uniquement celles-là, seront doublées ; si l'on trace une courbe avec l'option `line width=0.8pt`, elle aura une épaisseur de 1,6 pt alors que la courbe tracée avec l'option `very thick` aura encore l'épaisseur 0,8 pt.

## 1.2 Coordonnées polaires

Pas de modifications pour les coordonnées polaires [13.2.2] ; l'option `scale` ne porte évidemment que sur la coordonnée rayon.

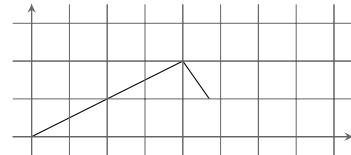
## 1.3 Déplacements et tracés relatifs

Pas de modification pour les définitions correspondantes [13.4.1].

## 1.4 Transfert de coordonnées entre T<sub>E</sub>X et Ti<sub>k</sub>Z et vice versa

Il peut arriver que l'on veuille utiliser des dimensions ou des compteurs déjà définis et affectés d'une valeur avant la figure [14.18] ; en quelque sorte, on peut dire qu'il s'agit d'un transfert de valeurs de T<sub>E</sub>X à Ti<sub>k</sub>Z. Pour cela, on procède comme dans l'exemple suivant :

```
\newlength{\longa}\newcounter{compta}
\setlength{\longa}{20mm}\setcounter{compta}{10}
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)--(\longa,\value{compta})
--(\longa+10,\value{compta}-5);
\end{tikzpicture}
```



Mais il se peut que l'on veuille faire un transfert dans le sens inverse, c'est-à-dire utiliser plus loin une donnée calculée au milieu d'une commande Ti<sub>k</sub>Z pour l'utiliser dans une commande T<sub>E</sub>X ; cela se fait avec la commande pgf<sup>(1)</sup> :

```
\pgfextra{commande TEX}
```

On donne à la section suivante un exemple de ce type de transfert utilisant cette commande.

---

<sup>(1)</sup> Il faut bien garder en mémoire que les commandes pgf sont des commandes T<sub>E</sub>X.

## 1.5 Désignation des points, des paires et des nombres

Voilà deux possibilités initialement motivées par la géométrie élémentaire, mais dont au moins la première peut rendre service pour beaucoup de types de figure.

On rappelle que, étant donnée une paire de coordonnées, on peut définir le point correspondant par [4.1.2] :

```
\coordinate(A)at(0,5);
```

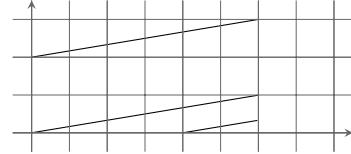
(A) peut être utilisé comme (0,5) dans les tracés.

On a la nouvelle possibilité de représenter une paire de coordonnées ou une paire de dimensions par les macros `\p1`, `\p2`, ... `\p9` ou encore `\p{nom}` en écrivant des définitions du type :

```
let \p1=($(A)$) in, let \p1=($(B)-(A)$) in, ...
```

Dans le premier exemple ci-dessus, `\p1` représente les coordonnées du point *A* alors que dans le deuxième `\p1` représente la paire de composantes du vecteur  $\vec{AB}$ ; lorsque `\p1`, `\p2...` est affecté d'une paire, alors `\x1`, `\x2...` est automatiquement affectée du premier élément de la paire et `\y1`, `\y2...` est affectée du deuxième élément. En outre, on dispose des macros `\n1`, `\n2...` pour transférer des nombres. Voilà des exemples d'utilisation :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\coordinate(A)at(0,10);\coordinate(B)at(30,15);
\draw let \p1=($(A)$),\p2=($(B)$)in (\p1)--(\x2,\y2);
\draw let \p1=($(B)-(A)$),\n1={veclen(\x1,\y1)},
\n2={atan(\y1/\x1)}in
\pgfextra{\xdef\angle{\n2}\xdef\longueur{\n1}}
(0,0)--(\n2:\n1);
%\draw(20,0)--+(\n2:10);%non
\draw(20,0)--+(\angle:10);%oui
\axesgrille(-2.5mm,-2.5mm)(42.5mm,17.5mm)
\end{tikzpicture}
```



Ici, le premier tracé montre bien que `\x2` et `\y2` représentent les éléments de la paire `\p2`. Contrairement à ce que l'on pourrait croire, (`\p1`) n'est pas tout à fait équivalent (A) (point délicat sur lequel on reviendra).

Dans le deuxième tracé, `\p1` représente la paire des composantes de  $\vec{AB}$ ; la longueur de ce vecteur, calculée par la macro `veclen`, est affectée à `\n1` et l'angle ( $\vec{Ox}, \vec{AB}$ ) est affecté à `\n2`. Il faut souligner que les macros du type `\p1`, `\x1`, `\y1` et `\n1` ne sont valables que pour le tracé dans la commande duquel elles sont définies.

Cet exemple permet de montrer comment, malgré la remarque précédente, on peut faire des transferts de TikZ vers TeX en utilisant la commande de transfert `\pgfextra` annoncée dans la section précédente. Placée dans la commande de tracé, après les affectations, elle permet d'exécuter des commandes TeX, des définitions dans l'exemple précédent, qui sauvegardent les valeurs affectées à `\n1` et `\n2` pour une utilisation ultérieure dans les figures suivantes !

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)--(\angle:\longueur);
\axesgrille(-2.5mm,-2.5mm)(42.5mm,7.5mm)
\end{tikzpicture}
```



# Lignes brisées

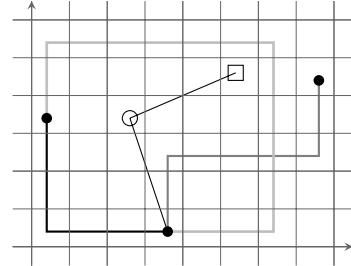
## 2.1 Syntaxe de base

Pas de modification pour ce type de tracé [14.2.1] ; on va simplement rappeler une possibilité intéressante qui sera revue avec les liaisons inter noeuds.

Le tracé d'un segment d'un point à un autre point, `--`, peut être remplacé par `-|` ou `|-` qui tracent respectivement un segment horizontal suivi d'un segment vertical ou un segment vertical suivi d'un segment horizontal [14.2.2] ; bien sûr, ceci n'est possible que si les deux coordonnées des deux points sont différentes. Avec la donnée d'un point supplémentaire, on peut faire des liaisons à 3 ou 4 segments.

Sur l'exemple, on teste aussi une nouvelle possibilité permettant de placer à chaque sommet de la ligne brisée un objet graphique uniquement constitué d'un chemin [14]. La complication nécessaire pour certains chemins, le rectangle par exemple, et l'impossibilité de remplir ce chemin s'il est fermé font perdre beaucoup d'intérêt à cette construction. Par contre, elle donne l'occasion d'introduire la nouvelle syntaxe pour créer des styles : on reverra cela en détails au chapitre 3.

```
\def\ptn(#1)(#2){\fill(#1)circle(2pt)}
\begin{tikzpicture}[x=1mm,y=1mm]
\coordinate(A) at (18,2); \coordinate(B) at (2,17);
\coordinate(C) at (38,22);
\draw[thick] (A)-|(B);
\draw[gray, line width=0.8pt] (A)--(18,12)-|(C);
\draw[lightgray, line width=1pt] (A)-|(32,27)-|(B);
\ptn(A)(1.5);\ptn(C)(1.5);\ptn(B)(2);
\tikzset{stycirc/.style={insert path={circle(1)}}}
\tikzset{styrect/.style={insert path={+(-1,-1)rectangle+(1,1)}}}
\draw(A)--(13,17)[stycirc]--(27,23)[styrect];
\axesgrille(-2.5mm,-2.5mm)(42.5mm,32.5mm)
\end{tikzpicture}
```



## 2.2 Options principales

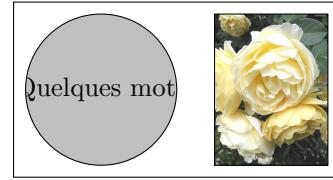
Pas de modification [15], mais trois nouvelles options possibles dont l'intérêt n'est pas évident pour un débutant mais qui peuvent être très utiles dans la production de certains types de figures.

La première permet de remplir un chemin fermé avec des motifs répétés disponibles uniquement en chargeant la bibliothèque `patterns` [15.4.1] ou en la complétant. Voilà des exemples :

```
\draw[pattern=dots](1,1)circle(20);
\draw[pattern=bricks](1,1)circle(20);
```

La suivante permet de remplir un chemin fermé avec un objet graphique disponible [15.5], cela grâce à l'utilisation de l'option `path picture` et de la commande `\includegraphics`. On donne deux exemples.

```
\begin{tikzpicture}[x=1mm,y=1mm,show background rectangle]
\draw[fill=lightgray](0,0)circle(10)[path
picture={\node at(path picture bounding box.center)
{Quelques mots};}];
\draw(15,-10)rectangle(30,10)[path picture=%
\node at (path picture bounding box.center)%
{\includegraphics[height=2cm]{begonia.jpg}};];
\end{tikzpicture}
```



Enfin la troisième nouvelle possibilité [15.7] : la boundingbox faite par TikZ est le plus petit rectangle contenant la figure, ce qui permet à TeX de réserver la place de la figure. Mais on pourrait vouloir, pour des travaux très spécifiques, que la figure déborde sur le texte environnant ou, au contraire, que la figure soit placée dans un espace vierge plus étendu que la figure. Il suffit pour cela d'écrire une commande du type :

```
\path[use as bounding box](0,0)rectangle(40,40);
```

## 2.3 Options de tracé

Mis à part quelques ajouts d'utilité très spécifique, on retrouve l'essentiel sans modification : épaisseur de traits, terminaisons, jonctions, pointillés et traitillés [13.3.1] et [13.3.2], flèches [4], traits doubles [5], angles arrondis [14.6]. On note seulement que l'on dispose d'une bibliothèque `arrows` particulièrement bien fournie pour les flèches.

