

☞ MANUEL DE PRISE EN MAIN POUR TIKZ

□ Yves SOULET

INTRODUCTION

Si l'on utilise beaucoup PSTricks, pour les figures mais aussi pour de très nombreux éléments de composition tels que numérotations sur des onglets, flèches adaptées à la longueur et à l'inclinaison des lettres, etc., on peut se poser la question suivante : est-ce que PDFTricks, avec sa multitude de fichiers PostScript intermédiaires et leurs transformés en PDF entre les deux passes, est la bonne solution pour produire directement du PDF ?

Le n° 48 (avril 2007) des Cahiers GUTenberg contient un tutoriel TikZ, écrit par Till Tantau et traduit par Yvon Henel, fort agréable à lire et donnant envie d'utiliser l'extension correspondante tikz de L^AT_EX en allant plus loin. Il semble bien que cette extension apporte une réponse à la question posée. Mais alors, « aller plus loin » signifie se plonger dans plusieurs centaines de pages de la documentation de référence très touffue et dans laquelle on a parfois des difficultés pour rassembler ce qui est utilisable et passer rapidement sur ce qui est superflu ou qui constitue des explications sur le fonctionnement interne des commandes.

Ces deux premiers paragraphes exposaient la motivation de l'écriture du Manuel de prise en main pour TikZ publié par les Cahiers GUTenberg (n° 50, avril 2008). TikZ est de plus en plus utilisé et une nouvelle version (version 2.10, 726 p. de documentation) vient d'être mise à disposition. Une syntaxe un peu modifiée et de nouvelles possibilités intéressantes (voir le résumé ci-dessous) justifient une mise à jour de ce manuel et ce d'autant plus que certaines nouvelles bibliothèques, moyennant quelques petites additions, permettent en plus des tracés particulièrement utiles.

Mise à jour article Cah. GUT n° 50 — avril 2008, p. 5-87.

RÉSUMÉ

Tout ce qui est nouveau dans la version 2.10 de TikZ par rapport à la version 1.10 est signalé par la donnée du numéro de la section correspondante. Des additions de l'auteur sont composées en italique.

Les parties reprises dans l'édition 2008 du manuel ont été le plus souvent rerédigées et amplifiées.

Chapitre 1 : Le début, inchangé, est consacré aux coordonnées ainsi qu'aux unités. Ensuite, on insiste sur la manière et l'utilité de nommer les points en donnant des exemples d'utilisation (sect. 4). Les transferts de données de \TeX vers TikZ et de TikZ vers \TeX sont présentés sur des exemples (sect. 5 et 6). De nouvelles représentations des points permettent d'écrire directement certains points, par exemple, étant donné A, B et C on peut directement définir P, projection de C sur AB (sect. 7).

Chapitre 2 : Il traite du tracé de lignes brisées et introduit toutes les opérations possibles : tracé, découpage, remplissage, ombré, etc. (si la ligne est fermée). Sont rajoutées le remplissage par des motifs ou des objets graphiques quelconques ainsi que l'utilisation comme boundingbox (fin sect. 9). Le reste du chapitre détaille toutes les options de tracé et donne un premier aperçu d'utilisation de la couleur avec `xcolor`. Il se termine par les tracés prédéfinis `rectangle` et `grid`.

Chapitre 3 : Le début concerne la mise en œuvre de TikZ : distribution, installation, environnement de base pour les figures et leur intégration dans les documents (avec, éventuellement, un fond, un cadre, etc). Viennent ensuite les explications concernant le positionnement des options : options par défaut, option de figure, etc. La définition de styles par l'utilisateur est présentée avec la nouvelle syntaxe dans divers exemples (sect. 16). L'ancienne commande `tikzstyle` est toujours reconnue. Le chapitre se termine par la présentation du nouveau système de clés-valeurs `pgfkeys` (plus général que `keyval` et `xkeyval`) pouvant être utilisé indépendamment de TikZ (sect. 17). La nouvelle définition des styles par l'utilisateur est replacée dans le contexte du système `pgfkeys` qui permet de disposer de clés définissant des styles ou exécutant du code avec des paramètres (sect. 17). Le cas d'une clé exécutant une commande à un paramètre déjà définie est testé sur un exemple (fin sect. 17).

Chapitre 4 : D'abord, les tracés prédéfinis `circle`, `ellipse`, `arc`, `sin` et `cos` sont présentés; s'y ajoutent les tracés de tangentes aux cercles

(sect. 18.3). Ensuite les courbes de Bézier sont présentées avec leurs syntaxes de tracé et leurs propriétés. La fin est consacrée au tracé pratique de courbes avec les deux options : `plot` et `to` ; plusieurs exemples sont présentés.

Chapitre 5 : La première partie est consacrée aux nœuds : syntaxe et options (présentation, positionnement et composition du contenu). La deuxième partie commence par présenter les liaisons possibles entre les nœuds (liaisons par un ou plusieurs segments ou par des courbes) et les labels attachés à ces liaisons. Elle termine avec la construction de labels et de pins attachés aux nœuds.

Chapitre 6 : La première partie développe le tracé de courbes (au sens de la visualisation des données) avec `plot` en proposant plusieurs présentations pour un même exemple (sect. 24.2), ainsi que la méthode de tracé à l'aide de `GNUPLOT`. La deuxième partie est complètement nouvelle et sans documentation : le texte correspondant est rédigé à partir de trois exemples sans explication de la documentation de la version « provisoire » 2.00 CVS et par comparaison du contenu de la bibliothèque de cette version avec le contenu de la bibliothèque de la version actuelle (sect. 25). Cette nouvelle version, très puissante et très facile à utiliser est testée dans plusieurs situations. Pour pouvoir tracer plusieurs courbes et plusieurs séries de points (points expérimentaux par exemple), il a fallu *modifier une commande de la distributions et définir des nouvelles commandes* pour choisir les types de tracés représentant les courbes et les types de glyphes représentant les points ; cela a conduit à montrer *comment définir des glyphes particuliers pour représenter des séries de points expérimentaux* (fin sect. 25).

Chapitre 7 : Le sujet de ce chapitre est entièrement nouveau : il s'agit des intersections de droites (sect. 26) et des intersections de courbes Bézier (sect. 27). Les syntaxes pour obtenir les coordonnées des points d'intersection sont expliquées en détail et testées sur des exemples. Par une *modification d'une commande de la distribution* et la *définition de nouvelles commandes* les temps d'intersection sont obtenus et permettent de définir des souscourbes limitées par des points d'intersection (sect. 28). Cela permet de colorer différemment une souscourbe (par rapport à la courbe complète) et de colorer un domaine limité par des souscourbes. Cette possibilité est *étendue aux courbes définies avec to* (sect. 29).

Chapitre 8 : Il ne contient rien de nouveau. La première partie est

consacrée à la construction des arbres : elle présente les modes de croissance linéaire et cyclique et toutes les possibilités typographiques concernant les nœuds et les liaisons entre générations afin d'améliorer la lisibilité. Deux exemples montrent comment on peut réussir le tracé de formules chimiques complexes. La dernière partie expose la construction de « mindmaps », ces organigrammes complexes (souvent très colorés) ayant la structure d'arbres à croissance cyclique. Après avoir décrit toutes les options disponibles, un exemple (à vocation pédagogique) est construit montrant l'utilisation de toutes ces options.

Chapitre 9 : Il rassemble des options dont certaines ont déjà été utilisées. On y trouve quelques conseils pour trouver de belles couleurs et on y expose en détail comment on obtient l'ombré (ou gradient de couleur). Il y a ensuite la construction de l'ombre portée traditionnelle d'un objet graphique (sect. 33.2) et d'une ombre-répétition (ou ombre multiple) (sect. 33.3). On y trouve aussi l'option d'opacité complétée dans la version actuelle par la construction de masques d'opacité (sect. 34). Vient ensuite l'étude du remplissage en fonction du type des chemins qui est présentée plus profondément que dans le manuel initial. Enfin, on reprend le découpage avec un exemple que l'on ne peut pas terminer de façon satisfaisante, ce qui suggère la définition des calques. Dans ce manuel, on montre comment l'utilisation conjointe des découpages et des calques permet aussi de résoudre les problèmes tels que : colorer différemment une souscourbe (par rapport à la courbe complète) et colorer un domaine limité par des souscourbes. Les exemples des sections 28 et 29 sont ainsi retraités et le lecteur peut se faire une opinion sur le choix de la méthode à utiliser pour les problèmes cités ci-dessus.

Chapitre 10 : On y trouve des outils classiques et de nouveaux outils. D'abord toutes les transformations disponibles sont détaillées et certaines de leurs propriétés sont développées dans cette version du manuel. Ensuite la syntaxe générale des boucles de répétition est exposée sur de nombreux exemples. Enfin deux outils nouveaux sont exposés sur des exemples. Le premier permet de faire des liaisons entre deux nœuds appartenant à deux figures différentes (un nœud pouvant n'être qu'un mot d'un paragraphe) (sect. 40) ; une variante permet de positionner dans la page un objet graphique indépendamment de la composition de la page (fin sect. 40). Le deuxième nouvel outil est la loupe qui permet d'agrandir directement une partie complexe d'une figure (sect. 41).

REMARQUES

Ces quelques pages ont le modeste objectif d'aider à avancer très vite dans l'utilisation de l'extension `tikz`. Elles contiennent l'essentiel pour la réalisation de figures courantes et pour l'utilisation des extensions disponibles.

On s'est imposé de faire suivre chaque nouvelle commande et chaque nouvelle option d'un exemple où le code se trouve juste à gauche de la figure. Compte tenu de l'étroitesse du miroir (identique à celui de la revue) les explications du code des figures sont parfois avant la figure, parfois après. Dans ces morceaux de code, des petites macros sont introduites pour diminuer la longueur de la saisie (on considère que le lecteur a une petite expérience dans l'utilisation de $\text{L}^{\text{T}}\text{E}^{\text{X}}$).

Les références `[x]`, `[x.x]` et `[x.x.x]` se rapportent à la documentation (sections, sous-sections et paragraphes) de la version 2.10 de `TikZ`.

L'index contient toutes les commandes et options et mentionne le numéro des pages de leurs introductions et utilisations principales. Il ne reprend pas les titres de chapitre, section et sous-section qui se trouvent dans la table des matières détaillée.

Certaines traductions méritent des explications :

— *node* relatif à une liaison inter-nœud et *label* relatif à un nœud ont été traduits par « label », plutôt que par étiquette.

— *node*, dans le sens usuel de lettrage (lettres ou autres caractères souvent mathématiques rajoutés sur les figures), a été traduit par « lettrage » ;

— *pin* a été conservé,

— *mindmap* a été également conservé (en réalité ce mot ne figure pas dans un des dictionnaires servant de référence pour les traductions anglais-français).

Certaines dénominations sont un peu anciennes, lignes brisées et lignes courbes par exemple ; en se limitant aux lignes brisées, on peut bien avancer dans « l'esprit `TikZ` » sans se compliquer la vie avec les courbes de Bézier : cela constitue un encouragement pour le débutant.

Pour avoir des petites figures face au code correspondant, on a toujours utilisé des petites dimensions et des petites tailles de caractère. Pour faciliter l'impression, la couleur a été évitée autant que possible mais ce choix n'a pas pu être respecté comme on l'aurait souhaité.

Il faut signaler au lecteur que `GSView` (et donc `GhostScript` également)

ne peuvent afficher convenablement certains effets de TikZ : fonds remplis avec des motifs (`patterns`), masques de transparence et opacité.

Enfin, il faut signaler que l'extension `tikz` est une couche de code permettant l'utilisation du langage PGF utilisé par d'autres styles ou classes, BEAMER par exemple. Ce langage comprend lui-même deux couches : la couche « de base » et la couche plus profonde « système », parties VII et VIII de la documentation. Il est parfois nécessaire d'utiliser des commandes de la couche de base (préfixe `\pgf`) mais le lecteur de ce manuel n'a pas besoin de consulter ces deux parties de la documentation.

Il faut reconnaître que le débutant peut être un peu affolé par le nombre de commandes et d'options, mais on « s'y fait vite » dès que l'on réalise que, pour disposer de nombreuses possibilités, il faut « en payer le prix » d'une façon ou d'une autre. La meilleure façon d'apprendre TikZ est de commencer à l'occasion d'une série de figure à faire. On aura peut-être l'impression de perdre du temps ; c'est faux ! Pour les séries de figures suivantes, même si l'on a l'impression d'avoir tout oublié, on s'apercevra que tout revient très vite en consultant ce que l'on a déjà fait.

L'expérience montre que pour travailler vite et bien, il est *très utile* d'écrire des petites macros simples du type `\pose (.) [] { . } { . }`, (cf. sect. 7), bien utiles pour le lettrage des figures, mais aussi des styles du type `tn`, `tr` (cf. sect. 16.1) ou encore du type `every rectangle node` (cf. sect. 16.2). Cela assure en outre la parfaite homogénéité des séries de figures.

AVERTISSEMENT

Quatre types de fichiers (en plus des fichiers de composition) sont associés (sous forme compactée) au fichier PDF de ce manuel :

- fichiers de données (`.dat`) : données associées à certaines figures,
- fichier `macsup.tex` : petites macros de composition du texte,
- fichier `mactikz.tex` : petites macros TikZ utiles dont certaines *exigent* le chargement du fichier suivant,
 - fichier `mactikz.sty` : macros de la distribution modifiées (ce qui évite de toucher aux fichiers originaux),
 - fichier `manualmindmap.tex` : macros pour créer des mindmaps dont les dimensions sont adaptées aux dimensions des présentes pages (format plus petit que A4 mais plus grand que A5).

TABLE DES MATIÈRES

Introduction	1
Résumé	2
Remarques	5
Avertissement	6
Chapitre 1. Points et leurs représentations	11
1. Coordonnées cartésiennes	11
2. Coordonnées polaires	12
3. Déplacements et tracés relatifs	12
4. Nommage des points	13
5. Transfert de données de \TeX vers \tikz	13
6. Transfert de données de \tikz vers \TeX	14
7. Formes spéciales de représentation des points	15
Chapitre 2. Lignes brisées	17
8. Syntaxe de base	17
9. Options principales	18
10. Options de tracé	20
10.1. Epaisseur de traits	20
10.2. Terminaison des traits	21
10.3. Jonction des traits	21
10.4. Arrondissement des angles	21
10.5. Pointillés et traitillés	22
10.6. Flèches	23
10.7. Double trait	24
10.8. Couleur	24
11. Tracés prédéfinis : rectangle et grid	25
12. Tracés particuliers	26
Chapitre 3. La machinerie \tikz	29
13. Préparations préliminaires	29
13.1. Installation des fichiers	29
13.2. Ajouts au préambule	29
14. Utilisation de \tikz	30
14.1. Environnement de base	30
14.2. Intégration des figures \tikz	30

15.	Manipulation des options	32
15.1.	Placement en option de figure	32
15.2.	Placement en option d'une partie de figure	32
15.3.	Placement en option de commande	33
15.4.	Options dans les spécifications d'une commande	34
16.	Définition de styles par l'utilisateur	34
16.1.	Commande de définition d'un style	35
16.2.	Styles attachés à des éléments graphiques	36
17.	Généralisation du système de clés-valeurs : pgfkeys	37
Chapitre 4. Lignes courbes		41
18.	Constructions prédéfinies : circle, ellipse et arc	41
18.1.	Cercle et ellipse	41
18.2.	Arc	42
18.3.	Tangentes aux cercles et aux arcs	42
18.4.	Sinus et cosinus	43
19.	Courbes de Bézier	43
19.1.	Syntaxe générale	43
19.2.	Syntaxe spéciale	45
19.3.	Premières applications	46
20.	Tracé pratique de courbes	47
20.1.	Tracer une courbe avec plot	47
20.2.	Tracer une courbe avec to	50
Chapitre 5. Nœuds, liaisons et labels :		
	organigrammes, algorithmes...	53
21.	Nœuds	53
21.1.	Syntaxe générale	53
21.2.	Options de présentation	54
21.3.	Options de positionnement	55
21.4.	Options de composition	56
22.	Liaisons et labels attachés	57
22.1.	Liaisons par un seul segment	58
22.2.	Liaisons à 2, 3 ou 4 segments parallèles aux axes	59
22.3.	Liaisons par lignes courbes	62
23.	Labels et « pins » associés aux nœuds	63
Chapitre 6. Visualisation des données		67

24. Tracé de courbe direct avec plot	67
24.1. Deuxième syntaxe	67
24.2. Troisième syntaxe	68
24.3. Quatrième syntaxe	70
25. Tracé de courbes : version avancée	72
25.1. Tracés « élémentaires »	73
25.2. Tracés dits « scientifiques »	75
Chapitre 7. Intersections de lignes	83
26. Intersection de droites	83
27. Intersection de deux courbes	84
28. Intersection et courbes partielles	86
29. Intersection des courbes avec to	88
Chapitre 8. Arbres et structures arborescentes	93
30. Arbres	93
30.1. Structure de base	93
30.2. Adaptation de la structure de base	95
30.3. Application à la composition de formules chimiques	97
31. Mindmaps	99
Chapitre 9. Colorer et ombrer, opacité et transparence, remplissage, découpages et calques	105
32. Couleur	105
33. Ombré et ombre portée	106
33.1. Ombré proprement dit	106
33.2. Ombre portée	107
33.3. Ombre-répétition	108
34. Opacité et masque de transparence	109
35. Remplissage	111
36. Découpages et calques	113
37. Découpages et calques : reprise des exemples	116
Chapitre 10. Principaux outils disponibles	119
38. Transformations	119
39. Boucles de répétition : foreach	122
40. Liaisons entre figures et positionnement sur la page	125
41. La loupe	126

42. Et les outils dont on n'a pas parlé...	128
Conclusion	128
Index	130

CHAPITRE 1

Points et leurs représentations

Dans ce premier chapitre, on introduit les notations pour la représentation d'un point par ses coordonnées ou par l'attribution d'un nom. Dans les exemples suivants, les points sont les extrémités des segments tracés avec la commande `\draw` et la spécification `--` qui seront vues dans le prochain chapitre. Une option de cette commande, `xshift`, est parfois utilisée pour translater l'origine afin de bien séparer les différentes parties de l'exemple. Une macro (nommée `\ppmm` et ne figurant pas sur les listings des figures), affiche une grille d'un pas de 1 millimètre et des axes gradués, permettent de visualiser plus facilement l'effet des nouvelles commandes introduites. On ajoute :

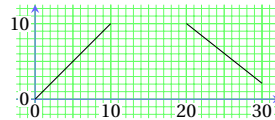
```
\usepackage{tikz}
```

dans le préambule et on est prêt à faire le premier pas !

1. COORDONNÉES CARTÉSIENNES

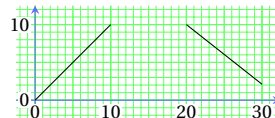
En cartésiennes, le point de coordonnées $x = a$ cm et $y = b$ cm est représenté par $(a \text{ cm}, b \text{ cm})$ ou, puisque le cm est l'unité de longueur par défaut, par (a, b) [2.15]. On peut aussi utiliser le mm ou le pt.

```
\begin{tikzpicture}
\draw(0,0)--(1,1);
\draw(2cm,28.5pt)--(30mm,6pt);
\end{tikzpicture}
```



On peut aussi changer l'unité par défaut en ajoutant `x=1mm,y=1mm` dans l'option de figure et utiliser la notation sans unité dans toute la figure [13.2.1]. Cependant, on peut encore utiliser d'autres unités dans les commandes.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)--(10,10);
\draw(2cm,10)--(3cm,6pt);
\end{tikzpicture}
```

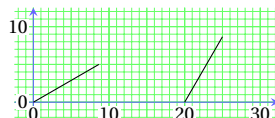


On peut aussi choisir des unités différentes pour les deux coordonnées, ce qui peut être parfois utile [13.2.1]. Par contre, l'ajout de `scale=(nombre)` à l'option de figure permet de redimensionner la figure [2.10]. Il faut savoir que toutes les dimensions sont concernées : avec `scale=2`, une ligne tracée avec l'option `line width=0.8pt` aura une épaisseur de 1,6 pt alors que, définie avec l'option `thick` (option d'épaisseur définie préalablement), l'épaisseur ne sera pas changée [2.10]. Cet inconvénient peut être évité en multipliant les unités des axes par 2 ... mais alors il faut adapter les graduations prédéfinies des axes s'il y en a.

2. COORDONNÉES POLAIRES

En polaires, le point de coordonnées $\rho = r$ cm et $\theta = w^\circ$ est noté $(w:r)$ cm) ou $(w:r)$ [2.15]. L'unité de longueur pour le rayon ρ est celle valant pour les coordonnées cartésiennes.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)--(30:10);
\draw[xshift=20mm](0,0)--(60:10);
\end{tikzpicture}
```



3. DÉPLACEMENTS ET TRACÉS RELATIFS

Après avoir introduit la représentation d'un point par ses coordonnées, on va voir maintenant un complément à cette notation particulièrement pratique [2.15] :

$+(a,b)$, resp. $+(w:r)$, désigne le point déplacé de a suivant les x et de b suivant les y , resp. déplacé de r suivant la direction w , par rapport à la position précédente qui est conservée,

$++(a,b)$, resp. $++(w:r)$, désigne le point déplacé de a suivant les x et de b suivant les y , resp. déplacé de r suivant la direction w , par rapport à la position précédente puis amène la position précédente au point déplacé. L'exemple montre un chemin noté simplement avec la notation initiale, puis avec deux variantes du code pour montrer le rôle des commandes $+$ et $++$.

```
\begin{tikzpicture}
\draw(0,0)--(1,0)--(2,1)--(3,0);
% ou \draw(0,0)--(1,0)---(1,1)--(3,0);
% ou \draw(0,0)--(1,0)---+(1,1)---+(1,-1);
\end{tikzpicture}
```



4. NOMMAGE DES POINTS

On peut nommer un point à partir de ses coordonnées à l'aide de la commande [4.1.2] :

```
\coordinate(A)at(a,b); (ou at(w:r);)
```

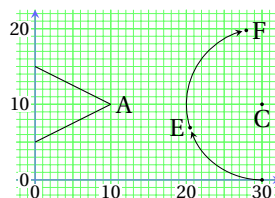
On peut utiliser (A) à la place de la paire de coordonnées (a,b) (ou (w:r)) dans les tracés ... si l'on dispose des coordonnées a et b (sur l'exemple, on constate que (10,10) et (A) représentent bien le même point). Par contre, il y a des situations où interviennent des points dont on ne dispose pas explicitement des coordonnées (on peut les extraire mais pour cela il faut manipuler des nombres décimaux); voilà deux cas de points où l'on ne connaît pas explicitement les coordonnées :

— point obtenu à partir d'un point initial de coordonnées connues (ou d'un point déjà nommé) par l'intermédiaire d'une transformation, par exemple une rotation (sur l'exemple, le point E est obtenu à partir du point de coordonnées (30,0) par une rotation de -72° autour du point C, F est obtenu à partir de E par une autre rotation du même type, toujours sans manipuler explicitement les coordonnées);

— point obtenu par une opération telle que le calcul des points d'intersection de deux courbes où la paire de coordonnées de ces points est représentée par un nom du type $pti-n$ (n -ième intersection), on écrit :

```
\coordinate(A)at(pti-n);
```

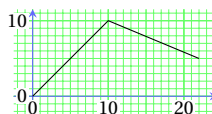
```
\begin{tikzpicture}[x=1mm,y=1mm]
\coord(A)at(10,10); %\coord:coordinate
\draw (0,5)--(10,10);\draw (0,15)--(A);
\coord(C)at(30,10);
\coord[rotate around={-72:(C)}](E)at(30,0);
\coord[rotate around={-96:(C)}](F)at(E);
\end{tikzpicture}
```



5. TRANSFERT DE DONNÉES DE T_EX VERS TikZ

On peut vouloir utiliser dans des commandes TikZ des dimensions ou des compteurs déjà définis en T_EX. Voilà un exemple :

```
\newlength{\lga}\newcounter{cpta}%
\setlength{\lga}{10mm}\setcounter{cpta}{10}%
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)--(\lga,\value{cpta})
--(\lga+12mm,\value{cpta}-5);
\end{tikzpicture}
```



Cet exemple montre que les données provenant de définitions $\text{T}_\text{E}\text{X}$ peuvent être additionnées avec longueurs TikZ à la condition que les longueurs soient données avec la même unité : par exemple $\backslash\text{ga}+12$ (cf. exemple précédent) serait traité comme $\backslash\text{ga}+12\text{pt}$ bien que l'unité par défaut soit le mm et que $\backslash\text{ga}$ soit définie en mm (voir la remarque dans [13.2.1]).

6. TRANSFERT DE DONNÉES DE TikZ VERS $\text{T}_\text{E}\text{X}$

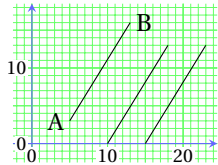
Inversement on peut vouloir utiliser en $\text{T}_\text{E}\text{X}$ des données calculées en TikZ . On va voir comment on peut récupérer les coordonnées de points définis par des opérations comme les points considérés dans la section précédente. On dispose des macros $\backslash\text{p1}$, $\backslash\text{p2}$, jusqu'à $\backslash\text{p9}$ et $\backslash\text{p}\{(\text{nom})\}$ pour représenter des points [4.1.3] [14.15] ; pour les définir on écrit : $\text{let } \backslash\text{p1}=(\$(\text{A})\$) \text{ in, let } \backslash\text{p2}=(\$(\text{B})-(\text{A})\$) \text{ in, etc.}$ Ces macros (toujours définies entre let et in) ne sont utilisables qu'à l'intérieur de la commande où elles sont définies (voir l'exemple).

Quand ces définitions sont posées, alors :

$\backslash\text{x1}$ et $\backslash\text{y1}$ sont les coordonnées du point A, $\vec{\text{AB}}$,
 $\backslash\text{x2}$ et $\backslash\text{y2}$ sont les composantes du vecteur $\vec{\text{AB}}$, etc.

On dispose aussi des macros $\backslash\text{n1}$, $\backslash\text{n2}$, etc. pour représenter des nombres. Voici un exemple de manipulation de ces macros :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\coord(A)at(5,3);\coord(B)at(13,16);
\draw let \p1=(\$(\text{A})\$), \p2=(\$(\text{B})\$) in (\p1)--(\x2,\y2);
\draw let \p1=(\$(\text{B})-(\text{A})\$), \n1={\veclen(\x1,\y1)},
      \n2={\atan(\y1/\x1)} in
      \pgfextra{\xdef\longueur{\n1}\xdef\angle{\n2}}
      (10,0)--+(\n2:\n1);
\draw (15,0)--+(\angle:\longueur);
\end{tikzpicture}
```



Le premier tracé montre que $\backslash\text{p1}$, resp. $\backslash\text{x2}$ et $\backslash\text{y2}$, représentent bien le point A, resp. le point B. Le deuxième tracé, excepté la ligne portant la commande $\backslash\text{pgfextra}$, montre que $\backslash\text{n1}$, resp. $\backslash\text{n2}$, est bien la longueur du vecteur $\vec{\text{AB}}$, resp. la direction de ce vecteur. On a utilisé la macro TikZ $\text{veclen}(a, b)$ qui donne la longueur du vecteur de composantes a et b .

Enfin, la ligne avec la macro $\backslash\text{pgfextra}$ permet d'extraire les valeurs de $\backslash\text{n1}$ et $\backslash\text{n2}$ en dehors de la commande $\backslash\text{draw}$. Pour cela on défini-

nit alors des macros \TeX qui prennent le contenu de ces macros et le conservent en dehors de tout environnement dans lequel elles sont définies (cf. la macro $\TeX \backslash xdef$) et on place ces définitions en argument de la macro \pgfextra qui finalise cette extraction [14.18]. Le tracé de la dernière ligne montre que la longueur et la direction du vecteur \overrightarrow{AB} ont bien été « transportées » (elles sont disponibles pour les figures suivantes).

7. FORMES SPÉCIALES DE REPRÉSENTATION DES POINTS

On a déjà écrit les formes ($\lga+12mm, \value{cptA}-5$) et ($\angle: \longueur$) pour les coordonnées d'un point (cf. sect. 5 et 6) : on a donc utilisé le fait que chacune des composantes peut être une somme et qu'un nombre peut être remplacé par une macro. En réalité, la paire de coordonnées peut avoir une forme bien plus générale, particulièrement avantageuse lorsque l'on travaille avec des boucles [13.5]. On donne d'abord quelques exemples :

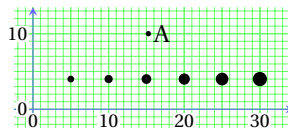
$(\$*(1,2)\$)$, $(\$3*\sin(60)*(1,2)\$)$, etc.,

$(\$(A)+3*(4,2)\$)$, etc.,

$\$(1, \i)\$, \$(\i*(2,3)\$)$, etc. où \i est un indice de boucle.

Ces formes exigent le chargement de la bibliothèque `calc`. On en explicite deux applications (on remarquera que les $\$$ entourant les coordonnées ne sont pas toujours nécessaires) :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\coord(A)at{(\$*(0.5*\longueur,10)\$)};
\ptn(A)(1)\pose(A){0}{A}
\foreach \i in {5,10,...,30}
  {\ptn(\i,12)(1+\i/18)}
\end{tikzpicture}
```



On a utilisé dans cet exemple la macro de boucle :

— $\foreach \i in \{...\}\{(commandes \text{ à effectuer})\}$;

et les deux macros privées suivantes (fichier de macros `mactikz.tex` :

— $\ptn(\#1)(\#2)$ trace un point noir en $(\#1)$ de rayon $\#2$ pt ; $(\#1)$ peut être du type (a,b) ou (A) ;

— $\pose(\#1)[\#4][\#2][\#3]$ place le texte $\#3$ au voisinage du point $\#1$ (cf. ci-dessus) dans la direction $\#2^\circ$; l'argument facultatif ajoute (à la demande) un éloignement supplémentaire de $\#4$ pt. Cette macro sert à poser le lettrage sur les figures ; la version \poseb de cette macro trace le

texte sur un fond blanc pour améliorer la lisibilité.

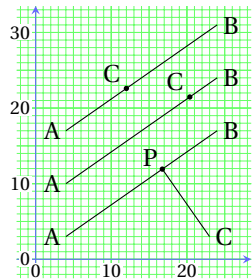
Il y a encore trois formes particulières pour les coordonnées qui peuvent être fort utiles :

$(\$ (A) ! 0.4 ! (B) \$)$ représente le point C situé sur la ligne AB, tel que $\vec{AC} = 0.4 \times \vec{AB}$,

$(\$ (A) ! 20mm ! (B) \$)$ représente le point C situé sur la ligne AB, tel que $|\vec{AC}| = 20mm$,

$(\$ (A) ! (C) ! (B) \$)$ représente la projection P du point C sur la ligne AB. Dans ces formes, les points A et B peuvent être aussi représentés par les paires de coordonnées (x,y). Voilà un exemple de chaque forme :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\catcode'\!=12
\coord(A)at(4,17);\coord(B)at(24,31);
\ptn($ (A) ! 0.4 ! (B) $) (1)\draw[thin] (A)--(B);
\pose(A){180}\pose(B){0}\pose(C){135}\pose(C){135}\coord(C)at($ (A) ! 0.4 ! (B) $);
\coord(A)at(4,10);\coord(B)at(24,24);
\ptn($ (A) ! 20mm ! (B) $) (1)\draw[thin] (A)--(B);
\pose(A){180}\pose(B){0}\pose(C){135}\pose(C){135}\coord(C)at($ (A) ! 20mm ! (B) $);
\coord(A)at(4,3);\coord(B)at(24,17);
\coord(C)at(23,3);\pose(C){0}\pose(C){0}\ptn($ (A) ! (C) ! (B) $) (1)\draw[thin] (A)--(B);
\draw[thin] ($ (A) ! (C) ! (B) $)--(C);
\pose(A){180}\pose(B){0}\pose(B){0}\coord(P)at($ (A) ! (C) ! (B) $);\pose(P)[-1]{135}\pose(P)[-1]{135}
\end{tikzpicture}
```



On notera le changement de catcode du point d'exclamation (qui est déclaré « actif » par l'extension frenchb); ce problème se pose quelques fois dans TikZ (il ne se pose pas pour les deux points de la notation des coordonnées polaires).

CHAPITRE 2

Lignes brisées

On a choisi de consacrer un chapitre aux lignes brisées, ouvertes ou fermées, car la syntaxe correspondante, relativement simple, se prête bien à l'introduction des options dont la plupart se retrouveront dans la syntaxe correspondant aux lignes courbes. Dans tous les exemples qui vont être proposés, on utilise le `mm` pour unité (ce qui permet d'éviter l'usage des nombres décimaux) et on ajoute presque toujours une grille et des axes comme précédemment.

8. SYNTAXE DE BASE

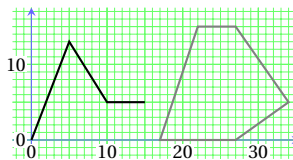
Cette syntaxe de base est la commande [14.2.1] :

`\path [options] (spécifications) ;`

Les options sont par exemple la commande de tracé¹, l'épaisseur du trait, la couleur du trait, etc. Les spécifications de base sont les coordonnées des extrémités des segments de la ligne brisée, le tracé des segments (`--`) et la fermeture (`--cycle`) s'il s'agit d'une ligne brisée fermée.

Voici un exemple de ligne brisée ouverte et de ligne brisée fermée :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,thick](0,0)
--(5,13)--(10,5)--(15,5);
\path[xshift=17mm,draw=gray,thick](0,0)--
(5,15)--(10,15)--(17,5)--(10,0)--cycle;
\end{tikzpicture}
```



Les options se trouvant dans les exemples ci-dessus et leur rôles sont :
`draw` : trace la ligne,
`very thick` : choisit l'épaisseur de la ligne (1.2 pt),

¹On peut définir un chemin pour le remplir ou pour découper une partie de figure et cela sans vouloir le tracer.

gray : choisit la couleur,
xshift : translate l'origine suivant les x .

9. OPTIONS PRINCIPALES

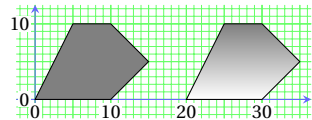
Les options principales pour une ligne sont [15.1] :

- draw : trace la ligne avec la couleur choisie[15.3],
- fill : colore l'intérieur de la ligne avec la couleur choisie [15.4],
- shade : ombre l'intérieur de la ligne fermée [15.6],
- clip : découpe l'intérieur de la ligne fermée [15.8],
- pattern : remplit l'intérieur de la ligne fermée de motifs répétitifs [15.4.1],
- path picture : remplit l'intérieur de la ligne fermée avec un objet ou une partie d'un objet graphique [15.5],
- use as bounding box : fait de la ligne fermée une bounding box prévalant sur la bounding automatiquement calculée [15.7].

On a donné des exemples de la première option sous sa forme abrégée au chapitre 1 (ces abréviations seront définies en fin de section).

Voici des exemples des trois options suivantes.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,fill=gray](0,0)--(5,10)
--(10,10)--(15,5)--(10,0)--cycle;
\path[xshift=20mm,draw,shade](0,0)--
(5,10)--(10,10)--(15,5)--(10,0)--cycle;
\end{tikzpicture}
```



Remarque : On note tout de suite que la spécification `--cycle` trace le dernier segment alors que la spécification `--(0,0)` fermerait aussi la ligne brisée mais imparfaitement sur le plan graphique. Cette imperfection n'est visible que pour les épaisseurs de trait assez grandes et ne pourra être comprise que plus loin.

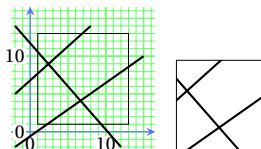
Remarque : On peut vouloir des couleurs différentes pour le tracé et le remplissage : il faut alors écrire `draw=(couleur)` et `fill=(couleur)`, la couleur par défaut étant le noir dans les deux cas.

L'option `clip` est telle que la ligne fermée découpe tous les tracés qui suivent et n'en conserve que les parties situées dans son intérieur. On peut tracer en même temps la ligne mais sans option de tracé : la commande `\path[draw,clip]` est acceptée mais on ne peut donner aucune option supplémentaire à l'option `draw`; c'est parce que l'on

voulait une ligne de découpe en trait très fin (`very thin`) que l'on a du faire le tracé avant de faire la découpe (une autre possibilité consisterait à déclarer l'option `very thin` comme option courante).

L'exemple montre la figure initiale et ensuite la partie découpée :

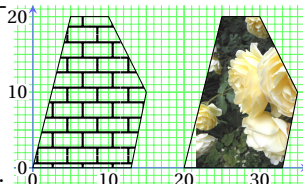
```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,thick](-2,-2)--(15,10) (-2,5)--(8,14) (-2,14)--(12,-2);
\path [draw,very thin](1,1)--(13,1)--(13,13)--(1,13)--cycle;
\end{tikzpicture}
\hspace{2mm}
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,very thin](1,1)--(13,1)
--(13,13)--(1,13)--cycle;
\path[clip](1,1)--(13,1)--(13,13)
--(1,13)--cycle;
\path[draw,thick](-2,-2)--(15,10) (-2,5)--(8,14) (-2,14)--(12,-2);
\end{tikzpicture}
```



Le remplissage d'une ligne fermée par des motifs répétitifs nécessite le chargement de la bibliothèque `patterns`; il suffit d'introduire `pattern=(motif choisi)` en option.

Le remplissage de la ligne fermée par un objet ou une partie d'objet graphique se fait selon la syntaxe un peu lourde ci-dessous; en fait elle est claire : la ligne fermée « clippe » une partie de l'objet graphique dont le centre est amené au centre de la boundingbox de la ligne fermée (on peut choisir un autre point bien évidemment). Dans l'exemple, l'objet graphique est une photo insérée avec `includegraphics`; on pourrait aussi utiliser du texte en remplaçant `{\includegra...}` par `{\parbox[.]{(largeur)}{(Texte)}}`.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\path[draw,pattern=bricks](0,0)--(5,20)--
(10,20)--(15,10)--(13,0)--cycle;
\path [xshift=20mm,draw,path picture={%
\node at (path picture bounding
box.center){\includegraphics%
[height=2cm]{begonia.jpg}};}] (0,0)--
(5,20)--(10,20)--(15,10)--(13,0)--cycle;
\end{tikzpicture}
```



La dernière option consiste à se donner un rectangle pour remplacer la boundingbox créée par TikZ. Cela permet de réserver du blanc autour

de la figure ou, au contraire, de superposer des parties de la figure sur le texte adjacent ; mais l'application importante de cette option est l'élimination des points de contrôle des courbes de Bézier que TikZ inclue dans la boundingbox ; ils se trouvent parfois en dehors du rectangle minimum contenant le tracé, ce qui oblige à réduire la boundingbox (les points de contrôle des courbes seront vus au chapitre 4). La syntaxe est :

```
\path[use as bounding box] (a,b) rectangle (c,d) ;
```

Enfin voici cinq abréviations qui sont souvent utilisées [11.3] :

```
— \draw [options] et \fill [options]
```

équivalent respectivement à

```
\path [draw,options] et \path [fill,options]
```

```
— \filldraw [options] et \shadedraw [options]
```

équivalent respectivement à

```
\path [draw,fill,options] et \path [draw,shade,options]
```

```
— \clip [options] équivaut à \path [draw,fill,options]
```

Remarque : Il faut être attentif dans le cas où on trace et remplit une ligne fermée. Pour un tracé seul, on peut écrire :

```
\draw[red,...] ... ;
```

mais, dans le cas où on demande les deux actions, il faut mieux préciser :

```
\path[draw=red,fill=blue,...] ... ;
```

10. OPTIONS DE TRACÉ

10.1. EPAISSEUR DE TRAIT

L'option pour l'épaisseur des traits est [15.3.1] :

```
line width=(dimension)
```

On dispose aussi de styles prédéfinis (défaut : thin) :

style=ultra thin, very thin, thin, semithick, thick, very thick et ultra thick (défaut : 0.4 pt)

qui correspondent respectivement à 0.1 pt, 0.2 pt, 0.4 pt, 0.6 pt, 0.8 pt, 1.2 pt et 1.6pt (défaut : 0.4pt).

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

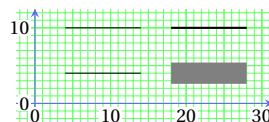
```
\draw[thin] (4,10)--+(10,0) ;
```

```
\draw(4,4)--+(10,0) ;
```

```
\draw[thick] (18,10)--+(10,0) ;
```

```
\draw[line width=8pt,gray] (18,4)--+(10,0) ;
```

```
\end{tikzpicture}
```



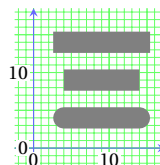
10.2. TERMINAISON DES TRAITES

Elle est produite par l'option [15.3.1] :

`cap=rect, butt` ou `round` (défaut : `butt`)

Les trois possibilités sont montrées sur l'exemple.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt, cap=rect, gray] (4,14)---+(10,0);
\draw[line width=8pt, gray] (4,9)---+(10,0) ;
\draw[line width=8pt, cap=round, gray] (4,4)---+(10,0) ;
\end{tikzpicture}
```



On constate que le premier et le dernier choix conduisent à un dépassement d'une demi épaisseur de trait. Il est évident que cette option joue un rôle important pour les grandes épaisseurs de trait.

10.3. JONCTION DES TRAITES

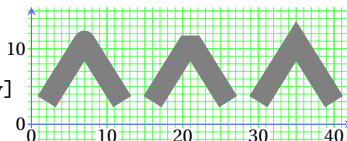
Elle est déterminée par l'option [15.3.1] :

`join=round, bevel` ou `miter` (défaut : `miter`)

L'option par défaut permet encore de revenir aux angles vifs si nécessaire.

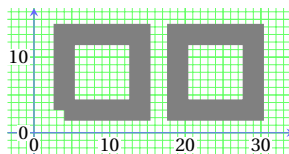
Les trois possibilités sont montrées sur l'exemple.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt, join=round, gray]
(2,3)---+(5,8)---+(5,-8);
\draw [line width=8pt, join=bevel, gray]
(16,3)-- ++(5,8)--+(5,-8);
\draw [line width=8pt, gray]
(30,3)---+(5,8)---+(5,-8); \end{tikzpicture}
```



Il est encore évident que cette option joue un rôle important pour les grandes épaisseurs de trait. Et voilà enfin la raison pour laquelle il faut toujours fermer une ligne (brisée ou courbe) par `--cycle` [11.4] :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=8pt, gray] (4,3)---+
(0,10)---+(10,0)---+(0,-10)---(-10,0);
\draw [line width=8pt, gray] (19,3)---+
(0,10)---+(10,0)---+(0,-10)--cycle;
\end{tikzpicture}
```



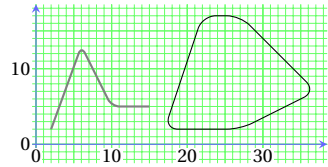
10.4. ARRONDISSEMENT DES ANGLES

Il est réalisé grâce à l'option [14.6] :

`rounded corners=(dimension)` (défaut : `sharp corners`)

où cette dimension est le rayon de courbure souhaité. L'option par défaut permet de revenir aux angles vifs si nécessaire.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,gray,rounded corners=3pt]
(2,2)--(6,13)--(10,5)--(15,5);
\draw[xshift=17mm,rounded corners=5pt]
(0,2)--(5,17)--(10,17)--(20,7)
--(10,2)--cycle;
\end{tikzpicture}
```



10.5. POINTILLÉS ET TRAITILLÉS

Les pointillés sont obtenus l'option [15.3.2] :

`densely dotted`, `dotted`, `loosely dotted` ou `solid` (défaut : `solid`, pas de pointillés).

Les traitillés sont obtenus avec l'option [15.3.2] :

`style=densely dashed`, `dashed`, `loosely dashed` ou `solid` (défaut : `solid`, pas de traitillés).

On abandonne les pointillés et traitillés avec le style `solid`.

On dispose aussi de l'option `dashdotted` et de quelques autres cousines. On peut aussi définir son propre motif de pointillé en déclarant un style qui donne une valeur à la clé `dash pattern` définissant ce motif (voir la première ligne du code de l'exemple pour ce type de déclaration qui sera vu au prochain chapitre). Ce motif se répète ensuite indéfiniment, comme en langage MetaFont ; dans l'exemple suivant, le nom choisi pour le style est, pour le style est suivant `perso dashed`.

Il possible de décaler le motif au début du traitillé (pour harmoniser les coupures des tirets extrêmes) avec l'option :

`dash phase=(dimension)`

```
\tikzset{perso dashed/.style={dash pattern=
{on 2pt off 3pt on 6pt off 3pt}}}%
\hfill\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,loosely dotted](3,23)--+(10,0);
\draw[thick,dotted](3,18)--+(10,0);
\draw[thick,densely dotted](3,13)--+(10,0);
\draw[thick,loosely dashed](20,23)--+(10,0);
\draw[thick,dashed](20,18)--+(10,0);
\draw[thick,densely dashed](20,13)--+(10,0);
\draw[thick,dashdotted](3,8)--(30,8);
\draw[thick,perso dashed](3,3)--(30,3);\end{tikzpicture}
```

10.6. FLÈCHES

On dispose d'un très grand nombre de possibilités pour composer des flèches en chargeant la bibliothèque `arrows`. Sans cette bibliothèque, il y a un choix suffisant pour les applications courantes où l'on n'utilise qu'un ou deux styles de flèches.

Voilà la syntaxe utilisée [15.3.4] [23] :

1) `>=(type de pointe)`

permet de définir la forme de la pointe de la flèche; les types les plus courants et définis dans l'extension de base sont `to` (forme élémentaire désuète), `latex` (forme des flèches des fontes `cm`), `stealth` (forme des flèches de `PSTricks`) et `|` (un simple petit trait perpendiculaire).

2) le choix du type de la flèche se fait avec les options :

`->` donne la flèche simple usuelle,

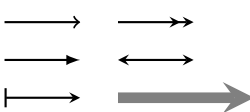
`<->` donne la flèche double usuelle,

`>->` donne la flèche avec les deux pointes dans le même sens,

`->>` donne une flèche avec double pointe,

`|<->|` donne la flèche pour coter en ajoutant la dimension avec la macro pose rencontrée à la section 7.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,->](0,10)--+(10,0);
\draw[thick,>=latex,->](0,5)--+(10,0);
\draw[thick,>=stealth,|>](0,0)--+(10,0);
\draw[thick,>=stealth,->>](15,10)--+(10,0);
\draw[thick,>=stealth,<->](20,5)--+(10,0);
\draw[line width=4pt,>=stealth,->,gray](20,0)--+(18,0);
\end{tikzpicture}
```

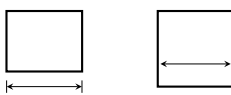


On dispose encore de la possibilité de placer l'origine et l'extrémité en retrait d'une distance donnée en utilisant les options :

`shorten<=(dimension)` et `shorten>=(dimension)`

où la dimension donnée est celle du retrait.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick](0,10)--++(10,0)--++(0,-8)--++(-10,0)--cycle;
\draw[very thin,>=stealth,|<->|]
(0,0)--++(10,0);
\draw[thick](20,10)--++(10,0)
--++(0,-10)--++(-10,0)--cycle;
\draw[very thin,shorten >=1pt,
shorten <=1pt,>=stealth,<->](20,3)--+(10,0);
\end{tikzpicture}
```

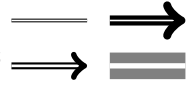


10.7. DOUBLE TRAIT

Pour avoir un double trait, on dispose de l'option [15.3.5] :
`double` ou `double=(couleur)` (défaut : `double=white`)
 Cette option trace deux traits d'épaisseur courante séparés par un trait
 d'épaisseur déterminée par l'option :
`double distance=(dimension)` (défaut : épaisseur de 0,6 pt).

La largeur du double trait est évidemment deux fois l'épaisseur de trait
 courante plus la distance courante de séparation.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[thick,double] (0,6)--+(10,0);
\draw[thick,double distance=1pt,->] (0,0)--+(10,0);
\draw[ultra thick,double,->] (13,6)--+(10,0);
\draw[line width=4pt,double distance=2pt,gray] (13,0)--+(10,0);
\end{tikzpicture}
```



10.8. COULEUR

Il est hors de question dans ce petit manuel de reprendre tout ce qui
 est disponible dans le domaine de la couleur [15.2]. On peut se reporter
 à la documentation de l'extension `xcolor` pour des listes de couleurs
 prédéfinies bien fournies.

On rappelle les couleurs de base prédéfinies dans cette extension :

Nom	Couleur	Non	Couleur	Non	Couleur
red		yellow		black	
green		orange		darkgray	
blue		violet		gray	
cyan		purple		lightgray	
magenta		brown		white	
pink		olive			

et la possibilité de varier l'intensité des couleurs par le remplacement :
 (couleur) par (couleur)!(nombre entre 100 et 0)

Nom	100	75	50	25
red				
blue				
green				

La syntaxe la plus générale pour l'option de couleur est [2.7] [15.2] :

```
color=(couleur)
```

et donc, on écrit :

```
\draw[color=(couleur),...] abrégé en \draw[(couleur),...]
```

Pour colorer la ligne fermée et son intérieur, on écrit :

```
\fill[(couleur),draw=(couleur),...]
```

ou la même chose en échangeant `\fill` et `draw=` en `\draw` et `fill=`.

L'utilisateur moyen peut difficilement utiliser les définitions du type :

```
\definecolor{(couleur){rgb}{a,b,c} où  $0 \leq a, b, c \leq 1$ 
```

Mais l'extension `xcolor` propose des tableaux de couleurs définies par des spécialistes ; on peut en diminuer l'intensité et les mélanger avec :

```
\colorlet{(nouvelle couleur)}{(couleur)!a}  $0 \leq a \leq 1$ 
```

```
\colorlet{(nouv.coul.)}{(coul.1)!a!(coul.2)!b}  $0 \leq a, b \leq 1, a + b = 1$ 
```

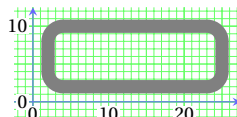
11. TRACÉS PRÉDÉFINIS : rectangle ET grid

La commande [2.6] :

```
\draw [(options)] (a,b) rectangle (c,d) ;
```

trace un rectangle dont les coordonnées des points bas-gauche et haut-droit sont respectivement (a,b) et (c,d) :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=5pt,gray,rounded
corners=2mm](0,0)rectangle(25,10);
\end{tikzpicture}
```



Il y a aussi un tracé prédéfini, constitué de droites parallèles aux axes et espacées de la distance `step` ; cette grille s'obtient avec [2.7] :

```
\draw[step=(dimension),(options)] (a,b) grid (c,d) ;
```

où (a,b) et (c,d) ont la même signification que dans le tracé du rectangle ci-dessus. En voici une application produisant la grille graduée utilisée dans les figures du présent document (à la grille, on a ajouté les axes et les graduations). Elle s'obtient par exemple avec :

```
\ppmm(-31.5mm,-21.5mm)(71.5mm,31.5mm)
```

dont le code est :

```
\tikzset{colorlines/.style={color=green!60}} % CHOIX
\tikzset{fondblanc/.style={rectangle,fill=white,inner sep=0.5pt}}
\def\ppmm(#1)(#2){%
\draw[step=1mm,line width=0.2pt,colorlines](#1)grid(#2);
%idem avec step=5mm,line width=0.3pt,step=10mm,line width=0.4pt
```

```

\def\xmin(##1,##2){##1}
%idem pour extraire \xmax, \ymin et \ymax
\draw[->,>=stealth,line width=0.4pt,colorlines]%
  (\xmin(##1),0mm)--(\xmax(##2),0mm);
%idem pour l'axe y
\foreach\i in{10,20,...,100}{\ifnum\i>\xmax(##2)\breakforeach
\else\node[fondblanc,anchor=north]at(\i mm,-0.5mm){$\sc{i}$};\fi}
%idem pour les graduations des demi-axes x<0, y>0 et y<0}

```

12. TRACÉS PARTICULIERS

On peut joindre deux points A et B par une ligne en équerre en remplaçant -- par -| ou |- selon que l'on veut le segment issu de A parallèle à l'axe des y et l'autre segment parallèle à l'axe des x ou inversement [14.2.2]. Avec la donnée d'un point supplémentaire, on peut faire des jonctions à trois ou quatre segments : c'est très pratique !

Une peut aussi placer à chaque sommet d'une ligne brisée un objet graphique constitué d'un chemin ; cela grâce à l'option [14] :

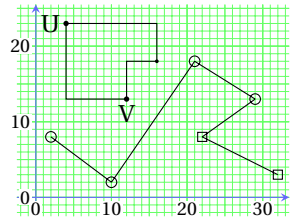
`insert path=(chemin)`

Cette possibilité peut devenir avantageuse lorsque l'on trace une ligne brisée ayant de nombreux de sommets.

```

\begin{tikzpicture}[x=1mm,y=1mm]
\tikzset{c/.style={insert path={circle(0.7)}}}%
\tikzset{r/.style={insert path={+(-.6,-.6)
  rectangle+ (.6,.6)+(0,0)}}}%
\ppmm(-2.5mm,-2.5mm)(33.5mm,25.5mm)
\coord(U)at(4,23);\coord(V)at(12,13);
\draw[thin](U)|-(V)(U)-|(16,18)-|(V);
\pose(U){180}{U}\pose(V){-90}{V}
\ptn(16,18)(0.7)\ptn(U)(1)\ptn(V)(1)
\draw(2,8)[c]--(10,2)[c]--(21,18)[c]--
(29,13)[c]--(22,8)[r]--(32,3)[r];
\end{tikzpicture}

```



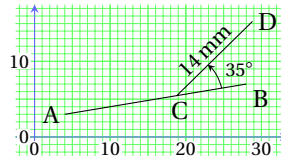
On a défini dans cette figure des styles avec la commande `tikzset` qui sera vue dans les sections suivantes.

On a encore la possibilité de réaliser la construction suivante : on a deux points A et B, on se donne un point C sur la ligne AB et on trace le point D tel que :

$|\overrightarrow{CD}| = (\text{longueur donnée})$ et $(\overrightarrow{CB}, \overrightarrow{CD}) = (\text{angle donné})$.

Pour cela on va utiliser la représentation spéciale des coordonnées des points vue à la section 7 ; il faut donc charger bibliothèque calc. Le code pour cette construction est :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\catcode'\!=12\catcode'\:=12
\coord(A)at(4,3);\coord(B)at(13:30);
\coord(C)at($(A)!15mm!(B)$);
\coord(D)at($(C)!14mm!35:(B)$);
\draw[thin](A)--(B)(C)--(D);
\pose(A){180}{A}\pose(B){0}{B}
\pose(C){-80}{C}\pose(D){0}{D}
\end{tikzpicture}
```



On remarque que cette construction peut être faite à partir de points A et B dont on ne connaîtrait pas explicitement les coordonnées (points résultant de transformations, points d'intersection de courbes, etc.).

On termine en rappelant que l'on peut tracer la perpendiculaire à un segment donné menée depuis un point quelconque non situé sur le segment [13.5.5] (cf. sect. 7).

CHAPITRE 3

La machinerie TikZ

Un titre inattendu pour un chapitre où on a rassemblé beaucoup de choses, qui n'ont pas toujours de relations directes entre elles, mais qui sont nécessaires pour pouvoir profiter rapidement des immenses possibilités graphiques de TikZ.

13. PRÉPARATIONS PRÉLIMINAIRES

13.1. INSTALLATION DES FICHIERS

On peut se procurer le fichier `pgf_2.10.tds.zip` sur le site :
sourceforge.net/projects/pgf/

Une fois décompacté, on a quatre sous-répertoires et un fichier. L'installation peut se faire automatiquement, mais l'auteur de ces quelques lignes a toujours une peur bleue de ces méthodes.

Voici donc comment installer TikZ à la main (distribution MiKTeX) :

- le sous-répertoire `doc\generic` contient un sous-répertoire `pgf` que l'on doit placer le sous-répertoire `MiKTeX\doc\generic`,
- le sous-répertoire `tex\generic` contient un sous-répertoire `pgf` qui est à placer dans le sous-répertoire `MiKTeX\tex\generic`,
- le sous-répertoire `tex\latex` contient un sous-répertoire `pgf` qu'il faut placer dans le sous-répertoire `MiKTeX\tex\latex`,
- on suit la même procédure avec les sous-répertoires `tex\plain` et `tex\context`,

Pour terminer, il ne faut surtout pas oublier de régénérer la base de données de T_EX pour qu'il puisse aller chercher les fichiers là où ils ont été placés!

13.2. AJOUTS AU PRÉAMBULE

Il faut appeler l'extension avec la commande [12.1] :

```
\usepackage{tikz}
```

L'extension `tikz` appelle, pour son « fonctionnement de base », les fichiers nécessaires, l'extension \LaTeX `xcolor` en particulier.

Pour des figures complexes, il faut utiliser les bibliothèques d'éléments disponibles que l'on charge avec la commande [12.1] :

```
\usetikzlibrary{(biblio)}
```

ou `(biblio)` est un nom ou une suite de noms de bibliothèques séparés par des virgules.

Il est évidemment astucieux de mettre les macros `TikZ` dans un fichier spécifique `mactikz.tex` que l'on appellera avec la commande `\input`.

14. UTILISATION DE `TikZ`

14.1. ENVIRONNEMENT DE BASE

Avec `TikZ`, on fait une figure en plaçant les commandes graphiques à l'intérieur de l'environnement graphique spécifique [12.2.1] :

```
\begin{tikzpicture}  
...commandes...  
\end{tikzpicture}
```

Toutes les commandes graphiques doivent être à l'intérieur de cet environnement, excepté la commande `\tikzset` permettant la définition de styles par l'utilisateur et abordée à la section suivante.

Heureusement, `TikZ` possède un moyen d'abrégier l'écriture pour une figure composée d'une seule commande graphique ou éventuellement d'un très petit nombre de commandes ; on écrit [12.2.2] :

```
\tikz [options] {(commandes)}
```

ou

```
\tikz [options] (commande) ;
```

C'est très commode : en voici un exemple de chaque syntaxe, la deuxième étant réservée au cas d'une seule commande.

```
\tikz[x=1mm,y=1mm]
```

```
{\draw(0,0)rectangle(15,2);
```

```
\draw(20,0)rectangle(35,2);}
```



```
\tikz[x=1mm,y=1mm]
```

```
\draw(0,0)rectangle(35,2);
```



14.2. INTÉGRATION DES FIGURES `TikZ`

Toute figure créée avec `TikZ` en respectant les directives précédentes se comporte comme une boîte \TeX ; en effet, une `boundingbox` a été créée

par l'environnement `tikzpicture`. Cette boîte se place dans la page \TeX comme un `\parbox` ou une `minipage` ; par défaut, le bas de la figure se place sur la ligne de base de la ligne courante [12.2.1].

On peut monter ou descendre la figure avec l'option de figure :
`baseline=(dimension)`

qui place l'origine des coordonnées de la figure à la distance (dimension) de la ligne de base de la ligne courante (par exemple `dimension=0` fait que l'origine des coordonnées est sur cette ligne de base). D'une manière générale, chacun peut utiliser son arsenal de méthodes de placement habituel et, en particulier, l'environnement `figure` de \LaTeX .

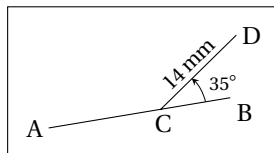
A propos du placement des figures, on dispose encore de l'option de figure [12.2.3] :

`show background rectangle`

qui permet « d'habiller » les figures avec un fond, un cadre, etc.

Voilà l'exemple du cadre (attention, il faut charger la bibliothèque `backgrounds` [25] qui contient beaucoup d'autres possibilités) :

```
\begin{tikzpicture}[x=1mm,y=1mm,
  show background rectangle]
% code dernière figure section 12
\end{tikzpicture}
```

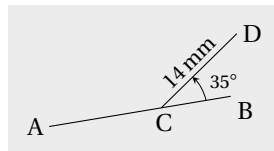


Et voilà l'exemple du fond légèrement grisé : il faut redéfinir le style `background rectangle` fourni par la bibliothèque avec la définition :

```
\tikzset{background rectangle/.style={fill=lightgray!30}}
```

ce qui donne le résultat :

```
\tikzset{background
  rectangle/.style={fill=lightgray!30}}
\begin{tikzpicture}[x=1mm,y=1mm,
  show background rectangle]
% code avant dernière figure section 12
\end{tikzpicture}
```



On peut en outre faire un contour en trait double, arrondir les angles, ombrer au lieu de remplir, etc. La distance entre la `boundingbox` initiale et le `background rectangle` peut être modifiée avec l'option :

`inner frame sep=(dimension)` (défaut 1 ex)

On peut prendre des valeurs différentes suivant l'axe des x et l'axe des y .

15. MANIPULATION DES OPTIONS

Jusqu'à maintenant, on a placé les options entre `[]` après les commandes `\path` ou `\draw` (excepté les options de choix d'unités et l'option de tracé d'un fond, placées comme option de l'environnement `tikzpicture`) : on va voir toutes les manières de placer les options ainsi que les propriétés qui résultent de ces placements. La bonne gestion des options facilite la saisie et est une aide précieuse pour une présentation homogène des documents.

15.1. PLACEMENT EN OPTION DE FIGURE

Les options qui seront presque toujours utilisées dans une figure vont être placées comme options de l'environnement `tikzpicture` (comme on l'a fait et fera dans les exemples pour le choix des unités). Si, par exemple, on veut avoir dans toute une figure des pointillés très fins, des flèches du type PostScript et une épaisseur de 1 pt pour presque tous les traits, on écrit [12.2.1] :

```
\begin{tikzpicture}[x=1mm,y=1mm,densely dotted,  
>=stealth,line width=1pt]
```

Toutes ces options deviennent des options par défaut pour cette figure et, bien entendu, prévalent sur les options par défaut de `TikZ`.

15.2. PLACEMENT EN OPTION D'UNE PARTIE DE FIGURE

Les options qui ne sont utilisées que pour une partie déterminée d'une figure (options différentes des options par défaut de `TikZ` et options différentes de celles installées par défaut comme options de figure) vont être placées comme options d'un environnement spécial prévu à cet effet. Par exemple, si on veut une partie d'une figure avec des traits gris et très fins, on écrit [12.3.1] :

```
\begin{scope}[draw=gray,very thin]  
... commandes de la partie de figure...  
\end{scope}
```

Ces deux options deviennent des options par défaut dans cette partie de la figure ; elles prévalent sur toutes les options par défaut courantes à l'entrée de l'environnement `scope`.

Cette méthode a une abréviation possible si l'on charge la bibliothèque `scopes` ; sa syntaxe (familière aux `TEX`nciens) est [12.3.2] :

```
{[draw=gray,very thin]...partie de figure...}
```


15.3. PLACEMENT EN OPTION DE COMMANDE

C'est ce qui avait été fait jusqu'à maintenant (sauf pour le choix des unités et le tracé d'un fond). Les options ainsi placées prévalent sur toutes les options par défaut courantes à la position de la commande qui les contient et sont utilisées jusqu'à la fin de cette commande.

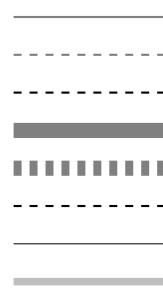
On trouvera ci-dessous des exemples des trois possibilités de placement précédemment détaillées. On écrit trois macros :

- `\traita` qui trace un trait sans options,
- `\traitb` qui trace un trait en traitillé épais (option de commande `dashed`),
- `\traitle` qui trace un trait en traitillé, épais et noir (options de commande `dashed`, `thick` et `black`).

On rappelle que par défaut le trait est continu et sa couleur et noire.

On voit que le `traitle` est invariant, il garde toujours ses trois options. Par contre le `traita` prend les options en vigueur à sa position. Le lecteur est invité, à titre d'exercice, à déterminer les options actives à chaque tracé en tenant compte des règles de prévalance.

```
\def\traita#1{\draw(0,#1)--+(20,0);}
\def\traitb#1{\draw[dashed](0,#1)--+(20,0);}
\def\traitle#1{\draw[dashed,thick,black]
(0,#1)--+(20,0);}
\hfill\begin{tikzpicture}[x=1mm,y=1mm,thick,gray]
\traita{35}\traitb{30}\traitle{25}
\begin{scope}[line width=2mm]
\traita{20}\traitb{15}\traitle{10}
\draw[ultra thin,black](0,5)--+(20,0);
\end{scope}
\draw[lightgray,line width=1mm](0,0)--+(20,0);
\end{tikzpicture}
```



Remarque : Cette notion de prévalance de certaines options sur d'autres (en fonction de leur positionnement) appelle une directive très importante pour l'écriture des macros. Si l'on écrit une macro sans aucune option, lors de son exécution, elle va utiliser les options par défaut courantes à la position de l'exécution et l'on risque d'avoir des résultats non souhaités ! Le seul moyen pour que la macro se comporte dans tous les cas comme on le veut est d'y introduire toutes les options souhaitées. Par exemple, la macro `\ppmm` qui trace la grille graduée, utilisée dans une

bonne partie des figures est définie avec le mm pour unité de longueur : aussi, sur la figure de la section 3 (qui garde le cm pour unité de longueur) la grille en question s’affiche normalement.

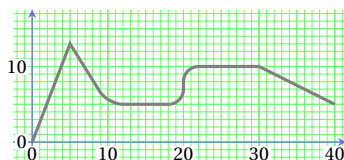
15.4. OPTIONS DANS LES SPÉCIFICATIONS D’UNE COMMANDE

Enfin, il y a des options qui peuvent être données dans les spécifications mêmes d’une commande [12.3.3].

Un cas typique d’option pour laquelle ce n’est pas possible est l’option de couleur : cette option est valable pour toute la commande ; si plusieurs couleurs sont données dans la spécification d’une commande, c’est la dernière couleur qui prévaut (l’explication de cette restriction exigerait d’entrer dans les entrailles de TikZ, ce que l’on ne va pas faire).

Par contre, cette possibilité est présente pour des options telles que celle commandant l’arrondissement des angles d’une ligne brisée, le dédoublement de la ligne, etc. Il suffit alors de préciser le domaine auquel on veut affecter l’option par des {}, comme on a l’habitude de faire avec T_EX. Voici un exemple de ligne brisée dont seulement trois sommets sont arrondis.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[very thick,gray](0,0)--(5,13)
  {[rounded corners=2mm]--(10,5)--
  (20,5)--(20,10)}--(30,10)--(40,5);
\end{tikzpicture}
```



Il faut aussi signaler que l’on a utilisé cette possibilité dans la première figure de la section 11 où l’on a placé, après chaque sommet et entre [], une « option généralisée » qui place un petit objet graphique au sommet correspondant. On vient de dire « option généralisée » car c’est plus qu’une option choisissant par exemple les caractéristiques d’un tracé (couleur, dimension, etc.), c’est, dans le cas rappelé ci-dessus, une opération traçant un élément de la figure. Cette remarque ouvre la voie à l’études des styles que l’on va aborder.

16. DÉFINITION DE STYLES PAR L’UTILISATEUR

Le contenu de cette section devient particulièrement utile lorsque, ayant acquis une certaine aisance, on se lance dans la production de nombreuses figures : la définition et l’utilisation de styles spécifiques

regroupant un ensemble d'options permet d'assurer une forte homogénéité de l'ensemble des figures d'un même document tout en respectant les directives de composition d'ordre pédagogique ou esthétique. On y gagne aussi un allègement de la saisie.

16.1. COMMANDE DE DÉFINITION D'UN STYLE

On commence par la commande de définition de style, déjà utilisée dans quelques figures, (sect. 11 et 12 notamment) [2.8] :

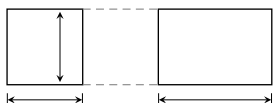
`\tikzset{(nom du style)/.style={options séparées par des virgules}}` qui remplace la commande `\tikzstyle` des premières versions de TikZ. A la section suivante, on rentrera plus profondément dans le puissant système de clés-valeurs sous-jacent. Ces styles sont ensuite appelées comme les options (ce ne sont en fait que des groupes d'options), là où ils sont nécessaires et avec la syntaxe :

`style=(nom du style)`
généralement abrégée en (non du style).

Si, par exemple, on a une série de figures, où quatre types de traits apparaissent systématiquement : traits normaux, de rappel, de cotation intérieure et de cotation extérieure, on va écrire quatre définitions correspondantes :

```
\tikzset{tn/.style={line width=0.5pt}}%
\tikzset{tr/.style={line width=0.4pt,densely dashed,gray}}%
\tikzset{cote/.style={line width=0.2pt,>=stealth,|<->|}}%
\tikzset{coti/.style={line width=0.2pt,>=stealth,
shorten <=1pt,shorten >=1pt,<->}}%
```

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[style={tn}] (0,0)rectangle(10,10);
\draw[tn] (20,0)rectangle(35,10);
\draw[tr] (10,10)--(20,10)(10,0)--(20,0);
\draw[coti] (7,0)--(7,10);
\draw[cote] (0,-2)--(10,-2); \draw[cote] (20,-2)--(35,-2);
\end{tikzpicture}
```



Cet exemple est suffisant pour donner une idée de l'allègement de saisie au bout de 50 ou 100 figures tout en conservant strictement l'homogénéité : il suffit de bien choisir définitions.

Cette commande de définition de style peut être placée à l'extérieur de l'environnement `tikzpicture`, le style est alors utilisable pour toutes les figures qui suivent (cf. la figure de la section 2.1) ; si elle est placée à

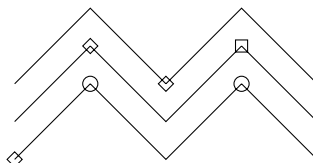
l'intérieur, le style n'est utilisable que jusqu'à la fin de l'environnement ; si elle est placée dans un environnement `scope`, le style ne pourra être utilisée que jusqu'à la fin de cet environnement.

Les définitions de styles peuvent également être placées comme options de figure, de partie de figure et de commande (sans utiliser la commande `\tikzset`) ; dans ce cas on écrit seulement :

`[(nom du style)/.style={options séparées par des virgules, . . .}]`

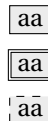
Voilà des exemples où l'on utilise l'option de tracé `insert path` qui modifie le tracé à chaque sommet d'une ligne brisée (cf. sect. 12) :

```
\begin{tikzpicture}[x=1mm,y=1mm,styl/.style={insert
  path={+(0,-1)--+(1,0)--+(0,1)--+(-1,0)--cycle+(0,0)}}]
\draw(0,10)--(10,20)--(20,10)[styl]--(30,20)--(40,10);
\begin{scope}[styr/.style={insert
  path={+(-.8,-.8)
  rectangle+.8,.8)+(0,0)}}]
\draw(0,5)--(10,15)[styl]--(20,5)
  --(30,15)[styr]--(40,5);
\draw[styc/.style={insert
  path={circle(1)}}](0,0)[styl]--
  (10,10)[styc]--(20,0)--(30,10)[styc]--(40,0);
\end{scope}
\end{tikzpicture}
```



Pour terminer la sous-section, on montre comment on peut modifier provisoirement un style déjà défini (style nommé `rect` dans l'exemple) : modification en option de figure, en option de partie de figure et en option de commande. On utilise la commande `node` et la forme `rectangle` servant à construire les nœuds contenant du texte (ce qui fait l'objet chapitre 5) ; cela nécessite de charger la bibliothèque `shapes`.

```
\tikzset{rect/.style={rectangle,draw}}%
\begin{tikzpicture}[x=1mm,y=1mm,rect/.append style={fill=gray!15}]
\node[rect](a)at(0,16){aa};
\begin{scope}[rect/.append style={double}]
\node[rect](a)at(0,8){aa};\end{scope}
\node[rect/.append style={dashed},rect](a)at(0,0){aa};
\end{tikzpicture}
```



16.2. STYLES ATTACHÉS À DES ÉLÉMENTS GRAPHIQUES

Les styles de ce type ne se différencient de ceux précédemment étudiés que par le fait que chacun d'entre eux ne joue un rôle que par rapport

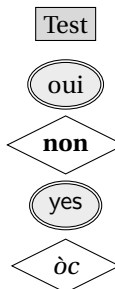
à un élément graphique bien déterminé : le nom de ces styles est de la forme `every` (dacos) où dacos est « l'élément graphique » qui sera affecté par ce nouveau style ; ce peut être un nœud, `node`, une forme, `circle`, un `label`, `label`, un type de construction de courbe, `to`, un environnement `scope`, etc. ; il suffit de se reporter à l'index de la documentation, mot *every*, pour voir le domaine d'application de ce type de style.

Dans l'exemple qui suit, on montre des définitions de ces styles et leurs modifications suivant la syntaxe exposée précédemment. On a choisi des styles associés à différentes formes de nœuds : les définitions et leurs modifications sont placées de diverses façons possibles : avant l'environnement `tikzpicture` et dans son intérieur (avec la commande `\tikzset{}`), en option de figure de cet environnement, en option de partie de figure et en option de commande.