En ce qui concerne la couleur, on dispose, en plus des quelques couleurs définies par défaut, de la doc du package `xcolor`, dont TikZ a adopté la syntaxe et dans laquelle on trouve plusieurs tables de couleurs.

## 2.4 Tracés prédéfinis

Les tracés prédéfinis, chemin fermé `rectangle` [2.6] et grille `grid` [2.7], ne font pas l'objet de modification.

# La machinerie TikZ

## 3.1 Préparations préliminaires

Installation des fichiers :

Il faut prendre la distribution à l'adresse<sup>(1)</sup> :

<http://www.texample.net/tikz/builds/> pour la version 2.00 ainsi que pour la version 2.00 cvs (version sous développement, pas forcément stable mais intéressante car proposant les trois nouvelles possibilités citées au troisième paragraphe de l'introduction. Ces distributions sont organisées comme celles de la version 1.10 (utilisée lors de la rédaction du petit manuel de prise en main). L'installation est donc celle du petit manuel.

Ajouts au préambule :

Il faut charger les packages `tikz` et `xcolor` et charger la bibliothèque pour les calculs avec la commande [12.2.1] :

`\usetikzlibrary{calc}`

(au lieu du package `calc`). Il faut aussi charger le package `pgfkeys` joint à la distribution (au lieu du package `keyval` ; il est aussi utilisable en dehors de TikZ par les sorciers de la technique des mots-clés) ; on verra plus loin son utilité.

Pour les autres bibliothèques, TikZ indique parfois dans les messages d'erreur la bibliothèque nécessaire. Dans le présent document les bibliothèques utilisées dans les exemples sont mentionnées dans l'avertissement. En dernier recours, il y a la doc de la version utilisée (il y a beaucoup de développeurs sur ce projet et donc, beaucoup de développements qui impliquent des changements complets des fichiers : divisions, regroupements, nouvelles bibliothèques, etc.).

## 3.2 Directives générales d'utilisation

Il n'y a pas de modification en ce qui concerne l'environnement de base des figures, leur intégration et leur positionnement par rapport à la ligne de base T<sub>E</sub>X [12.2.1]. Par contre l'adjontion d'un fond [12.2.3] et [25] est modifiée : il faut obligatoirement charger la bibliothèque `backgrounds` et ajouter l'option `show background rectangle` qui, par défaut, trace un rectangle extérieur à la boundingbox (premier exemple). On peut aussi très simplement définir le style `background rectangle` pour obtenir une présentation personnelle (un fond gris sur le deuxième exemple). On peut, comme auparavant, modifier la distance entre la boundingbox et la limite du fond.

```
\begin{tikzpicture}[x=1mm,y=1mm,show background rectangle]
\draw(0,0)--(30,5);
\end{tikzpicture}

\tikzset{background rectangle/.style={fill=gray!20}}
\begin{tikzpicture}[x=1mm,y=1mm,show background rectangle]
\draw(0,0)--(30,5);
\end{tikzpicture}
```




---

<sup>(1)</sup>Merci à A. Mattes pour m'avoir communiqué cette adresse.

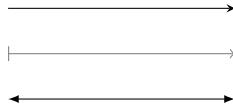
### 3.3 Manipulation des options

Il n'y a pas de modification concernant le placement des options en option de figure, de partie de figure (environnement `scope`, cf. [12.3.1]), de commande et parfois de spécification de commande. Il faut simplement signaler une abréviation [12.3.1] pour l'environnement `scope` (qui ne fonctionne que si l'on charge la bibliothèque `scopes`) ; pour l'utiliser, il suffit alors d'écrire :  
`\tikzset{[options]}` ensemble des commandes de la partie de figure  
comme cela était possible pour le placement en option de spécification de commande (exemple de la ligne brisée dont on ne veut arrondir qu'une suite de quelques angles).

On doit maintenant, pour déclarer des options, utiliser la commande [12.4.1] :  
`\tikzset{liste d'options séparées par des virgules}`

Si cette commande est placée à l'extérieur de l'environnement `tikzpicture`, les options sont utilisées pour toutes les commandes qui suivent, y compris celles des figures suivantes ; si elle est placée à l'intérieur de l'environnement `tikzpicture` ou d'un environnement `scope`, elles sont utilisées jusqu'à la fin de cet environnement. Voici quelques exemples.

```
\tikzset{>=stealth}
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[->](0,12)--(30,12);
\begin{scope}[gray]
\tikzset{>=to}
\draw[|->](0,6)--(30,6);
\end{scope}
\draw[<-,>=latex](0,0)--(30,0);
\begin{scope}[gray]
\end{scope}
\end{tikzpicture}
```



### 3.4 Définition de styles par l'utilisateur

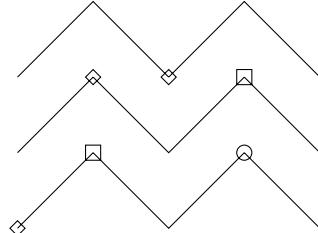
Commande de définition d'un style :

La syntaxe de définition d'un style est devenue [2.8] :

```
\tikzset{styleperso/.style={liste d'options séparées par des virgules}}
```

Si cette définition est placée à l'extérieur de l'environnement `tikzpicture`, elle est utilisable pour toutes les figures qui suivent (cf. fig. sect. 2.1) ; si elle est placée à l'intérieur, elle est utilisable jusqu'à la fin de l'environnement (cf. 2ième fig. sect. 3.2). Mais elle peut elle-même être placée comme une option de figure [14], de partie de figure [6.4] et de commande [13.3.2]. Voilà des exemples :

```
\begin{tikzpicture}[x=1mm,y=1mm,
  stylos/.style={insert path={%
    +(0,-1)---+(1,0)---+(0,1)---+(-1,0)---cycle+(0,0)}},
  \draw(0,20)--(10,30)--(20,20)[stylos]--(30,30)--(40,20);
  \begin{scope}%
  [styrec/.style={insert path={+(-1,-1)rectangle+(1,1)+(0,0)}},
  \draw(0,10)--(10,20)[stylos]--(20,10)--(30,20)[styrec]--(40,10);
  \draw[stycir/.style={insert path={circle(1)}}]%
  (0,0)[stylos]--(10,10)[styrec]--(20,0)--(30,10)[stycir]--(40,0);
  \end{scope}
  \end{tikzpicture}
```

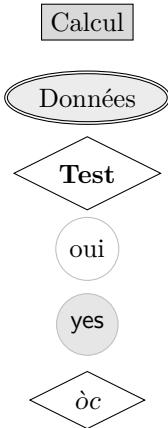


Commande de définition de styles s'utilisant automatiquement :

Le principe de ce type de style est conservé ; bien entendu c'est la syntaxe de définition qui change : dans la définition précédente, le nom du style défini est remplacé par `every dacos` où `dacos` est « l'élément graphique » qui sera affecté par ce nouveau style ; ça peut être un noeud, `node`, une forme, `circle`, un label, `label`, un environnement tel que `scope`, etc. ; il suffit de se reporter à l'index, mot `every`, pour voir le domaine d'application de ce type de style.

Dans l'exemple qui suit, repris du petit manuel, on a montré différentes positions possibles des définitions des styles associés aux noeuds suivant leur forme : avant l'environnement `tikzpicture`, à l'intérieur de cet environnement puis en option de figure.

```
\tikzset{cercle/.style={circle,draw=gray!50}}
%\pgfkeys{/tikz/cercle/.style={circle,draw=gray!50}}
\tikzset{every rectangle node/.style={draw,fill=gray!30}}
\begin{tikzpicture}[x=1mm,y=1mm,
    every ellipse node/.style={draw,double,fill=gray!15}]
\tikzset{every diamond node/.style={aspect=2,draw,font=\bfseries}}
\node (a) at (20,30) [rectangle] {Calcul} ;
\node (b) at (20,20) [ellipse] {Données} ;
\node (c) at (20,10) [diamond] {Test} ;
\node (d) at (20,0) [cercle] {oui} ;
\tikzset{cercle/.append style={fill=gray!15,font=\sfseries}}
\node (e) at (20,-10) [cercle] {yes} ;
\tikzset{every diamond node/.append style={font=\slshape}}
\node (f) at (20,-20) [diamond] {\`oc} ;
\end{tikzpicture}
```



Ensuite, on montre comment un style, `cercle` dans le cas présent, peut être modifié, avec le préfixe `append`, par un ajout de fond gris et un changement de fonte.

Enfin, on montre que l'on peut aussi modifier les styles définis avec le préfixe `every`, dans le cas présent `every diamond node`, par un changement de fonte.

Toutes ces manipulations sont montrées ici pour familiariser le lecteur avec la « machinerie TikZ » ; il est bien évident que pour produire un lot de figures pour un document, dans un but pédagogique, on définit des styles correspondant à des classes d'éléments bien déterminés et on les utilise sans les modifier en cours de route !

### 3.5 Généralisation du système de clés-valeurs

Comme déjà annoncé, les nouvelles versions de TikZ utilisent un système de clés-valeurs, implementé par le package `pgfkeys`, plus général que celui géré par les packages `keyval` ou `xkeyval`.

Les utilisateurs de `\includegraphics` connaissent par exemple les clés :

`draf=true` ou `false`

`bb= xx yy uu vv` (4 dimensions exprimées en pt, sans unité), etc.

Les utilisateurs des premières versions de TikZ connaissent par exemple les clés :

`join=round` ou `bevel` ou `mitter`

`inner sep=xx` (une dimension exprimée avec unité), etc.

Maintenant, TikZ peut gérer un ensemble de clés-valeurs plus général comme on va le voir dans les lignes suivantes. Cette généralisation conduit à de nombreuses nouvelles possibilités [57] ; ses caractéristiques et ses avantages sont décrits [57.1.1] en des termes difficiles pour les non informaticiens. Peu importe puisque l'on va se cantonner aux quelques définitions indispensables à l'utilisateur moyen. Ces clés-valeurs se placent dans une arborescence dont la syntaxe est identique à celle de l'arborescence d'un ensemble de fichiers sous unix.