```

\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={shape aspect=2,draw,font=\bfseries}}
\node[rectangle] (a) at (0,32){Test} ;
\node[ellipse] (b) at (0,24) [ellipse]{oui} ;
\node[diamond] (c) at (0,16){non} ;
\begin{scope}[every ellipse node
/.append style={font=\sffamily}]
\node[ellipse] (e) at (0,8){yes};\end{scope}
\node[every diamond node
/.append style={font=\slshape},diamond] (g) at (0,0){\`oc};
\end{tikzpicture}

```



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, on définit des styles correspondant à des classes d'éléments bien déterminés et on les utilise sans les modifier en cours de route !

17. GÉNÉRALISATION DU SYSTÈME DE CLÉS-VALEURS : `pgfkeys`

Les nouvelles versions de TikZ utilisent un système de clés-valeurs, implémenté par le package `pgfkeys`, plus général que celui géré par `keyval` ou `xkeyval`. Il peut être utilisé indépendamment de TikZ.

Les utilisateurs de `\includegraphics` connaissent les clés :

`draf=true` ou `false`

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

Dans le chapitre 2, on a utilisé la clé :

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

Dans le chapitre 5, on verra la clé :

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

Maintenant, TikZ utilise un système de clés-valeurs plus général comme on va le voir dans les lignes suivantes. Cette généralisation conduit à de nombreuses nouvelles possibilités [55] ; ses caractéristiques et ses avantages sont décrits [55.1.1] en des termes difficiles pour les non informaticiens. Peu importe puisque l'on va se cantonner aux quelques propriétés indispensables pour permettre d'utiliser TikZ avec profit. Ces clés-valeurs se placent dans une arborescence dont la syntaxe est identique à celle de l'arborescence d'un ensemble de fichiers (séparation des niveaux avec / comme sous Unix, même si on est sous Windows).

On commence par faire la liaison avec la commande `\tikzset` déjà largement utilisée. La création d'une clé définissant un style se fait avec la commande :

```
\pgfkeys{(chemin et nom de clé)/.style={liste des options...}}
```

qui place la clé dans l'arborescence des clés [55.2]. La définition de la commande `\tikzset` peut être simplifiée dans un but pédagogique à :

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

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

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

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

Il faut noter que l'utilisateur peut placer chaque clé qu'il définit dans dans le répertoire de l'arborescence des clés qu'il souhaite, répertoire qui est créé (si nécessaire) par la définition elle-même. Pour utiliser les clés ainsi définies, il n'est pas nécessaire d'indiquer le chemin d'accès (cf. l'utilisation des styles `circle`, `every rectangle node`, etc. des figures précédentes). Les mots réservés préfixés par un point sont :

— `.style` pour définir un style,

— `.append style` pour compléter un style (cf. encore les deux dernières figures pour la syntaxe).

Mais il y en d'autres, en particulier :

- `.default` pour donner une valeur par défaut aux clés,
- `.value forbidden` pour avertir que la clé ne prend pas de valeur,
- `.value required` pour avertir qu'il faut affecter une valeur à la clé,
- `.is choice` pour que la valeur qui suit devienne une des valeurs que peut prendre la clé,
- etc.
- `.code` qui sera abordé plus loin, en fin de sous-section.

Pour première application, on choisit pour tout le document l'option `stealth` pour les têtes de flèche en écrivant [55.4.2] :

```
\pgfkeys{/>/.default=stealth}
```

car la clé `>` est dans la racine de l'arborescence des clés.

La première généralisation importante concerne la possibilité de définition des styles avec un paramètre [55.4.4] avec la syntaxe suivante :

```
\pgfkeys{(chemin-clé)/.style={options avec 0 ou 1 argument}}
```

Pour application, on propose la définition d'un style constituant une simple abréviation particulièrement commode que l'on peut utiliser dans tout le document :

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

qui sera, par exemple, utilisé de la manière suivante :

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

Cette possibilité est généralisée jusqu'à 9 arguments par les commandes de définition suivantes ($n \leq 9$) :

```
\pgfkeys{(chemin-clé)/.style 2 args={options avec 2 arguments}}
```

```
\pgfkeys{(chemin-clé)/.style n args={n}{options avec n arguments}}
```

Les clés correspondantes s'utilisent avec la syntaxe :

```
(nom de clé)=(valeur) ou (nom de clé)={valeur 1}{valeur 2}...
```

et se placent comme les styles sans arguments, c'est-à-dire comme les options (cf. sect. 15).

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. Les commandes correspondantes utilisent le dernier mot réservé cité : `code` [55.4.3].

Voici la syntaxe de définition de ces clés :

```
\pgfkeys{(chemin-clé)/.code{code avec 0 ou 1 paramètre}}
```

```
\pgfkeys{(chemin-clé)/.code 2 args={code à 2 paramètres}}
```

```
\pgfkeys{(chemin-clé)/.code n args={n}{code à n paramètres}}
```

Et voilà les commandes d'utilisation correspondantes dans les cas de un paramètre, deux paramètres et plus de deux paramètres :

```
\pgfkeys{(nom de clé)}
\pgfkeys{(nom de clé)=(valeur du paramètre)}
\pgfkeys{(nom de clé)={val. param. 1}{val. param. 2} ... }
```

On les utilise lorsque l'on a besoin d'un résultat produit par l'exécution du code de la clé.

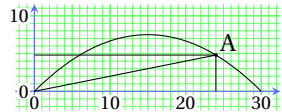
On signale le cas particulier de définitions de clés exécutant une commande déjà définie (avec paramètres éventuellement) [55.3.3] :

```
\pgfkeysdef{(chemin-clé)}{(commande à 0 ou 1 parm.)}
\pgfkeysdefnargs{(chemin-clé)}{(commande à 9 param. au plus)}
```

Toutes ces commandes généralisées ont une version expansée (pour les deux dernières, `def` y est changé en `edef`).

On en donne un exemple avec un paramètre : on a besoin de connaître l'abscisse et/ou l'ordonnée du point correspondant au temps 0.8 d'une courbe de Bézier donnée :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\newlength{\xxx}\newlength{\yyy}
\draw(0,0)..controls(10,10)and(20,10)..(30,0);
\pgfkeysdef{/pouint}{\pgfpointcurveattime{#1}{\pgfpoint{0}{0}}
{\pgfpoint{1cm}{1cm}}{\pgfpoint{2cm}{1cm}}{\pgfpoint{3cm}{0}}}
\pgfextractx{\xxx}{\pgfkeys{pouint=0.8}}
\pgfextracty{\yyy}{\pgfkeys{pouint=0.8}}
\coord(A)at(\xxx,\yyy);
\draw(A)--(0,0);\ptn(A)(0.7)
\draw[very thin](\xxx,\yyy)--(\xxx,0);
\draw[very thin](\xxx,\yyy)--(0,\yyy);\pose(A)[-1]{45}{A}
\end{tikzpicture}
```



Dans cet exemple, la macro `\pgfpointcurveattime{#1}{-}{-}{-}` [70.5.2] donne le point de la courbe de Bézier correspondant au temps `#1` ; les 4 autres paramètres sont les 4 points définissant la courbe en syntaxe PGF. Le point résultant est aussi donné en syntaxe PGF, ce qui exige l'utilisation des macros `\pgfextractx{-}{-}` et `\pgfextracty{-}{-}` [70.6] qui, à partir de la clé pour la valeur 0.8 (`pouint=0.8`), donne l'abscisse `\xxx` et l'ordonnée `\yyy` du point souhaité, sous la forme d'une distance définie en \TeX et utilisable en *TikZ* comme vu en section 5 ; ces deux macros PGF assurent le transfert de PGF à \TeX comme la macro `\pgfextra` assure le transfert de *TikZ* à \TeX (cf. sect. 6).

CHAPITRE 4

Lignes courbes

Ce chapitre expose les différentes manières pratiques de tracer des courbes. Contrairement à ce qu'on avait fait pour le chapitre 2 (lignes brisées), on commence par donner les constructions prédéfinies.

18. CONSTRUCTIONS PRÉDÉFINIES : cercle, ellipse ET arc

18.1. CERCLE ET ELLIPSE

Les commandes [2.5] :

```
\draw (a,b) circle (r) ;
```

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

construisent respectivement un cercle centré en $x = a$, $y = b$ et de rayon r et une ellipse centrée en $x = a$, $y = b$ et de rayons respectifs r suivant les x et s suivant les y . Si l'on ne donne pas d'unités, ce sont les unités par défaut qui sont utilisées; le cercle n'est un cercle que si les unités suivant x et suivant y sont identiques. On ne peut utiliser que les options de tracé vues pour les lignes brisées qui gardent un sens : épaisseur du trait, pointillés et traitillés, double trait et couleur.

Ces anciennes commandes sont illogiques parce que les parenthèses sont utilisées pour autre chose que pour des coordonnées, (r) et (r and s) ; mais elles sont plus pratiques que les nouvelles [14.7] :

```
\draw(a,b) circle [radius=r] ;
```

```
\draw(a,b) circle [x radius=r,y radius=s] ;
```

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

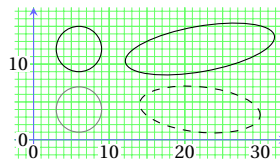
```
\draw(6,12)circle (3);
```

```
\draw[rotate around={10:(22,12)}]%  
      (22,12)ellipse(10and3);
```

```
\draw[radius=3,gray](6,4)circle;
```

```
\draw(22,4)[rotate around={-6:(22,4)},  
           x radius=8,y radius=3,dashed]circle;
```

```
\end{tikzpicture}
```



18.2. ARC

La commande [2.10] :

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

trace un arc d'origine en $x = a$, $y = b$, d'angle à l'origine u° , d'angle à l'extrémité v° et de rayon r . Si l'on ne donne pas d'unité, les unités par défaut sont utilisées. Aux options de tracé citées plus haut il faut bien évidemment ajouter les options terminaison et flèche; cette remarque vaut pour tous les tracés de courbes fermées.

Si la paire (a, b) désigne le centre de l'arc, il suffit d'écrire :

```
\draw (a,b)+(u:r) arc (u:v:r)
```

pour tracer l'arc centré en (a, b), d'angle d'origine u° , d'angle d'extrémité v° et de rayon r .

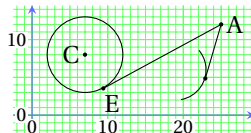
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(15,0)arc(5:105:10);
\draw[thick](17,0)+(10:10)arc(10:105:10);
\ptn(17,0)(0.8)
\draw[dashed](17,0)--+(10:10);
\end{tikzpicture}
```



18.3. TANGENTES AUX CERCLES ET AUX ARCS

Soit à tracer la tangente à un cercle, de centre C et de rayon r , issue d'un point A extérieur au cercle. La méthode consiste d'abord à construire le cercle en tant que nœud vide (commande `\node` terminée par la paire `{}`), d'où la nécessité d'utiliser l'option `minimum size=2r`. Ensuite, les coordonnées du point de tangence sont calculées grâce à l'utilisation des coordonnées tangentes (bibliothèque `calc` nécessaire) [13.2.4]. La méthode s'étend aux arcs de cercle : on considère (sans le tracer) le cercle ayant le même rayon que l'arc pour obtenir le point de tangence, puis on termine en superposant l'arc au cercle. Voici l'exemple :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\coord(C)at(7,8);\coord(A)at(25,12);
\ptn(C)(0.8)\ptn(A)(0.8)
\pose(C){180}\pose(A){0}\A}
\node[circle,draw](X)at(C)
[minimum size=10mm]{};
\coordinate(E)at(tangent cs:node=X,
point={(A)},solution=2); % solution=1 donne l'autre tangente
\draw(E)--(A);\ptn(E)(0.8);\pose(E){-60}\E}
\end{tikzpicture}
```



On dispose aussi de la possibilité de tracer un cercle centré en un point C, passant par un point B et de rayon CB. Ce tracé pourrait être fait après avoir calculé la distance CB (macro `vecLen`, sect. 6). Mais on dispose d'une méthode directe utilisant une option spécifique à la construction des nœuds : `circle through` (dit plus simplement, on construit un nœud circulaire de centre donné dont la circonférence « traverse » un point donné). Cette méthode exige le chargement de la bibliothèque `through` et la syntaxe correspondante est [4.1.3] :

```
\node[draw,circle through=(B)] at (A) {};
```

où l'argument contenu du nœud est encore laissé vide.

18.4. SINUS ET COSINUS

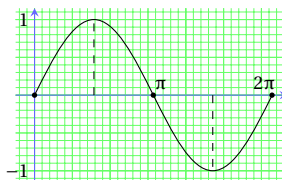
Les commandes [14.11] :

```
\draw (a,b) sin (c,d) ;
```

```
\draw (a,b) cos (c,d) ;
```

tracent respectivement le sinus et le cosinus sur l'intervalle $[0, \pi/2]$ ($[0, 0.7854]$) transformés linéairement de telle manière que le point correspondant à l'angle 0 devient le point $x = a, y = b$ et le point correspondant à l'angle $\pi/2$ devient le point $x = c, y = d$. On donne le tracé d'une sinusoïde sur l'intervalle $[0, 2\pi]$:

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)sin(7.854,10)cos(15.708,0)
      sin(23.562,-10)cos(31.416,0);
\draw[dashed,thin](7.854,0)--+(0,10);
\draw[dashed,thin](23.562,0)--+(0,-10);
\ptn(15.7,0)(1)\pose(16.7,0){90}{\sc\pi$}
\ptn(31.4,0)(1)
\pose(30.4,0){90}{\sc2\pi$}\ptn(0,0)(1)
\end{tikzpicture}
```



19. COURBES DE BÉZIER

19.1. SYNTAXE GÉNÉRALE

Ces courbes ont été initialement utilisées par Bézier pour dessiner des formes de carrosseries de voiture chez Renault. Depuis que l'informatique s'est introduite dans toutes les disciplines, les courbes de Bézier sont très utilisées dans les disciplines graphiques, la création de fontes notamment. On abordera certains aspects mathématiques élémentaires

par la suite ; on commence en donnant la méthode pour les construire et en remarquant quelques propriétés.

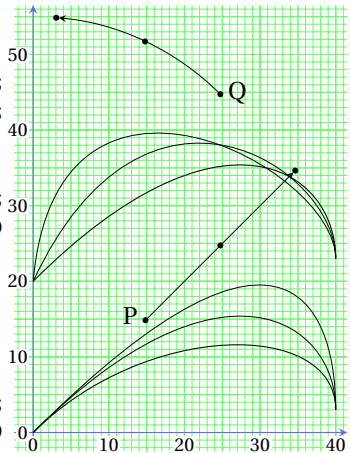
Une courbe de Bézier est obtenue par la commande [2.4] [14.3] :

```
\draw (a,b) .. controls (u,v) and (w,t) .. (c,d) ;
```

qui trace une courbe allant du point (a,b) au point (c,d) et dont la forme est déterminée par les points de contrôle (u,v) et (w,t) : la courbe part du point (a,b) dans la direction du point (u,v) et arrive au point (c,d) depuis la direction du point (w,t).

Pour comprendre le rôle des points de contrôle, on construit une courbe de Bézier dont on va déplacer le premier point de contrôle (noté Q). Le déplacement de Q suivant la flèche arcquée produit deux autres courbes qui montrent bien que Q commande la direction de la courbe au point de départ, coordonnées (0,20). La courbe initiale est translatée vers le bas pour des raisons de clarté de la figure. Le déplacement du premier point de contrôle (noté maintenant P) suivant la flèche produit deux autres courbes et l'on constate qu'en s'éloignant du point de départ, coordonnées (0,0), il semble « tirer » la courbe dans son déplacement.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\def\point(#1){\fill(#1)circle(1pt);}
% dr=draw,T=thin,c=controls,a=and
\dr(0,20) .. c+(45:35)a(40,35) .. (40,20);
\drT(0,20) .. c+(65:35)a(40,35) .. (40,20);
\drT(0,20) .. c+(85:35)a(40,35) .. (40,20);
\begin{scope}[yshift=20mm]
\draw[>=stealth,->,very thin,
shorten >=1pt](45:35)arc(45:85:35);
\point(45:35)\point(65:35)\point(85:35)
\end{scope}
\dr(0,0) .. c(45:21)a(40,15) .. (40,0);
\drT(0,0) .. c(45:35)a(40,15) .. (40,0);
\drT(0,0) .. c(45:49)a(40,15) .. (40,0);
\draw[>=stealth,->,very thin,
shorten >=1pt](45:21)--(45:49);
\point(45:21)\point(45:35)\point(45:49)
\pose(14,15){180}{P}\pose(25,45){0}{Q}
\end{tikzpicture}
```



On va aborder deux propriétés mathématiques simples. La première est que les courbes de Bézier sont la représentation graphique des po-

lynômes de Bernstein (utilisés par cet auteur, en 1912, lors des premiers pas des méthodes d'approximation) :

$$z(t) = (1-t)^3 z_1 + 3(1-t)^2 t z_2 + 3(1-t) t^2 z_3 + t^3 z_4 \quad \text{pour } 0 \leq t \leq 1,$$

où les $z_1, z_2, z_3,$ et z_4 représentent les paires de coordonnées de quatre points (un point de départ, deux points de contrôle et un point d'arrivée).

La deuxième propriété intéressante à connaître est que ces courbes de Bézier sont la limite d'un ensemble dénombrable de points construits comme suit :

On prend 4 points A, B, C et D ;

On construit les 3 milieux I, J, et K des segments AB, BC et CD ;

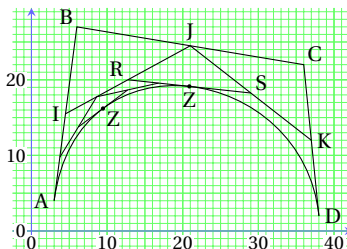
On construit les 2 milieux R et S des segments IJ et JK ;

On considère le milieu Z du segment RS.

On constate alors que l'on a reconstruit en double la situation initiale avec les groupes de points (A, I, R et Z) et (Z, S, K et D).

On recommence et on obtient 2 points du type Z, puis 4 points, puis 8, etc. La limite des points du type Z est une courbe de Bézier (A point de départ, B et C points de contrôle et D point d'arrivée) ; ceci est très ancien et date du temps de gloire de la géométrie. Voici le début du code :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\catcode'\!=12
\coord(A)at(3,4);\coord(B)at(6,27);
\coord(C)at(36,22);\coord(D)at(38,2);
\coord(I)at($(A)!0.5!(B)$);
\coord(J)at($(B)!0.5!(C)$);
\coord(K)at($(C)!0.5!(D)$);
\coord(R)at($(I)!0.5!(J)$);
\coord(S)at($(J)!0.5!(K)$);
\coord(Z)at($(R)!0.5!(S)$);
\draw[thin](A)..controls(B)and(C)..(D);
% plus les tracés des segments et les lettrages
\end{tikzpicture}
```



19.2. SYNTAXE SPÉCIALE

Dans la commande de tracé d'une courbe de Bézier, les déplacements relatifs prennent un sens différent et très commode pour les applications. Si on écrit [13.4.1] :

```
\draw (a,b) .. controls +(u,v) and +(w,t) .. +(c,d);
```

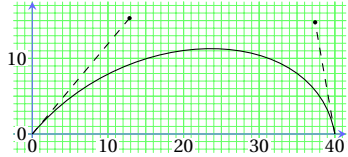
alors les notations $+(. , .)$ ont les significations suivantes :

$+(u, v)$ désigne alors un point déplacé par rapport au point $\{a, b\}$;

$+(w, t)$ désigne un point déplacé par rapport au point d'arrivée de la courbe ;

$+(c, d)$ désigne un point déplacé par rapport au point (a, b) ; le point d'arrivée de la courbe n'est pas alors (c, d) mais $(a, b)+(c, d)$.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(0,0)..controls+(50:20)
and +(100:15)..(40,0);
\draw[dashed,thin](0,0)--+(50:20);
\draw[dashed,thin](40,0)--+(100:15);
\ptn(50:20)(0.8)
\ptn({40,0}+(100:15})(0.8)
\end{tikzpicture}
```



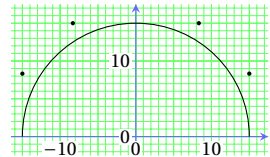
Cette syntaxe est très commode ; elle permet par exemple d'imposer facilement la direction des tangentes aux extrémités de la courbe en utilisant les coordonnées polaires, comme fait dans l'exemple.

19.3. PREMIÈRES APPLICATIONS

D'une façon générale, on appelle courbe de Bézier une courbe constituée de plusieurs courbes de Bézier élémentaires (qui viennent d'être définies) mises bout à bout.

Le premier exemple (très important) est le cercle qui se construit avec deux exemplaires de la courbe de Bézier élémentaire suivante [2.4] :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(-15,0)..controls(-15,8.325)
and(-8.325,15)..(0,15)..controls
(8.325,15)and(15,8.325)..(15,0);
\point(-15,8.325)\point(-8.325,15)
\point(8.325,15)\point(15,8.325)
%\draw[red](15,0)arc(0:180:15);
\end{tikzpicture}
```

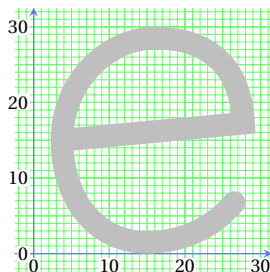


En enlevant le % (avant dernière ligne) on pourra constater que le demi cercle recouvre la courbe de Bézier.

L'exemple suivant montre le dessin de la lettre minuscule e constituée d'une courbe de Bézier et d'un segment, soit : 8 points de contrôle et 5 points guides (ou points de passage, c'est-à-dire extrémités des courbes de Bézier élémentaires et des segments). Le lecteur est invité à mettre à

son goût ce caractère en modifiant les points de contrôle mais aussi et les points guides.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=3mm,lightgray,cap=round]
(26.5,6.75)..controls+(-130:4.5)
and+(0:4.5)..(15,1.5)..controls+(180:7)
and+(-90:7.5)..(3.75,15)..controls+(90:8)
and+(180:6)..(16,28.5)..controls+(0:9)
and+(93:5.5)..(27.75,17.25)--(3.75,15);
\end{tikzpicture}
```



Ces courbes de Bézier sont à la base de tous les dessins de caractères des fontes logicielles disponibles ; un exemple est donné à la fin de ce chapitre : on sera frappé par le peu de nombres nécessaires pour obtenir un très beau caractère.

20. TRACÉ PRATIQUE DE COURBES

C'est sûr que les courbes de Bézier sont bien belles quand on choisit bien les points de contrôle. Mais ces points de contrôle sont difficiles à trouver et généralement on ne dispose que des points guides et éventuellement de quelques autres données. Comme expliqué dans le manuel de MetaFont, l'idée de base est que le programme puisse, avec les coordonnées des points guides et un peu d'information supplémentaire disponible, déterminer des points de contrôle pour tracer (sans qu'on le sache) des courbes de Bézier.

20.1. TRACER UNE COURBE AVEC PLOT

Les commandes [19.2] [19.8] :

```
\draw plot [smooth, (options)] coordinates{(a,b) (c,d) ... (y,z)};
\draw plot [smooth cycle, (options)]
coordinates{(a,b) (c,d) ... (y,z)};
```

tracent respectivement les courbes ouverte et fermée passant par les points de coordonnées (a,b), (c,d), ..., (y,z).

Si les coordonnées des points guides sont donnés dans un fichier, les commandes deviennent [19.4] :

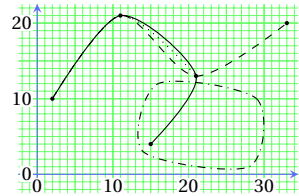
```
\draw plot [smooth, (options)] file{(chemin/fichier)};
\draw plot [smooth cycle, (options)] file{(chemin/fichier)};
```

Sans les options `smooth` ou `smooth cycle` [19.8], on aurait une ligne brisée, ce qui s'obtient plus simplement comme on a vu à la section 4.

Une autre possibilité consiste à donner la fonction à représenter à l'aide d'une syntaxe qui demande à \TeX de faire une pause pour faire produire à `GNUPLOT` un tableau de coordonnées de points guides qui seront utilisées de la même manière que les coordonnées données directement ou lues dans un fichier. Une dernière possibilité consiste à donner l'expression de la fonction à représenter suivant une syntaxe appropriée. On verra ces possibilités plus loin.

Dans l'exemple suivant, on commence par illustrer comment `TikZ` sait trouver les points de contrôle nécessaires pour tracer une courbe continue à tangente continue passant par des points guides.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw plot[smooth]coordinates
  {(2,10)(11,21)(21,13)(15,4)};
\draw[dashed]plot[smooth]coordinates
  {(2,10)(11,21)(21,13)(33,20)};
\draw[dotted]plot[smooth]coordinates
  {(2,10)(11,21)(21,13)};
\ptn(2,10)(0.8)\ptn(11,21)(0.8)
\ptn(21,13)(0.8)\ptn(15,4)(0.8)\ptn(33,20)(0.8)
\draw[dashdotted]plot[smooth cycle]file{f0.dat};
\end{tikzpicture}
```



On voit bien sur la figure que le deuxième tronçon de la courbe est dépendant de la position du quatrième point (courbes en trait plein et en traitillé) ainsi que de son éventuelle absence (courbe en pointillé) : les points de contrôle d'un tronçon entre les points guides n et $n + 1$ sont déterminés en fonction des points guides $n - 1$ et $n + 2$.

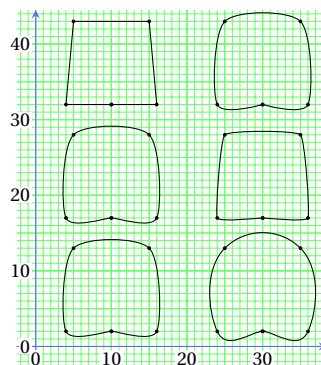
Sur cet exemple, on a testé la première syntaxe de tracé qui prend directement les coordonnées des points guides. Mais on a aussi testé la deuxième syntaxe de tracé (courbe en traitillé-pointillé) qui lit les coordonnées dans un fichier. Le fichier utilisé, nommé `f0.dat` pour cet exemple, a le format exigé suivant :

```
# ligne de commentaire
2 16 i (soit abscisse espace ordonnée espace lettre i pour chaque ligne)
10 7 i
20 2.75 i
30 1 i
```


On verra sur l'exemple suivant comment on peut marquer automatiquement les points guides avec une option appropriée. Cet exemple est constitué de tracés d'une courbe fermée sur lesquels on teste l'option `tension` et l'option `smooth cycle`. On a dans l'ordre :

- pas d'option `smooth` (ligne brisée),
- option `smooth` (le point (10, 2) est anguleux),
- option `smooth cycle` (le point anguleux a disparu),
- trois valeurs de l'option `tension` (0.2, 0.5 et 0.9) qui illustrent bien son rôle et son intérêt.

```
\begin{tikzpicture}[x=1mm,y=1mm,mark=*,mark size=0.5pt]
\draw plot[yshift=30mm]coordinates{(10,2)(4,2)
(5,13)(15,13)(16,2)(10,2)};
\draw plot[smooth,xshift=20mm,
yshift=30mm]coordinates{(10,2)
(4,2)(5,13)(15,13)(16,2)(10,2)};
\draw plot[smooth cycle,
yshift=15mm]coordinates{(10,2)
(4,2)(5,13)(15,13)(16,2)(10,2)};
\draw plot[smooth cycle,tension=0.2,
xshift=20mm,yshift=15mm]coordinates
{(10,2)(4,2)(5,13)(15,13)(16,2)(10,2)};
\draw plot[smooth cycle,tension=0.5]
coordinates{(10,2)(4,2)(5,13)(15,13)
(16,2)(10,2)};
\draw plot[smooth cycle,tension=0.9,xshift=20mm]
coordinates{(10,2)(4,2)(5,13)(15,13)(16,2)(10,2)};
\end{tikzpicture}
```



L'option des trois dernières courbes [19.8] :

`tension=(nombre entre 0 et 1)`

commande le « gonflement » de la courbe (qu'elle soit ouverte ou fermée). La courbe passant par les sommets d'un carré est pratiquement le cercle circonscrit du carré si on prend l'option `tension=1` ; si cette option est absente mais si l'option `smooth` est donnée, alors la valeur donnée à la tension est (par défaut) 0,55 ; sans l'option `smooth`, la valeur donnée est 0, la courbe est une suite de segments rectilignes (en interne un segment est une courbe de Bézier particulière).

Les points guides sont marqués avec un petit cercle plein, un signe plus et une petite croix de Saint-André grâce à l'option [19.7] :

`mark=*, + ou x`

Avec la bibliothèque `plotmarks`, on dispose de beaucoup d'autres glyphes. Leur dimension est choisie avec l'option [19.7] :

`mark size=(dimension)`

On peut ne marquer qu'un point sur 2 ou 3 ou etc. en écrivant [19.7] :

`mark repeat=2 ou 3 ou . . .`

On peut aussi décaler le début du marquage ou ne marquer que certains points avec les options [19.7] ;

`mark phase=(nombre de points non marqués au début)`

`mark indices={ (indices des points choisis séparés par des virgules) }`

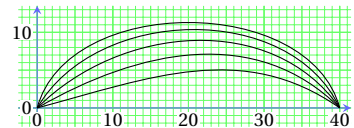
20.2. TRACER UNE COURBE AVEC TO

On aborde une deuxième possibilité pour tracer une courbe lorsqu'on dispose, en plus des coordonnées des points guides, des directions des tangentes en ces points. En effet, la commande [14.14] :

`draw (a,b) to [out= u , in= v , (options)] (c,d)`

trace une courbe allant du point (a,b) au point (c,d) et qui part du premier point dans la direction u degrés et qui arrive vers le deuxième point depuis la direction v degrés. Voici quelques courbes ayant les mêmes extrémités.

```
\begin{tikzpicture} [x=1mm,y=1mm]
\draw(0,0)to[out=15,in=145](40,0);
\draw(0,0)to[out=30,in=135](40,0);
\draw(0,0)to[out=45,in=125](40,0);
\draw(0,0)to[out=60,in=115](40,0);
\draw(0,0)to[out=75,in=105](40,0);
\end{tikzpicture}
```

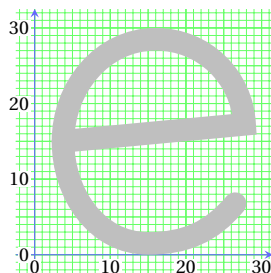


On se rend compte de la grande différence avec le tracé des courbes de Bézier de la section 12.1 : malgré l'augmentation des angles de départ et d'arrivée, la courbe reste relativement « collée » à la ligne joignant le point de départ au point d'arrivée ; rien d'étonnant : prenons l'exemple de la première courbe. Pour la tracer, TikZ trace la courbe de Bézier avec les points de départ et d'arrivée $(0,0)$ et $(40,0)$ et avec les points de contrôle $(15:r)$ et $(145:s)$ où les longueurs r et s sont *déterminées* en fonction des données (les deux points et les deux angles). Le tracé avec `to` est donc plus limité.

On reprend le tracé de la lettre minuscule `e` de la section 19.3 en gardant les mêmes points guides et les mêmes directions des tangentes en

ces points ... et quelle surprise ! Ainsi on constate que, si on décompose une ligne courbe en suffisamment d'éléments et si, pour chacun de ces éléments, la direction de la tangente orientée entre ses extrémités varie « raisonnablement », on peut espérer un résultat visuellement agréable comme dans cet exemple.

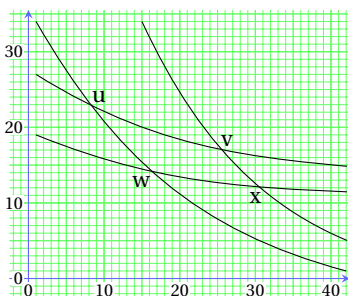
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=3mm,lightgray,cap=round]
(25.5,6.75)to[out=-130,in=0](15,1.5);
to[out=180,in=-90](3.75,15);
to[out=90,in=180](15,28.5);
to[out=0,in=93](27.75,17.25);
--(3.75,15);
\end{tikzpicture}
```



Pour la courbe de Bézier de la section 12.2, on a donné non seulement les orientations des tangentes aux points guides mais aussi les distances entre ces points et les points de contrôle voisins (les longueurs r et s de l'explication précédente) : on constate que le résultat du tracé avec l'opération `to` correspond approximativement avec le résultat obtenu avec la détermination par « tâtonnement » de ces distances.

L'utilisation de cette syntaxe (avec l'opération `to`) pour tracer des courbes semble fastidieuse ; il n'en est rien, en témoigne ce « cycle de Carnot » bien connu des thermodynamiciens ; la coloration du rectangle curviligne $uvwx$ (qui représente une grandeur physique importante) sera abordée plus loin et de deux manières différentes (intersection des courbes, section 29, découpages et calques, section 37).

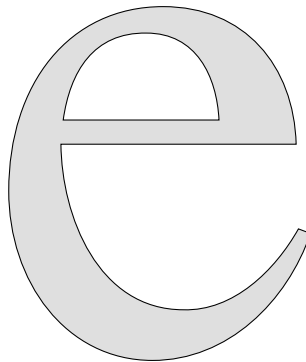
```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw(1,34)to[out=-60,in=165](42,1);
\draw(52,2)to[out=168,in=-65](15,34);
\draw(1,27)to[out=-32,in=175](52,14);
\draw(52,11)to[out=177,in=-22](1,19);
\pose(8.2,23)[-2]{45}{u}
\pose(25.7,17)[-2]{45}{v}
\pose(16.4,14.3)[-0.5]{-135}{w}
\pose(30.7,12)[-1]{-120}{x}
\end{tikzpicture}
```



Pour terminer, voici une lettre minuscule e où se mêlent art et infor-

matique (exemple cité par J. André, Cahiers GUTenberg, **26**, p. 24 et présenté ici codé en TikZ).

```
\begin{tikzpicture}[x=0.1mm,y=0.1mm]
\draw[fill=gray!25] (402,276)..controls(399,380)and(334,458)
..(226,458)..controls(102,458)and(22,356)..(22,214)..controls
(22,95)and(97,-10)..(212,-10)..controls(312,-10)and(390,68)..
(421,158)--(405,164)..controls(374,109)and(319,55)..(253,57)
..controls(140,57)and(92,181)..(91,276)--cycle;
\draw[fill=white] (94,308)..controls(103,372)and(134,424)..
(204,423)..controls(270,423)and(297,366)..(300,308)--cycle;
\end{tikzpicture}
```



CHAPITRE 5

Nœuds, liaisons et labels : organigrammes, algorithmes...

Ce chapitre est consacré aux nœuds, aux liaisons inter-nœud et aux labels, ce qui permet de construire des organigrammes, des algorithmes de toutes sortes, mais aussi de placer des lettrages sur les figures.

21. NŒUDS

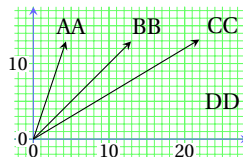
21.1. SYNTAXE GÉNÉRALE

La syntaxe pour la création d'un nœud au point de référence de coordonnées $x = a$ et $y = b$ et contenant du matériel (c'est-à-dire, au sens du \TeX Book, du texte, des formules et/ou du graphisme) est [13.2.3] :

```
\node [options] (nom du nœud) at (a,b) {matériel} ;
```

Le nom du nœud peut être une chaîne de lettres et de nombres avec des espaces (sans lettre accentuée bien entendu) ; on peut par exemple utiliser des noms du type P_1 , N_a , etc. On peut aussi ne pas donner de nom mais alors on perd la possibilité d'utiliser ce nœud par la suite. Les options possibles vont être examinées ci-dessous. Sans aucune option, on a le résultat suivant où l'on remarque que le matériel du nœud est simplement centré sur le point de référence ; on verra plus loin, lorsque l'on étudiera les liaisons entre les nœuds, pourquoi les flèches s'arrêtent à une certaine distance du contenu des nœuds.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node(a)at(10,15){AA};
\node(a b)at(20,15){BB};
\node(a_1)at(30,15){CC};
\node at(30,5){DD};
\draw[thin,>=stealth,->](0,0)--(a);
\draw[thin,>=stealth,->](0,0)--(a b);
\draw[thin,>=stealth,->](0,0)--(a_1);
\end{tikzpicture}
```



21.2. OPTIONS DE PRÉSENTATION

Voici la liste des options de présentation dont certaines ont déjà été vues dans les chapitres précédents et ne seront données que par leur nom générique.

— `draw=(couleur)` trace un contour de couleur spécifiée autour du matériel ; `draw` trace le contour en noir.

— si l'option `draw` est présente, on peut utiliser toutes les options de tracé des courbes fermées : épaisseur des traits, double trait, pointillés, traitillés et arrondissement des angles pour les formes qui en ont.

— `shape=rectangle, circle, ellipse` ou encore `diamond` définit la forme du contour [16.2.1]. Pour les dernières formes, il faut charger la bibliothèque `shapes` [48] qui contient d'autres formes intéressantes dont des formes « à plusieurs pages d'affichage » [16.3].

— `shape aspect=(nombre)` fixe le rapport (demi-axe horizontal)/(demi-axe vertical) pour la forme `diamond` [16.2.2].

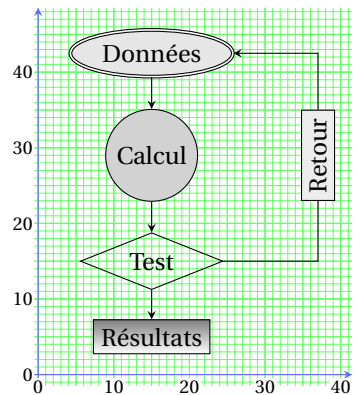
— `fill=(couleur)` remplit l'intérieur du contour tracé avec la couleur spécifiée.

— `shade` suivie des options d'ombrage (facultatives) sert à ombrer l'intérieur du contour (ces options seront vues par la suite).

— `rotate=(angle)` tourne le nœud de l'angle spécifié exprimé en degrés autour de son centre.

L'exemple qui suit montre l'action d'options des différents types décrits ci-dessus.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,ellipse,double,
fill=gray!20](a)at(15,42.5){Données};
\node[draw,circle,fill=gray!35]
(b)at(15,29){Calcul};
\node[draw,diamond,shape aspect=2.5]
(c)at(15,15){Test};
\node[draw,rectangle,shade]
(d)at(15,5){Résultats};
\node[draw,fill=gray!15,
rotate=90](e)at(37,29){Retour};
\draw[>=stealth,->](a)--(b);
\draw[>=stealth,->](b)--(c);
\draw[>=stealth,->](c)--(d);
\draw[>=stealth,->](c)-|(e)|-(a);
\end{tikzpicture}
```

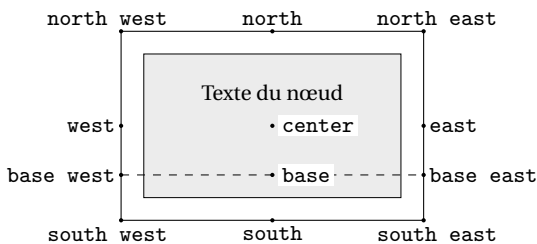


21.3. OPTIONS DE POSITIONNEMENT

Il s'agit du positionnement du nœud par rapport au point de référence ; on a vu que, sans option de positionnement, le nœud est centré sur le point de référence, ce que l'on peut exprimer en disant que le point de référence est au centre du nœud. La figure ci-dessous montre toutes les positions possibles du point de référence permettant tous les alignements possibles [16.5]. L'option d'alignement s'écrit :

`\node [anchor=(position)] (nom) at (a,b) {(matériel)};`

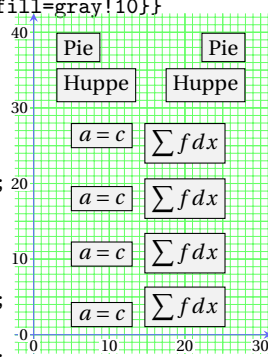
où la position est à choisir sur la figure ; `center` correspond au centre du nœud et est la position par défaut ; `base` correspond à la ligne de base du matériel contenu dans le nœud.



```

\begin{tikzpicture}[x=1mm,y=1mm]
\tikzset{every node/.style={draw,rectangle,fill=gray!10}}
\node[anchor=east]at(28,38){Pie};
\node[anchor=east]at(28,33){Huppe};
\node[anchor=west]at(3,38){Pie};
\node[anchor=west]at(3,33){Huppe};
\node[anchor=north]at(9,28){$a=c$};
\node[anchor=north]at(20,28){$\ds\sum f dx$};
\node at(9,18){$a=c$};
\node at(20,18){$\ds\sum f dx$};
\node[anchor=base]at(9,10){$a=c$};
\node[anchor=base]at(20,10){$\ds\sum f dx$};
\node[anchor=south]at(9,1){$a=c$};
\node[anchor=south]at(20,1){$\ds\sum f dx$};
\end{tikzpicture}

```



L'alignement vertical centré ayant été utilisé dans la figure précédente, l'exemple montre seulement deux alignements verticaux, l'un à gauche et l'autre à droite. Ensuite viennent quatre alignements horizontaux : le premier et le dernier sont des alignements par le haut et par le bas ; les

deux intermédiaires sont des alignements à la hauteur du centre et à la hauteur de ligne de base (ligne située à l'ordonnée $y = 10$). Ainsi sont testés les alignements utilisés dans des documents traditionnels. Il y a un autre alignement horizontal qui se fait à la mi-hauteur de la lettre x; il ne donne qu'un déplacement d'ensemble par rapport à l'alignement sur la ligne de base : il n'a donc aucun effet visible. Les alignements horizontaux à la ligne de base sont toujours utilisés pour les diagrammes des mathématiciens. On note que l'on a créé et utilisé le style `every node` pour abréger la saisie.

21.4. OPTIONS DE COMPOSITION

Ces options concernent ce que nous avons appelé le matériel du nœud; l'action de chacune de ces options est, d'après le nom, suffisamment claire pour les utilisateurs de \TeX pour qu'il soit suffisant de donner quelques courtes explications [16.4].

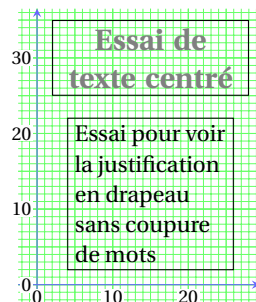
— `text=(couleur)` détermine la couleur du texte (noir par défaut) [16.4.1].

— `font=(taille-forme-graisse)`; par exemple, pour mettre tout le texte en petit italique gras, on écrit : `font=\small\itshape\bfseries` [16.4.2].

Pour une partie du texte seulement, il faut utiliser les commandes usuelles de \LaTeX dans le texte du nœud.

— `text width=(dimension)` détermine la largeur du texte du nœud [16.4.3].

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,rectangle,text width=24mm,
      align=center,text=gray,
      font=\Large\bfseries](b)at(20,30)
      {Essai de texte centré};
\node[draw,rectangle,text width=20mm,
      align=flush left](a)at(20,10)
      {Essai pour voir la justification
      en drapeau sans coupure de mots};
\end{tikzpicture}
```



— `align=justified` et `align=center` peuvent s'utiliser quand la largeur du texte est imposée; il peut y avoir des césures [16.4.3].

— `align=left` et `align=right` font des justifications en drapeau

avec césures et avec des lignes le plus possible égales entre elles [16.4.3].

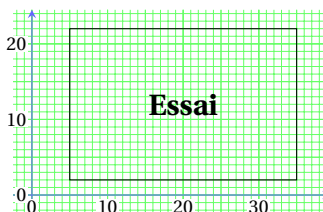
— `align=flush left` et `align=flush right` font des justifications en drapeau avec des lignes pouvant être très inégales mais en général sans césure [16.4.3].

— `inner sep=(dimension)` détermine la distance entre le texte et le contour du nœud (défaut : un espace inter mot normal) ; on a là l'explication du fait que les flèches s'arrêtent avant le nœud (cf. figure de la sect. 21). Cette distance peut être choisie différente dans le sens horizontal et le sens vertical grâce aux deux options : `inner xsep=(dimension)` et `inner ysep=(dimension)`.

— `minimum size=(dimension)` sert à assurer que toute dimension a une taille minimum (option utilisée dans la cas d'un nœud circulaire sur la figure de la sect. 18.3) [16.2.2].

— `minimum width=(dimension)` et `minimum height=(dimension)` servent à assurer des dimensions minimales (en largeur et en hauteur) au nœud complet [16.2.2].

```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[draw,rectangle,minimum height
=20mm,minimum width=30mm,font=
\Large\bfseries] (a) at (20,17){Essai};
\end{tikzpicture}
```



— `outer sep=(dimension)` (défaut : 0 pt) est une option qui caractérise aussi les nœuds bien que n'affectant pas l'apparence du nœud ; c'est la distance minimum à laquelle un tracé extérieur au nœud peut s'approcher de la frontière du nœud ; sa valeur par défaut doit être au moins la demi-épaisseur du contour du nœud pour que les flèches des liaisons ne donnent pas l'impression de « piquer » dans le contour du nœud [16.2.2]. Comme pour `inner sep=`, on dispose des deux options : `outer xsep=(dimension)` et `outer ysep=(dimension)`.

22. LIAISONS ET LABELS ATTACHÉS

Cette section est consacrée aux liaisons inter nœuds et aux labels qui peuvent leur être attachés. On traite successivement les liaisons à un segment puis à plusieurs segments. On commence par examiner sur le premier exemple quelques propriétés et notations générales.

— On remarque d’abord sur les deux liaisons AA vers BB et CC vers DD du premier exemple qu’il y a deux variantes de la syntaxe : le label est construit avec l’expression [16.8] :

node [options] {matériel}

qui est placée, soit après la commande qui trace la ligne de liaison, soit à l’intérieur des spécifications de cette commande, juste avant le deuxième nœud. Nous préférons la première syntaxe qui, *lorsqu’elle peut être utilisée*, est plus claire : la liaison d’abord, le label ensuite.

— On remarque ensuite que les labels sont des nœuds particuliers avec une syntaxe de positionnement spécifique qui assure leur attachement à une liaison.

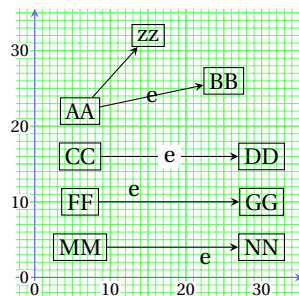
— Le nombre p , $0 \leq p \leq 1$, désigne la position du label le long de la ligne de liaison ; 0 correspond à l’origine et 1 à l’extrémité ; 0.5 est la valeur par défaut ; l’option correspondante s’écrit `pos=p`.

— La position latérale du label par rapport à la ligne de liaison est ainsi définie : en parcourant la ligne de liaison de l’origine vers l’extrémité, pour que le label se trouve à gauche (à droite), il faut donner l’option `above` (`below`) ; sans option, il est sur la ligne de liaison (dans ce cas, il faut prévoir un fond blanc pour améliorer la lisibilité du label).

22.1. LIAISONS PAR UN SEUL SEGMENT

On remarque encore sur l’exemple la direction par défaut du segment de liaison (il provient du centre du nœud de départ et va vers le centre du nœud d’arrivée), les différentes possibilités de position du label et la présence d’un fond blanc pour le label de la liaison de CC vers DD.

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=2pt,>=stealth]
\small\nrec(A)(6,22)\nrec(B)(25,26)
\draw[->](A)--(B)node[pos=0.5]{e};
\nrec(C)(6,16)\nrec(D)(30,16)
\draw[->](C)--node[fill=white]{e}(D);
\nrec(F)(6,10)\nrec(G)(30,10)
\draw[->](F)--(G)node
[pos=0.25,above]{e};
\nrec(M)(6,4)\nrec(N)(30,4)
\draw[->](M)--(N)node[pos=0.75,below]{e};
\end{tikzpicture}
```

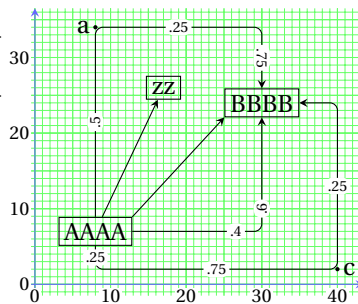


Pour obtenir ce fond blanc, quand le label est sur la liaison (et ceci s'applique aussi aux cas où le label se superpose à des parties de la figure), on ajoute les options : `rectangle` ou `circle`, puis `fill=white` et éventuellement `inner sep=1.5pt` (l'espace inter mot est trop grand dans ce cas) qui découpent ce fond blanc rendant le label mieux visible.

22.2. LIAISONS À 2, 3 OU 4 SEGMENTS PARALLÈLES AUX AXES

On note d'abord sur l'exemple que remplacer `AA` par `AA.north east` fait que la liaison du nœud `AAAA` vers le nœud `BBBB` part de la position nord-est du nœud `AAAA` au lieu de donner l'impression de partir de son centre comme le fait la liaison du nœud `AAAA` vers le nœud `zz`; une modification semblable fait que cette liaison arrive à l'angle sud-ouest du nœud `BBBB` et ne se dirige pas vers le centre de ce nœud. Il y a ensuite une liaison à trois segments et une liaison à quatre segments; ces liaisons, qui nécessitent la donnée d'un point supplémentaire (`a` pour la première et `c` pour la seconde), seront décrites en détail après la figure correspondante :

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=2pt,>=stealth]
\nrec(AA)(8,7)\nrec(BB)(30,24)\nrec(z)(17,26)
\coord(a)at(8,34);\coord(c)at(40,1);
\tikzset{every path/.style={rounded corners=1mm,->}}
\tikzset{every node/.style={fill=white,scale=0.6}}
\draw(AA.north east)
--(BB.south west);
\draw(AA)--(z);
\draw(AA)-|(BB)node[pos=0.4,sloped]{.4}node[pos=0.6,sloped]{.6};
\draw(AA)--node[pos=0.5,sloped]{.5}(a)-|(BB)
node[pos=0.25,sloped]{.25}node[pos=0.75,sloped]{.75}(BB);
\draw(AA)-|(BB)node[pos=0.25]{.25}
node[pos=0.75](c)-|(BB)node[pos=0.25]{.25}(BB);
\end{tikzpicture}
```



Ces liaisons à plus de deux segments, nécessitant un point supplémentaire, doivent être considérées comme formées de deux éléments différents (l'un avant ce point, l'autre après). Cela joue un rôle dans le

positionnement des labels (p repart à 0 en ces points supplémentaires) mais aussi dans la syntaxe de construction des labels. Il faut alors utiliser la deuxième syntaxe pour poser les labels le long des liaisons : l'expression `node [.] { . }` doit être placée à la fin de chaque élément, c'est-à-dire juste avant le nœud d'arrivée de la liaison. Il faut aussi prendre en compte le fait que, pour les éléments contenant un angle, l'angle est à la position $p = 0.5$. Sur la figure les labels donnent la valeur de p déterminant leur position. L'option `sloped` assure que le label a sa ligne de base parallèle à la liaison (option également valable pour les liaisons par des courbes).

La position des commandes `\tikzset` appelle deux remarques. D'une part, si on avait donné l'option `rounded corners=1mm` comme option de figure, les « vrais » nœuds auraient eu aussi les angles arrondis alors que seuls les liaisons et les petits rectangles blancs des labels (qui sont des nœuds du point de vue code comme on l'a déjà fait remarquer) ont les angles arrondis. D'autre part, l'option `scale=0.6` ne s'applique qu'au matériel des labels mais pas au matériel des « vrais » nœuds définis auparavant.

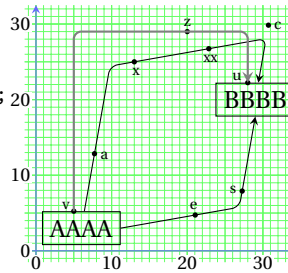
Pour continuer, il faut utiliser les commandes suivantes ;
— `\node [coordinate, shift={(c,d)}] (N) at (a,b) {} ;`
qui crée un nœud vide (un point invisible mais utilisable pour les tracés) de nom N déplacé de (c, d) [16.2.1] par rapport au point (a, b) ; le point (a, b) peut être remplacé par (A) où A est le nom d'un nœud déjà défini ; ensuite (c, d) peut être exprimé en coordonnées polaires, soit $(w:r)$: cela permet de définir une droite passant par un point donné et faisant un angle de w degrés avec l'axe des x (par exemple droite passant par (A) et (N) définie par la formule précédente avec (c, d) remplacé par $(w:r)$, avec $r > 0$). L'option `shift={...}` peut être remplacée par `xshift=...` ou `yshift=...` en donnant des dimensions avec unité sans $()$ ni `{}`.
— `(intersection of a,b--c,d and e,f--g,h)`
donne le point d'intersection de la droite passant par (a, b) et (c, d) et la droite passant par (e, f) et (g, h) (on remarque la syntaxe spéciale : pas de $()$ entourant les coordonnées).

L'exemple suivant montre trois liaisons, une à deux segments et deux à trois segments dans le cas où ces segments sont inclinés par rapport aux axes : on y détaille en particulier comment on peut faire pour faire partir (arriver) plusieurs segments d'un (sur) un même nœud.

```

\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\tikzset{evp/.style={rounded corners=1mm,->,>=stealth}}
\begin{tikzpicture}[x=1mm,y=1mm]
\nrec(AA)(8,3)\nrec(BB)(30,20)
\node[shift={(10:10)}](e)at(AA.east){};
\node[shift={(-100:10)}](s)at(BB.south){};
\draw[evp](AA.east)--(intersection of
AA.east--e and BB.south--s)--(BB.south);
\node[shift={(80:10)}](a)at(AA){};
\node[shift={(80:10)}](c)at(BB){};
\coord(x)at(18,25);
\node[shift={(10:10)}](xx)at(x){};
\draw[evp](AA)--(intersection of AA--a and x--xx)
--(intersection of x--xx and BB--c)--(BB);
\coord(z)at(20,29);
\node[coordinate,xshift=-1](v)at(AA.north){};
\node[coordinate,xshift=-1](u)at(BB.north){};
\draw[evp,gray,thick](v)|-(z)-|(u);

```



La figure montre d'abord deux liaisons AAAA vers BBBB de la figure précédente faites cette fois avec des segments faisant un angle de 10° avec les axes (axe des x tourné de $+10^\circ$ et axe des y tourné de -10°). On remarque que l'expression `coordinate` n'est pas toujours nécessaire dans la définition des nœuds d'aide au tracé (cas où le point d'aide est à l'extérieur du nœud).

On détaille la liaison par le bas (deux segments) : la liaison part de `AA.east` vers `e`; elle arrive en `BB.south` depuis le direction de `s`; ces droites `AA.east--e` et `BB.south--s` se coupent au point `(intersection of AA.east--e and BB.south--s)` d'où le tracé de la liaison.

La liaison par le haut part de `AA` vers `a` et arrive en `BB` depuis la direction de `c`; on se donne le point intermédiaire `x`, ce qui permet de définir le point `xx`; le point `x` et le point `xx` définissent une droite qui va porter le troisième segment de la liaison : cette liaison part de `AA` passe par les points `(intersection of AA--a and x--xx)` puis par le point `(intersection of x--xx and BB--c)` et arrive enfin en `BB`.

Ensuite, en gris épais, on a une autre liaison par le haut avec des segments parallèles aux axes. Pour brancher une deuxième liaison sur un nœud, il suffit de prendre un point sur la même face du nœud mais légèrement décalé : ici on prendra `v` décalé de 1 mm vers les x négatifs

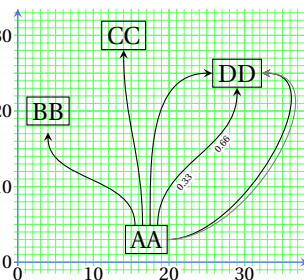
par rapport à `aa.north` et `u` pareillement décalé par rapport à `BB.north`. Cette méthode est très facile à mettre en œuvre dans le cas où il y a de nombreuses liaisons sur un même nœud.

En conclusion, on peut dire que les exemples donnés présentent le nécessaire pour construire des algorithmes, des organigrammes et d'autres structures semblables. On a approfondi les problèmes posés par constructions « denses » où la réussite du contenant est importante pour transmettre le contenu.

22.3. LIAISONS PAR LIGNES COURBES

Les cinq liaisons en trait noir de l'exemple sont obtenues avec l'opération « `to` », c'est-à-dire en donnant seulement les directions des tangentes aux extrémités ; on se trouve alors obligés d'utiliser la deuxième syntaxe pour poser les labels : les expressions `node [.] { . }` sont placées à la fin de chaque élément de tracé. La dernière liaison, en trait gris, est une courbe de Bézier dont les points de contrôle ont été choisis pour assurer la bonne direction des tangentes aux extrémités (les distances entre le premier, resp. second, point de contrôle et l'origine, resp. l'extrémité, de la courbe sont de l'ordre de la demi distance entre les extrémités).

```
\def\nrec(#1)(#2){\node(#1)at(#2)[rectangle,draw]{#1#1};}
\begin{tikzpicture}[x=1mm,y=1mm,inner sep=6pt,>=stealth]
\nrec(A)(17,3)\nrec(C)(14,30)\nrec(D)(29,25)
\node[rectangle,draw,outer sep=1mm](B)at(4,20){BB};
\node[coordinate,xshift=-1.5mm](A1)at(A.north){};
\node[coordinate,xshift=-0.5mm](A2)at(A.north){};
\node[coordinate,xshift=0.5mm](A3)at(A.north){};
\node[coordinate,xshift=1.5mm]
(A4)at(A.north){};
\tikzstyle{every node}=[fill=white,
scale=0.4,sloped,below,outer sep=3pt]
\draw[->](A1)to[out=90,in=-90](B);
\draw[->](A2)to[out=90,in=-90](C);
\draw[->](A3)to[out=90,in=180](D);
\draw[->](A4)to[out=90,in=-90](D)
node[pos=.33]{.33}node[pos=.66]{.66};
\draw[->](A)to[out=0,in=0](D);
\draw[->,gray](A)..controls+(0:14)and+(0:14)..(D);
\end{tikzpicture}
```



Un point important est à souligner : on a choisi de placer les labels

le long de la courbe sur un petit rectangle blanc pour les rendre bien visibles (options `rectangle`, `fill=white`, `inner sep=2pt` et `below`) ; si on s'en tient là, à cause de la concavité ou de la convexité de la courbe, les petits fonds blancs rectangulaires vont « mordre » légèrement sur la courbe : on a ajouté l'option `outer sep=3pt` qui éloigne le label de la courbe (un pt paraît suffire, il ne faut pas trop éloigner le label de la courbe). Le rôle primordial de cette option [13.13] est d'écarter les liaisons de leur nœud de départ et de leur nœud d'arrivée, cas du nœud BB sur la figure, mais on voit qu'elle permet aussi d'éloigner les labels des liaisons auxquelles ils sont attachés.

Il semble bien que, pour des organigrammes ou des algorithmes complexes, ce sont les liaisons par segments parallèles aux axes qui assurent le plus de clarté ; les liaisons courbes peuvent, par exemple, trouver une place en très petit nombre au milieu de blocs de texte.

23. LABELS ET « PINS » ASSOCIÉS AUX NŒUDS

Il est assez évident que, pour ne pas surcharger les figures, les labels et les pins associés aux nœuds doivent être relativement « sobres » (comme les labels attachés aux liaisons). On va retrouver pour ces labels et ces pins une syntaxe rappelant (à une importante différence près) celle des labels attachés aux liaisons : les commandes de labels et de pins sont des options de la commande des nœuds auxquels ils sont attachés [16.10] ; elles s'écrivent :

```
label={ [options] u : (matériel) }
```

```
pin={ [options] u : (matériel) }
```

où `u` est la direction en degrés dans laquelle le label ou le pin doit être placé ; on peut aussi utiliser pour `u` une des huit directions `above`, `below`, etc. Avec des nœuds de rattachement tournés, il peut être utile d'ajouter l'option `absolute` pour maîser cette direction (cf. [16.10]).

La distance du label ou du pin au nœud de rattachement (partie visible du rayon entre le label ou le pin et le nœud) est déterminée par :

```
label distance=(dimension) (défaut : 0 pt)
```

```
pin distance=(dimension) (défaut : 0 pt)
```

Pour cet exemple, il a fallu augmenter ces dimensions pour compenser la petite valeur choisie pour `inner sep`, sinon les labels et les pins ne se détacheraient pas suffisamment du nœud (car la distance `outer sep` est nulle par défaut).

Pour bien voir comment se fait le positionnement des labels, on donne une deuxième version du nœud entouré de labels où on a tracé (dans un but pédagogique) en traitillé fin et gris, des rayons montrant les directions dans lesquelles les labels sont placés et en trait fin, le contour des labels. On remarque alors que le positionnement directionnel du label se fait de la manière suivante : à l'exception d'une position dans une direction voisine des quatre directions principales, c'est le coin le plus près du centre du nœud qui est placé dans la direction imposée.

Le codage de cet exemple soulève deux difficultés :

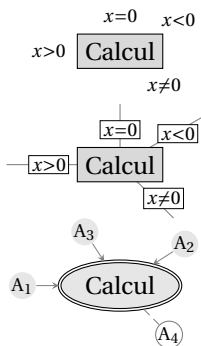
— On a été obligé de remettre à 12 le `\catcode` du caractère deux-points (séparateur entre l'angle et le matériel) sinon on peut obtenir, sans erreur de traitement, un résultat *très surprenant* !

— Le caractère = n'étant pas accepté dans la partie matériel, il a fallu enfermer l'égalité dans une `\hbox` (méthode bien connue et utilisée dans d'autres situations où un caractère donné joue un rôle spécial dans une syntaxe particulière).

```

\catcode'\:=12
\def\Nrec(#1)(#2)#3#4{\node(#1) at (#2)
  [rectangle,draw,fill=gray!30,#4] {#3};}
\def\Nell(#1)(#2)#3#4{\node(#1) at (#2)
  [ellipse,draw,double,fill=gray!20,#4] {#3};}
\tikzset{every label/.style={rectangle,
  inner sep=1pt,fill=white,label distance=3pt}}
\tikzset{every pin/.style={circle,inner sep=0pt,
  fill=gray!20,pin distance=3mm,
  pin edge={>=stealth,<-,shorten <=0pt}}}
\begin{tikzpicture} [x=1mm,y=1mm]
\draw[thin,gray] (30,16)--+(180:15)(30,16)--
  +(30:13)(30,16)--+(90:8)(30,16)--+(-45:10);
\Nrec(a)(30,31){Calcul}{label=left:$\sc x>0$,label=30:$\sc x<0$,
  label=90:\hbox{$\sc x=0$},label=-45:\hbox{$\sc x\not=0$}}
\Nrec(aa)(30,16){Calcul}{label={[draw,very thin]left:$\sc
  x>0$},label={[draw,very thin]30:$\sc x<0$},
  label={[draw,very thin]90:\hbox{$\sc x=0$}},
  label={[draw,very thin]-45:\hbox{$\sc x\not=0$}}}}
\Nell(b)(30,0){Calcul}{pin=left:$\sc A_{1}$,pin=30:$\sc A_{2}$,
  pin=120:$\sc A_{3}$,pin={[fill=none,draw=gray,
  pin edge={densely dashed,shorten <=0.5pt}]-45:$\sc A_{4}$}}
\end{tikzpicture}

```



On a défini un style `every label` pour les labels : il n'y a pas de tracé du contour (`draw`) et l'on y trouve l'option de forme (`rectangle`), la distance de séparation texte-contour (`inner sep`), la couleur du fond (`fill`) et la distance au nœud de rattachement (`label distance`). L'écriture du code des labels se fait sans invoquer d'option supplémentaires à celle contenues dans le style déclaré `every label`.

Les pins sont en fait constitués de la tête et de l'épingle (ou du pied).

— Pour les têtes, un style `every pin` a été défini : il n'y a pas de tracé du contour (`draw`) et l'on y trouve l'option de forme (`circle`), le fond gris (`fill`), la distance de séparation texte-contour (`inner sep`), la distance au nœud de rattachement (`label distance`) et le style `pin edge` qui contient des options pour l'épingle.

— Pour l'épingle, le style `pin edge` contient, pour cet exemple, des options habituelles de tracé de flèches ; l'option `shorten <=1mm` est utilisée pour que la flèche n'arrive pas au trait central blanc du contour du nœud de rattachement.

L'écriture du code pour les labels n'invoque pas d'option supplémentaire à celles contenues dans le style `every label` en vigueur ; par contre pour, pour la deuxième version, on a invoqué les options `draw` et `thin` pour visualiser le positionnement des labels par rapport aux rayons représentant les directions choisies.

L'écriture du code pour les trois premiers pins n'invoque pas d'option supplémentaire à celles contenues dans le style `every pin` en vigueur ; par contre pour le dernier pin, on laisse le fond en blanc et on trace le contour (`fill` et `draw`), puis on redéfinit entièrement le style `pin edge` : un simple trait en traitillé remplace la flèche des trois premiers pins.

L'utilisation de deux types de labels ou de pins sur un groupe de nœuds identiques peut avoir un apport pédagogique : on peut imaginer une situation où un type de nœud représenterait un élément ayant deux propriétés différentes dont les valeurs associées seraient mentionnées sur des pins de deux styles différents.

Là encore, on a abordé l'essentiel pour être en mesure de créer des documents de bonne qualité pédagogique. Pour faciliter le travail, il ne faut jamais oublier de définir quelques macros \TeX (telles que `\Nrec`) et des styles (tels que `every label`) : on y gagne en facilité et en qualité.

CHAPITRE 6

Visualisation des données

Ce nom de chapitre (traduction mot à mot) désigne ce que les physiciens, les chimistes, les ingénieurs, etc. appellent « tracé de courbes », c'est-à-dire la représentation graphique de variations (théoriques et/ou expérimentales) de grandeurs en fonction de variations de paramètres.

On commence avec la commande `plot` déjà vue à la section 20.1.

24. TRACÉ DE COURBE DIRECT AVEC `plot`

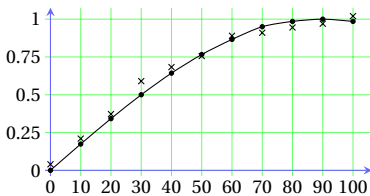
Les syntaxes des commandes de tracé de courbes avec `plot`, dont la première et brièvement la deuxième ont été testées (cf. sect 20.1), sont :

```
\draw plot [smooth, (options)] coordinates{(a,b) ... (x,y)};  
\draw plot [smooth, (options)] file{(chemin/fichier)};  
\draw plot [smooth, (options)] ("fonction tikz");  
\draw plot [smooth, (options)] fonction{"fonction gnuplot"};
```

24.1. DEUXIÈME SYNTAXE

On dispose des fichiers `f1.dat` et `f1a.dat` au format voulu (cf. sect. 20.1) [19.4] donnant la valeur y d'une grandeur en fonction du paramètre x ; on constate sur le fichier que $0 \leq x \leq 100$ et $0 \leq y \leq 1$ et l'on veut une figure de dimensions approximatives : largeur = 40 mm et hauteur = 20 mm : on choisit les unités $x=0.4\text{mm}$ et $y=20\text{mm}$:

```
\begin{tikzpicture}[x=0.4mm,y=20mm]  
\draw plot [smooth,mark=*,mark  
size=0.8pt]file{f1.dat};  
\draw plot [only marks,mark=x,mark  
size=1.6pt]file{f1a.dat}  
\foreach\x in {0,10,...,100}  
{\poseb(\x,0){-90}{\sc\x$}} ...  
\foreach\y in {0,0.25,...,1}{\poseb(0,\y){180}{\sc\y$}}  
\end{tikzpicture}
```



Les différentes options utilisées pour ces tracés à partir de fichiers ont été vues à la section 20.1. La nouvelle option :

`only marks`

trace uniquement les points dont les coordonnées sont contenues dans le fichier `f1a.dat` : on peut faire ainsi figurer les résultats expérimentaux sur la même figure que la courbe théorique. Pour améliorer la lisibilité, on a introduit une grille d'un pas de 10 unités suivant les x et d'un pas de 0.25 unités suivant les y avec la macro `\pps` qui fonctionne comme la macro `\ppm` (pas de graduation) sauf qu'il faut donner deux arguments supplémentaires : les pas de grille suivant les x et les y :

```
\pps(-2.5mm,-2.5mm)(42.5mm,21.5mm){10}{0.25}
```

Cette commande est construite avec la commande TikZ :

```
\draw[xstep=(dimension),ystep=(dimension),(options)]grid...;
```

Enfin, deux boucles `\foreach` et la macro `\poseb` sont utilisées pour placer les graduations.

24.2. TROISIÈME SYNTAXE

Pour tester cette syntaxe [19.4], on considère la fonction $\sin(x)$ pour $0 \leq x \leq 100$ (c'est la fonction qui donne le tableau utilisé pour le test précédent, fichier `f1.dat`) : on choisit donc les mêmes unités. On doit donner en plus les paramètres :

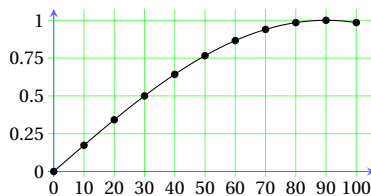
`domain=(valeur initiale) : (valeur finale) : domaine de variation,`

`samples=(nombre) : nombre de points calculés,`

`mark repeat=(nombre) : affichage de certains point uniquement.`

Cela donne :

```
\begin{tikzpicture}[x=0.4mm,y=20mm]
\draw plot[mark repeat=4,
mark=*,mark size=1.2pt,samples=41,
domain=0:100](\x,{sin(\x)});
\foreach\x in {0,10,...,100}
{\poseb(\x,0){-90}{\sc\x$}}
% selon l'axe y : comme ci-dessus
\end{tikzpicture}
```



Dans ce tracé, on n'a marqué qu'un point sur quatre avec l'option `mark repeat=4`. On portera son attention sur la syntaxe de la fonction donnée; les exemples suivants donneront plus de détails.

On va maintenant tester le tracé de courbes en « représentation paramétrique ». On veut tracer, par exemple, la fonction en coordonnées

polaires par $r = \cos^2(w)$ avec $0 \leq w \leq 90$; on va donc tracer la courbe en représentation paramétrique définie par $x = \cos^3(w)$ et $y = \cos^2 \sin(w)$. Pour cela, il faut donner l'option :

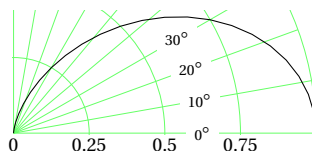
`parametric` (défaut : `false`)

et nommer le paramètre utilisé dans les expressions des deux fonctions : `variable=\w` (`\x` et `\y` sont définies par défaut)

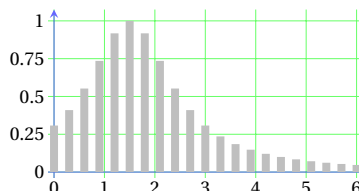
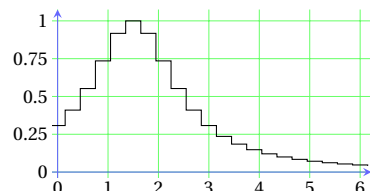
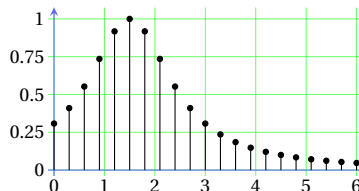
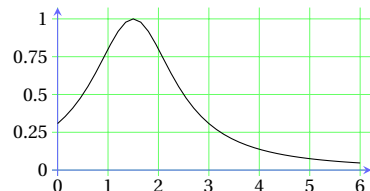
Le maximum de x est 1 pour $w = 0$: on choisi donc `x=40mm` et `y=40mm`.

Voilà le tracé présenté avec une grille adaptée aux coordonnées polaires :

```
\begin{tikzpicture}[x=40mm,y=40mm]
\begin{scope}[colorlines,very tin]
\path[clip](-0.2mm,-0.2mm)rectangle(40.2mm,16.2mm);
\foreach\i in {1,2,...,4}
    {\draw(0,0)circle(0.25*\i);}
\foreach\i in {0,10,...,90}
    {\draw(0,0)--(\i:1);}
\end{scope}
\draw plot[parametric,
variable=\w,domain=0:90]
({cos(\w)^3},{sin(\w)*cos(\w)^2});
\end{tikzpicture}
```



Toujours avec la troisième syntaxe, on va donner plusieurs présentations de la même courbe [19.8]. Dans un même document, il est possible d'utiliser plusieurs présentations des courbes associées à des données de type différent. La fonction choisie est $1/((x-1.5)^2 + 1)$ pour $0 \leq x \leq 6$; son maximum est 1; on prend donc `x=6.6667mm` et `y=20mm` :



Voici le code de ces quatre « looks » :

— courbe simple (bien entendu, on peut donner à `\draw` les options usuelles de tracé de ligne : épaisseur, couleur, etc.)