On commence par une définition déjà rencontrée. Il s'agit de la création d'une clé définissant un style avec la commande :

`\pgfkeys{/nom de clé/.style={liste des options}}`

qui place la clé dans la racine de l'arborescence (préfixe `/`). La commande `\tikzset` a une définition dont la version simplifiée (à but pédagogique) est :

`\def\tikzset#1{\pgfkeys{/tikz/#1}}`

Elle place donc les styles (clés particulières) dans le répertoire `/tikz` de l'arborescence ; son utilisation pour définir un style revient à écrire :

`\pgfkeys{/tikz/nom de clé/.style={liste des options}}`

On peut tester cela sur l'exemple précédent en remplaçant la première ligne de code par la suivante.

Avant de continuer, il faut remarquer que l'utilisateur peut placer les clés qu'il définit dans des répertoires (répertoire de l'arborescence des clés !) qui sont créés au moment des définitions

des clés. Pour utiliser les clés ainsi définies, il n'est pas nécessaire d'indiquer le chemin d'accès (cf. l'utilisation des styles `cercle`, `every rectangle node`, etc. de la figure précédente). On note aussi le rôle joué par certains mots réservés préfixés par un point : `.style` pour définir un style, `.append style` pour compléter un style (cf. encore la figure ci-dessus) ; on va en voir quelques autres.

La première définition rencontrée est généralisée dans le sens que la liste des options peut contenir un paramètre [57.4.4] ; on peut par exemple définir le style `lw` par la commande :

```
\pgfkeys{/lw/.style={line width=#1pt}}
```

qui sera utilisé de la manière suivante :

```
\draw[lw=1.5,gray!50] (0,0)--(1,1);
```

Cette possibilité est généralisable par les commandes de définition suivantes :

```
\pgfkeys{/nom de clé/.style 2 args={liste des options avec 2 arguments}}
```

```
\pgfkeys{/nom de clé/.style args={n}{liste des options avec n arguments}}
```

où  $n \leq 9$ .

Pour ces deux dernières définitions, les clés correspondantes sont utilisées avec la syntaxe :  
nom de clé={val. 1}{val. 2} ... }

L'aspect le plus puissant de cette généralisation est que l'on dispose aussi de la possibilité de définir des clés qui exécutent du code avec des paramètres [57.3.2]. Ce code peut même contenir des définitions. Voici les commandes utiles et facilement utilisables.

Commandes de définition :

```
\pgfkeysdef{/nom de clé}{code sans où avec un paramètre à exécuter}
```

définit une clé qui exécutera du code sans paramètre ou avec un paramètre.

```
\pgfkeysdefnargs{/nom de clé}{n}{code à n paramètres à exécuter}
```

définit une clé qui exécutera du code avec  $n$  paramètres ( $n \leq 9$ ).

Ces deux commandes ont une variante où `def` est remplacé par `edef` : `\pgfkeysedef`, etc. Leur effet par rapport aux commandes initiales est exactement celui des définitions avec `\edef` par rapport aux définitions avec `\def`.

Commandes d'utilisation :

```
\pgfkeys{nom de clé}
```

dans le cas où le code est sans paramètre ;

```
\pgfkeys{nom de clé=valeur du paramètre}
```

dans le cas où le code a un paramètre ;

```
\pgfkeys{nom de clé={val. 1}{val. 2} ... }
```

dans le cas où le code a plusieurs paramètres.

Ces commandes sont utilisées dans le cas où l'exécution du code de la clé donne un résultat utilisable plus loin : par exemple un nombre ou une paire de dimensions (cas des deux figures de la section 4.3 où l'exécution de la clé donne une paire de coordonnées).

On va présenter brièvement une autre possibilité de définition des clés grâce à des mots réservés [57.4.3] d'une manière semblable à la définition des styles. Les commandes correspondantes de définition s'écrivent à l'aide du nouveau mot réservé `code` :

```
\pgfkeys{/nom de clé/.code{code sans où avec un paramètre à exécuter}}
```

```
\pgfkeys{/nom de clé/.code 2 args={code à 2 paramètres à exécuter}}
```

```
\pgfkeys{/nom de clé/.code n args={n}{code à n paramètres à exécuter}}
```

Pour apprécier la puissance de cette gestion de paires clés-valeurs il faudrait de très nombreux exemples et beaucoup de pages ! On en reste donc à ces quelques lignes.

# Lignes courbes

Il y a quelques légères modification et quelques possibilités supplémentaires.

## 4.1 Tracés prédéfinis

Pour le cercle et l'ellipse, on a toujours [2.5] :

```
\draw(a,b)circle(r); et \draw(a,b)ellipse(r and s)
```

où l'ensemble **a,b** peut être remplacé par **A**, point de coordonnées **a** et **b**. On remarque le **and** remplaçant la virgule séparant les deux rayons. Pour l'arc de cercle on a toujours [2.10] :

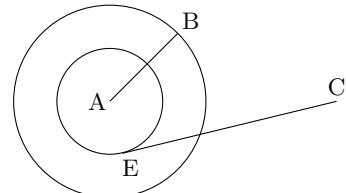
```
\draw(a,b)arc(u:v:w);
```

mais il y a une syntaxe similaire pour l'arc d'ellipse [2.10] :

```
\draw(a,b)arc(u:v:r and s);
```

Deux tracés nouveaux peuvent être utiles : cercle centré au point **A** et passant par **B** [4.1.3] et tangente à un cercle de centre **A** et issue du point **B** extérieur au cercle [13.2.4] ; ils sont l'objet de la figure suivante. On note que, pour les deux tracés, les cercles doivent être définis comme noeuds de forme circulaire. On donne aussi, cachée par des %, la construction de la tangente issue d'un point à un cercle défini par son centre et un autre point.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\small\begin{tikzpicture}[x=1mm,y=1mm]
% Cercle centre A passant par B
\coordinate(A)at(0,0);\coordinate(B)at(9,9);
\node[draw,circle through=(B)]at(A){};
\draw(A)--(B);\pose(A)[-1]{180}{A};\pose(B)[-1]{45}{B};
% Tangente au cercle centre A issue de C
\coordinate(C)at(30,0);\pose(C){90}{C};
\node[draw,circle,minimum size=14mm](D)at(A){};
\coordinate(E)at(tangent cs:node=D,point={(C)},solution=2);
\draw(C)--(E);\pose(E){-60}{E};
% Tang. au cercle centre A passant par B issue de C
\node[draw,circle through=(B)](F)at(A){};
\draw(C)--(tangent cs:node=F,point={(C)},solution=2);
\end{tikzpicture}
```



Il n'y a pas de modification pour les courbes sinus et cosinus.

## 4.2 Courbes de Bézier

La syntaxe générale et la syntaxe spéciale particulièrement commode et utile n'ont pas subi de modification [2.4].

### 4.3 Courbes passant par des points donnés

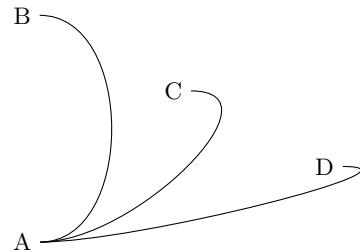
Courbes définies par deux points guides avec directions imposées.

Il n'y a pas de modification pour le tracé de courbes passant par deux points donnés avec l'opération :

`\draw[...](...)to[out=...,in=...](...)` [14.14].

On note cependant une possibilité de définir un style permettant de tracer un ensemble de telles courbes ayant certaines caractéristiques communes [14.14] : voilà un exemple qui peut, peut-être, inspirer le lecteur pour des applications spécifiques.

```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\path[use as bounding box](-1.5,-1.5)rectangle(45,31.5);
\tikzset{allure/.style={to path={.. controls
+(15,0)and+(15,0)..(\tikztotarget)}}}
\node(a)at(0,0){A};\node(b)at(0,30){B};
\node(c)at(20,20){C};\node(d)at(40,10){D};
\draw(a)to[allure](b);\draw(a)to[allure](c);
\draw(a)to[allure](d);
%\axespapiermilli(-5,-3)(60,35)
\end{tikzpicture}
```



Cet exemple tombe, bien involontairement, dans le cas où il faut modifier la boundingbox par la commande spéciale [15.7] vue à la fin de la section 2.2. La raison est invisible sur la figure : la boudingbox par défaut inclut les points de contrôles des courbes de Bézier ; pour mesurer les dimensions de la figure, on ajoute, en fin de code, la macro `\axesgrillefine()` avec des dimensions suffisamment grandes pour recouvrir largement la figure y compris les points de contrôles ; il suffit alors de relever les dimensions de la figure et de les reporter dans la macro `\path[use as bound...`. On pourra vérifier la nécessité de cette opération en plaçant un % devant cette macro.

Courbes avec de nombreux points guides.

Il n'y a pas non plus de modification pour le tracé de courbes passant par des points donnés avec les opérations (pour abréger, on ne recopie pas `\draw[...]`) :

`plot[...]coordinates{()(...)}` [19.3] ( coordonnées à la suite),

`plot[...]file{...}` [19.4] ( coordonnées dans un fichier)

et les options `smooth`, `smooth cycle`, `mark=...` et `tension=...` [19.8].

Ensuite, on note que le tracé de courbes avec Gnuplot (sect. 6.10 du Manuel) ne subit pas de modification ; on en rappelle la syntaxe [19.6] :

`plot[...]function{...}` (fonction écrite en syntaxe Gnuplot)

avec les options rappelées ci-dessus et les options spécifiques `domain=...:...`, `samples=...`, `prefix=...`, `id=...` et, éventuellement, `parametric=true`.

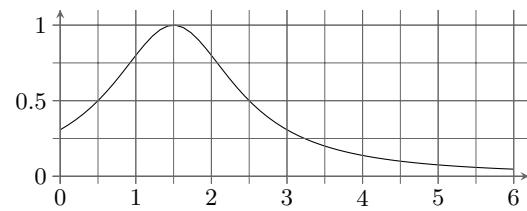
Enfin, les nouvelles versions de Ti $k$ Z permettent de tracer des courbes directement à partir des fonctions les définissant. La syntaxe est la suivante [19.5] :

`plot[variable=macro, domain=nombre:nombre, samples=nombre,...](a,b) ou (u:r)`

Par défaut, `variable=\x`, `samples=25` et `domain=-5:5`; `a` et `b` sont les coordonnées catésiennes des points de la courbe.

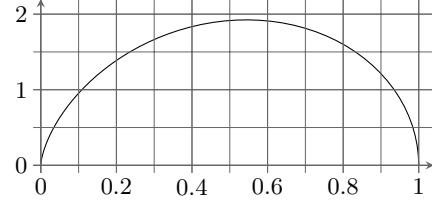
La première courbe est définie par la fonction  $1/((x-2)^2+1)$  pour  $x$  variant de 0 à 6 : on choisit `scale=10` et 2 pour le coefficient multiplicatif de l'ordonnée (afin de mieux visualiser la courbe comme disent les physiciens).