```
\begin{tikzpicture}[x=6.6667mm,y=20mm]
\pps(-2.5mm,-2.5mm)(41.5mm,21.5mm){1}{0.25}
\draw plot[domain=0:6,samples=41](\x,{1/((\x-1.5)^2+1)});
\foreach\i in {0,1,...,6}{\poseb(\i,0){-90}{$\sc\i$}}
\foreach\i in {0,0.25,...,1}{\poseb(0,\i){180}{$\sc\i$}}
\end{tikzpicture}
```

— avec les « champignons » :

options de `plot` : ajout de `ycomb,mark=*,mark size=1pt`

— avec les « marches d'escalier » :

options de `\draw` : `xshift=-1mm`

options de `plot` : `domain=0:6.3,samples=22,const plot`

— avec les barres grises :

options de `\draw` : `draw=none,fill=lightgray`

options de `plot` : ajout de `ybar,bar width=3pt`

24.3. QUATRIÈME SYNTAXE

Le tracé de courbes à l'aide de cette syntaxe, nécessitant la disponibilité de `GNUPLOT`, se fait suivant un des deux processus suivants :

A — Si la possibilité de `TEX` de faire des appels système existe et est activée², alors `TEX` peut arrêter sa propre tâche, lancer un autre programme par l'intermédiaire d'un fichier de commande qu'il a écrit lui-même, lire le fichier de résultats écrit par ce programme et enfin reprendre et terminer sa propre tâche en utilisant ces résultats.

Dans le cas qui nous intéresse, `TEX` commence par chercher le fichier `(préfixe)(nom).table`

a) s'il ne le trouve pas, alors il écrit un fichier

`(préfixe)(nom).gnuplot`

a1) il utilise ce fichier pour lancer `GNUPLOT`, lui faire calculer les coordonnées des points guides et les lui faire écrire dans un fichier

`(préfixe)(nom).table`

a2) il lit ce dernier fichier et termine en traçant la courbe

²Il faut pour cela que le compilateur `TEX` soit lancé avec une option en ligne de commande (`-enable-write18` dans le cas de `MIKTEX`).

a3) dans ces notations, (préfixe) peut être un chemin (plots/ par exemple) ou même rester vide et (id) est un nom repérant la figure elle-même ;

b) s'il trouve ce fichier d'extension .table, alors T_EX trace la courbe passant par les points dont les coordonnées sont contenues dans ce fichier (ce fichier n'est donc créé qu'une seule fois).

B — Si l'on ne dispose pas de cette possibilité (si elle n'est pas activée ou si le système d'exploitation est mono tâche), T_EX cherche d'abord le fichier (préfixe)(nom).table

a) s'il ne le trouve pas

a1) il écrit le fichier (préfixe)(nom).gnuplot

a2) à l'aide de ce fichier (qui contient en fait la commande complète avec toutes les options nécessaires) on peut alors lancer GNU-PLOT à la main (éventuellement sur un autre ordinateur) et obtenir le fichier d'extension .table

contenant les coordonnées des points guides

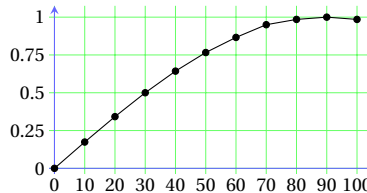
a3) enfin, on relance T_EX en s'assurant que ce fichier de coordonnées est bien accessible, le chemin d'accès doit conforme à celui donné dans (préfixe) ;

b) s'il trouve ce fichier d'extension .table, alors il trace la courbe correspondante.

Bien entendu, il faut donner à plot les deux options supplémentaires :
prefix=(chemin)
id=(nom figure)

On reprend la première figure de la présente sous-section : il faut prendre en compte que, sous GNU-PLOT l'argument de la fonction doit être donné en radians (pas en en degrés), ce qui explique le facteur 0.017453. Voici cette figure :

```
\begin{tikzpicture}[x=0.4mm,y=20mm]
\draw plot[mark=*,mark size=1.2pt,
domain=0:100,samples=11,prefix=t/,
id=sinus]function{sin(x*0.017453)};
\foreach\x in {0,10,...,100}
{\poseb(\x,0){-90}{\sc\x}}
% idem avec \y
\end{tikzpicture}
```



Remarque : On déduit de ce qui a été expliqué ci-dessus que, si l'on dispose du fichier (prefixe)(nom) .table, on peut recompiler le fichier du document sans disposer de GNUPLOT.

Remarque : Il est précisé dans la documentation que, dans la cas de fonctions complexes (en écriture), il faut travailler en trois étapes, comme si les appels système n'était pas possibles.

25. TRACÉ DE COURBES : VERSION AVANCÉE

Le qualificatif avancé est là pour dire que cette approche est beaucoup plus puissante que celle précédemment exposée et se prête à des développements supplémentaires. Le « hic » c'est qu'il n'y a pas une seule ligne de documentation alors que la distribution de la version 2.10 de TikZ (version actuelle) contient tous les fichiers nécessaires.

En conséquence, cette fin de chapitre est rédigée à partir de :

— la documentation très réduite de la version 2.00 CVS : une page d'introduction, cinq pages sur les formats d'entrée des données et deux pages contenant trois exemples sans aucune explication ;

— la lecture (il faudrait dire le déchiffrage!) de deux fichiers qui, par comparaison aux deux fichiers correspondants de la version 2.00 CVS, ont permis de trouver les changements de syntaxe par rapport à la version 2.00 CVS. Ces fichiers sont :

`datavisualization.code.tex`

`datavisualization.formats.functions.code.tex`

situés dans la distribution de la version 2.10 (printemps 2012).

Pour cette version avancée, le `\catcode` du point-virgule et celui des deux-points doivent être remis à 12.

La syntaxe de base est du type suivant :

```
\datavisulization[
```

```
(type d'axes) school book axes, scientific --, etc.
```

```
(type de tracé) visualize as smooth line, -- scatter
```

```
(options pour l'axe x : grille, tiks, labels) x axis={...}
```

```
(options pour l'axe y : grille, tiks, labels) y axis={...}
```

```
data[format=function]{(description fonction)};
```

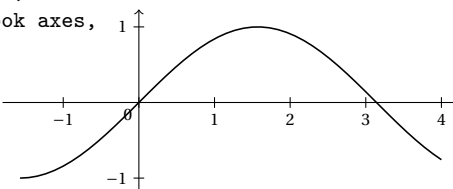
```
ou data[source=(chemin/fichier), format=(format de tableau)];
```

```
ou data[format=(format de tableau)]{(tableau en suivant)};
```


25.1. TRACÉS « ÉLÉMENTAIRES »

Le but des auteurs de la méthode est de formuler des « petits ensembles prêts à utiliser » sans aucune préparation. On prend l'exemple suivant : tracer la courbe représentant $y = \cos(x)$ pour $\pi/2 \leq x \leq 4$. Par défaut, l'unité suivant les deux axes est 10 mm : en conséquence, la largeur de la figure déterminée par le domaine de variation de la variable x est inférieure à 60 mm, ce qui est acceptable. On peut donc conserver toutes les options par défaut :

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12
\datavisualization[school book axes,
visualize as smooth line]
data[format=function]{
  var x:interval[-0.5*pi:4]
  samples 50;
  func y=sin(\value x r);}
\end{tikzpicture}
```

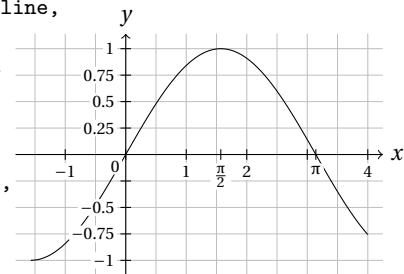


Les choix faits pour ce premier exemple sont :

- school book axes : axes et graduations ;
- visualize as smooth line : tracé de la courbe ;
- samples = . . . : qualité visuelle (50 points guides) ;
- var x : interval [. . .] : variable et son domaine de variation ;
- func y = (fonction) . . . : fonction à représenter.

On reprend le même exemple en l'améliorant ; les ajouts seront expliqués ligne par ligne par la suite.

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12
\datavisualization[school book axes,
clean ticks, visualize as smooth line,
x axis={unit length=8mm,
grid={step=1, minor steps between
steps=1}, ticks={some,
major={also at=(.5*pi)as$\pi/2$,
also at=(pi)as$\pi$,
options at=(3)as[no tick text]}},
label=$x$},
y axis={unit length=18mm,
grid={step=1, minor steps between
steps=3}, ticks={few,
major={options at=(-0.25)as[no tick text]}}, label=$y$}}
% data : voir ci-dessus \end{tikzpicture}
```



On détaille toutes les lignes du code ci-dessus :

Il n'y a aucune option pour ce type d'axes ; il y a d'autres types d'axes qui seront testés plus loin.

`clean ticks` fait que les ticks (les graduations) sont écrits sur un fond blanc recouvrant la grille et la courbe (amélioration de la lisibilité) ;

`visualize as smooth line` trace la courbe lissée ;

`visualize as scatter` tracerait seulement les points guides ;

`x axis={...}` contient les options pour l'axe des x :

`unit length=...` fixe l'unité de longueur (défaut : 10 mm) à déterminer en fonction du domaine de variation de 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={...}` contient les options pour la grille horizontale :

`step=...` fixe le pas de la grille exprimé avec l'unité choisie

`minor steps between steps=...` est le nombre de traits de la grille secondaire entre les traits de la grille primaire ;

`ticks={...}` contient les options de graduations :

`some` ainsi que `none`, `few` et `many` déterminent le nombre de graduations (*TikZ* calcule lui-même le nombre de graduations qu'il va tracer : on ne lui donne qu'une préférence, à apprécier soi-même à l'usage)

`major={also at=(...)as$. . .$, ...}` permet de rajouter des graduations

`options at=(...)as[no tick text]` supprime le texte d'une graduation ;

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

`y axis{...}` contient les options pour l'axe des y : on dispose des mêmes possibilités que pour l'axe des x ;

`all axes={...}` contiendrait des options communes aux deux axes ;

`var x : interval[. . .] samples ...` ; spécifications de la variable,

`func y = ... (\value x)` ; fonction $f(x)$.

Pour une représentation paramétrique, on aurait les trois lignes :

`var t : interval[. . .] samples ...` ; paramètre t ;

`func x = ... (\value t)` ; fonction $x = f(t)$,

`func y = ... (\value t)` ; fonction $y = g(t)$.

Le r qui se trouve après `\value x` dans l'expression de la fonction correspond à la multiplication par le facteur 57.2956 car les fonctions trigo-

nométriques prennent les angles exprimés en degrés.

Pour ces figures et pour les suivantes, ne disposant d'aucune documentation, on est amené à définir deux macros que l'on place dans un fichier nommé `mactikz.sty` qui sera chargé après tous les fichiers `TikZ`, ceci pour diminuer la taille des fontes utilisées pour les labels et le texte des ticks :

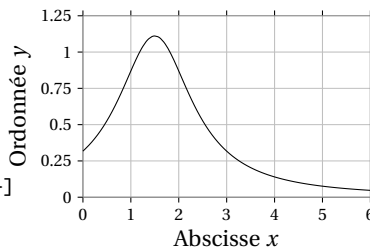
```
\def\pgfutil@font@footnotesize{\footnotesize}  
et une macro semblable avec la taille \small.
```

25.2. TRACÉS DITS « SCIENTIFIQUES »

On a vu les principes de base de cette approche du tracé de courbes. Il reste à voir les autres types d'axes (qui calculent automatiquement la valeur des unités suivant les axes), l'affichage de plusieurs courbes et/ou de plusieurs suites de résultats sur la même figure et les formats de lecture des données (en fin de code de la figure ou dans un fichier de données).

On commence par tester le type d'axe `scientific axes` :

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12  
\datavisualization[scientific axes,  
visualize as smooth line,  
x axis={length=38mm,grid={step=1},  
ticks={some},label=Abscisse $x$},  
y axis={length=24mm,  
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);};  
\end{tikzpicture}
```



On a remplacé la donnée des unités suivant les x et les y , par les dimensions souhaitées pour la figure :

`length=(dimension)`

pour les dimensions selon les x puis selon les y . Ces dimensions ne comprennent pas les graduations ni les labels. Le calcul que l'on avait fait précédemment est fait par `TikZ`. On a aussi ajouté :

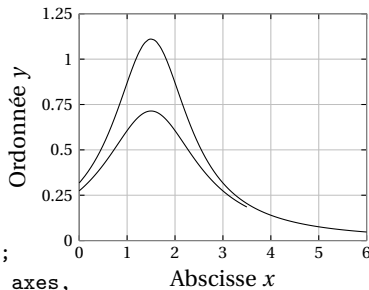
```
include values{a,b}
```

qui étend l'axe des y jusqu'à a et jusqu'à b : on convient que le trait

de grille de plus grande ordonnée ne peut pas être tangent au pic de la courbe tracée et que le point d'abscisse $x = 6$ ne peut être sur l'axe des x .

On continue avec le type d'axes `scientific inner axes` qui ne se différencie du précédent que par le fait que le petit tiret des ticks est tourné vers l'intérieur (au lieu d'être tourné vers l'extérieur) :

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12
\datavisualization[scientific inner axes,
visualize as smooth line,
x axis={grid={step=1},length=38mm,
ticks={some},label=Abscisse $$},
y axis={length=30mm,grid={step=1,
minor steps between steps=3},
ticks={some},label=Ordonnée $$,
include values={0,1.25}}]
data [format=function]{
var x : interval[0:6] samples 50;
func y = 1/((\value x-1.5)^2+0.9);};
\datavisualization[scientific inner axes,
visualize as smooth line,
x axis={length=38mm,ticks={none},include values={6}},
y axis={length=30mm,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}
```



Pour ajouter d'autres courbes sur la même figure, il faut une nouvelle macro `\datavisualisation` pour chaque courbe supplémentaire car l'option `visualize as ...` 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 courbe ! Il n'est pas nécessaire de reprendre toutes les options de `\datavisualisation`, en particulier celles concernant la grille et les graduations ; par contre, il faut s'assurer que, pour les deux axes, les données déterminées par `length` et `include values` forment les mêmes intervalles ; dans le cas de l'exemple, l'intervalle des x est `[0 : 6]` pour la première courbe et `[0 : 3.5]` pour la seconde : TikZ va donc répartir sur la largeur de 38 mm 6 unités pour la première courbe et 3.5 unités pour la seconde : il faut donc rajouter `include value={6}` aux options de l'axe des x de la deuxième courbe ; pour l'axe des y , on reprend les deux « valeurs incluses » 0 et 1.25.

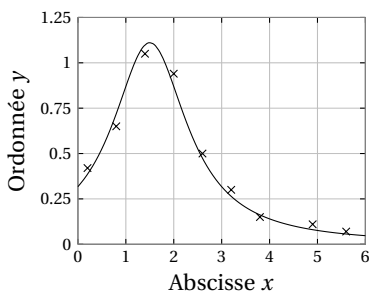
On continue encore avec les mêmes axes en reprenant la première courbe de la figure précédente et en y ajoutant une suite de points, par exemple un ensemble de points « expérimentaux » à confronter à la courbe théorique. Comme déjà vu au début de la présente section, il faut disposer d'un format de tableau pour que TikZ puisse afficher les points aux bonnes coordonnées sur la figure. Un format nommé `table` donné dans les quelques pages de la documentation de la version 2.00 CVS n'étant pas reconnu, on crée un format nommé `tableau` avec les quelques explications données dans ces quelques pages :

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

(la première paire d'accolades est destinée à recevoir un caractère de commentaire si on le souhaite).

On présente d'abord le cas où les données sont placées en fin du code de la figure. Comme pour une deuxième courbe, il faut renouveler la commande `\datavisualization` en veillant bien qu'il y ait similitude des intervalles de variation suivant le x et les y : dans le cas présent, il faut rajouter `include values{0,6}` et reprendre `include values{0,1.25}` respectivement pour les axes x et y :

```
\begin{tikzpicture}
\catcode'\;=12\catcode'\:=12
\datavisualization[scientific inner axes,
% courbe : voir les code ci-dessus
\datavisualization[scientific inner axes,visualize as scatter,
x axis={length=38mm,ticks={none},include values={0,6}},
y axis={length=30mm,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
4.9 0.11
5.6 0.07
};
\end{tikzpicture}
```

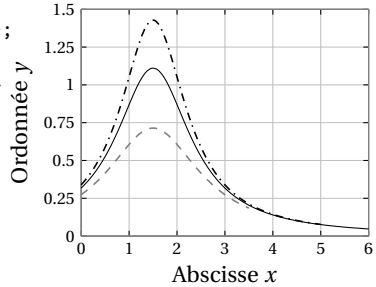


La version 2.10 de TikZ n'a pas de documentation pour la visualisation des données; elle ne permet pas, telle qu'elle est, de changer de type de ligne ni de glyphe pour l'affichage des courbes et des points. Un ajout, des modifications et les macros (le tout dans le fichier `mactikz.sty`) :

```
\MARK[(dimension)]{(glyphe)}
\TRAIT{(options de trait)}
```

permettent l'utilisation de `x`, `+` et `*` (mais aussi de `o`, `square`, etc. si la bibliothèque `plotmarks` est chargée) ainsi que des options d'épaisseur, de couleur, de traitillé, etc. On teste d'abord la macro `TRAIT{.}` :

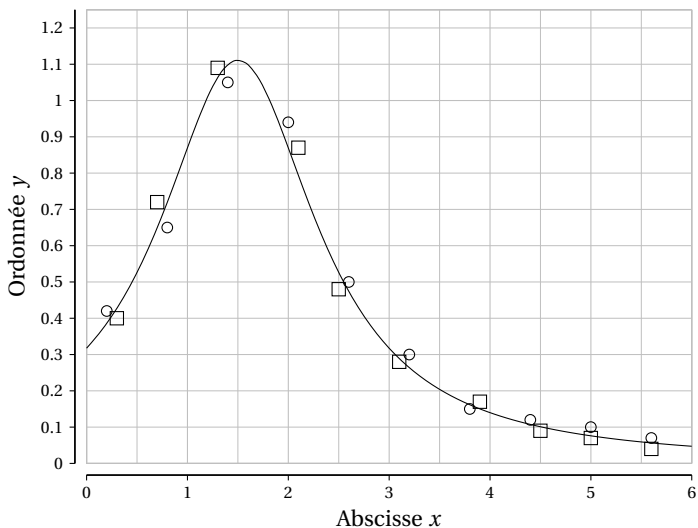
```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12
\datavisualization[scientific inner axes,visualize as smooth line,
x axis={grid={step=1},length=38mm,ticks={some},label=Abscisse $x$},
y axis={length=30mm,grid={step=1,minor steps between steps=3},
ticks={some},include values={0,1.5},label=Ordonnée $y$}]
data [format=function]{
var x : interval[0:6] samples 50;
func y = 1/((\value x-1.5)^2+0.9);};
\TRAIT{dashed,gray}
\datavisualization[scientific inner
axes,visualize as smooth line,
x axis={length=38mm,ticks={none},
include values={6}},
y axis={length=30mm,ticks={none},
include values={0,1.5}}]
data [format=function]{
var x : interval[0:3.5] samples 50;
func y = 1/((\value x-1.5)^2+1.4);};
\TRAIT{dashdotted}
\datavisualization[scientific inner axes,
visualize as smooth line,
x axis={length=38mm,ticks={none},include values={6}},
y axis={length=30mm,ticks={none},include values={0,1.5}}]
data [format=function]{
var x : interval[0:5] samples 50;
func y = 1/((\value x-1.5)^2+0.7);};
\end{tikzpicture}
```



Pour terminer, on utilise le type d'axes `scientific clean axes` dans l'exemple suivant où l'on trace une courbe et deux suites de points qu'il faut, bien entendu, afficher avec des glyphes différents. On peut voir la différence entre `many` (tick à tous les traits de grille) et `some` (tick à un trait

de grille sur deux). On teste ainsi la macro `\MARK[.]{.}` avec argument facultatif (le carré est volontairement agrandi).

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12
\datavisualization[scientific clean axes,visualize as smooth line,
x axis={length=80mm,grid={step=1,minor steps
between steps=1},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)};
\MARK{o}\datavisualization[scientific clean axes,
visualize as scatter,
x axis={length=80mm,ticks={none},include values={0,6}},
y axis={length=60mm,include values={0,1.25},ticks={none}}]
data [source=f3.dat,format=tableau];
\MARK[2.5]{square}\datavisualization[scientific clean axes,
visualize as scatter,
x axis={length=80mm,ticks={none},include values={0,6}},
y axis={length=60mm,include values={0,1.25},ticks={none}}]
data [source=f4.dat,format=tableau];
\end{tikzpicture}
```

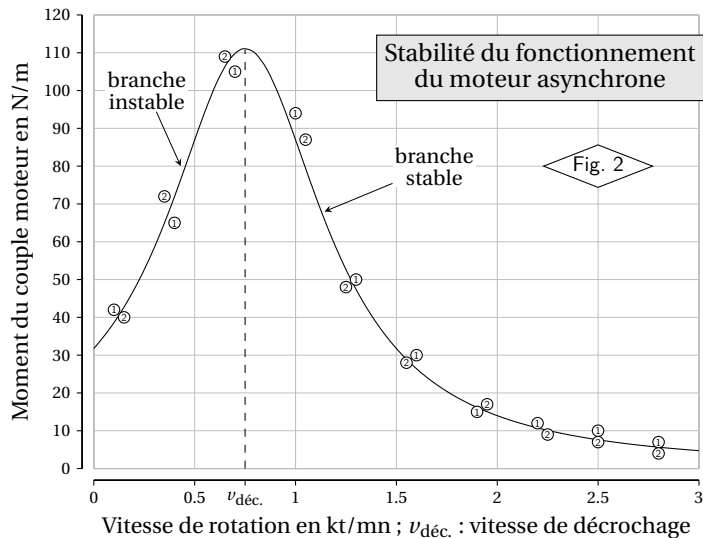


On peut souhaiter utiliser des glyphes pour marquer les points expérimentaux permettant une identification de la provenance des données : par exemple des glyphes qui porteraient un chiffre ou une lettre (voir les glyphes de l'exemple suivant). Voilà la construction d'un tel glyphe qui s'utilise comme les glyphes prédéfinis avec la macro `\MARK{.}` mais sans paramètre optionnel ; ici la taille du glyphe est déterminée par l'option `scale=...` qui figure dans sa commande de construction :

```
\def\faitmarku{\pgfdeclareplotmark{marku}%
{\node[circle,draw,thin,inner sep=0.8pt,
scale=0.4,font=\sffamily](0,0){1};\pgfusepathqstroke}}
```

Il reste un problème à résoudre : positionner des informations de quelques mots sur la figure : en effet on ne connaît pas les longueurs des unités que TikZ a calculées. La solution logique est fastidieuse : il faut considérer le bloc d'information à placer sur le graphique comme un glyphe de représentation d'une suite de données (ici suite à un seul élément : losange contenant le texte : Fig. 2) :

```
\def\faitmarkinfo{\pgfdeclareplotmark{markinfo}{\node[diamond,
aspect=2.5,draw,thin,inner sep=0.8pt,outer sep=1pt,fill=white,
scale=0.8,font=\sffamily](0,0){Fig. 2};\pgfusepathqstroke}}
```



Heureusement, il y a une solution beaucoup plus pratique permettant de rajouter des informations : on superpose (provisoirement bien entendu) la grille `\ppmm` utilisée dans certaines figures ; on repère à l'écran les positions adéquates (en millimètres) et on les utilise pour placer les blocs comme l'on fait d'habitude ; il faut préciser les unités ou bien ajouter l'option de figure `[x=1mm, ...]` qui ne perturbe pas les calculs internes de TikZ. Voilà le code de l'exemple :

```
\begin{tikzpicture}[x=1mm,y=1mm]\catcode'\;=12\catcode'\:=12
\datavisualization[scientific clean axes,visualize as smooth line,
  x axis={length=80mm,grid={step=1,minor steps between steps=1},
  ticks={some,major={also at=0.75as$v_{\text{déc.}}$}},
  label={Vitesse de rotation en kt/mn ;
    $v_{\text{déc.}}$ : vitesse de décrochage}},
  y axis={length=60mm,grid={step=100,minor steps between steps=9},
  ticks={many},include values={0,120},
  label=Moment du couple moteur en tour/mn}]
data [format=function]{
  var x : interval[0:3] samples 50;
  func y = 100/((2*value x-1.5)^2+0.9);};
\faitmarku\MARK{marku}
\datavisualization[scientific clean axes,
  visualize as scatter,
  x axis={length=80mm,ticks={none},include values={0,3}},
  y axis={length=60mm,include values={0,120},ticks={none}}]
data [source=f3a.dat,format=tableau];
\faitmarkd\MARK{markd}
\datavisualization[... 3 lignes comme ci-dessus
data [source=f4a.dat,format=tableau];
\faitmarkinfo\MARK{markinfo} % définition plus haut
\datavisualization[... 3 lignes comme ci-dessus
data [format=tableau]{2.5 80}; % tableau d'une seule ligne
%\ppmm(0mm,0mm)(80mm,60mm)
\draw[thin,dashed](20,0)--(20,55.4);
\node[draw,rectangle,fill=gray!20,align=center]at(59,53)%
  {Stabilité du fonctionnement\[-1mm] du moteur asynchrone};
\montre(45,40)(30.5,34){branche\[-1mm]stable}
\montre(7,50)(11.5,40){branche\[-1mm]instable}
\end{tikzpicture}
```

On remarque l'utilisation de la petite macro :

```
\montre((pointe))((queue)){(texte)}
```

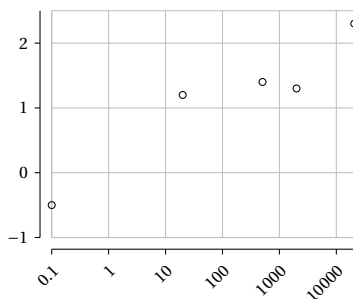
particulièrement commode dont la définition est :

```
\def\montre(#1)(#2)#3{\node(nd)at(#1)[fill=white,
font=\footnotesize,align=center,inner sep=0.7pt]{#3};
\draw[>=stealth,->,very thin](nd)--(#2);}
```

Sur ces derniers exemples, on teste le cas d'une graduation logarithmique de l'axe des x grâce à l'option `logarithmic` ainsi que le cas où les zéros des deux axes ne sont pas dans les domaines de variation de la variable et de la fonction.

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12\MARK{o}
\datavisualization[scientific clean axes,visualize as scatter,
x axis={logarithmic,length=40mm,grid={step=1},
ticks={some,node style={rotate=45,anchor=north east}}},
y axis={length=30mm,grid={step=1},
ticks={some},include
values={-1,2.5}}]
```

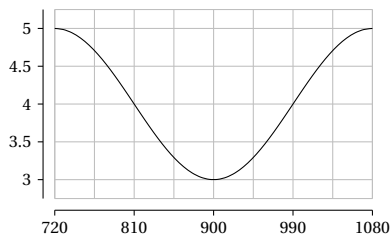
```
data [format=tableau]{
0.1 -0.5
20 1.2
505 1.4
2000 1.3
20670 2.3
};
```



```
\end{tikzpicture}
```

```
\begin{tikzpicture}\catcode'\;=12\catcode'\:=12\MARK[1.3]{o}
\datavisualization[scientific clean axes,visualize as smooth line,
x axis={length=42mm,grid={step=45},include values={720,1080},
ticks={few,step=45}},
y axis={length=25mm,
grid={step=0.5},ticks={some},
include values={2.75,5.25}}]
```

```
data [format=function]{
var x : interval[720:1080]
samples 38;
func y = cos(\value x)+4;};
```



```
\end{tikzpicture}
```

Afin d'avoir une certaine cohérence entre les lignes de grille et les ticks, on a du donner une valeur de `step` pour les ticks de l'axe des x et, pour supprimer la virgule des mille, on a du utiliser l'option `mille` [66.1] :

```
\tikzset{mille/.style={/pgf/number format/1000 sep={}}}
```

CHAPITRE 7

Intersections de lignes

26. INTERSECTION DE DROITES

On a déjà vu, section 22.2, que les coordonnées du point d'intersection de la droite passant par les points (a, b) et (c, d) et de la droite passant par les points (e, f) et (g, h) sont données par l'expression :

`(intersection of a,b--c,d and e,f--g,h)`

On peut utiliser directement ces coordonnées pour :

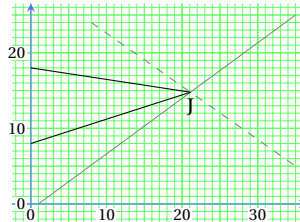
— faire un tracé : `\draw(inters...)--(x,y)` (cf. sect. 22.2),

— définir un point : `\coord(P)at(inters...)` ;

— définir un nœud : `\node[coordinate](N)at(inters...){(texte)}` ;

Ce problème a, dans la version 2.10, une formulation plus générale qui permet de traiter le problème de l'intersection de courbes quelconques. L'exemple montre deux variations légèrement différentes de l'intersection de deux droites [13.3.2]. Il faut charger la bibliothèque `intersections`. On explique le code après la figure :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[gray,dashed](8,24)--(35,5);
\draw[name path=droiteA,very thin,
      gray](1,0)--(35,25);
\path[name path=droiteB](8,24)--(35,5);
\draw[name intersections={of=droiteA
      and droiteB,by={I}}](I)--(0,8);
\node[coordinate,name intersections={%
      of=droiteA and droiteB,name=pinter}](J)at(pinter-1){};
\draw(J)--(0,18);\pose(J){-90}{J}
\end{tikzpicture}
```



On doit d'abord nommer les droites, ce qui se fait avec la clé :

`name path=(nom)`

placée en option des commandes `\draw` ou `\path` suivant que l'on veut, en même temps, la tracer ou non.

Le calcul est exécuté (on peut dire déclenché) par la clé principale :

`name intersections={...}`

(introduite comme option de la commande `\draw`) avec les sous-clés :

`of=(droite1)and(droite2)`

`by=(nom-du-point-d'intersection)`

Le point d'intersection ainsi calculé est disponible pour tout tracé, y compris pour la commande ayant donné lieu à sa détermination.

On peut aussi procéder d'une autre manière consistant à remplacer la sous-clé `by` par la sous-clé :

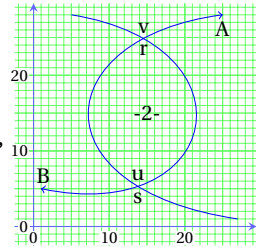
`name=(préfixe)`

Le nom du point d'intersection est alors (`pintersec-1`) si l'on a choisi `name=pintersec`. C'est cette manière de procéder, qui va être généralisée au cas de l'intersection des lignes courbes.

27. INTERSECTION DE DEUX COURBES

Pour deux courbes, la syntaxe est celle décrite ci-dessus pour deux droites sauf qu'il peut y avoir aucune, une ou plusieurs solutions (les cas triviaux de deux droites ou deux courbes confondues étant exclus) [13.3.2]. Le sens de croissance du paramètre temps des deux courbes de l'exemple est repéré par une flèche :

```
\begin{tikzpicture}[x=1mm,y=1mm]\footnotesize
\draw[->,very thin,blue,name path=pathA]
(27,1)..controls(0,5)and(2,25)..(25,28);
\draw[->,very thin,blue,name path=pathB]
(5,28)..controls(30,24)and(25,0)..(1,5);
\pose(25,28){-90}{A}\pose(1.3,5){90}{B}
\path[name intersections={of=pathA and pathB,
total=\nbr,sort by=pathA,by={u,v}}]
\pgfextra{\xdef\nombre{\nbr}};
\pose(u){90}{u}\pose(v){90}{v}
\node at (15,15){-\nombre-};
\path[name intersections={of=pathA and pathB,
sort by=pathB,by={r,s}}];\pose(r){-90}{r}\pose(s){-90}{s}
\end{tikzpicture}
```



On reconnaît le nommage des courbes, la clé principale et la clé secondaire `by` dont la syntaxe générale est :

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

Ces noms sont les noms des points d'intersection dans l'ordre où ils sont

calculés. Il y a ensuite la clé secondaire :

```
sort by={ (nom-d'une-des-deux-courbes) }
```

qui provoque, en fin de calcul, le classement des points d'intersection par temps d'intersection croissants par rapport à la courbe choisie : pour le premier calcul (classement par rapport à la courbe A), le nom `u` est attribué au point d'intersection ayant le plus petit temps d'intersection en tant que point de la courbe A, ce que montre la figure. Le deuxième calcul (classement par rapport à la courbe B) confirme le « fonctionnement » du classement. La sous-clé :

```
total=(macro-contenant-le-nombre-d'intersections)
```

contient un résultat pouvant être utile dans des situations particulières ; ici, on l'a simplement affiché, ce qui ne peut être direct car la macro `\nbr` n'est valable qu'à l'intérieur de la commande `TikZ` où elle a été définie (ici, la commande `\path`, dite fictive car traçant un chemin nul) : il faut donc utiliser la commande `\pgfextra` (voir sect. 6) pour placer son contenu dans une macro `TeX` `\nombre` afin de pouvoir l'afficher (ici au centre de la figure, entre deux tirets).

L'autre manière de travailler consiste à remplacer les lignes 7 à 10 par :

```
\path[name intersections={of=pathA and pathB,total=\nbr,  
  sort by=pathA,name=pintersec}] \pgfextra{\xdef\nombre{\nbr}};  
\pose(pintersec-1){90}{u}\pose(pintersec-2){90}{v}  
puis à faire un remplacement semblable pour le deuxième calcul.
```

Ce qui vient d'être dit est encore valable si une des deux lignes est une ligne droite : on la définit avec `\path` ou on la trace avec `\draw` et dans les deux cas on la nomme suivant la syntaxe :

```
... [ ... ,name path=(nom)] (a,b)--(c,d) ;
```

où (a,b) et (c,d) sont les points définissant la droite ; ces points peuvent aussi être donné par leur nom : (A) et (B).

Si l'une des deux lignes est une droite et s'il y a deux points d'intersection P et Q entre la droite et la courbe, on peut mettre en évidence les parties de la droite entre A et P, entre P et Q et entre Q et B par un simple tracé surchargeant le tracé initial (épaisseur, couleur, etc.). Mais on ne peut pas faire la même opération pour la courbe : la seule solution consiste à tracer les parties de la courbe entre les temps d'intersection relatifs aux deux points d'intersection ; c'est possible avec la macro [71.4] :

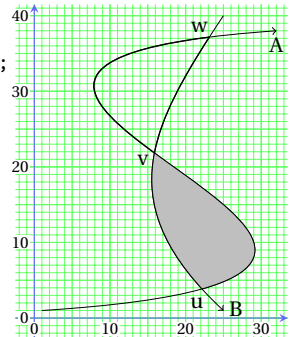
```
\pgfpathcurvebetweentime{(t1)}{(t1)}{(dép.){(ctrl.1)}{(ctrl.2)}{(arr.)}
```

si l'on dispose des temps t_1 et t_2 correspondant aux points d'intersection.

28. INTERSECTION ET COURBES PARTIELLES

Cette section est consacrée à la récupération des temps d'intersection de deux courbes et à leur utilisation pour mettre en évidence des parties de ces courbes et des domaines limités par ces courbes. D'abord il faut noter qu'un point d'intersection a deux temps d'intersection (un temps relatif à chaque courbe). Ces temps sont calculés et utilisés pour le classement des points d'intersection à temps d'intersection croissants (cf. `sort by`). Il a été possible d'écrire le nécessaire pour extraire ces temps d'intersection des fichiers de la distribution. Pour l'utilisateur, tout est transparent : ces fichiers ne sont pas touchés, les macros modifiées et les nouvelles macros sont dans le fichier `mactikz.sty`. On va expliquer la méthode de travail et le code sur l'exemple suivant. Pour simplifier la suite, on fait un tracé préalable pour connaître le nombre de points d'intersection, trois dans cet exemple).

```
\begin{tikzpicture}[x=1mm,y=1mm]\footnotesize
\useasboundingbox(-3mm,-2.5mm)rectangle(33.5mm,41.5mm);
\ppmm(-2.5mm,-2.5mm)(33.5mm,41.5mm)
\draw[->,very thin, name path=pA]
(1,1)..controls(80,6)and(-40,33)..(32,38);
\draw[->,very thin, name path=pB]
(25,40)..controls(15,25)and(10,15)..(25,1);
\pose(32,38){-90}{A}\pose(25,1.3){0}{B}
\path[name intersections={of=pA and pB,
total=\nbr,sort by=pA,name=iABA}];
\coord(u)at(iABA-1);\coord(v)at(iABA-2);
\coord(w)at(iABA-3);\pose(u)[1]{-110}{u};
\pose(v){-150}{v}\pose(w)[-1]{135}{w}
\CALCULDESTEMPS{tABA}
\path[name intersections={of=pA and pB,
total=\nbr,sort by=pB,name=iABB}];
\CALCULDESTEMPS{tABB}
{\souscourbe(1,1)(80,6)(-40,33)(32,38){\tABA-1,\tABA-2}
\souscourbe(25,40)(15,25)(10,15)(25,1){\tABB-2,\tABB-3}
\pgfsetlinewidth{0.2pt}\pgfsetfillcolor{lightgray}
\pgfusepath{fill,stroke}}
{\souscourbe(1,1)(80,6)(-40,33)(32,38){\tABA-2,\tABA-3}
\souscourbe(25,40)(15,25)(10,15)(25,1){\tABB-2,\tABB-1}
\pgfsetlinewidth{0.5pt}\pgfusepath{stroke}}
\end{tikzpicture}
```



On choisit les noms des courbes se terminant par une capitale : A, B, etc., pA et pB par exemple. Ensuite, les temps des points d'intersection de pA et pB sont nommés tABA ou tABB suivant que le classement par temps croissants se fait relativement à la courbe pA ou pB.