```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\draw plot[variable=\x,domain=0:6,
samples=60,scale=10]
(({\x},{2/((\x-1.5)^2+1)});
\pose(-1,0){180}{0}\pose(-1,10){180}{0.5}
\pose(-1,20){180}{1}\pose(0,-1){-90}{0}
\pose(10,-1){-90}{1}\pose(20,-1){-90}{2}
\pose(30,-1){-90}{3}\pose(40,-1){-90}{4}
\pose(50,-1){-90}{5}\pose(60,-1){-90}{6}
\axesgrille(-1,-1)(62,32)
\end{tikzpicture}
```



La deuxième courbe est définie sous forme paramétrique par les deux fonctions  $\cos^3(t)$  et  $\sin(t)\cos^2(t)$  pour  $t$  variant de 0 à 90 degrés : on choisit `scale=50` et 1 pour le coefficient multiplicatif de l'ordonnée.

```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\draw plot [parametric,variable=\t,domain=0:90,
samples=60,scale=60]
  ({cos(\t)^3},{sin(\t)*cos(\t)^2});
\pose(-1,0){180}{0}\pose(-1,10){180}{1}
\pose(-1,20){180}{2}\pose(0,-1){-90}{0}
\pose(10,-1){-90}{0.2}\pose(20,-1){-90}{0.4}
\pose(30,-1){-90}{0.6}\pose(40,-1){-90}{0.8}
\pose(50,-1){-90}{1}
\axesgrille(-1,-1)(52,22)
\end{tikzpicture}
```



Il faut noter que, par exemple pour la première courbe, on aurait pu écrire `x=10mm` et `y=20` au lieu de prendre `scale=60` et 2 pour coefficient multiplicatif de l'ordonnée ; mais alors la macro `\axesgrille`, qui n'est pas adaptées à ce type de changement d'unité, aurait été inutilisable.

On peut aussi tracer une courbe donnée en coordonnées polaires à condition que la fonction correspondante soit une des fonctions prédéfinies de TikZ, par exemple :

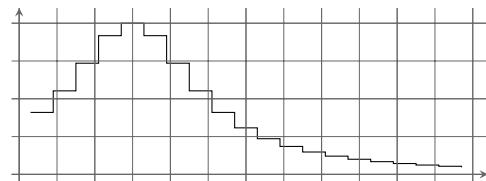
```
\draw plot [...,domain=0:90,samples=60] ({\t}:{cos(\t)});
```

Sinon, il faudrait peut-être définir la fonction en syntaxe TikZ [65.1 et 2]. De toute manière, on dispose de la solution suivante ; la courbe correspondant à  $r = f(t)$  peut être considérée comme la courbe paramétrique définie par les deux fonctions  $\cos(t)f(t)$  et  $\sin(t)f(t)$  ; cela a été fait ci-dessus pour tracer la courbe définie par  $r = \cos^2(t)$ .

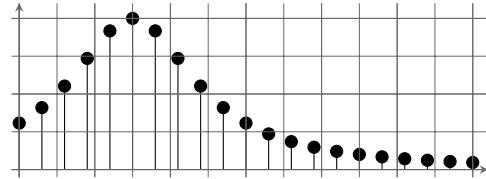
Il faut signaler de nombreuses possibilités pour le style du tracé. D'abord on dispose d'un grand nombre de glyphes pour marquer les points guides de la courbe (cas de tracés avec peu de points guides, tracés avec `coordinates` ou `file` notamment). Il faut charger la bibliothèque `plotmarks` [43] pour disposer de ces glyphes à l'aide de l'option `mark=...`

Pour le tracé au sens strict lui-même, on dispose de nombreuses possibilités [19.8] : on n'en présente que trois (relativement à la mode en ces temps-ci) ;

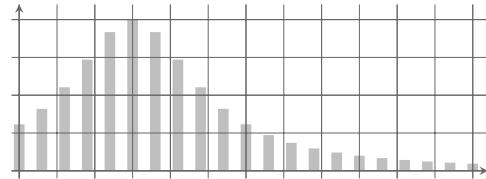
```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\draw plot [const plot,xshift=-1.5mm,variable=\x,
domain=0:3.6,samples=20,scale=10]
  ({\x},{2/((\x-1.5)^2+1)});
\axesgrille(-1,-1)(62,22)
\end{tikzpicture}
```



```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\pgfsetplotmarksize{0.08mm}
\draw plot [ycomb,variable=\x,domain=0:6,
samples=21,mark=*,scale=10]
  ({\x},{2/((\x-1.5)^2+1)});
\axesgrille(-1,-1)(62,22)
\end{tikzpicture}
```



```
\small\begin{tikzpicture}[x=1mm,y=1mm]
\draw[draw=none,fill=lightgray] plot[ybar,
  bar width=0.4pt,variable=\x,domain=0:6,
  samples=21,scale=10] ({\x},{2/((\x-1.5)^2+1)});
\axesgrille(-1,-1)(62,22)
\end{tikzpicture}
```



Ces tracés appellent des remarques. Il a fallu déplacer la courbe en escalier d'une demi distance horizontale entre les points guides vers les  $x$  négatifs afin que les points guides correspondent au milieu des segments horizontaux. Pour les deux dernières courbes, à cause de la présence de l'option `scale=10`, il a fallu réduire par le facteur 10 la dimension des disques noirs et la largeur des barres grises.

On fera attention à la position des options : les options de `plot` doivent être placées après `plot`, les options de couleur (qui sont valables pour tous les tracés exécutés par la commande `draw`) doivent être placées après `\draw`. Pour les autres options, la position est assez libre.

On arrive à la fin du chapitre 4 du Manuel. On verra au chapitre 6 une tout autre manière d'aborder le tracé de courbes permettant la « visualisation de données ».

# Intersections de courbes

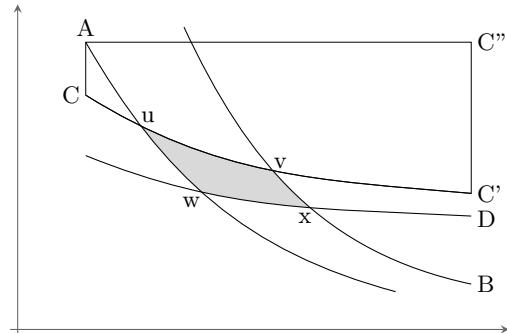
(TikZ 2.00 cvs du 02/06/10)

## 5.1 Colorer une surface découpée par des courbes

Cette méthode n'est pas nouvelle ; dans le petit manuel, il y a un exemple d'utilisation des calques mais il est banal : s'il est utile sur le plan pédagogique, il est bien moins intéressant que l'exemple suivant (cycle de Carnot des thermodynamiciens) qui, outre son intérêt pratique, va permettre d'expliquer ce qui manquait à TikZ jusqu'à la version 2.00 comprise (ici, on travaille avec la version 2.00 cvs, peut-être il y aura quelques modifications mineures de syntaxe dans la version stabilisée).

La méthode des calques (très utilisée, surtout par les architectes, et portée dans tous les logiciels de dessin) ne fait, pour cet exemple, que ce que ferait un gamin à l'école maternelle : il trace les courbes, sur du papier coloré, il découpe la bande limitée par les courbes A et B, puis il découpe cette bande suivant les parties centrales des courbes C et D, c'est-à-dire uv et wx ; enfin, il colle le découpage sur le papier blanc où sont tracées les quatre courbes !

```
\pgfdeclarelayer{carnot}
\pgfsetlayers{carnot,main}
\begin{tikzpicture}[x=1mm,y=1mm]
\def\pathA{(9,38)to[out=-60,in=165](50,5)}
\def\pathB{(60,6)to[out=168,in=-65](22,40)}
\def\pathC{(9,31)to[out=-32,in=175](60,18)}
\def\pathD{(60,15)to[out=177,in=-22](9,23)}
\begin{pgfonlayer}{carnot}
\path[clip]\pathA--\pathB--cycle;
\path[fill=gray!30]\pathC--\pathD--cycle;
\end{pgfonlayer}
\draw\pathA;\draw\pathB;
\draw\pathC;\draw\pathD;
%%% fin du remplissage
\pose(9,38){90}{A}\pose(60,6){0}{B}
\pose(9,31){180}{C}\pose(60,14){18}{D}
\pose(16,2,27)[-2]{45}{u}
\pose(33,7,21)[-2]{45}{v}
\pose(24,4,18,3)[-0.5]{-135}{w}
\pose(38,7,16)[-1]{-120}{x}
\draw[thin]\pathC--(60,38)--(9,38)--cycle;
\pose(60,18){0}{C'}\pose(60,38){0}{C''}
\end{tikzpicture}
```



Cette coloration n'est pas, dans ce cas, très lourde à coder mais on peut facilement imaginer des situations plus complexes. Par contre, la coloration des différentes parties des courbes, même dans cet exemple, devient fastidieuse : prenons le cas où les parties centrales des courbes sont en traitillé et les parties extrêmes sont en trait fin. Pour chaque tronçon de courbe, il faut construire un contour pour le découper : sur l'exemple, le contour CC'C''A permet le découpage du tronçon Au, et ainsi de suite !

Pour travailler « raisonnablement », il faudrait pouvoir avoir accès aux points u,v, w et x pour disposer des tronçons uv, vx, uw et wx ainsi que du chemin fermé uwvx. On va voir la manière dont TikZ permet ce genre de manipulation.

## 5.2 Intersection de courbes

On donne pour commencer un exemple de détermination des points d'intersection des courbes de l'exemple précédent. Il faut, bien entendu, charger la bibliothèque `intersections`; on pourra se reporter à l'exemple de la doc [13.3.2].

On remarque qu'il faut d'abord nommer les courbes avec :

`name path=nom de la courbe`

et désigner les courbes dont on veut déterminer les intersections avec :

`name intersections={of=nom de courbe and nom de courbe}`

Les coordonnées des points d'intersections sont représentées par la macro nommée par :

`name=nom de macro (i ici et dans l'exemple de la doc)([13.3.2], page 131).`

Le nombre total de points d'intersection est représenté par la macro nommée par :

`total=nom de la macro (\t dans l'exemple de la doc).`

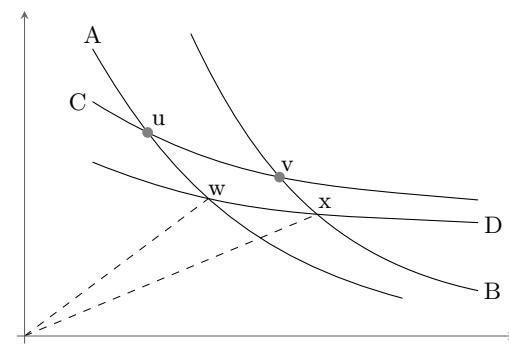
Pour pouvoir s'y retrouver, on peut classer les points d'intersection dans l'ordre croissant du temps<sup>(1)</sup> avec l'option :

`sort by=nom de la courbe choisie`

c'est-à-dire par temps croissant sur la courbe choisie. Il y a quelques autres possibilités bien moins intéressantes.

L'exemple montre comment placer des points gris aux points d'intersection u et v [13.3.2], mais il montre aussi comment procéder pour pouvoir utiliser les coordonnées de ces points par l'intermédiaire d'un noeud fantôme.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[name path=pathA] (9,38)to[out=-60,in=165](50,5);
\draw[name path=pathB] (60,6)to[out=168,in=-65](22,40);
\draw[name path=pathC] (9,31)to[out=-32,in=175](60,18);
\draw[name path=pathD] (60,15)to[out=177,in=-22](9,23);
% place un point gris aux points u et v (4 lignes)
\fill[name intersections={of=pathA and pathC,name=i}]
[gray](i-1)circle(2pt);
\fill[name intersections={of=pathB and pathC,name=i}]
[gray](i-1)circle(2pt);
% définit les points d'intersection w et x pour
% les utiliser dans un tracé ultérieur (5 lignes)
\node[name intersections={of=pathA and pathD,name=i}]
at(i-1){};\coordinate(w)at(i-1);
\node[name intersections={of=pathB and pathD,name=i}]
at(i-1){};\coordinate(x)at(i-1);
\draw[very thin](0,0)--(w);\draw[very thin](0,0)--(x);
\pose(16.7,27.5)[-2]{45}{v}\pose(33.7,21.5)[-2]{45}{v}
\pose(24.4,18.3)[-2]{45}{w}\pose(38.7,15.8)[1]{60}{x}
\pose(9,38){90}{A}\pose(60,6){0}{B}
\pose(9,31){180}{C}\pose(60,14){18}{D}
\end{tikzpicture}
```



## 5.3 Travail avec les intersections des courbes

Le programme évoqué à l'avant dernière section est loin d'être résolu. La difficulté à résoudre est que les macros disponibles donnent les coordonnées des points d'intersection mais ne donne pas le temps de ces points bien que l'on en ait besoin pour utiliser la macro pgf [73.4] :

`\pgfpathcurvebetween{temps 1}{temps 2}{}{}{}{}{}{} (quatre points définissant la courbe)` qui permet de tracer la partie de la courbe entre deux valeurs du temps. On dispose aussi de la macro pgf [72.5.2] :

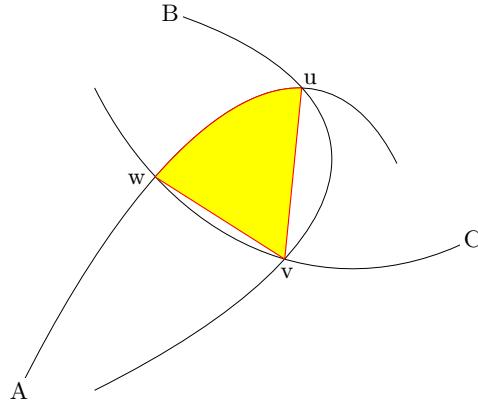
`\pgfpointcurveat{temps}{}{}{}{}{}{} (quatre points définissant la courbe)`

<sup>(1)</sup>On rappelle que les courbes de Bézier sont des courbes du troisième degré en représentation paramétrique ; le paramètre est traditionnellement nommé le temps.

qui donne les coordonnées du point correspondant à une valeur du temps. On se rend compte qu'il va falloir travailler au niveau de la couche de base. Voilà une proposition pour tracer et colorer la surface découpée par les courbes qui fonctionne avec la version 2.00 cvs.

On ne prend, pour simplifier, que trois courbes dont chacune n'intersecte qu'une fois les deux autres. Les parties du code sont signalées par des lignes de commentaires, les explications des points délicats vont suivre dans l'ordre.

```
\begin{tikzpicture}[x=1mm,y=1mm]
%%%%%% courbes en notation complète A, B et C
\draw[very thin,name path=pathA] (0,0)..controls (20,40) and (40,50)..(50,30);
\draw[very thin,name path=pathB] (20,50)..controls (50,40) and (50,20)..(10,0);
\draw[very thin,name path=pathC] (60,20)..controls (40,10) and (20,20)..(10,40);
%%%%% détermination des points d'intersection u, v et w
\node[name intersections={of=pathA and pathB,name=i}]at(i-1){};\coordinate(u)at(i-1);
\node[name intersections={of=pathB and pathC,name=i}]at(i-1){};\coordinate(v)at(i-1);
%% détermination des deux coordonnées de v (2 l.)
\newlength{\xv}\newlength{\yv}
\path let \p1=(i-1) in \pgfextra{\xdef\xv{\x1}\xdef\yv{\y1}}(0,0);
\node[name intersections={of=pathC and pathA,name=i}]at(i-1){};\coordinate(w)at(i-1);
%%%%% pour faire un test
%\draw(u)--(v)--(w)--cycle;
%%%%% calcul des temps d'intersection en couche de base
\newlength{\xvar}\newlength{\yvar}
% boucle de calcul de \tuA, \xuA et \yuA
% définition d'une clé exécutant une macro (3 l.)
\pgfkeysdef{/clef}{\pgfpointcurveattime{#1}
  {\pgfpoint{0mm}{0mm}}{\pgfpoint{20mm}{40mm}}
  {\pgfpoint{40mm}{50mm}}{\pgfpoint{50mm}{30mm}}}
% autre définition de la m^eme clé (3 l.)
%\pgfkeys{/clef/.code={\pgfpointcurveattime{#1}
%  {\pgfpoint{0mm}{0mm}}{\pgfpoint{20mm}{40mm}}
%  {\pgfpoint{40mm}{50mm}}{\pgfpoint{50mm}{30mm}}}}
\edef\Dcal{70mm} %% distance max possible
\foreach\tvar in {0.501,0.502,...,1}
{\pgfextractx{\xvar}{\pgfkeys{clef=\tvar}}
\pgfextracty{\yvar}{\pgfkeys{clef=\tvar}}
\coordinate(Uvar) at (\xvar,\yvar);
\path let \p1=($(Uvar)-(u)$),\n1={veclen(\x1,\y1)}in
  \pgfextra{\xdef\Uvar{\n1}}(0,0)--(10,0);
\ifdim\Uvar>0.25pt\xdef\Dcal{\Uvar}
  \else\xdef\tuA{\tvar}\breakforeach\fi
% \newlength{\xuA}\newlength{\yuA} %in.
% \pgfextractx{\xuA}{\pgfkeys{clef=\tuA}} %in.
% \pgfextracty{\yuA}{\pgfkeys{clef=\tuA}} %in.
% boucle de calcul de \twA, \xwA et \ywA
\edef\Dcal{70mm} %% remise à 70mm de \Dcal
\foreach\tvar in {0.301,0.302,...,1}
{\pgfextractx{\xvar}{\pgfkeys{clef=\tvar}}\pgfextracty{\yvar}{\pgfkeys{clef=\tvar}}
\coordinate(Wvar) at (\xvar,\yvar);
\path let \p1=($(Wvar)-(w)$),\n1={veclen(\x1,\y1)}in\pgfextra{\xdef\Uvar{\n1}}(0,0)--(10,0);
\ifdim\Uvar>0.25pt\xdef\Dcal{\Uvar}\else\xdef\twA{\tvar}\breakforeach\fi
% \newlength{\xwA}\newlength{\ywA} %in.
% \pgfextractx{\xwA}{\pgfkeys{clef=\twA}} %in.
% \pgfextracty{\ywA}{\pgfkeys{clef=\twA}} %in.
%%%%% tracé et remplissage du chemin uvw : un tronçon de courbe et deux segments
% tracé de la partie centrale de la courbe A (3 l.)
\pgfpathcurvebetweenetime{\twA}{\tuA}{\pgfpoint{0mm}{0mm}}{\pgfpoint{20mm}{40mm}}
  {\pgfpoint{40mm}{50mm}}{\pgfpoint{50mm}{30mm}}
\pgflineto{\pgfpoint{\xv}{\yv}}\pgfclosepath
\pgfsetstrokecolor{red}\pgfsetfillcolor{yellow}\pgfusepath{fill,stroke}
\end{tikzpicture}
```



On constate que le code en couche de base ci-dessus est fastidieux ; il serait bien simplifié si la commande TikZ déterminant les points d'intersection donnait aussi leurs temps d'intersection. Mieux encore, tout serait parfait si les commandes fondamentales `\pgfpathcurvebetweenetime` et `\pgfpointcurveattime` avaient une version en couche TikZ.