Pour commencer, on nomme et trace les deux courbes pA et pB.

Ensuite, on déclenche le calcul des points d'intersection par une commande `\path`, fictive dans le sens qu'elle ne définit rien. On y introduit l'option `sort by=(nom-d'une-courbe)` (pA par exemple) et l'option `name=iABA`; les coordonnées des points d'intersection sont alors (iABA-1), (iABA-2) et (iABA-3).

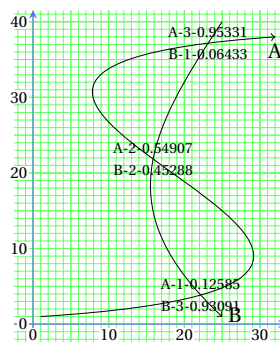
Enfin, la macro `\CALCULDESTEMPS{tABA}` récupère les temps d'intersection relativement à la première courbe et les met à disposition dans les macros `\tABA-1`, `\tABA-2` et `\tABA-3`.

Les deux paragraphes ci-dessus doivent être repris avec l'option `sort by=(nom-de-l'autre-courbe)` (donc pB pour l'exemple) afin de disposer, grâce à la macro `\CALCULDESTEMPS{tABB}` des temps d'intersection relativement à la deuxième courbe : `\tABB-1`, `\tABB-2` et `\tABB-3`.

On termine par la définition de sous courbes et leur utilisation comme annoncé en début de section. Pour faciliter la saisie, on utilise la macro `\souscourbe` donnée plus loin qui permet de tracer une partie de courbe entre deux valeurs du paramètre.

On reprend le même exemple pour savoir à quels points d'intersection obtenus au premier calcul (avec `sort by=pA`) il faut affecter les temps d'intersections obtenus au deuxième calcul (avec `sort by=pB`);

```
\begin{tikzpicture}[x=1mm,y=1mm]\footnotesize
% 7 lignes de la figure précédente
\pose(32,38){-90}{A}\pose(25,1.3){0}{B}
\path[name intersections={of=pA and pB,
total=\nbr,sort by=pA,name=iABA}]%
\pgfextra{\xdef\nombre{\nbr}};
\CALCULDESTEMPS{tABA}\tiny
\foreach\i in{1,...,\nombre}%
{\pose(iABA-\i){90}{A-\i-\tABA-\i}}
\path[name intersections={of=pA and pB,
total=\nbr,sort by=pB,name=iABB}];
\CALCULDESTEMPS{tABB}
\foreach\i in{1,...,\nombre}
{\pose(iABB-\i){-90}{B-\i-\tABB-\i}}\end{tikzpicture}
```



On affiche à chaque point d'intersection son numéro et le temps correspondant relativement à chaque courbe. On voit bien sur la figure que le temps $\backslash\text{tABB-1}$ (0.06433) doit être affecté au point d'intersection w : autrement dit, le temps correspondant à w est $\backslash\text{tABB-1}$ relativement à la courbe B et tABA-3 relativement à la courbe A, ce qui était évident dans cet exemple après avoir fait le tracé préalable.

29. INTERSECTION DES COURBES AVEC `to`

Bien souvent, pour des raisons de rapidité d'exécution, on utilise des courbes de Bézier construites avec `to`. Ce sont des courbes de Bézier dont la syntaxe de tracé est (cf. sect. 20.2) :

```
\draw [...] (a,b) [out=v,in=w] (c,d)
```

En interne et de façon transparente, TikZ trace la courbe de Bézier notée en syntaxe spéciale (cf. sect. 19.2) :

```
\path [...] (a,b) .. controls +(v:r) and +(w:s) ... (c,d)
```

où les longueurs r et s sont calculées par TikZ lui-même en fonction des autres données (les angles v et w étant bien entendu les angles donnés dans la commande de tracé écrite par l'utilisateur, cf. sect. 20.2).

Pour tracer des courbes partielles de ces courbes, il faut connaître les points de contrôle que TikZ a calculé de façon transparente pour leur tracé complet. Un ajout dans la définition de deux macros (dont la version modifiée est reportée dans le fichier `mactikz.sty`) et la macro `\CALCULDESCONTROLES{A}` permettent de mettre à disposition, pour la courbe pA par exemple, les quatre coordonnées `\xctrlaA`, `\yctrlaA`, `\xctrlbA` et `\yctrlbA`; cette notation inclut une unité. Cette propriété implique la création d'une adaptation de la macro `\souscourbe` nommée `\souscourbeto`. Voici ces deux macros :

```
\def\souscourbe(#1,#2)(#3,#4)(#5,#6)(#7,#8)#9{%
  \edef\argua##1,##2|{##1}\edef\argub##1,##2|{##2}%
  \pgfpathcurvebetweentime{\argua#9|}{\argub#9|}%
  {\pgfpoint{#1mm}{#2mm}}{\pgfpoint{#3mm}{#4mm}}%
  {\pgfpoint{#5mm}{#6mm}}{\pgfpoint{#7mm}{#8mm}}}%
```

Dans la macro `souscourbeto`, les deux dernières lignes ci-dessus sont modifiées en :

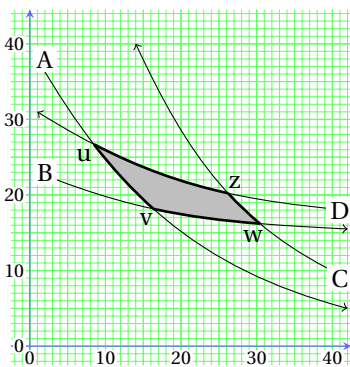
```
{\pgfpoint{#1mm}{#2mm}}{\pgfpoint{#3}{#4}}%
{\pgfpoint{#5}{#6}}{\pgfpoint{#7mm}{#8mm}}}%
(pas d'unité après les argument #3, #4, #5 et #6).
```


Le « cycle de Carnot » est constitué de quatre courbes délimitant une surface dont l'aire représente l'énergie produite par cycle. On calcule les quatre points d'intersection et les temps correspondants pour définir les sous courbes limitant cette aire que l'on veut colorer :

```

\begin{tikzpicture}[x=1mm,y=1mm]
\ppmm(-2.5mm,-2.5mm)(42.5mm,44.5mm)
\draw[name path=pA,->](1,38)%
    to[out=-60,in=165](42,5);
\CALCULDESCONTRÔLES{A}
\draw[name path=pB,->](1,23)%
    to[out=-22,in=177](42,15.5);
\CALCULDESCONTRÔLES{B}
\draw[name path=pC,->](42,9)%
    to[out=155,in=-65](14,40);
\CALCULDESCONTRÔLES{C}
\draw[name path=pD,->](42,18)%
    to[out=175,in=-32](1,31);
\CALCULDESCONTRÔLES{D}
\posb(2,38){A}\posb(2,23){B}\posb(41,9){C}\posb(41,18){D}
\path[name intersections={of=pD and pA,sort by=pA,name=iDAA}];
\CALCULDESTEMPS{tDAA}\pose(iDAA-1)[-1.5]{-135}{u}
\path[name intersections={of=pA and pB,sort by=pA,name=iABA}];
\CALCULDESTEMPS{tABA}
\path[name intersections={of=pA and pB,sort by=pB,name=iABB}];
\CALCULDESTEMPS{tABB}\pose(iABB-1)[-1]{-125}{v}
\path[name intersections={of=pB and pC,sort by=pB,name=iBCB}];
\CALCULDESTEMPS{tBCB}
\path[name intersections={of=pB and pC,sort by=pC,name=iBCC}];
\CALCULDESTEMPS{tBCC}\pose(iBCC-1){-120}{w}
\path[name intersections={of=pC and pD,sort by=pC,name=iCDC}];
\CALCULDESTEMPS{tCDC}
\path[name intersections={of=pC and pD,sort by=pD,name=iCDD}];
\CALCULDESTEMPS{tCDD}\pose(iCDD-1)[-0.5]{60}{z}
\path[name intersections={of=pD and pA,sort by=pD,name=iDAD}];
\CALCULDESTEMPS{tDAD}
% partie coloration de la surface
% partie tracé du périmètre de la surface
\end{tikzpicture}

```



On trouve d'abord dans le code ci-dessus les définitions des quatre courbes, chacune *immédiatement* suivie par le calcul des coordonnées

des points de contrôle. Ensuite, pour chaque segment curviligne et pour chacune de ses extrémités, on trouve le calcul du temps correspondant (on y trouve aussi le lettrage des extrémités des segments). Enfin, on va discuter la suite du code qui assure la coloration et le tracé.

```
% partie coloration de la surface
\pgfsetfillcolor{lightgray}
\def\scA{\souscoubeto(1,38)%
  (\xctrlaA,\yctrlaA)(\xctrlbA,\yctrlbA)(42,5){\tDAA-1,\tABA-1}}
\scA\pgf\lineto{\pgfpoint{17mm}{20mm}}
\pgf\closepath
\def\scB{\souscoubeto(1,23)%
  (\xctrlaB,\yctrlaB)(\xctrlbB,\yctrlbB)(42,15.5){\tABB-1,\tBCB-1}}
\scB\pgf\lineto{\pgfpoint{17mm}{20mm}}
\pgf\closepath
\def\scC{\souscoubeto(42,9)%
  (\xctrlaC,\yctrlaC)(\xctrlbC,\yctrlbC)(14,40){\tBCC-1,\tCDC-1}}
\scC\pgf\lineto{\pgfpoint{17mm}{20mm}}
\pgf\closepath
\def\scD{\souscoubeto(42,18)%
  (\xctrlaD,\yctrlaD)(\xctrlbD,\yctrlbD)(1,31){\tCDD-1,\tDAD-1}}
\scD\pgf\lineto{\pgfpoint{17mm}{20mm}}
\pgf\closepath\pgf\usepath{fill}
% partie tracé du périmètre de la surface
\pgfsetlinewidth{1pt}
\scA\pgf\usepath{stroke}
\scB\pgf\usepath{stroke}
\scC\pgf\usepath{stroke}
\scD\pgf\usepath{stroke}
```

Sans se noyer dans la « machinerie » PGE on va simplement expliquer les raisons qui conduisent à la méthode du remplissage du « parallélogramme curviligne » limité par les quatre courbes de l'exemple. On suggère de considérer l'exercice contenant les deux tracés suivants :

```
\draw[fill=lightgray,yshift=6mm](0,0)--(5,0)--(2.5,0)--(0,0);
\draw[fill=lightgray](0,0)--(5,0)(5,0)--(2.5,0)(2.5,0)--(0,0);
```

On constate que la première ligne construit le remplissage du triangle mais pas la deuxième ligne qui, en réalité, construit trois segments juxtaposés mais qui pourraient être séparés (juxtaposés ou séparés, ils sont indépendants du point de vue logique). On dira que le premier exemple constitue un chemin continu dans le sens que les coordonnées de la fin d'un segment sont aussi les coordonnées du début de la partie suivante.

Le cas de l'exemple est du type de la deuxième ligne : quatre segments de courbe juxtaposés, mais indépendants. Pour le remplissage, il faut utiliser la macro `\pgfusepath{fill}` qui, étant donné un chemin continu (au sens du premier exemple ci-dessus) mais non fermé, un arc de courbe par exemple, remplit la surface comprise entre l'arc de courbe et la corde [71.4] (on peut s'en rendre compte en supprimant les quatre macros `\pgflineto{.}` dans les lignes ci-dessus) ... et l'on comprend aussi pourquoi le remplissage de la surface de l'exemple précédent n'a pas posé de problème ! Par contre la commande `\pgflineto{.}` ajoute un segment de droite à partir de la deuxième extrémité de la sous-courbe et la commande `\pgfclosepath` trace un segment de droite partant du point atteint par `\pgflineto{.}` et arrivant à la première extrémité du segment curviligne : ainsi est réalisé un triangle à base curviligne du type de la première ligne de l'exercice suggéré. La macro `\pgfusepath{fill}` peut remplir ce triangle.

On termine la coloration en faisant trois fois encore la même opération et les quatre triangles recouvrent exactement la surface à colorer.

Enfin, on fait le tracé des quatre arcs de courbes à l'aide de la macro : `\pgfusepath{stroke}`.

On verra plus loin une autre manière moins laborieuse de réaliser cette figure.

CHAPITRE 8

Arbres et structures arborescentes

Non, il n'est pas question dans ce chapitre de fougères mais de graphiques en forme d'arbres destinés à représenter des structures hiérarchisées. On va donner des exemples simples pour montrer ce qui peut être fait avec les commandes spécifiques de TikZ et leurs options. Ensuite, ce sera à chaque lecteur de ces lignes, en fonction de sa spécialité, d'adapter l'utilisation de ces commandes et options au style des structures utilisées dans sa discipline.

30. ARBRES

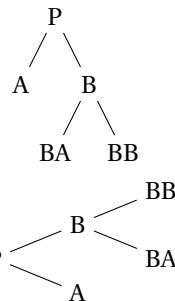
30.1. STRUCTURE DE BASE

On va entrer dans le vif du sujet en proposant une première structure simple où n'apparaissent que les deux commandes, [6.2] [11.6] :

`node` : pour créer le nœud « parent »

`child` : pour créer les nœuds « fils », « petit-fils », etc.

```
\begin{tikzpicture}[level distance=9mm,
                    sibling distance=9mm]
\node{P}
  child{node{A}} child{node{B}}
  child{node{BA}} child{node{BB}}};
\end{tikzpicture}\hspace*{5mm}\vskip1pt
\begin{tikzpicture}[grow=right,
                    level distance=11mm,sibling distance=9mm]
\node{P}
  child{node{A}} child{node{B}}
  child{node{BA}} child{node{BB}}};
\end{tikzpicture}
```



On remarque les trois options de figure :

`grow=(angle)` (défaut : -90)

donne la direction de croissance de l'arbre ; au lieu de donner un angle

(en degrés), on peut donner un des 4 points cardinaux north,... (défaut : south) ou une des 4 directions up,...(défaut down) ;

level distance=(dimension)

détermine la distance entre les générations dans la direction de croissance ;

sibling distance=(dimension)

détermine la distance entre les points d’ancrage des descendants ; cette distance ne prend pas en compte la largeur des nœuds des descendants, la distance de séparation entre ces nœuds ni le nombre de ces nœuds. En effet, on se rend tout de suite compte sur l’exemple précédent que l’on est très limité pour compléter la structure : par exemple, on ne peut ajouter le petit-fils AB car la place est occupée par le petit-fils BA.

Enfin, il faut noter que l’on n’a pas « placé » ni nommé directement les nœuds : la racine est placée par défaut à l’origine, ce qui se vérifie immédiatement en rajoutant la grille graduée `\ppmm (.) (.)` ; les autres nœuds sont placés successivement en partant du nœud racine et en utilisant les deux distances imposées (level distance et sibling distance). Les nœuds ont par défaut un nom déterminé à partir du nom du nœud racine de la façon suivante [18.3] : soit root le nom donné à ce nœud, les autres nœud ont respectivement les noms :

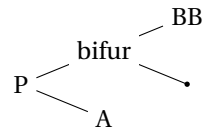
root-i, root-i-j, root-i-j-k, etc.

où -i, -i-j, -i-j-k, etc. représentent les « chemins d’accès » des nœuds en partant du nœud racine. Il faut noter que le nœud racine n’a pas de nom par défaut accessible

On reprend la fin de l’exemple précédent pour montrer, d’une part, que ces noms sont bien définis et, d’autre part, que changer node par coordinate

change le nœud correspondant en un point sans structure situé au point d’ancrage de ce nœud et affecté du nom de ce nœud :

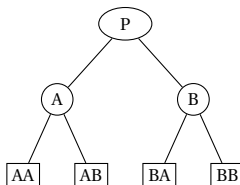
```
\begin{tikzpicture}[grow=right,
  level distance=11mm,sibling distance=9mm]
\node(root){P}
  child{node{A}} child{coordinate
    child{coordinate} child{node{BB}}};
\fill(root-2-1)circle(1pt);
\node[fill=white]at(root-2){bifur};
\end{tikzpicture}
```



30.2. ADAPTATION DE LA STRUCTURE DE BASE

On a vu que l'on devait, pour évaluer la distance entre les descendants de même génération, prendre en compte le nombre de nœuds de cette génération ainsi que les dimensions des nœuds et leurs distances de séparation. Cela est fait sur l'exemple suivant où l'on donne, pour chaque niveau de descendance, une valeur spécifique de l'option `sibling distance` : cela se fait à l'aide des styles `level 1`, `level 2`, etc. que l'on définit avec la commande `tikzset` [18.1 à 4] :

```
\tikzset{every node/.style={draw}}
\tikzset{level 1/.style={sibling distance=18mm,nodes={circle}}}
\tikzset{level 2/.style={sibling distance=9mm,nodes={rectangle}}}
\begin{tikzpicture}[level distance=10mm]
\node[ellipse]{\kern1ex P\kern1ex}
  child{node{A}
    child{node{AA}} child{node{AB}}}
  child{node{B}
    child{node{BA}} child{node{BB}}};
\end{tikzpicture}
```



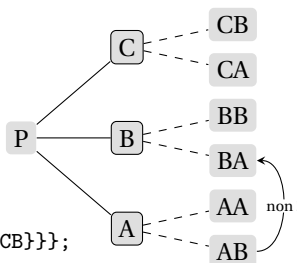
La syntaxe générale pour définir des styles pour les différentes générations est la suivante [18.4] (*i* : numéro de la génération) :

```
\tikzset{level i/.style={sibling distance=(distance),...,
  nodes={(style pour les nodes)},
  edge from parent={(style pour les liaisons vers le parent)}}}
```

Voici un exemple avec différents styles de nœuds et de liaisons :

```
\tikzset{every node/.style={rectangle,rounded
  corners=2pt,fill=gray!25}}
\tikzset{level 1/.style={sibling distance=12mm,nodes={draw,thin}}}
\tikzset{level 2/.style={sibling distance=6mm,nodes={draw=none},
  edge from parent/.style={draw,dashed}}}
```

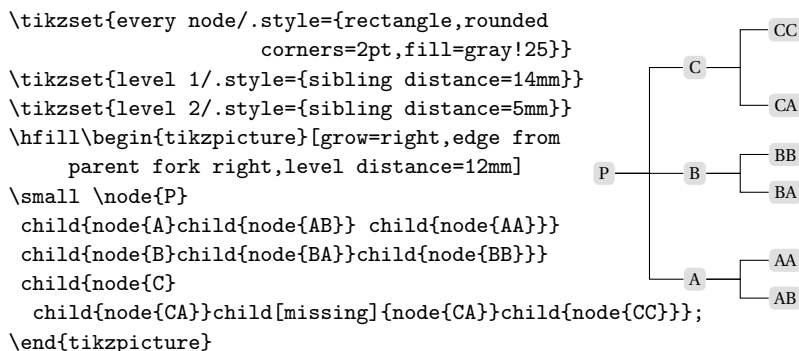
```
\begin{tikzpicture}[grow=right,
  level distance=14mm]
\node(p){P}
  child{node{A}
    child{node{AB}} child{node{AA}}}
  child{node{B}
    child{node{BA}}child{node{BB}}}
  child{node{C}child{node{CA}}child{node{CB}}};
\draw[>=stealth,->](p-1-1)to[out=0,in=0]
  node[fill=white,pos=0.5,inner sep=0.5pt]{\tiny non !}(p-2-1);
\end{tikzpicture}
```



Il faut noter que les options se transmettent aux niveaux inférieurs d'où la nécessité de `nodes={draw=none}` au niveau 2 (pour annuler un traitillé du niveau 2 on écrirait `nodes={dashed=solid}` au niveau 3).

On a en outre tracé une liaison courbe entre deux nœuds avec un label associé comme cela à été vu à la section 22.3 ; dans ce cas, on note que c'est la deuxième syntaxe qui doit être utilisée (celle où la pose du label se place juste avant le nœud où aboutit la liaison).

D'autres présentations sont disponibles [18.5 et 6], mais pour cela, il faut charger la bibliothèque `\usepackage{trees}`. On choisi d'abord un exemple d'arbre de forme classique et on prend la croissance horizontale mais la croissance verticale est aussi d'un bel effet.



L'option de nœud `missing` garde la place du nœud correspondant.

Il y a de nombreuses possibilités pour l'option de croissance, `grow`, ainsi que pour les options de liaison entre parents et descendants [18.6] : forme de la liaison elle-même et points d'ancrage de la liaison sur le nœud de départ et sur le nœud d'arrivée. Ci-dessous, on présente encore un dernier exemple plus original que les précédents pour lequel on adopte une autre manière de définir les styles nécessaires : on les place en option de figure. Pour cet exemple, on choisit l'option `grow cyclic` qui place les descendants de même génération sur un cercle autour du parent ; l'option `sibling distance` est alors remplacée par l'option : `sibling angle= (angle)`

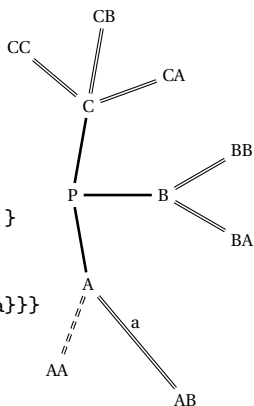
Il faut en plus donner le sens du positionnement angulaire des descendants de même génération et l'angle de départ de ce positionnement avec l'option :

`counterclockwise from (angle-de-départ)`


```

\begin{tikzpicture}[grow cyclic,level distance=12mm,
level 1/.style={sibling angle=80,edge from
parent/.style={draw,line width=1pt}},
level 2/.style={sibling angle=60,edge from
parent/.style={draw,double}}]\small
\node{P}[counterclockwise from=-80]
child{node{A}[counterclockwise from=-110]
child{node{AA}
edge from parent[densely dashed]}
child[level distance=20mm]{node{AB}
edge from parent node[fill=white,inner
sep=0.5pt,outer sep=3pt,right,pos=0.3]{a}}}
child{node{B}[counterclockwise from=-30]
child{node{BA}}child{node{BB}}}}
child{node{C}[counterclockwise from=20]
child{node{CA}}child{node{CB}}child{node{CC}}}};
\end{tikzpicture}

```



On remarque la syntaxe de la modification des liaisons entre les générations par le style :

`edge from parent`

défini dans l'argument d'une commande `\tikzset` (avant dernier exemple) ou en option de figure (présent exemple).

Comme toujours, une modification exceptionnelle d'une liaison est possible en plaçant l'option :

— pour la longueur : `level distance=(dimension)` en option de la commande `child` correspondante,

— pour le tracé : `edge from parent=[densely dashed]` juste après le matériel du nœud d'arrivée de la liaison concernée.

On remarque enfin que l'on peut ajouter un label aux liaisons entre parents et descendants en plaçant, toujours à la même position, l'option : `edge from parent node [(options de nœud), pos=0.3]`

30.3. APPLICATION À LA COMPOSITION DE FORMULES CHIMIQUES

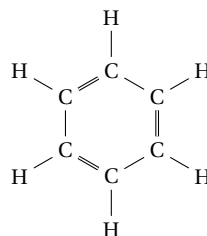
On va terminer ce chapitre en montrant comment l'option `grow` utilisée en tant qu'option de la commande `child` permet de composer des formules chimiques.

On commence avec la formule déployée du benzène qui rappellera à certains des temps heureux.

```

\begin{tikzpicture}[level distance=7mm,inner sep=2pt]
\def\double{edge from parent[double]}
\node{C}
  child[grow=up]{node{H}} %haut
  child[grow=-150]{node{C}} %gauche
    child[grow=150]{node{H}}
    child[grow=-90]{node{C}} %gauche
      child[grow=-150]{node{H}}
      child[grow=-30]{node{C}} %bas
        child[grow=-90]{node{H}}
          \double}\double}
  child[grow=-30]{node{C}} %droite
    child[grow=30]{node{H}}
    child[grow=-90]{node{C}} %droite
      child[grow=-30]{node{H}}
      child[grow=-150]{node{\phantom{C}}}\double}};
\end{tikzpicture}

```



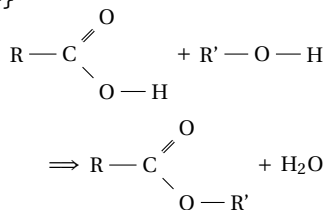
Sur cette formule déployée, on remarque que le C du bas est en troisième génération par l'intermédiaire de la demi chaîne de gauche mais aussi en troisième génération par l'intermédiaire de la chaîne de droite d'où l'utilisation de `` pour avoir une liaison parfaitement positionnée et pour éviter une superposition de deux lettres C.

On termine avec une très classique réaction de substitution :

```

\def\double{edge from parent[double]}
\def\bfor{\vcenter\bgroup\hbox\bgroup
  \begin{tikzpicture}[level distance=7mm,inner sep=2pt]}
\def\efor{\end{tikzpicture}\egroup\egroup}
$\bfor\node{C}
  child[grow=left]{node{R}}child[grow=45]{node{O}\double}
  child[grow=-45]{node{O}} child[grow=right]{node{H}};\efor
+\bfor\node{O}child[grow=left]{node{R'}}
  child[grow=right]{node{H}};\efor$
\vskip1mm$\Longrightarrow
\bfor\node{C}
  child[grow=left]{node{R}}
  child[grow=45]{node{O}\double}
  child[grow=-45]{node{O}}
  child[grow=right]{node{R'}};\efor
+\bfor\node{H$}_{2}$0};\efor$

```



Ces exemples sont donnés sans prétendre fournir une méthode infaillible pour les formules chimiques : ils donnent une idée du possible.

31. MINDMAPS

Cette dénomination, *mindmap* en anglais, est utilisée pour désigner des ensembles d'éléments en relation hiérarchisée sous la forme d'arbres ayant une structure et surtout un style particulier : les nœuds sont des cercles colorés et les liaisons ressemblent à des liaisons entre les organes des êtres vivants. La structure de base de ces mindmaps [39.2 à 5] est un cas bien particulier de la structure des arbres à croissance cyclique : on va retrouver certains aspects de la syntaxe utilisée dans ces structures. Il faut d'abord charger la bibliothèque `mindmap`.

Il y a d'abord les styles d'ensemble concernant le mindmap dans son ensemble :

`mindmap` est le style de base convenant pour le format A4, `small mindmap`, `large mindmap`, et `huge mindmap` sont les adaptations du style `mindmap` aux formats respectifs A5, A3 et A2.

Le terme adaptation signifie ici que l'on peut produire un mindmap sans donner des dimensions (distances entre les nœuds, diamètres des nœuds, etc.) : le document produit sera lisible dans les conditions habituelles de lecture d'un document au format choisi. Pour les exemples du présent manuel, on utilisera toujours le style `manual mindmap` très voisin du style `small mindmap` mais un peu plus « élégant » (il est disponible en chargeant le fichier `manmind.tex`).

Il y a ensuite les styles de nœuds concernant les éléments du mindmap (qui sont des nœuds au sens de TikZ) :

`concept` est le style pour composer les nœuds,

`root concept` est le style pour composer le nœud racine : il contient le précédent et des options supplémentaires,

`extra concept` est le style pour composer des nœuds n'appartenant pas à l'arborescence mais pouvant être connectés à certains de ses nœuds par une liaison spécifique,

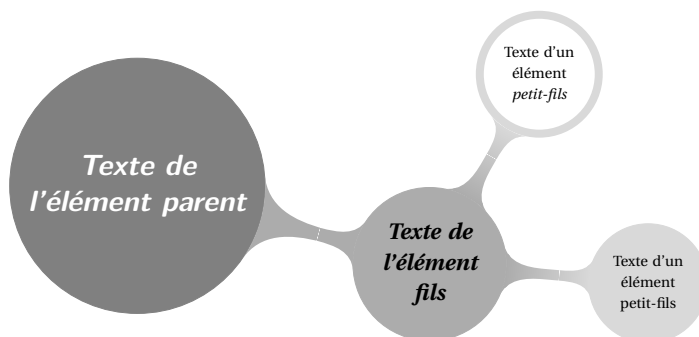
`level i concept`, $i = 1, 2, \dots$ contient des options supplémentaires spécifiques pour composer les nœuds de la génération de niveau i ,

`concept color=(couleur)` est la couleur de remplissage du nœud ; si l'on veut que le nœud ait une bordure et un remplissage avec des couleurs différentes, alors il faut donner l'épaisseur de la bordure et sa couleur sera celle donnée avec `concept color` ; ensuite le remplissage se fera comme pour les nœuds avec `fill=(couleur)` ; la particularité de syntaxe

pour la bordure provient du fait que sa couleur se prolonge dans la liaison : on ne doit pas écrire `draw=(couleur)`. Par contre, on peut utiliser toutes les autres options pour les nœuds, en particulier `text=(couleur)` pour colorer le texte.

Sans attendre, voici le premier exemple pour illustrer les manipulations des styles venant d'être présentées :

```
\begin{tikzpicture}[small mindmap,line width=0pt,
root concept/.style={concept,concept color=gray,text=white,
font=\large,\sffamily,\bfseries\itshape},
level 1 concept/.append style={concept,concept color=gray!65},
level 2 concept/.append style={every
child/.style={concept color=gray!30,font=\tiny}}}
\node[root concept]{Texte de\\l'élément\\parent}
child[grow=-15]{node[concept]{Texte de\\l'élément\\fils}
child[grow=-5]{node[concept]{Texte d'un élément petit-fils}}
child[grow=60]{node[concept, line width=3pt,fill=none,
outer sep=1.5pt]{Texte d'un élément \textit{petit-fils}}}};
\end{tikzpicture}
```



On reprend ligne par ligne le code de l'exemple :

- 1) – pour des raisons d'encombrement et de « look », on choisit `manual mindmap`; l'autre option est expliquée à la ligne 10;
- 2) – le style `root concept` est le style `concept` avec en plus un choix de couleur pour le fond et le texte;
- 3) – on ajoute un choix de couleur pour le fond des fils;
- 4) et 5) – on ajoute encore un choix de couleur pour le fond des petits-fils mais d'une façon différente telle que cette couleur soit placée en option de `child` et non de `node`; cela a pour conséquence que cette couleur

se fond dans la couleur du nœud parent le long de la liaison (comparer avec la génération précédente) ;

6) à 9) – on pose les nœuds en donnant leur position angulaire en option de la commande `grow` ;

10) – on limite le fond à une bordure, le centre reste non coloré ; les liaisons sont calculées pour s’appliquer sur l’axe de la bordure : pour des bordures épaisses la liaison semble s’enfoncer dans le nœud : il faut donc donner à `outer sep` la moitié de l’épaisseur de la bordure du nœud ; cet effet se retrouve dans toutes les situations où il y a des nœuds avec des bordures épaisses et des liaisons qui pointent vers ces nœuds : l’imperfection est décelable sur ce mindmap avec l’épaisseur des lignes par défaut (0.4pt), d’où l’annulation de cette épaisseur ; si on enlève l’annulation, l’imperfection sera particulièrement visible au départ de la liaison du nœud racine vers le nœud fils.

Pour illustrer le rôle des styles d’ensemble on considère le style `manual mindmap` utilisé pour l’exemple, on prend le cas de la deuxième génération : sa composition est gouvernée par défaut par le style `level 2 concept` qui contient (voir le fichier `manmind.tex`) :

```
minimum size=1.48cm,level distance=2.9cm,text width=1.275cm,  
sibling angle=60,font=\footnotesize
```

On vérifie facilement sur la figure que le diamètre du nœud petit-fils gris a bien un diamètre de 15 mm et que sa distance au fils est de 29 mm. Mais on peut recouvrir toutes ces données en ajoutant d’autres données.

On va examiner les trois possibilités de personnalisation des mindmaps (la seconde est destinée à des modifications exceptionnelles) :

— Pour le nœud racine on a modifié la taille, la graisse, la famille, la forme et la couleur du texte en définissant le style `root concept`. Pour les nœuds petits-fils, on a modifié la taille du texte en modifiant le style `level 2 concept`. On a aussi modifié le style `level 1 concept` pour donner une couleur aux nœuds fils d’une manière spéciale.

— Pour mettre un mot en italique dans le nœud petit-fils avec fond blanc, on a utilisé dans le texte la macro \LaTeX habituelle. Il faut noter qu’un changement de taille placé en avant du texte ne change pas l’interligne ; il doit être placé dans le style de niveau correspondant.

— Si on prévoit de faire un assez grand nombre de mindmaps avec une présentation spécifique, il faut intervenir au niveau du style d’ensemble, c’est-à-dire faire un fichier du type `manmind.tex` comportant les options

voulues : dimensions, fontes, couleurs, etc. (cf. fichier `manmind.tex`).

On termine par un exemple plus important dans le but de montrer de nombreuses variantes possibles pas forcément artistiquement compatibles entre elles. C'est à l'utilisateur de choisir les textes, la structure et les couleurs, en fonction du contenu et des lecteurs potentiels.