Voilà les commentaires et explications annoncées :

- Des lignes, avec un % et la mention `in`. ne sont donnée que pour une éventuelle utilisation.
- On a besoin des deux coordonnées `\xv` et `\yv` du point `v` (à cause de la syntaxe de la commande `\pgfpoint`) pour le tracé en couche de base, d'où leur extraction avec un chemin fantôme, la commande `TikZ let...in` et la commande `\pgfextra`, commandes présentées à la section 1.5.
- On définit la clé `/clef` avec la commande (cf. sect. 3.5) :
 

```
\pgfkeysdef{/clef}{code à un paramètre}
```

 On trouve juste après la deuxième possibilité de définition la même clé (cf. sect. 3.5). Par la suite, l'utilisation de cette clé exécute le code pour la valeur du paramètre fournie en écrivant (cf. sect. 3.5).
 

```
\pgfkeys{clef=valeur souhaitée du paramètre}
```
- Enfin, on utilise les macros [72.6] :
 

```
\pgfextractx{longueur TeX}{ coordonnées du point}
```

`\pgfextracty{longueur TeX}{ coordonnées du point}`
 pour disposer *séparément* des valeurs des deux coordonnées d'un point.
- Pour abréger, on remplace les autres tronçons de courbe par deux segments en utilisant les macros pgf appropriées [73]. Il faudrait calculer les temps `\tuB`, `\tvB`, `\tvC` et `\twC` car chaque point d'intersection a deux temps, un pour chaque courbe ... et ils sont nécessaires pour tracer les tronçons des courbes B et C.
- Il est bien entendu qu'il ne s'agit ici que d'une proposition : en particulier la boucle de calcul des temps d'intersection reste à améliorer.

## 5.4 Petite plongée en couche de base

Pour terminer, on donne cinq exemples de calcul d'intersection qui vont faire l'objet de la figure suivante.

Le premier exemple est l'intersection de deux droites codée en couche TikZ (comme dans le petit manuel). Les coordonnées du point d'intersection sont utilisées de deux manières différentes :

- directement dans une commande de tracé d'un cercle (rayon 2 pt),
- indirectement en définissant un point (commande `TikZ \coordinate`) utilisé ensuite comme centre d'un cercle (rayon 4 pt).

La coïncidence semble triviale ... et elle l'est. Elle l'est beaucoup moins pour les trois autres exemples écrits en code pgf : c'est un moyen d'apprendre à maîtriser le passage de résultats de la couche pgf à la couche TikZ.

Le deuxième exemple traite de la même intersection que le premier, excepté qu'il est écrit en pgf. Les coordonnées du point d'intersection sont données par la macro pgf [72.5.4] :

```
\pgfpointintersectionoflines{}{}{}{}
```

où les deux premiers arguments sont deux points définissant la première droite et les deux derniers deux points définissant la deuxième ; ces points sont codés par la macro pgf [72.2] :

```
\pgfpoint{}{}
```

où chaque coordonnée (suivant les *x* et les *y*) est un nombre suivi d'une unité.

Le troisième exemple présente l'intersection d'une droite et d'une courbe de Bézier en couche TikZ. On nomme la droite puis la courbe de Bézier et on continue d'une manière légèrement différente par rapport à la détermination des points d'intersections de la section 2 du présent chapitre ; on utilise l'option [13.3.2] :

`by={suite de noms séparés par des virgules}`

ce qui permet de définir les points d'intersection pour une utilisation directe à condition que l'on en connaisse le nombre. Si le nombre de noms est plus petit que le nombre de points d'intersection, seuls les premiers sont nommés. Cette option est particulièrement intéressante dans plusieurs cas, particulièrement dans le cas d'un seul point d'intersection, ce qui est le cas présent.

Le quatrième exemple présente l'intersection précédente en couche pgf. La macro pgf [72.5.6] : `\pgfintersectionofpaths{}{}`

exécute le calcul du point d'intersection ; le premier argument est le code pgf de la droite [73.3] avec la donnée de 2 points<sup>(2)</sup> ; le deuxième argument est le code pgf de la courbe de Bézier [72.4] avec la donnée de 4 points. Une fois le calcul exécuté, on dispose des coordonnées des points d'intersection avec la macro pgf [72.5.6] :

`\pgfpointintersectionsolution{}`

où l'argument est le numéro du point d'intersection voulu ; le nombre de ces points (1 dans le cas présent et le cas suivant) est donné par la macro pgf [72.5.6] :

`\pgfpointintersectionsolutions`

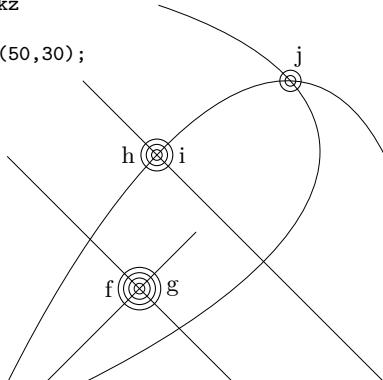
Pour relier à ce qui a été vu à la section 4 du présent chapitre, on donne les correspondances suivantes :

`\pgfpointintersectionsolution{1}  $\Leftrightarrow$  i-1`

`\pgfpointintersectionsolutions  $\Leftrightarrow$  \t`

Le dernier exemple présente l'intersection de deux courbes de Bézier en couche pgf. Tout a été vu dans l'exemple précédent, il n'y a rien à rajouter.

```
\small\begin{tikzpicture}[x=1mm,y=1mm]
%%%%%%%
% Intersection de 2 droites en couche tikz (frontend layer)
\draw[very thin](0,30)--(30,0);\draw[very thin](5,0)--(25,20);
\draw[very thin](intersection of 0,30--30,0 and 5,0--25,20)circle(2pt);
\coordinate(f)at(intersection of 0,30--30,0 and 5,0--25,20);
\draw(f)circle(4pt);\pose(f)[8]{180}{f};
%%%%%%
% M^eme intersection en couche pgf (basic layer)
\newlength{\xxx}\newlength{\yyy}
\pgfkeysdef{/clai}{\pgfpointintersectionoflines{\pgfpoint{0mm}{30mm}}
{\pgfpoint{30mm}{0mm}}{\pgfpoint{5mm}{0mm}}{\pgfpoint{25mm}{20mm}}}
\pgfpathcircle{/clai}{6pt}\pgfusepath{stroke}
\pgfextractx{\xxx}{/clai}\pgfextracty{\yyy}{/clai}
\coordinate(g)at(\xxx,\yyy);\draw(g)circle(8pt);\pose(g)[8]{0}{g};
%%%%%%
% Intersection d'une droite et d'une courbe en couche tikz
\draw[very thin, name path=DR](10,40)--(50,0);
\draw[very thin, name path=CO](0,0)..controls (20,40) and (40,50)..(50,30);
\path[name intersections={of=DR and CO, by={h}}](0,0);
\draw(h)circle(2pt);\pose(h)[6]{0}{i};
%%%%%%
% M^eme intersection en couche pgf
\pgfintersectionofpaths{\pgfpathmoveto{\pgfpoint{10mm}{40mm}}
{\pgfpathlineto{\pgfpoint{50mm}{0mm}}}
{\pgfpathmoveto{\pgfpoint{0mm}{0mm}}
{\pgfpathcurveto{\pgfpoint{20mm}{40mm}}
{\pgfpoint{40mm}{50mm}}{\pgfpoint{50mm}{30mm}}}}
\pgfpathcircle{/pgfpointintersectionsolution{1}}{4pt}
\pgfusepath{stroke}
\pgfextractx{\xxx}{/pgfpointintersectionsolution{1}}
\pgfextracty{\yyy}{/pgfpointintersectionsolution{1}}
\coordinate(i)at(\xxx,\yyy);\draw(i)circle(6pt);
\pose(i)[6]{180}{h};
%%%%%%
% Intersection de deux courbes en couche pgf
\draw[very thin](20,50)..controls (50,40) and (50,20)..(10,0);
\pgfintersectionofpaths%
{\pgfpathmoveto{\pgfpoint{0mm}{0mm}}\pgfpathcurveto{\pgfpoint{20mm}{40mm}}
{\pgfpoint{40mm}{50mm}}{\pgfpoint{50mm}{30mm}}}
{\pgfpathmoveto{\pgfpoint{20mm}{50mm}}\pgfpathcurveto{\pgfpoint{50mm}{40mm}}
{\pgfpoint{50mm}{20mm}}{\pgfpoint{10mm}{0mm}}}
\pgfpathcircle{/pgfpointintersectionsolution{1}}{2pt}\pgfusepath{stroke}
\pgfextractx{\xxx}{/pgfpointintersectionsolution{1}}
\pgfextracty{\yyy}{/pgfpointintersectionsolution{1}}
\coordinate(j)at(\xxx,\yyy);\draw(j)circle(4pt);\pose(j)[3.3]{70}{j};
\end{tikzpicture}
```



<sup>(2)</sup>Ce code va rappeler des souvenirs à ceux qui ont, un jour, touché du Postscript !



## CHAPITRE 6

# Visualisation des données

Ce titre de chapitre (traduction mot à mot) désigne tout simplement ce qui est exposé à la section 4.4 travaillé avec une autre approche (qui rappelle un peu celle de MetaPost). Il manque encore la partie la plus intéressante de la doc [55.6 à 9]. L'auteur de ces lignes ne fait que donner un léger mode d'emploi basé sur les trois exemples donnés [56.6], la partie rédigée de la doc [54 à 56] ... et la lecture (aride) de quelques fichiers, notamment :

```
\datavisualization.code.tex (fichier A)
\datavisualization.formats.functions.code.tex
situées respectivement dans les sous-réertoires :
\tex\generic\pgf\frontendlayer\tikz\libraries\datavisualization\
\tex\generic\libraries\datavisualisation\
```

On notera qu'il faut remettre (à l'intérieur de l'environnement `tikzpicture` bien entendu) le `catcode` du point-virgule à 12 s'il a été changé (cas du français). Sur la version disponible, 2.00 cvs, il faut pourcenter les lignes 1580, 1581 et 1582 du premier fichier A cité ci-dessus et mettre `\def\ninerm{}` dans le fichier de travail (tout cela sera réparé dans les prochaines versions).

La macro fondamentale est :

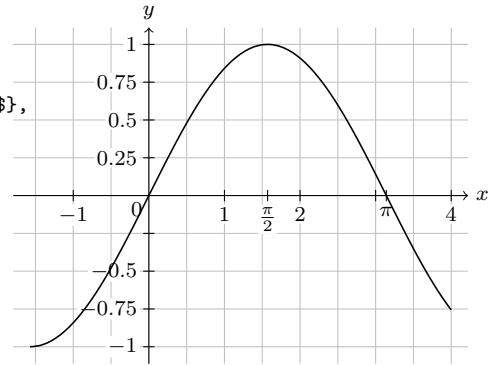
```
\datavisualization choix graphiques data calcul ou lecture des données;
```

### 6.1 Premier exemple pour se lancer

On donne le premier exemple de la doc largement modifié pour des raisons pédagogiques ; les explications seront données ligne par ligne à la suite.

```
\small\begin{tikzpicture}
\catcode`;=12
\datavisualization[school book axes,    %l. 1
smooth line plot,clean ticks,
x axis={unit length=10mm, %par défaut
grid={step=1,minor steps between steps=1},
ticks={some,major={also at={(0.5*pi)}as$\frac{\pi}{2}$},
also at={(pi)}as$\pi$},           %l. 6
options at=3as[no tick text]},    %l. 8
label=$x$},                      %l. 8
y axis={unit length=2cm, %pour plus de visibilité
grid={step=1,minor steps between steps=3},
ticks={some,               %l. 11
major={options at=-0.25as[no tick text]}},
label=$ y $}]

data [format=function] {          %l. 14
var x : interval[-0.5*pi:4] samples 50;
func y = sin(\value x r);};      %l. 16
\end{tikzpicture}
```



### 6.1.1 Choix graphiques

- l. 1) Aucune option pour ce type d'axes ; il y a d'autres types qui seront testés plus loin.
- l. 2) `line plot` donne une ligne brisée au lieu d'être lissée et `scatter plot` trace seulement les points guides avec une croix de Saint-André (on verra plus loin comment on peut utiliser les glyphes définis dans la bibliothèque `plotmarks`) ; `clean ticks` fait que les ticks sont écrits sur un fond blanc recouvrant la grille mais pas la courbe.
- l. 3 à 8) `x axis=...` options pour l'axe des  $x$  :

`unit length=...` : à déterminer en fonction du domaine de variation des  $x$  et de la largeur voulue pour la figure ; on verra plus loin que ce calcul est automatiquement fait par `TikZ` si l'on choisit d'autres types d'axes ;

`grid=...` : pas de 1 (unité choisie) avec un pas secondaire de 0.5 (unité choisie), ce qui correspond bien à une graduation secondaire entre deux graduations ;

`ticks=...` : les possibilités sont `none`, `few`, `some`, ou `many` à apprécier soi-même ; ensuite on ajoute des ticks en des abscisses importantes avec `also...` et on peut enlever certaines étiquettes avec `options...` pour éviter des recouvrements ;

`label=...` écrit la notation choisie pour les abscisses dans le prolongement de l'axe.

- l. 9 à 13) `y axis=...` options pour l'axe des  $x$  : possibilités identiques à celle pour l'axe des  $x$  ;

Il faut noter que les options communes aux deux axes peuvent être données comme suit :

`all axes={options communes aux deux axes}`

et alors `x axis` et `y axis` ne sont utiles que si les axes ont des options spécifiques.

### 6.1.2 Données calculées à partir de fonctions

l.14 à 16) `format=function` signale que l'on donne la fonction selon une syntaxe à respecter scrupuleusement ; `interval` est l'intervalle souhaité à partir duquel on détermine les valeurs de `unit length` pour les axes comme expliqué ci-dessus ; `sample...` est le nombre de points guides souhaités (défaut : 25) ; la fonction elle-même est donnée en syntaxe `TikZ` [65.1 et 2]).

On peut aussi donner la forme paramétrique : dans ce cas les deux lignes (15 et 16) deviennent :

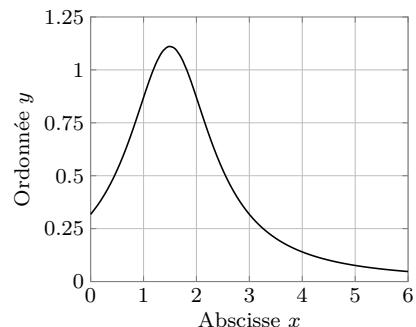
```
var t : interval[.....] sample....;
func x = f(\value t);
func y = g(\value t);
```

On verra par la suite les cas où les coordonnées des points guides sont données à la suite ou lues dans un fichier.

## 6.2 Des exemples pour continuer

Dans la section précédente, on a développé en détail les options des axes et les données sous forme de fonctions ; il reste à voir en détail les autres types d'axes, les types d'affichage des points guides (résultats expérimentaux) et les types de format de lecture (à la suite ou dans un fichier) des données. Chacun des exemples suivants est suivi (parfois précédé) des explications concernant les nouvelles options introduites.

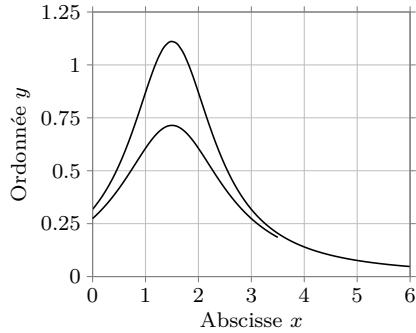
```
\small\begin{tikzpicture}
\catcode`\;=12
\datavisualization[scientific inner axes,
smooth line plot,
x axis={unit length=7mm,grid={step=1},
ticks={some},label={Abscisse $x$}},
y axis={unit length=28mm,grid={step=1,minor
steps between steps=4},ticks={some},
include values={0,1.25},label={Ordonnée $y$}}]
data [format=function]{
  var x : interval[0:6] samples 50;
  func y = 1/((\value x-1.5)^2+0.9);};
\end{tikzpicture}
```



On teste ci-dessus :

- le type d'axes **scientific inner axes**
- l'option **include values={a,b}** qui étend l'axe des  $y$  de manière à ce que ses extrémités soient en  $a$  et en  $b$ ; cette option peut aussi être utilisée pour l'axe des  $x$ ; pour bien voir l'intérêt de cette option, il faut la supprimer et comparer.

```
\hfill\footnotesize\begin{tikzpicture}
\catcode`\;=12
\datavisualization[scientific axes,
    smooth line plot,
    x axis=[grid={step=1},length=42mm,
        ticks={some},label=Abscisse $x$],
    y axis=[length=35mm,grid={step=1,minor
        steps between steps=3},ticks={some},
        include values={0,1.25},label=Ordonnée $y$}]
data [format=function]{
    var x : interval[0:6] samples 50;
    func y = 1/((\value x-1.5)^2+0.9);};
\datavisualization[scientific axes,
    smooth line plot,
    x axis=[length=42mm,ticks={none},include values={6}],
    y axis=[length=35mm],ticks={none},include values={0,1.25}]
data [format=function]{
    var x : interval[0:3.5] samples 50;
    func y = 1/((\value x-1.5)^2+1.4);};
\end{tikzpicture}
```



On teste ci-dessus :

- le type d'axes **scientific axes**
- le calcul automatique des longueurs des unités (7 mm et 28 mm dans l'exemple précédent où elles ont été calculées à la main pour avoir une figure de dimensions correctes). Il suffit de donner les dimensions souhaitées avec les options **length=...** pour les deux axes. Ces deux dimensions sont respectivement la largeur et la hauteur du rectangle contenant le tracé (il faut prévoir que les graduations et les labels viennent en plus).

- le tracé d'une deuxième courbe sur le même graphique. Pour cela, il faut une nouvelle macro **\datavisualization** pour chaque courbe supplémentaire car l'option **smooth line plot** est toujours en action ce qui fait que TikZ continue la première courbe à partir de son dernier point jusqu'au premier point de la deuxième! Il n'est pas nécessaire de reprendre toutes les options de **\datavisualization**, en particulier celles concernant la grille et les graduations; par contre, il faut s'assurer que les options **include values** et **length** des deux axes sont répétées à l'identique, ceci afin que les axes et les graduations (calculés mais non tracés) de la deuxième courbe coïncident avec ceux de la première. Pour la première courbe, la borne supérieure de la variation de la variable est 6, ce qui joue le rôle de **include values={6}** d'où l'absence de cette dernière option.