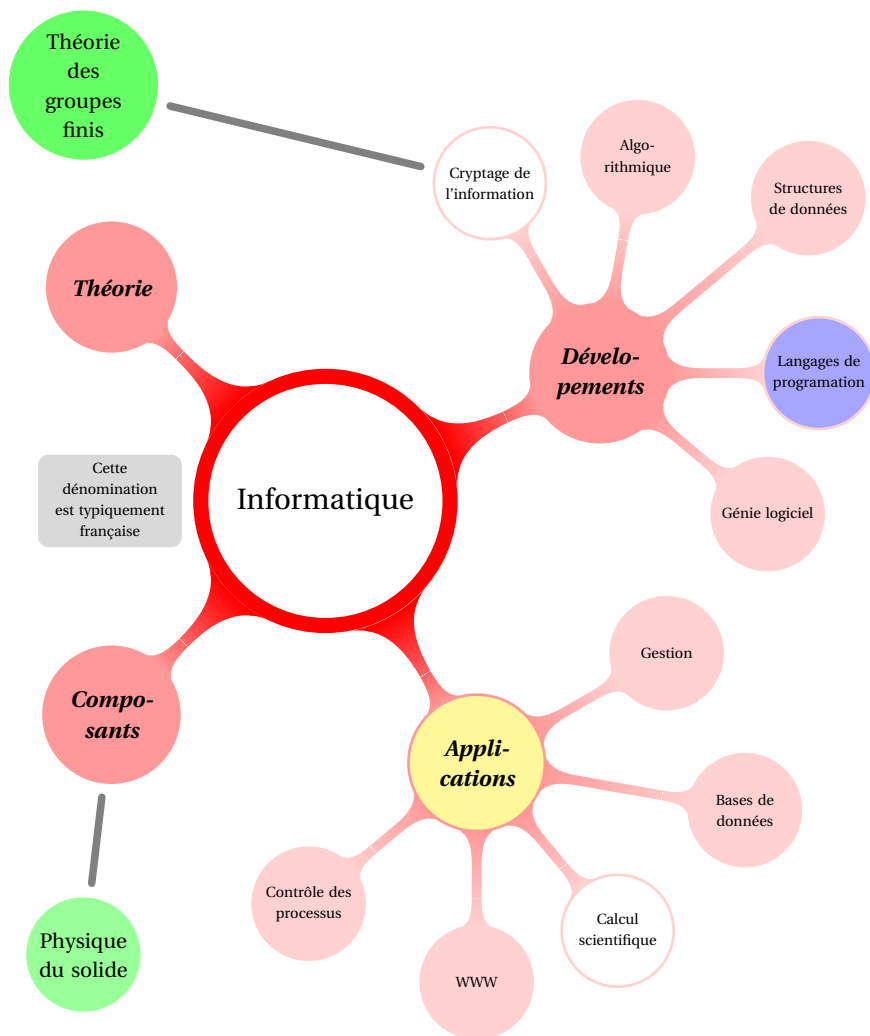
Cet exemple contient deux éléments (option `extra concept`) concernant des disciplines en relation avec certaines disciplines du mindmap (option `concept connection`) et une note interne (option `annotation`) contenant une remarque. On peut modifier la longueur des liaisons vers un même niveau : on peut imaginer que les liaisons d'un élément vers ses fils soient alternativement de longueurs différentes. Il suffit de se rappeler que les options concernant l'intérieur et le contenu du nœud doivent être placées en option de la commande `node` et celles concernant la couleur principale du nœud ainsi que la liaison doivent être placées en option de la commande `child`.

```
\begin{tikzpicture}[manual mindmap,line width=0pt,concept color=red,
root concept/.style={concept,fill=none,outer sep=1mm,line width=2mm},
level 1 concept/.append style={every child/.style={concept color=red!40,
line width=1pt,draw=red!40}},
level 2 concept/.append style={every child/.style={concept color=red!18,
font=\tiny}}]

\node[root concept](n5){\Large Informatique}
  child[grow=135]{node[concept]{Théorie}}
  child[grow=-135]{node[concept](n1){Compo\sants}}
  child[grow=-60]{node[concept,fill=yellow!50]{Appli\cations}
  child[grow=30]{node[concept]{Gestion}}
  child[grow=-10,level distance=36mm]{node[concept]{Bases de données}}
  child[grow=-50]{node[concept,fill=none]{Calcul scientifique}}
  child[grow=-90]{node[concept]{WWW}}
  child[grow=-140]{node[concept]{Contrôle des processus}}}
  child[grow=25]{node[concept]{Dévelo\pements}
  child[grow=120]{node[concept,fill=none](n3){Cryptage de l'information}}
  child[grow=80]{node[concept]{Algo\rithmique }}
  child[grow=40,level distance=36mm]{node[concept]{Structures de données}}
  child[grow=0]{node[concept,fill=blue!35]{Langages de programmation}}
  child[grow=-40]{node[concept]{Génie logiciel}}};

\node[extra concept,concept color=green!60](n2)
  at(-32mm,-60mm){Physique du solide};
\node[extra concept,concept color=green!40](n4)
  at(-32mm,55mm){Théorie des groupes finis};

\draw[concept connection](n1)edge(n2);
\draw[concept connection](n3)edge(n4);
\node[annotation,left,fill=gray!30,text width=17mm,text centered]
  at(n5.west){Cette dénomination\est typiquement\française};
\end{tikzpicture}
```



Un mindmap au contenu « improvisé ».

On note que la couleur du nœud racine est donnée en option de figure (et non dans le style `root concept`). Cette couleur s'applique à toutes les générations dont le style spécifique ne contient pas d'option de couleur.

CHAPITRE 9

Colorer et ombrer, opacité et transparence, remplissage, découpages et calques

Ce chapitre donne quelques compléments sur la couleur et détaille la façon de construire un « gradient » de couleurs dont un cas particulier est l'ombré par défaut utilisé à la première figure de la section 9 (passage continu du noir du haut de la surface ombrée au blanc du bas de cette surface). Il donne aussi l'essentiel concernant l'opacité et la transparence (masques de transparence). La suite est consacrée au remplissage de contours fermés plus complexes qu'un simple contour fermé comme celui de la figure citée ci-dessus. Le chapitre se termine en montrant les possibilités du découpage allié à l'utilisation des calques dont on expose la syntaxe de création et la méthode pour les utiliser.

32. COULEUR

Quelques éléments ont été donnés à la sous-section 10.8. On a vu que le chargement de l'extension `xcolor` met à disposition 17 couleurs (dont le blanc) dont on peut faire varier l'intensité avec la syntaxe :
(couleur)!xx où xx est un nombre entre 0 et 100.

La sous-section citée ci-dessus montre aussi comment on peut faire des mélanges de couleurs. Ces mélanges sont facile à faire du point de vue syntaxe mais il vaut mieux, pour disposer rapidement des couleurs désirées, utiliser les couleurs définies par les options de l'extension `xcolor` :

- `dvipsnames` charge un ensemble 68 couleurs (définies dans le standard `rgb`) qui sont les couleurs définies dans les drivers PostScript.
- `svgnames` charge un ensemble de 151 couleurs (définies dans le standard `cmymk`) selon la spécification SVG.

— `x11names` charge 317 couleurs (définies dans le standard `rgb`). Cet ensemble est largement suffisant pour le non professionnel ; il contient beaucoup de groupes de deux ou trois couleurs permettent des effets visuellement agréables : par exemple, la variante 1 à 20% pour le fond de page, la variante 2 à 50% pour colorer les boutons et la variante 3 pour le texte des boutons (il s'agit de boutons à cliquer de l'extension `hyperref`).

Pour gagner du temps, il est conseillé au lecteur de se reporter directement aux pages 38 à 40 de la documentation de `xcolor` qui affichent les couleurs chargées par les options citées ci-dessus.

33. OMBRÉ ET OMBRE PORTÉE

33.1. OMBRÉ PROPREMENT DIT

On a déjà obtenu un ombré, à la section 9, avec l'option :

`shade`

En fait, cet exemple est un cas particulier d'une coloration dont la couleur varie continuellement entre deux couleurs (noir et blanc pour cet exemple), en passant éventuellement par une troisième couleur.

L'ombré est d'abord caractérisé par un aspect géométrique : les lignes de même couleur sont des droites parallèles ou des cercles concentriques ; un troisième aspect simule une boule éclairée par une source lumineuse située en haut et à gauche. Il y a d'autres possibilités intéressantes, mais plutôt orientées vers le dessin d'art [46]. Les aspects géométriques présentés ci-dessus sont choisis à l'aide de l'option [15.6] [46] : `shading=axis`, `radial` ou `ball` (défaut `axis`).

Pour le premier choix, l'angle que font les lignes d'égale couleur avec l'axe des x peut être choisi grâce à l'option [15.6] :

`shading angle=xx` où `xx` est compris entre 0 et 90° (défaut=0).

Les angles différents de 0 et 90° ne peuvent être choisis que dans le cas des couleurs par défaut (noir et blanc), voir l'exemple suivant.

On peut choisir la couleur du haut ou de gauche avec les options [46] : `top color=(couleur)` ou `left color=(couleur)` (défaut noir)

et la couleur du bas ou de droite avec les options :

`bottom color=(couleur)` ou `right color=(couleur)` (défaut blanc)

suivant la direction choisie (0 et 90° seulement). La coloration varie régulièrement de la couleur du haut (de gauche) vers la couleur du bas (de droite), sauf si on impose le passage par une troisième couleur grâce à l'option :

`middle color=(couleur)`

qui doit être donnée après les deux autres options de couleur.

Le deuxième choix n'a que deux options de couleurs [46] :

`inner color=(couleur)` couleur du centre (défaut noir),

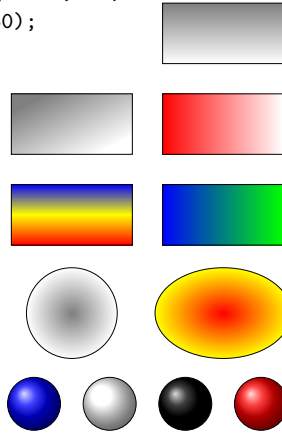
`outer color=(couleur)` couleur de la périphérie (défaut blanc).

Le troisième choix n'a qu'une option de couleur [46] :

`ball color=(couleur)` (défaut bleu, blanc et noir possibles).

On constate sur l'exemple ci-dessous que l'option `shade`, déjà utilisée à la section 9, choisit les options `shading=axis`, `shading angle=0` et `top color=black` par défaut.

```
\begin{tikzpicture}[x=1mm,y=1mm,very thin]
\draw[shade](22,54)rectangle(38,62);%cas de la sect. 9
\draw[shade,shading angle=45](2,42)rectangle(18,50);
\draw[left color=red](22,42)rectangle(38,50);
\draw[top color=blue,bottom color=red,
middle color=yellow](2,30)
rectangle(18,38);
\draw[left color=blue,right color=red]
(22,30)rectangle(38,38);
\draw[shading=radial](10,21)circle(6);
\draw[inner color=red,outer color
=yellow](30,21)ellipse(9 and 6);
\draw[shading=ball](5,9)circle(3.5);
\draw[ball color=white](15,9)
circle(3.5);
\draw[ball color=black](25,9)
circle(3.5);
\draw[ball color=red](35,9)circle(3.5);
\end{tikzpicture}
```



On constate encore qu'une option seule choisit les valeurs par défaut des options qu'elle nécessite : par exemple, `top color=red` active automatiquement `shading=axis` et `shading angle=0`, ce qui simplifie la saisie. De même, `inner color=red` active `shading=radial` et `ball color=red` active `shading=ball`.

33.2. OMBRE PORTÉE

Pour créer l'ombre portée d'un objet graphique on dispose de l'option [47.3] :

`drop shadow`

qui prend en compte les options suivantes :

```
shadow scale=1
shadow xshift=0.5ex
shadow yshift=0.5ex
opacity=50
fill=black!50
every shadow
```

Logiquement, on garde la valeur 1 pour la première option ; les deuxième et troisième options commandent la direction et la longueur de l'ombre portée ; les quatrième et cinquième option fixent l'opacité de l'ombre portée (pour ne pas cacher complètement le voisinage immédiat de l'objet, voir la section suivante pour la syntaxe correspondante) et sa couleur (y compris son intensité).

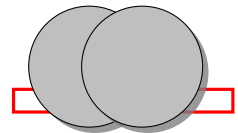
Ces options pourront être modifiées avec la définition :

```
every shadow/.style={options modifiées}
```

placée en option de figure ou dans une commande `\tikzset` et qui sera exécutée en tant que sixième option de `drop shadow` (cf. ci-dessus). On donne un exemple.

```
\begin{tikzpicture}[x=1mm,y=1mm, every
  shadow/.style={opacity=0.8,fill=gray}]
\node[drop shadow,fill=lightgray!40,
  font=\bfseries]at(16.5,21)
  {Essai d'ombre portée};
\draw[very thick,red](3,2)rectangle(32,5);
\draw[fill=lightgray,drop shadow]
  (13,8)circle(8);
\draw[fill=lightgray,drop shadow]
  (20,8)circle(8);
\end{tikzpicture}
```

Essai d'ombre portée



33.3. OMBRE-RÉPÉTITION

On dispose d'une autre possibilité pour mettre en valeur un objet graphique : dit en quelques mots, il s'agit de créer une (plusieurs) ombre(s) portée(s) constituée(s) d'une copie de l'objet. La méthode proposée est strictement semblable à celle créant l'ombre portée : on dispose de l'option [47.3] :

```
copy shadow
```


qui prend en compte les options suivantes :

```
shadow scale=1
shadow xshift=0.5ex
shadow yshift=0.5ex
every shadow
```

Ces options peuvent se modifier comme dans le cas de l'ombre portée (cf. ci-dessus).

L'option `double copy shadow` répète une deuxième copie. On dispose aussi d'autres artifices plutôt destinés au dessin d'art [47.4]. Voici un exemple de double répétition : les ombres et les copies ne sont pas incluses dans la bounding box créée par TikZ : il faut donc créer le point N, déplacé du point (`current bounding box.north east`) par $(1ex, 1ex)$ pour que les copies soient incluses dans la bounding box.

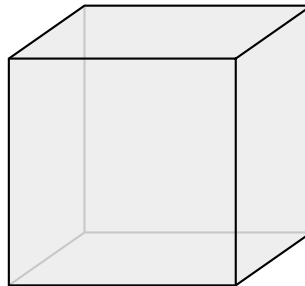
```
\begin{tikzpicture}[x=1mm,y=1mm]
\node[double copy shadow,draw,fill=lightgray!40,font=\bfseries]
{Essai de répétition};
\node[coordinate,shift={(1ex,1ex)}] (N) at
(current bounding box.north east){}; \path(N);
\end{tikzpicture}
```



34. OPACITÉ ET MASQUE DE TRANSPARENCE

L'opacité est une propriété très importante surtout pour des travaux à but pédagogique : elle permet de montrer en intensité affaiblie des parties cachées d'une figure en trois dimensions que l'on veut représenter. Voici un exemple typique :

```
\begin{tikzpicture}[x=1mm,y=1mm,thick]
\draw(0,0)--(30,0);
\draw(0,0)--(0,30);
\draw(0,0)--(-10,-7);
\filldraw[fill=lightgray!30,
fill opacity=0.85](-10,-7)--(20,-7)--
(30,0)--(30,30)--(0,30)--(-10,23)--
cycle;
\draw(-10,23)--(20,23)--(20,-7);
\draw(20,23)--(30,30);
\end{tikzpicture}
```



Ce choix ne signifie pas que cette propriété d'opacité n'a pas d'intérêt en deux dimensions. Grâce aux calques et à un bon choix d'opacité, on peut montrer, par exemple, un élément de figure ainsi que sa sous-structure correspondante.

L'opacité est déterminée par l'option [20.2] :
`opacity=xx` (où `xx` est un nombre entre 0 et 1).
 Le nombre 0 donne la transparence totale c'est-à-dire l'opacité nulle!
 Cette option d'opacité est valable pour toutes les opérations de la commande où on l'a donnée : il faut donc être attentif; avec `\draw` et `\fill`, il n'y a pas d'ambiguïté, mais avec `\filldraw`, il faut utiliser l'option d'opacité sous la forme :
`draw opacity=xx` ou `fill opacity=xx`
 suivant que l'on veut que ce soit le contour ou le fond qui ait la propriété d'opacité; dans l'exemple, l'opacité n'est appliquée qu'au trois faces visibles et non aux neuf arêtes visibles.

Dans ce qui suit, on va voir comment on définit un « masque de transparence » et comment on l'utilise [20.3]. Voici un exemple très simple.

On commence par faire le masque à l'aide de l'environnement :

```
tikzfadingfrompicture
```

Le masque est en fait un dessin fictif : son nom est défini en option de figure par l'option :

```
name=(nom-du-masque)
```

Les objets tracés et/ou remplis sont construits avec les syntaxes habituelles : la seule différence est que l'on doit colorer avec la couleur fictive qui sera choisie au moment de l'utilisation :

```
transparent!xx
```

où `xx` est un nombre entre 0 et 100,

(100 donne la transparence complète). Dans l'exemple, on a un rectangle transparent dans lequel on a placé une lettre transparente : on remarque que l'on peut faire varier la transparence horizontalement ou verticalement avec :

```
left color=transparent!xx
```

```
right color=transparent!xx
```

```
ou top color=transparent!xx|
```

```
bottom color=transparent!xx|
```

de la même façon que lors de la création d'un ombré avec deux couleurs.

En fin de code de la figure à « masquer », on écrit la ligne suivante :

```
\fill [path fading=(nom-du-masque), (couleur)] (chemin) ;
```

où (chemin) représente le code de construction du chemin entourant la surface à masquer. Sur l'exemple suivant, pour la première utilisation du masque, la surface à masquer est un rectangle de mêmes dimensions que le rectangle du masque. La deuxième utilisation montre que l'on peut masquer une surface de forme quelconque : on constate que les

dimensions du masque s'adaptent de manière à masquer le « mieux possible » la surface choisie (ici, on constate un changement d'échelle et un centrage du masque sur le centre de la surface à masquer si bien que la lettre T est présente en entier).

```
\begin{tikzfadingfrompicture}[x=1mm,y=1mm,name=masque]
\shade[left color=transparent!0,right color=transparent!100]
```

```
(0,0)rectangle(36,26);
```

```
\node[font=\Huge\bfseries,scale=2.5,
text=transparent!90]at(18,13){T};
```

```
\end{tikzfadingfrompicture}
```

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

```
\node[draw,rectangle,minimum width=36mm,
minimum height=26mm,text width=30mm,
text badly centered,font=\LARGE\bfseries]
at(18,13){Essai d'utilisation du masque};
```

```
\fill[path fading=masque,red]
```

```
(0,0)rectangle(36,26);
```

```
\node...idem décalé de 27mm vers le bas;
```

```
\fill[path fading=masque,red]
```

```
(10,-17)circle(10);
```

```
\end{tikzpicture}
```



L'utilisation des ces masques, qui semble plutôt destinée au dessin d'art, peut être utilisée avantageusement dans un but pédagogique.

35. REMPLISSAGE

On rappelle d'abord la notion de chemin fermé : si le chemin est constitué de plusieurs parties, il est nécessaire que les coordonnées de la fin d'une partie soient aussi les coordonnées du début de la suivante. Par exemple, le triangle défini avec :

```
(0,-30)--(20,-30)--(10,-40)--(0,-30)
```

peut être tracé et rempli, mais seulement tracé s'il est défini avec :

```
(0,-30)--(20,-30)(20,-30)--(10,-40)(10,-40)--(0,-30)
```

Dans ce dernier cas, on a en fait trois segments indépendants juxtaposés et non un triangle fermé (cf. discussion de la fin de la section 29).

L'action de remplissage a une propriété remarquable : si l'on a une courbe non fermée, extrémités (a, b) et (c, d), constituée d'une suite de segments et/ou de courbes telle que les coordonnées des extrémités des parties respectent la prescription ci-dessus, l'action de remplis-

sage trace en interne le segment entre les points (a,b) et (c,d) et remplit le chemin fermé ainsi constitué (c'est l'action de la commande `\pgfnclosepath` utilisée à la fin de la section 29).

Le remplissage est une opération très simple lorsque l'on a un domaine simplement connexe limité par un contour fermé, par exemple un disque limité par un cercle. Mais tout se complique dès que l'on a un domaine multiplement connexe (c'est à dire avec des « trous » comme on dit en langage de la rue) ou encore pire quand le domaine n'est pas connexe.

On prend l'exemple d'un disque possédant un trou circulaire ; on est tenté de dire : facile, ! on colore le disque en entier, ensuite on colore le petit disque intérieur en blanc ... ce qui ne serait satisfaisant que si la couleur `white` redonnait exactement la couleur du papier. On peut utiliser cette approche dans le cas où l'on travaille sur un fond coloré (le petit disque est alors coloré avec la couleur du fond).

La solution proposée par `TikZ` consiste à donner tous les contours (fermés bien entendu) à l'aide de commandes `\path` (ou `\draw` suivant que l'on ne veut pas ou que l'on veut tracer tous les contours) avec l'option `fill` suivie par le choix de la couleur de remplissage. `TikZ` détermine ensuite les points intérieurs qui vont être colorés suivant une méthode que l'on peut illustrer de la manière suivante [15.4.2] :

Soit un point donné A ; on considère un rayon Ax issu de ce point et allant à l'infini ; ce point est intérieur, et donc coloré, uniquement si le rayon Ax rencontre :

- un nombre impair de contours : option `even odd rule` qui est la plus simple à utiliser),
- autant de contour orientés dans un sens que dans l'autre : option par défaut `nonzero rule` qui est plus complète que la précédente mais plus exigeante puisque les sens de tracé des contours jouent un rôle fondamental.

La coloration pour les deux options en fonction du « niveau de profondeur » est résumée sur le tableau suivant (possible signifiant : sous réserve d'un choix adéquat des orientations des chemins fermés) :

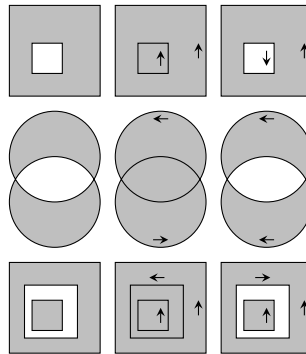
Niveau	<code>even odd rule</code>	<code>nonzero rule</code>
niveau 1	certaine	impossible
niveau 2	impossible	possible
niveau 3	certaine	impossible

Voilà quelques exemples traités, dans l'ordre even odd rule puis puis nonzero rule pour deux ensembles d'orientations différents.

```

\begin{tikzpicture}[x=1mm,y=1mm]
\def\vh{#1}{\draw[>stealth,->] (#1)--+(0,2mm);}
\def\vb{#1}{\draw[>stealth,->] (#1)--+(0,-2mm);}
\def\vg{#1}{\draw[>stealth,->] (#1)--+(-2mm,0);}
\def\vd{#1}{\draw[>stealth,->] (#1)--+(2mm,0);}
\path[draw,fill=lightgray,even odd rule]
(0,0)--(12,0)--(12,12)--(0,12)--cycle (3,3)--(7,3)--(7,7)--(3,7)--cycle;
\path[draw,fill=lightgray,shift={(14,0)}]
(0,0)--(12,0)--(12,12)--(0,12)--cycle (3,3)--(7,3)--(7,7)--(3,7)--cycle;
\vh(20,4)\vh(25,5)
\path[draw,fill=lightgray,shift={(28,0)}]
(0,0)--(12,0)--(12,12)--(0,12)--cycle (3,3)--(3,7)--(7,7)--(7,3)--cycle;
\vb(34,6)\vh(39,5)
\path[draw,fill=lightgray,even odd rule]
(6,-8)circle(6)(6,-14)circle(6);
\path[draw,fill=lightgray,shift={(14,0)}]
(6,-8)circle(6)(6,-14)circle(6);
\vg(21,-3)\vd(19,-19)
\path[draw,fill=lightgray,shift={(28,0)},->]
(6,-8)circle(6)(12,-14)arc(360:0:6);
\vg(35,-3)\vg(35,-19)
\path[draw,fill=lightgray,even odd rule,
shift={(0,-34)}] (0,0)--(12,0)--(12,12)--(0,12)
--cycle (3,3)--(7,3)--(7,7)--(3,7)--cycle
(2,2)--(9,2)--(9,9)--(2,9)--cycle;
\path[draw,fill=lightgray,shift={(14,-34)}]
(0,0)--(12,0)--(12,12)--(0,12)--cycle
(3,3)--(7,3)--(7,7)--(3,7)--cycle (2,2)--(9,2)--(9,9)--(2,9)--cycle;
\vh(20,-30)\vh(25,-29)\vg(20.5,-24)
\path[draw,fill=lightgray,shift={(28,-34)}] (0,0)--(12,0)--(12,12)--(0,12)--cycle
(3,3)--(7,3)--(7,7)--(3,7)--cycle (2,2)--(2,9)--(9,9)--(9,2)--cycle;
\vh(34,-30)\vh(39,-29)\vd(32.5,-24)
\end{tikzpicture}

```



On constate que l'on est limité; par exemple, on ne peut colorer le domaine commun aux deux cercles du deuxième cas de l'exemple. Ce sera possible avec d'autres moyens que l'on va examiner.

36. DÉCOUPAGES ET CALQUES

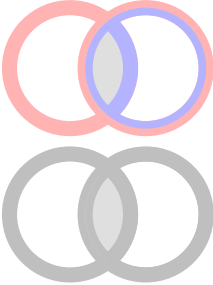
On rappelle que l'option `clip` (cf. sect. 9) affectée à un chemin fermé découpe tous les tracés suivants et ne conserve de ces tracés que la partie intérieure au chemin fermé. Cette possibilité est assez limitée; certes, elle est utile pour réduire une figure déjà faite à une partie de cette figure, mais on ne peut guère faire davantage sans utiliser un outil supplémentaire que l'on va aborder.

Prenons le cas des deux cercles sécants examiné ci-dessus dont on voudrait uniquement colorer la partie commune. On colore le cercle de gauche et on découpe avec le cercle de droite : on obtient donc la partie commune colorée mais on ne peut pas tracer « normalement » les cercles : il faudrait pouvoir arrêter l'effet du découpage, ce sera le rôle des calques. Dans la première partie de l'exemple, les deux cercles ont été tracés avant découpage en rouge et après découpage en bleu ; l'interprétation du résultat est laissée au lecteur ; la deuxième partie montre que l'on peut tout de même obtenir le bon résultat si les cercles sont tracés avec la même couleur (éventuellement à quelques imperfections près).

```

\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=2mm,red!30](8,0)circle(8)(18,0)circle(8);
\path[clip](18,0)circle(8);
\fill[lightgray!50](8,0)circle(8);
\draw[line width=2mm,blue!30]
(8,0)circle(8)(18,0)circle(8);
\end{tikzpicture}\vskip1mm
\begin{tikzpicture}[x=1mm,y=1mm]
\draw[line width=2mm,lightgray]
(8,-18)circle(8)(18,-18)circle(8);
\path[clip](18,-18)circle(8);
\fill[lightgray!50](8,-18)circle(8);
\draw[line width=2mm,lightgray](8,-18)circle(8)(18,-18)circle(8);
\end{tikzpicture}

```



Pour résoudre « rationnellement » le problème rencontré ci-dessus, on va utiliser les calques (*layers* en anglais).

Les dessinateurs du bâtiment ont utilisé pendant longtemps les calques ; pour une maison par exemple, il y avait un plan de masse avec les murs, portes et fenêtres ; ensuite il pouvait y avoir un calque pour le sanitaire (tuyaux, éviers, baignoires, etc.), un calque pour le chauffage (tuyaux, chaudière, radiateurs, etc.), un calque pour l'appareillage électrique (câbles, etc.)... Cela évitait d'avoir des plans surchargés : en reproduisant le premier calque posé sur le plan de masse on obtenait le plan pour les plombiers et ainsi de suite. Cette méthode est toujours utilisée par les outils modernes de DAO. TikZ peut aussi faire des calques, chaque calque est un dessin indépendant et c'est la superposition de ces dessins qui donne le résultat voulu. On expose la syntaxe de la méthode des calques avec l'exemple précédent : coloration de disques sécants et

de leur intersection [82].

On doit, avant d'ouvrir l'environnement `tikzpicture`, déclarer chaque calque (excepté le calque `main` déclaré par défaut) avec la commande

```
\pgfdeclarelayer{(nom du calque)}
```

et donner la liste des calques à utiliser pour la figure avec la commande

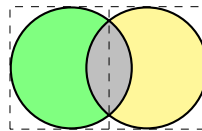
```
\pgfsetlayers{(liste des calques)}
```

Chaque calque est constitué par des commandes situées entre

```
\begin{pgfonlayer}{(nom du calque)} et \end{pgfonlayer}
```

Il constitue un objet graphique complètement indépendant. Toutes les commandes situées en dehors des environnements `pgfonlayer` constituent par défaut le calque nommé `main`.

```
\pgfdeclarelayer{gauche}\pgfdeclarelayer{droite}%
\pgfdeclarelayer{intersection}%
\pgfsetlayers{gauche,droite,intersection,main}%
\begin{tikzpicture}[x=1mm,y=1mm]
\begin{pgfonlayer}{gauche}\path[clip](-0.5,-8.1)rectangle(13,8.1);
\path[fill=green!50,even odd rule](8,0)circle(8)(18,0)circle(8);
\end{pgfonlayer}
\begin{pgfonlayer}{droite}
\path[clip](13,-8.1)rectangle(26.1,8.1);
\path[fill=yellow!50,even odd rule]
(8,0)circle(8)(18,0)circle(8);\end{pgfonlayer}
\begin{pgfonlayer}{intersection}\path[clip](18,0)circle(8);
\fill[gray!50](8,0)circle(8);\end{pgfonlayer}
\draw[thick](8,0)circle(8)(18,0)circle(8);
\draw[very thin,dashed](-0.1,-8.1)rectangle(26.1,8.1);
\draw[very thin,dashed](13,-8.1)--(13,8.1);
\end{tikzpicture}
```



Voici les actions des calques ; soient G et D les disques de gauche et de droite, soient $G \cup D$ et $G \cap D$ leur union et leur intersection et notons par \setminus le moins ensembliste :

- le calque gauche découpe $G \setminus (G \cap D)$ (partie de G colorée en vert) dans $(G \cup D) \setminus (G \cap D)$ (cf. l'exemple précédent) coloré en vert (le découpage utilise le rectangle de gauche) ;

- le calque droit découpe $D \setminus (G \cap D)$ (partie de D colorée en jaune) dans $(G \cup D) \setminus (G \cap D)$ (cf. l'exemple précédent) coloré en jaune (le découpage utilise le rectangle de droite) ;

— le calque intersection découpe ($G \cap D$) dans G , coloré en gris, avec le cercle D ;

— le calque main trace les contours des disques en noir (couleur par défaut) ainsi que les deux rectangles de découpage pour faciliter l'interprétation du code.

On note que les commandes de déclaration et d'utilisation des calques sont situées avant l'environnement `tikzpicture` et introduisent donc un espace si elles ne sont pas terminées par un `%`.

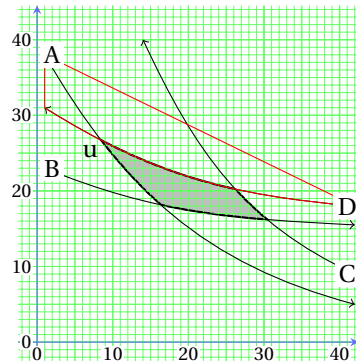
37. DÉCOUPAGES ET CALQUES : REPRISE DES EXEMPLES

On reprend le tracé du « cycle de Carnot » traité à la section 29 dans le cadre des intersections de courbes. La coloration de la surface découpée et le tracé de son contour est incontestablement beaucoup plus court et plus facile à mettre en œuvre avec la méthode des découpages et des calques ; mais on verra que cette méthode a aussi des inconvénients.

```

\pgfdeclarelayer{aire}\pgfdeclarelayer{limiteshor}%
\pgfdeclarelayer{limitesver}%
\pgfsetlayers{aire,limiteshor,limitesver,main}%
\begin{tikzpicture}[x=1mm,y=1mm]
\ppmm(-2.5mm,-2.5mm)(42.5mm,44.5mm)
\def\pA{(1,38)to[out=-60,in=165](42,5)}
\def\pB{(1,23)to[out=-22,in=177](42,15.5)}
\def\pC{(42,9)to[out=155,in=-65](14,40)}
\def\pD{(42,18)to[out=175,in=-32](1,31)}
\draw[thin,->]\pA;\draw[thin,->]\pB;
\draw[thin,->]\pC;\draw[thin,->]\pD;
\posb(2,38){A}\posb(2,23){B}
\posb(41,9){C}\posb(41,18){D}
\begin{pgfonlayer}{aire}
\path[clip]\pB--\pD--cycle;
\fill[lightgray]\pA--\pC--cycle;
\end{pgfonlayer}
\begin{pgfonlayer}{limitesver}
\path[clip]\pB--\pD--cycle;
\draw[thick]\pA\pC;
\end{pgfonlayer}
\begin{pgfonlayer}{limiteshor}
\path[clip]\pA--\pC--cycle;
\draw[thick]\pB\pD;
\end{tikzpicture}

```



On décrit brièvement le code correspondant :

- on commence par définir les quatre courbes et les tracer ;
- le calque `aire découpe`, selon le quadrilatère construit avec les courbes B et D, la surface concernée dans le quadrilatère construit avec les courbes A et C et coloré ;
- le calque `limitesver découpe`, selon le quadrilatère construit avec les courbes B et D, les arcs de courbe « verticaux » limitant la surface concernée dans les courbes A et C tracées avec l’option `thick` ;
- le calque `limiteshor découpe`, selon le quadrilatère construit avec les courbes A et C, les arcs de courbe « horizontaux » limitant la surface concernée dans les courbes B et D tracées avec l’option `thick` ;
- en rouge, un chemin pour colorer la partie Au de la courbe A.

Sur l’exemple de la section 28, on va constater que la méthode des découpages et calques n’est pas toujours aussi facile à mettre en œuvre.

La première figure de l’exemple montre les chemins fermés nécessaires pour réaliser exactement la figure de la section 28 : les deux courbes limitant la « poche » supérieure sont tracées en trait plus épais tandis que la surface de la « poche » inférieure est simplement colorée. Ces chemins (dont le code de construction peut être reconstruit facilement par le lecteur) sont :

- `\crouge` : courbe A plus trait plein rouge,
- `\crouged` : courbe B plus traitillé rouge,
- `\cbleu` : courbe A plus trait plein bleu,
- `\cbleud` : courbe B plus traitillé bleu,
- `(0,0)rectangle(30,25)` et `(0,20)rectangle(30,40)` : rectangles de découpe permettant d’éliminer la partie non désirée de la découpe immédiatement suivante (le lecteur pourra pourcenter une découpe suivant un de ces rectangles et vérifier que la découpe suivante contient deux parties dont l’une est indésirable).

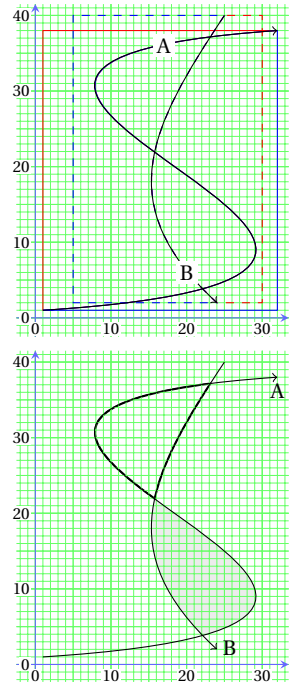
Voici, dans l’ordre, les calques et leur contenu :

- haut : découpe, avec le contour `\cbleud` (plus un rectangle), la courbe `\pA` épaissie pour donner la frontière gauche de la poche supérieure ;
- bas : découpe, avec `\crouged` (plus un rectangle), la surface entourée par `\crouge` et colorée et donne la coloration de la poche inférieure ;
- hautt : découpe, avec `\cbleu` (plus un rectangle), la courbe `\pB` épaissie et donne la frontière droite de la poche supérieure.

```

\pgfdeclarelayer{haut}\pgfdeclarelayer{bas}%
\pgfdeclarelayer{hautt}%
\pgfsetlayers{haut,hautt,bas,main}%
\begin{tikzpicture}[x=1mm,y=1mm]\footnotesize
\ppmm(-2.5mm,-2.5mm)(33.5mm,41.5mm)
\def\pA{(1,1)..controls(80,6)
(-40,33)..(32,38)}
\def\pB{(25,40)..controls(15,25)
and(10,15)..(24,2)}
\def\cbleu{\pA--(32,1)--cycle}
\def\crouge{\pA--(1,38)--cycle}
\def\cbleud{\pB--(5,2)--(5,40)--cycle}
\def\crouged{\pB--(30,2)--(30,40)--cycle}
\pose(32,38){-90}{A}\pose(24,2.3){0}{B}
\begin{pgfonlayer}{haut}
\path[clip](0,20)rectangle(30,40);
\path[clip]\cbleud;\path[draw,thick]\pA;
\end{pgfonlayer}
\begin{pgfonlayer}{bas}
\path[clip](0,0)rectangle(30,25);
\path[clip]\crouged;
\path[fill=lightgray!30]\crouge;
\end{pgfonlayer}
\begin{pgfonlayer}{hautt}
\path[clip](0,20)rectangle(30,40);
\path[clip]\cbleu;\draw[thick]\pB;
\end{pgfonlayer}
\draw[thin,->]\pA;\draw[->]\pB;
\end{tikzpicture}

```



On laisse au lecteur le soin de réfléchir aux complications supplémentaires nécessaires si les limites des poches étaient en trait épais et coloré en une autre couleur que le noir (le lecteur peut simplement prendre les options [red, ultra thick] au lieu de [thick] pour le tracé du dernier calque pour découvrir que les extrémités des courbes A et B devait être aussi tracées par des calques supplémentaires (cf. la coloration du segment de courbe Au de l'exemple précédent).

Finalement, on constate que le choix de la méthode la plus facile à mettre en œuvre (méthodes des sous courbes ou des découpages et des calques) dépend beaucoup du type des figures concernées ... et ce n'est qu'avec un peu d'expérience que l'on peut faire ce choix valablement.

Principaux outils disponibles

Ce dernier chapitre est consacré à la présentation de quelques outils dont les principaux ont été parfois utilisés dans les exemples, notamment les transformations et les boucles de répétition. Il se termine par un court exposé de deux outils dont on pourrait se passer mais pour lesquels certains découvriront probablement une judicieuse utilisation.

38. TRANSFORMATIONS

On a déjà utilisé certaines des translations pour déplacer des parties de figures, par exemple des parties presque identiques mais dont on veut mettre clairement en évidence leur différence. On donne la liste des transformations avec leur définition [22.3].