On va tester ci-après :

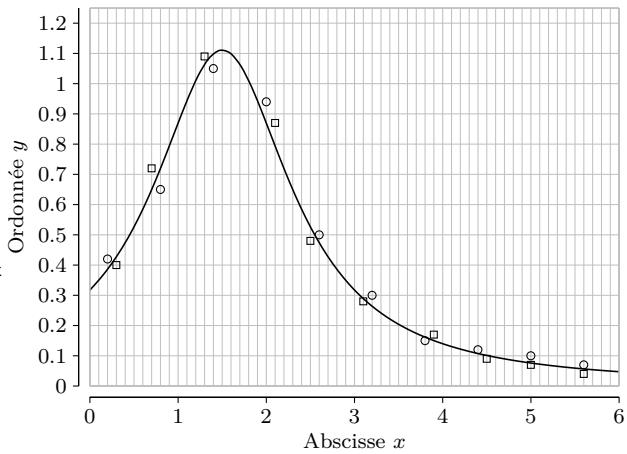
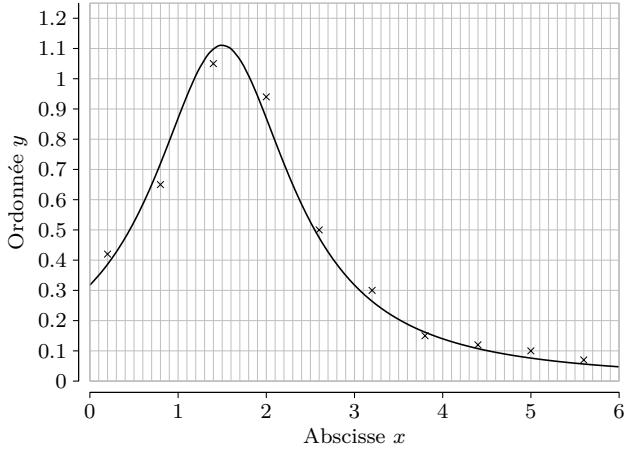
- la dernière variante du type d'axes disponible **scientific clear axes**
- l'ajout sur un graphique où l'on a déjà placé une courbe de points repérés par un glyphe (une petite croix de Saint-André par défaut). On dispose de plusieurs formats pour remplacer le format **function** dans les options de la commande TikZ **data**: les formats **coordinates** (donné dans la doc), **key value pairs** et **TeX code** [56.4 et 5] exigent une saisie fastidieuse; le format **table** ne semble fonctionner qu'avec le type d'axes **school book axes**<sup>(1)</sup>. Peu importe car c'est très facile de faire un format avec les explications de la doc [56.6]; voilà le plus simple suivi de l'exemple montrant la syntaxe correspondante pour la saisie des coordonnées :

```
\pgfdeclaredataformat{tableau}
{}{}{\#1 \#2}
{\pgfkeyssetvalue{/data point/x}{\#1}\pgfkeyssetvalue{/data point/y}{\#2}
 \pgfdatapoint}
{}
```

<sup>(1)</sup>C'est ce que pense l'auteur de ces lignes après avoir fouillé dans les fichiers cités.

```
\footnotesize\begin{tikzpicture}
\catcode`\:=12
\datavisualization[scientific clear axes,smooth line plot,
  x axis={length=70mm,grid={step=1,minor steps between steps=9},
    ticks={some},label=Abscisse $x$},
  y axis={length=50mm,grid={step=1,minor steps between steps=9},
    ticks={many},include values={0,1.25},label=Ordonnée $y$}]
data [format=function]{
  var x : interval[0:6] samples 50;
  func y = 1/((\value x-1.5)^2+0.9);}
\datavisualization[scientific clear axes,
  scatter plot,
  x axis={length=70mm,ticks={none},
    include values={0,6}},
  y axis={length=50mm,ticks={none},
    include values={0,1.25}}]
data [format=tableau]
{
  0.2 0.42
  0.8 0.65
  1.4 1.05
  2 0.94
  2.6 0.5
  3.2 0.3
  3.8 0.15
  4.4 0.12
  5 0.1
  5.6 0.07
};
\end{tikzpicture}

\footnotesize\begin{tikzpicture}
\catcode`\:=12
\datavisualization[scientific clear axes,smooth line plot,
  x axis={length=70mm,grid={step=1,minor steps between steps=9},
    ticks={some},label=Abscisse $x$},
  y axis={length=50mm,grid={step=1,minor steps between steps=9},
    ticks={many},include values={0,1.25},label=Ordonnée $y$}]
data [format=function]{
  var x : interval[0:6] samples 50;
  func y = 1/((\value x-1.5)^2+0.9);}
\tikzset{choisitmark/.style={mark=o,mark
  options=thin,mark size=1.5pt}}
\datavisualization[scientific clear axes,
  scatter plot,
  x axis={length=70mm,ticks={none},
    include values={0,6}},
  y axis={length=50mm,ticks={none},
    include values={0,1.25}}]
data [format=tableau,source=t/tableau1.dat];
\tikzset{choisitmark/.style={mark=square,mark
  options=thin,mark size=1.3pt}}
\datavisualization[scientific clear axes,
  scatter plot,
  x axis={length=70mm,ticks={none},
    include values={0,6}},
  y axis={length=50mm,ticks={none},
    include values={0,1.25}}]
data [format=tableau,source=t/tableau2.dat];
\end{tikzpicture}
```



On teste ci-dessus :

- la possibilité de placer plusieurs groupes de données sur le même graphique ;
- la possibilité de lire les données dans des fichiers en utilisant l'option :  
`source={nom de fichier}`
- la possibilité d'utiliser les différents glyphes déjà définis dans la bibliothèque `plotmarks` [43]. En attendant une nouvelle version de TikZ, on propose pour cela une très petite modification du fichier A.

Cette modification consiste en l'ajout de deux lignes :

- entre les lignes 1867 et 1868 (`style sheet` etc. et `{mark=x,mark options` etc.), on rajoute la ligne :

```
{choisitmark}
```

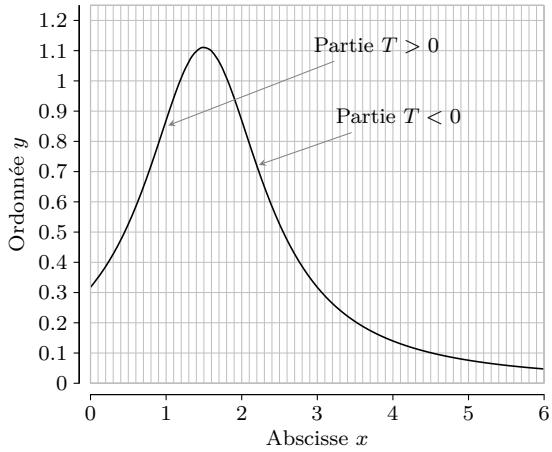
- après la ligne 1577 (`}`), on rajoute la ligne :

```
\tikzset{choisitmark/.style={mark=x,mark options=very thin,mark size=1.5pt}}
```

Dans la longue macro avant modification, c'est la ligne 1868, c'est-à-dire le premier élément de la liste de 6 éléments entre accolades (`{...}`), qui, par défaut, est seul pris en compte. Donc, il est aisément de voir que, par défaut, on aura le même résultat avec la modification. Par contre, on pourra utiliser les glyphes définis dans la bibliothèque `plotmarks` en modifiant certaines propriétés grâce à de nouvelles définitions du style `choisitmark`.

On va terminer ce chapitre en montrant comment on peut signaler une courbe, une partie de courbe ou encore un point (pour donner une explication supplémentaire sur le graphique lui-même). Pour cela, on va tout simplement définir une nouvelle valeur pour la clé `mark` avec les deux macros suivantes :

```
{\catcode`\:=12
\global\def\Pin#1#2#3{\node[inner sep=0pt,pin={[pin distance=#1mm,
    pin edge={>=stealth,<-}]\#2:#3}] (A) at (0,0){};}
\def\FaitPin#1#2#3{\pgfdeclareplotmark{Comment}%
  {\Pin{#1}{#2}{#3}\pgfusepathqstroke}}
\footnotesize\begin{tikzpicture}
\catcode`\:=12
\datavisualization[scientific clear axes,smooth line plot,
    x axis={length=60mm,grid={step=1,minor steps between steps=9},
        ticks={some},label=Abscisse $x$},
    y axis={length=60mm,grid={step=1,minor steps between steps=9},
        ticks={many},include values={0,1.25},label=Ordonnée $y$}]
data [format=function]{
    var x : interval[0:6] samples 50;
    func y = 1/((\value x-1.5)^2+0.9);};
\tikzset{choisitmark/.style={mark=Comment}}
\FaitPin{20}{30}{Partie $T>0$}
\datavisualization[scientific clear axes,
    scatter plot,
    x axis={length=60mm,ticks={none},
        include values={0,6}},
    y axis={length=50mm,include values={0,1.25},
        ticks={none}}]
data [format=tableau]{1 0.85};
\FaitPin{10}{30}{Partie $T<0$}
\datavisualization[scientific clear axes,
    scatter plot,
    x axis={length=60mm,ticks={none},
        include values={0,6}},
    y axis={length=50mm,include values={0,1.25},
        ticks={none}}]
data [format=tableau]{2.2 0.72};
```



Le fonctionnement est simple : la modification fait que la clé `choisimark` est exécutée ; elle est redéfinie pour donner la valeur `Comment` à la clé `mark` ; ensuite, la commande `\FaitPin` configure `Comment` à la demande grâce à ses trois paramètres. Ça paraît un peu long (en fait il y a simplement à recopier en changeant les trois paramètres de `\FaitPin` et les coordonnées du point visé) ... et ça dépanne bien en attendant la suite. Il faut aussi penser que dans un document donné, on aura un petit nombre de styles de figure ce qui permettra de définir des clés à plusieurs paramètres qui vont beaucoup raccourcir la saisie.

Cric-crac ! mon conte es acabat !