On commence par les translations, les changement d'échelle et les « penchés » ; on donne la syntaxe suivie par les coordonnées avant transformation puis les coordonnées après transformation :

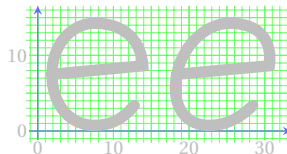
- `xshift=a` : x, y ; $x + a, y$; translation suivant l'axe des x .
- `yshift=b` : x, y ; $x, y + b$; translation suivant l'axe des y .
- `shift={ (a,b) }` : x, y ; $x + a, y + b$; translation de vecteur de composantes (a, b) .
- `xscale=k` : x, y ; kx, y ; dilatation suivant l'axe des x ;
- `yscale=k` : x, y ; x, ky ; dilatation suivant l'axe des y ;
- `scale=k` : x, y ; kx, ky ; dilatation isotrope autour de l'origine ;
- `scale around={k: (a,b) }` : x, y ; $a + k(x - a), b = k(y - b)$; dilatation isotrope autour du point (a, b) ;
- `xslant=k` : x, y ; $x + ky, y$; « penché » suivant l'axe de x ;
- `yslant=k` : x, y ; $x, y + kx$; « penché » suivant l'axe de y ;

Le penché est utilisée dans les dessins des caractères des fontes italiques et penchées. Voilà un exemple avec la lettre e tracée au dernier exemple de la section 19 :

```

\begin{tikzpicture}[x=0.5mm,y=0.5mm,
  line width=1.5mm,lightgray,cap=round]
\def\letrree{(25.5,6.75)..controls+(-130:4.5)and+(0:4.5)..(15,1.5)
..controls+(180:7)and+(-90:7.5)..(3.75,15)..controls+(90:8)and
+(180:6)..(15,28.5)..controls+(0:9)and+(93:5.5)..(27.75,17.25)
--(3.75,15);}
\draw\letrree
\begin{scope}[xshift=15mm,xslant=0.2]
\draw\letrree
\end{scope}
\end{tikzpicture}

```



Voici enfin les rotations :

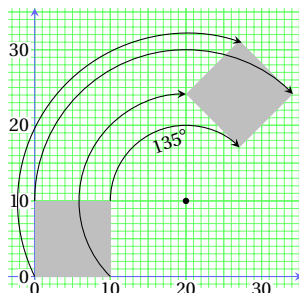
- `rotate=u` : rotation de u° autour de l'origine.
- `rotate around={u:(a,b)}` : rotation de u° autour du point de coordonnées $x = a$ et $y = b$.

La figure suivante illustre la rotation autour d'un point donné (un bon exemple pour tracer des arcs de cercle) :

```

\begin{tikzpicture}[x=1mm,y=1mm]
\fill[lightgray](0,0)rectangle(10,10);
\fill[rotate around={-135:(20,10)},
lightgray](0,0)rectangle(10,10);
\draw[thin,->,>=stealth](0,0)
arc(206:71:22.3607);
\draw[thin,->,>=stealth]
(0,10)arc(180:45:20);
\draw[thin,->,>=stealth]
(10,10)arc(180:45:10);
\draw[thin,->,>=stealth](10,0)
arc(225:90:14.1421);\filldraw(20,10)circle(0.35);
\node at (18,18)[rotate=202]{\sc135^\circ};
\end{tikzpicture}

```



On dispose aussi de la transformation linéaire la plus générale ainsi que d'une option qui ramène la transformation courante à la transformation unité.

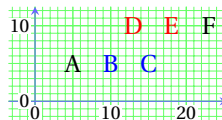
- `cm={a,b,c,d,(e,f)}` ; effectue la transformation :

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} \times \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} e \\ f \end{vmatrix}$$

- `reset cm` annule l'effet de la transformation courante ; « à ne pas utiliser sauf si l'on sait vraiment ce que l'on fait ! » dit la documentation.

La figure suivante illustre le comportement des transformations par rapport à celui des options. On trace la lettre A au point de coordonnées (5,5). Ensuite, on ouvre un environnement `scope` avec une translation de 5 mm suivant l'axe des x et le choix de la couleur bleu ; on trace B en (5,5) et C en (5,5) avec une translation de 5 mm suivant les x ; ils apparaissent en bleu respectivement en (10,5) et (15,5). A l'intérieur de ce `scope`, on ouvre encore un `scope` avec une translation de 3 mm suivant les x et 5 mm suivant les y et le choix de la couleur rouge : la transformation courante est alors une translation de 8 mm suivant l'axe des x et 5 mm suivant l'axe des y et la couleur courante est le rouge comme le montre les tracés de D et E. On constate alors que les translations successives « s'ajoutent » (se composent) alors que les choix de couleur se « remplacent » successivement. On ouvre enfin, à l'intérieur du deuxième `scope`, un autre `scope` avec l'option `reset cm` qui change la transformation courante en la transformation unité comme on peut le constater en traçant F en (23,10) qui apparaît bien en noir en (23,10) :

```
\begin{tikzpicture}[x=1mm,y=1mm]
\pos(5,5){A}
\begin{scope}[xshift=5mm]\pos(5,5){B}
\begin{scope}[xshift=5mm]at(5,5){C};
\begin{scope}[xshift=5mm,yshift=5mm]
\pos(5,5){D}\node[xshift=5mm]at(5,5){E};
\begin{scope}[reset cm]\node at(25,10){F};
\end{scope}\end{scope}\end{scope}
\end{tikzpicture}
```



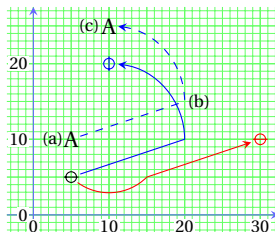
Dans l'exemple suivant, on montre la composition d'une translation et d'une rotation, ainsi que la composition dans l'ordre inverse, appliquées à un tracé initialement placé en un point quelconque (tracé noir). Le tracé déduit par la première transformation (tracé bleu) est tourné ; l'ajout d'une rotation initiale inverse autour de l'origine du tracé initial donne un tracé non tourné (tracé rouge).

Cette composition, possible pour les tracés comme vu ci-dessus, n'est pas possible pour les points ni pour les nœuds ; sur la figure, pour passer du point initial (a) au point transformé (c) on procède par étapes : (a)→(b) puis (b)→(c) ; le point (c) est alors disponible pour continuer, placer un nœud par exemple. Dans ce cas, on remarque la nécessité de donner le centre de rotation sous la forme (0) et non avec la forme (10, 15) (cette nécessité n'est pas mentionnée dans la documentation).

```

\begin{tikzpicture}[x=1mm,y=1mm]
\def\omoins{(5,5)circle(2pt)--(6,5)(5,5)--(4,5)}
\draw\omoins;
\draw[rotate around={90:(10,10)},shift={(15,5)},blue]\omoins;
\draw[shift={(15,5)},rotate around={90:(10,10)},
      rotate around={-90:(5,5)},red]\omoins;
\coord(0)at(10,15);\node(a)at(5,10){A};
\coord[shift={(15,5)}](b)at(a);
\coord[rotate around={90:(0)}](c)at(b);
\node at(c){A};
\draw[thin,blue,>stealth,->]
      (6.2,5.4)--(20,10)arc(0:83:10);
\draw[thin,red,>stealth,<-]
      (28.8,9.6)--(15,5)arc(-45:-126:7.071);
\draw[thin,blue,dashed,>stealth,->]
      (6.2,10.4)--(20,15)arc(0:83:10);
\end{tikzpicture}

```



39. BOUCLES DE RÉPÉTITION : `foreach`

Dans tous les langages on trouve une syntaxe spéciale pour répéter un certain nombre de fois une opération ; pour TikZ, cette répétition se fait, avec la commande [56] :

`\foreach (variable) in {(liste de valeurs)} {(commandes à répéter)}`
où (variable) s'écrit comme une commande \TeX sans argument ; (liste de valeurs) a plusieurs formes possibles :

— `{m,n,...,p}` avec m , n et p : nombres réels ; la variable prend les valeurs de m à p avec un pas déterminé par $|n - m|$; exemples de syntaxe et valeurs prises :

`{1,2,...,6}` : 1, 2, 3, 4, 5, 6 ;

`{1,3,...,11}` : 1, 3, 5, 7, 9, 11 ;

`{0,0.1,...,0.5}` ; 0, 0,1, 0,2, 0,3, 0,4, 0,5 ;

— `{m,...,p}` avec m et p : nombres réels ; la variable prend les valeurs de m à p avec un pas égal à 1 ;

— `{a,...,s}` avec a et s : lettres latines (capitales ou minuscules) ; la variable prend toutes les lettres de a à s (ou de A à S) ;

— `{a,b,c,d,7,8,9,uv,uw,ux,uy,90,102,254}` : toutes les valeurs données dans la liste sont prises successivement (chaînes de lettres et/ou de chiffres) ; les éléments de la liste peuvent donc être des noms de nœuds déjà définis ; bien que le séparateur soit la virgule, les paires de

coordonnées peuvent, malgré la virgule séparatrice interne, constituer des listes valables : la liste $\{(0, 0), (0, 1), (1, 1), (1, 2)\}$ est correcte. La forme suivante est encore possible :

— $\{a, b, 9, 8, \dots, 1, 2, 2.125, \dots, 2.5\}$: $a, b, 9, 8, 7, 6, 5, 4, 3, 2, 1, 2, 2, 125, 2, 25, 2, 375, 2, 5$ qui est en quelque sorte un mélange des première et quatrième syntaxes.

Mais on a aussi la possibilité de prendre les listes de valeurs du type :

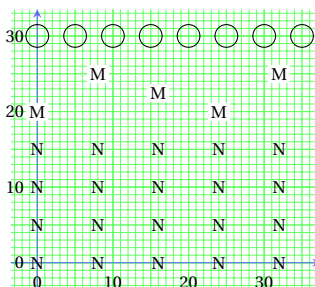
— $\{0\pi, 0.5\pi, \dots, 3.5\pi\}$ qui correspond à la liste : $0\pi, 0.5\pi$, jusqu'à 3.5π ;

— $\{2^1, 2^2, \dots, 2^5\}$ qui correspond à la liste : $2^1, 2^2$ jusqu'à 2^5 ;

— $\{A_1, \dots, H_1\}$ qui correspond à la liste : A_1, B_1 jusqu'à H_1 ;

On remarque sur l'exemple suivant l'utilisation de deux boucles `foreach` emboîtées, pour répéter des opérations avec deux variables de répétition.

```
\begin{tikzpicture}[x=1mm,y=1mm]
\foreach\i in{0,5,...,35}
  {\draw(\i,30)circle(1.5);}
\scriptsize
\foreach\i in{(0,20),(8,25),(16,22.5),
(24,20),(32,25)}{\node[rectangle,
fill=white,inner sep=1pt] at \i {M};}
\foreach\i in{0,8,...,32}
\foreach\j in{0,5,...,15}
  {\node at (\i,\j){N};}
\end{tikzpicture}
```



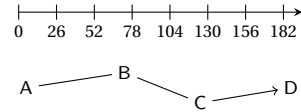
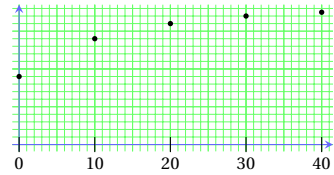
L'exemple suivant utilise encore `foreach` mais avec une variable double ; si les deux variables simples correspondantes croissent linéairement, il est plus intéressant de procéder avec ces variables simples (cf. l'exemple précédent) car on peut alors utiliser la propriété des `...` pour abréger la saisie de certaines listes de variables ; on remarque une difficulté introduite par l'extension `calc` : pour multiplier par 5,2 il a fallu multiplier par 5,20001 pour obtenir les valeurs correctes.

La fin de l'exemple montre l'utilisation d'une variable quadruple pour placer des nœuds en des points donnés, ayant un nom donné et un texte donné. Cependant, il faut être prudent lorsque l'on utilise des variables multiples : la dernière partie du dernier exemple ne fonctionne pas si l'on remplace les variables `\x` et `\y` par une variable paire `\x/\y`.

```

\newcounter{xx}
\begin{tikzpicture}[x=1mm,y=1mm]
\scriptsize
\foreach\x/\y in{0/9,10/14,20/16,
30/17,40/17.5}{\draw(\x,-1)--(\x,1);
\pose(\x,-1){-90}{\x}\ptn(\x,\y)(1)}
\draw[>=stealth,->]
(1,-10)--(38.5,-10);
\foreach\x in{1,6,...,36}
{\draw(\x,-11)--(\x,-9);}
\foreach\x in{1,6,...,36}
{\setcounter{xx}
{(\number\x-1)*real{5.20001}}
\pose(\x,-11){-90}{\the\value{xx}}}
\foreach\x/\y/\l in{2/-20/a/A,15/-18/b/B,25/-22/c/C,
37/-20/d/D}{\node(\l)at(\x,\y){\textsf{\L}};}
\draw[->](a)--(b)--(c)--(d);
\end{tikzpicture}

```



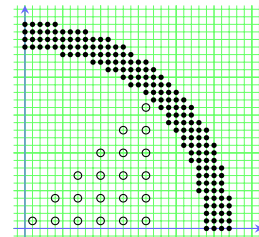
Pour terminer, on va montrer comment introduire des conditions de sortie à l'intérieur des boucles de répétitions. On donne deux exemples :

- le premier utilise la commande `\breakforeach` qui arrête la répétition lorsque la condition est vérifiée ;
- pour le second, `\breakforeach` ne convient pas ; on opère différemment : la boucle continue et c'est seulement l'action déclenché à chaque pas de boucle qui s'arrête (cette méthode est plus générale et permet de traiter des cas plus complexes).

```

\newcounter{cx}
\begin{tikzpicture}[x=1mm,y=1mm]
\foreach\x in{1,4,...,16}
\foreach\y in{1,4,...,16}
{\draw(\x,\y)circle(0.5);
\setcounter{cx}{\number\y+3}
\ifnum\x<\value{cx}\breakforeach\fi}
\foreach\x in{0,1,...,42}
\foreach\y in{0,1,...,42}
{\setcounter{cx}{\number\x*\number\x+
\number\y*\number\y}
\ifnum\value{cx}<550\else\ifnum
\value{cx}>750\ptn(\x,\y)(1)\fi\fi}
\end{tikzpicture}

```



40. LIAISONS ENTRE FIGURES ET POSITIONNEMENT SUR LA PAGE

TikZ offre la possibilité de faire des liaisons entre des figures, plus précisément des **liaisons** sur la même page. Le principe est simple : TikZ sauvegarde dans le fichier .aux les positions par rapport à la page courante des nœuds concernés ; *au cours de la compilation suivante*, il dispose donc de ces positions pour tracer les liaisons demandées : ce tracé des liaisons se fait alors de la façon habituelle (bien entendu, il ne faut rien modifier qui puisse changer la composition de la page).

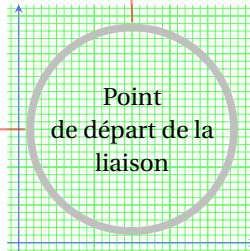
Dans chaque figure concernée, on prend l'option de figure [16.13] :
remember picture

et on donne un nom à chaque nœud de départ ou d'arrivée d'une liaison. La commande de tracé de la liaison (ou des liaisons) est placée dans l'une des figures concernées (ce choix est en général dicté par l'aspect pédagogique) ; elles s'écrivent :

`\draw[overlay, (options de tracé)] désignation habituelle du tracé ;`

Voici un exemple simple de liaisons d'une figure vers deux mots, chacun de ces mots constituant une figure particulière comprenant uniquement un nœud ; le codage de ces mots est donné juste avant le code de la figure :

```
\tikz[remember picture]{\node[rectangle,anchor=base,
  inner sep=0pt,outer sep=0.5pt,fill=lightgray!50](n2){mot};}
\begin{tikzpicture}[x=1mm,y=1mm,remember picture]
\ppm(-1.5mm,-1.5mm)(31.5mm,31.5mm)
\node[circle,draw=lightgray,
  line width=1mm,align=center,outer sep=1pt]
(n3)at(15,15){Point\de départ de
la\\liaison};
\tikzset{every draw/.style={thick,red,
  opacity=0.5,>=stealth}}
\draw[overlay, every draw, ->] (n3) -| (n2);
\draw[overlay, every draw, <->] (n2) -- (n1);
\draw[overlay, every draw, ->] (n3) to[out=90,in=-45] (n2);
\end{tikzpicture}
```



Il est assez évident qu'il ne faut pas abuser de cette opportunité, cependant, dans des cas bien particuliers, il semble qu'elle peut apporter une aide pédagogique en permettant un allègement du texte.

Par contre, on dispose d'une autre possibilité qui peut s'avérer utile :

positionner des objets graphiques (tracés et/ou nœuds) en un point donné de la page courante (quelle que soit la position de la commande correspondante). Cela est encore possible car TikZ peut se référer à la page courante (cf. les explications ci-dessus). La syntaxe est la suivante : on fait une figure avec les options de figure :

`remember picture, overlay`

Ensuite, on écrit les commandes de tracés ou de nœuds avec le positionnement par rapport à la page (les points prédéfinis sont les huit points nommés avec les points cardinaux et le centre). On donne un exemple de tracé et un exemple de nœud :

```
\begin{tikzpicture}[x=1mm,y=1mm,remember picture,overlay]
\node[font=\bfseries,scale=2,opacity=0.5,yshift=60,rotate=30,
text=red]at(current page.center){Qu'es ac\ 'o, \TikZ ?}.
\node[coordinate,shift={(35,140)}](n)at(current page.south west){};
\draw[>=stealth,thick,blue,->](n)--+(0,50);
\end{tikzpicture}
```

41. LA LOUPE

Il s'agit ici d'une opportunité intéressante utilisée depuis toujours par les dessinateurs techniques : une partie importante (du point de vue compréhension) d'une figure est grossie et représentée, soit à son emplacement (si son agrandissement ne masque pas d'autres détails importants), soit en dehors de la figure initiale. L'exemple choisi est la figure traditionnelle illustrant la définition des coordonnées sphériques r , θ , φ et montre en particulier les vecteurs unitaires $[\vec{r}]$, $[\vec{\theta}]$ et $[\vec{\varphi}]$. La syntaxe à utiliser est très simple [49] :

A) On place en option de figure une des deux options :

`spy using outlines={(options de loupe)}`

`spy using overlays={(options de loupe)}`

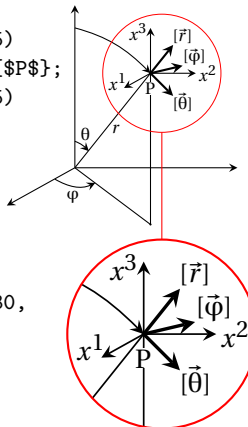
La partie concernée et son agrandissement sont présentées entourées par un contour avec la première option ou sur un fond coloré (ayant une opacité de 20%) avec la deuxième option. Les options de loupe concernent :

- la forme des parties agrandies : `circle`, `rectangle`, etc.,
- l'agrandissement désiré : `magnification=x` (x facteur d'échelle),
- les dimensions de la partie à agrandir, soit `size=(dimension)`, soit `height=(dimension)` et `width=(dimension)`,

```

\begin{tikzpicture}[x=1mm,y=1mm,
spy using outlines={circle,magnification=1.6,size=25mm,
connect spies}]\catcode'\;=12\scriptsize
\def\vct(#1,#2)(#3,#4){\draw[thick,->,>=stealth](#1,#2)--(#3,#4);}
\def\vctf(#1,#2)(#3,#4){\draw[thin,->,>=stealth](#1,#2)--(#3,#4);}
\def\traitf(#1,#2)(#3,#4){\draw[thin](#1,#2)--(#3,#4);}
\vctf(0,0)(21.5,0)\vctf(0,0)(0,21.5)\vctf(0,0)(-9,-5)
\ptn(10,-7.5)(0.4)\ptn(10,12.5)(0.5)
\traitf(0,0)(10,-7.5)\traitf(10,-7.5)(10,12.5)
\traitf(0,0)(10,12.5)\vct(10,12.5)(13,16.25)
\vct(10,12.5)(13,9.5)\vct(10,12.5)(14.25,13.5)
\node[inner sep=0.1pt,fill=white]at(10,10.5){\mathbb{P}};
\vctf(10,12.5)(10,18.5)\vctf(10,12.5)(16,12.5)
\vctf(10,12.5)(6.5,10.55)
\pose(10.4,18){180}{\mathbb{P} x^3}
\pose(15.525,12.5){0}{\mathbb{P} x^2}
\pose(6.5,10.55)[-4.5]{135}{\mathbb{P} x^1}
\draw[thin,->,>=stealth](0,18.5)to[out=-15,
in=135](10,12.5);
\draw[thin,->,>=stealth](-2.65,-1.5)to[out=-30,
in=-155](2.7,-2);
\draw[thin,->,>=stealth](0,3.5)to[out=-20,
in=135](1.85,2.22);
\pose(-0.25,-2.3){-90}{\mathbb{P} \varphi}
\pose(0,2.8){50}{\mathbb{P} \theta}
\pose(4.5,6)[-2]{-45}{\mathbb{P} r}
\pose(13,16.25)[-2.5]{45}{\mathbb{P} \vec{r}}
\pose(14.25,13.1)[-1.7]{45}{\mathbb{P} \vec{\varphi}}
\pose(13,9.955)[-3.5]{-45}{\mathbb{P} \vec{\theta}}
\spy[red] on (11.5,12.5) in node at (7.5,-22);
\end{tikzpicture}

```



— la connexion par un trait entre la partie à agrandir et son agrandissement : `connect spies` (cette connexion est facultative).

B) On place en fin de figure la commande :

```
\spy [(couleur)] on (a,b) in node at (c,d)
```

— la couleur donnée concerne le trait de jonction entre la partie à agrandir et son agrandissement ainsi que leur contour ou leur fond suivant l'option choisie

— (a,b) et (c,d) représentent les positions des centres de la partie à agrandir et de son agrandissement.

Dans l'estomac de Tikz, comme dit le \TeX Book, la partie à agrandir

et son agrandissement sont traités en tant que nœuds : cela permet de définir un style (`every style in node`) pour la partie à agrandir et un style (`every style on node`) pour son agrandissement. Une troisième option (`spy using mag glass`) présente la partie à agrandir avec une loupe et son ombre portée sur la figure (cf. la documentation pour ces autres possibilités).

On a du remettre le `catcode` du point-virgule à 12 pour cette figure. Le point-virgule de fin de commande `TikZ` ne pose pas de problème sauf dans le cas présent qui mérite d'être expliqué : dans la bibliothèque `spy`, on trouve que ce que l'on a écrit en fin de code de la figure, soit :

```
\spy[#1]on#2in node#3; avec #1 : red, #2 : (11.5, 12.5) et #3 : at(11.5, -22)  
conduit quelques lignes plus bas à l'équivalent de :
```

```
\spy[#1]on#2in node#3; {\macroA\macroB{\macroC{#1}{#2}{#3}}}
```

Le point-virgule fait ici office de limiteur de troisième argument qu'il ne peut assurer quand il est déclaré actif pour la typographie française.

42. ET LES OUTILS DONT ON N'A PAS PARLÉ...

On n'en cite que deux dont un qu'il a fallu utiliser.

— Construction de matrices de nœuds [17] [38] : permet de réaliser des présentations pédagogiques de diagrammes et d'organigrammes complexes avec de larges possibilités d'adaptation aux situations traitées ;

— Affichage des nombres flottants [36] : nécessaire pour gérer l'affichage des nombres dans les figures de visualisation des données ; pour la dernière figure de la section 25.2, l'option `mille` a permis d'écrire les nombres sans le point des mille (option par défaut, on aurait pu remplacer le point par un petit espace comme cela devrait se faire en français) ;

CONCLUSION

On cite d'abord des applications et des « aides » qui n'ont pas été exposées dans ces pages (ou qui ont été simplement citées) :

- Circuits électroniques, norme IEC [29] ;
- Réseaux logiques, normes IEC et US [29] ;
- Réseaux de Petri [3] ;
- Fond de figures (bibliothèque signalée fin sect. 14) [25] ;
- Motifs de remplissage (bibliothèque signalée fin sect. 10) [41] ;

- Lignes décoratives (en zig-zag, ondulées, fractales, etc.) et décoration de texte (notamment texte le long d'une courbe) [30] ;
- Flèches spéciales (bibliothèque signalée sect. 10.6) [23] ;
- Automates [24] ;
- Habillage d'un ensemble de points [16.6] ;
- Composition et impression de toutes sortes de calendriers [27] ;

pour se détendre : impression sur un dodécaèdre en papier [40] (l'impression se fait sur une feuille unique avec le contour de découpage, tous les traits de pliage ... et même toutes les bandes de collage!).

Tant pour les ouvrages d'information scientifique et technique que pour les livres d'enseignement, le côté pédagogique doit primer et il faut savoir que l'excès de couleurs, de fioritures et de fontes nuit beaucoup à la concentration des apprenants. On dispose maintenant de nombreuses possibilités graphiques, par exemple avec *TikZ*, mais il faut savoir les utiliser, les doser pour qu'elles apportent une aide sans dévier l'attention du lecteur. Il faut aussi veiller à respecter une certaine logique dans la présentation en se fixant des règles qui seront facilement respectées grâce à l'écriture et à l'utilisation de commandes et de styles spécifiques à l'ouvrage composé. Il apparaît clairement que *TikZ* permet de faire des figures de grande qualité pédagogique respectant tous ces conseils.

Dans la première version du manuel, on regrettait de ne pas avoir accès aux coordonnées des points d'intersection des courbes de Bézier : le regret tombe avec la version 2.2 ; il restera à intégrer, dans la distribution et sous forme améliorée, le calcul des temps d'intersection proposé dans ces pages, afin de disposer des mêmes possibilités qu'avec *METAPOST*.

Mais la phrase finale est reprise sans modification : *TikZ* est un outil utile, intéressant et agréable à utiliser tout en étant, du point de vue complexité, quelques crans en dessous de *METAPOST*.

INDEX

Symbols

! (difficulté avec -), 16
(A), 13
\$(A)!0.4!(B)\$, 16
\$(A)!20mm!(B)\$, 16
\$(A)!(C)!(B)\$, 16
(a,b), 11
(intersection of ... and ...), 60
(w:r), 12
+(a,b), 12
+(a,b) pour les courbes de Bézier, 45
+(w:r), 12
++(a,b), 12
++(w:r), 12
->, 23
->>, 23
--, 11, 17, 26
--cycle, 17, 18, 21
-|, 26
.append style, 38
.code, 39
.default, 39
.gnuplot, 70, 71
.is choice, 39
.style, 38
.table, 70, 71
.value forbidden, 39
.value required, 39
: (difficulté avec -), 64, 72
; (difficulté avec -), 72, 128
<->, 23
|<->|, 23
= (difficulté avec -), 64
>->, 23
>=, 23
\$...\$, 15
|, 23
|-, 26

A

above, 58
ajouter une autre courbe sur une figure, 76
all axes={.}, 74
also at=, 74
anchor, 55
annotation, 102
appel système, 70
arc, 42
arc tracé à partir du centre, 42
arrows, 23
automates, 129
axes, 72

B

back ground rectangle, 31
backgrounds, 31
ball color=, 107
bar width=, 70
base, 55
baseline=, 31
below, 58
bevel, 21
bottom color=, 106, 110
boucles foreach emboîtées, 123
boundingbox, 19
boundingbox, 30
\breakforeach, 124
butt, 21
by=, 84

C

cadre de figure, 31
calc, 15, 27, 42
calc (problème de calcul avec -), 123
\CALCULDESCONTROLES{.}, 88
\CALCULDESTEMPS{.}, 87
calendrier, 129

`cap=`, 21
`\catcode`, 16, 64, 72, 128
`center`, 55, 56
`child`, 93
`circle`, 41, 54
`circle through`, 43
`clean ticks`, 74
`clip`, 18, 113
`\clip`, 20
`cm`, 11
`cm={a,b,c,d,(e,f)}`, 120
`color=`, 25
`\colorlet`, 25
`concept`, 99
`concept color=`, 99
`concept connection`, 102
`connect spies`, 127
`const plot`, 70
`construction de glyphes`, 80
`controls(.)and(.)`, 44
`\coordinate`, 13
`coordinate`, 94
`coordinates`, 67
`copy shadow`, 108
`cos`, 43
`(couleur)!(nombre)`, 24
`couleur de tracé et de remplissage`,
18
`couleurs de base`, 24
`counterclockwise from`, 96
`courbe de Bézier`, 43, 44
`current bounding box.north`
`east`, 109
`current page.center`, 126
`current page.south.west`, 126

D

`dash phase=`, 22
`dashdotted`, 22
`dashed`, 22
`dashed pattern`, 22
`data[...]`, 72

`datavisualization.code.tex`, 72
`datavisualization.formats`
`.functions.code.tex`, 72
`\datavisulization`, 72
`\definecolor`, 25
`diamond`, 54
`distance level`, 97
`domain=... :...`, 68
`domaines de variation s'il y a deux`
`courbes`, 76
`dotted`, 22
`double copy shadow`, 109
`double distance=`, 24
`double=`, 24
`down`, 94
`draw opacity=xx`, 110
`\draw plot.`, 67
`\draw plot[.]coordinates{.}`,
67
`\draw plot[.]file{.}`, 67
`\draw plot[.]function{.}`, 67
`draw=`, 17, 18, 54
`\draw[overlay,...]`, 125
`\draw[path fading=.,...]`, 110
`\draw`, 11, 20, 25
`drop shadow`, 107
`dvipsnames`, 105

E

`east`, 55
`edge from parent`, 95, 97
`edge from parent node`, 97
`ellipse`, 41, 54
`environnement scope`, 32
`environnement tikzpicture`, 30
`even odd rule`, 112
`every ...`, 37
`every label`, 65
`every pin`, 65
`every shadow`, 108, 109
`every style in node`, 128
`every style on node`, 128

extra concept, 99, 102

F

`\faitmarkinfo`, 80
`\FAITMARKU`, 80
few, 74
file, 47, 67
fill opacity=xx, 110
fill=, 18, 54, 99, 108
`\fill`, 20, 25
`\filldraw`, 20
flèches spéciales, 129
flush right, 57
fond de figure, 31
font=, 56
`\foreach\ .in{.}{.}`, 15, 68, 122
format de fichier de données, 48
formule chimique, 97
func y = . . . , 73
function, 67
fush left, 57

G

gnuplot (tracé avec -), 70
gray, 18
grid, 25, 68
grid={.}, 74
grow cyclic, 96
grow=, 93

H

`\hbox` (utilisation de -), 64
height=, 126

I

id, 71
include values={.}, 75
`\includegraphics`, 19
inner color=, 107
inner frame sep=, 31
inner sep=, 57
inner xsep=, 57
inner ysep=, 57

insert path, 26
intensité des couleurs, 24
intersection of ... and ... ,
83

J

join, 21
justified, 56

L

label d'une liaison entre générations,
97
label distance=, 63, 65
label=, 74
label={ [.] }, 63
latex, 23
left, 56
left color=, 106, 110
length=, 75
level distance=, 94
level i, 95
level i concept, 99
liaisons, modification des – entre
générations, 97
lignes décoratives, 129
line width=, 12, 20
liste de valeurs de variables de
répétition, 122
logarithmic, 82

M

magnification=, 126
main, 115
major={.}, 74
manmind.tex, 99
manual mindmap, 99
manual mindmp, 100
many, 74, 78
`\MARK`, 78, 79
mark indices={.}, 50
mark phase=, 50
mark repeat=, 50, 68

mark size=, 50, 70
 mark=, 49, 70
 masque de transparence, 110
 matrices de nœuds, 128
 middle color=, 107
 mille, 82
 mindmap, small -, large -, huge -,
 99
 mindmap, bibliothèque -, 99
 minimum height=, 57
 minimum size=, 42
 minimum width=, 57
 minor steps between steps=, 74
 missing, 96
 miter, 21
 mm, 11, 17
 \montre, 81

N

\n1, ..., 14
 name intersections={.}, 84
 name path, 83
 name=, 84, 110
 node, 58, 93, 94
 \node[.] (.) at (.){.}, 42, 43, 53
 nodes=, 95
 \node[coordinate, ...], 60
 nombres flottants (affichage des, 128
 none, 74
 nonzero rule, 112
 north, 55, 94
 north east, ..., 55
 \nrec, 58

O

of=...and..., 84
 only mark, 68
 opacity=, 108, 110
 options at=(.)as[...], 74
 outer color=, 107
 outer sep=, 57, 63, 101
 outer xsep, 57

outer ysep, 57
 overlay, 126

P

\p[.], 14
 \p1, ..., 14
 parametric, 69
 \path, 17, 18
 path picture, 18
 pattern, 18, 19
 patterns, 19
 perso dashed, 22
 personnalisation des mindmaps, 101
 \pgfclosepath, 91
 \pgfdeclaredataformat, 77
 \pgfdeclarelayer{.}, 115
 \pgfextra, 14
 \pgfextractx, 40
 \pgfextracty, 40
 \pgfkeys, 38, 40
 pgfkeys, 37
 pgfkeysdef, 40
 pgfkeysdefnargs, 40
 \pgflineto{.}, 91
 pgfonlayer, 115
 \pgfpointcurveatime, 40
 \pgfsetlayers{.}, 115
 \pgfusepath{fill}, 91
 \pgfusepath{stroke}, 91
 pin distance=, 63
 pin edge={.}, 65
 pin={[.]...:..}, 63
 placement d'options en option de
 child ou de node, 102
 plot, 67
 plot[.]coordinates, 47
 plot[.]file, 47
 plotmarks, 50
 point intérieur coloré, 112
 points de contrôle, 44
 points de contrôle en dehors de la
 figure, 20

polynômes de Bernstein, 45
pos, 58
`\pose`, 15
`\poseb`, 15, 68
`\ppm`, 81
`\ppmm`, 11, 25
`\pps`, 68
prefix, 71
problème si bordures épaisses et
liaisons, 101
propriété géométrique des courbes
de Bézier, 45
pt, 11
`\ptn`, 15

R

radius=, 41
rect, 21
rectangle, 25, 54
remember picture, 125, 126
réseaux de Petri, 128
réseaux électroniques, 128
réseaux logiques, 128
reset cm, 120
right, 56
right color=, 106, 110
root concept, 99, 100
rotate around{.}, 120
rotate=, 54, 120
round, 21
rounded corners=, 21

S

samples=, 68, 70, 73
scale around={.}, 119
scale=, 12, 80, 119
school book axes, 73
scientific axes, 75
scientific clean axes, 78
scientific inner axes, 76
scope, 32
shade, 18, 54, 106

`\shadedraw`, 20
shading angle=, 106
shading=axis, 106
shading=ball, 106
shading=radial, 106
shadow scale=, 108, 109
shadow xshift=, 108, 109
shadow yshift=, 108, 109
shape, 54
shape aspect=, 54
shapes, 54
sharp corners, 21
shift={.}, 60, 119
shorten <=, 23
shorten >=, 23, 65
show background rectangle, 31
sibling angle=, 96
sibling distance=, 94
sin, 43
size=, 126
sloped, 60
smooth, 47–49
smooth cycle, 47–49
solid, 22
some, 74, 78
sort by={.}, 84, 85
`\souscourbe`, 87, 88
`\souscourbeto`, 88
south, 55, 94
spy using mag glass, 128
spy using outlines={.}, 126
spy using overlays={.}, 126
`\spy[.]on(.)in node at(.)`, 127
stealth, 23
step=, 25, 74, 82
style=, 35
styles de nœud de différentes
générations, 95
svgnames, 105

T

tableau, 77

tension=, 49
 text width=, 56
 text=, 56, 100
 texte sur une courbe, 129
 thick, 17, 20
 thin, 20
 ticks={.}, 74
 \tikz, 30
 tikz, 30
 tikzfadingfrompicture, 110
 tikzpicture, 30–32, 115
 \tikzset, 31, 35
 \tikzset (remarque sur le
 positionnement de -), 60
 to, 23
 to[out=...,in=...,...], 50
 top color=, 106, 110
 \TRAIT, 78
 transparent!xx, 110
 trees, 96

U

unit length, 74
 unités de longueur, 11
 up, 94
 use as bounding box, 18
 \usepackage[tikz], 11, 29
 \usetikzlibrary, 30

V

var x : interval[... :...], 73
 variable=\..., 69
 variable foreach double, 123
 variable de répétition foreach, 122
 veclen, 14, 43
 very thick, 12
 visualize as ..., 72
 visualize as scatter, 74
 visualize as smooth line, 73,

74

W

west, 55
 width=, 126

X

x axis={.}, 72
 x axis={.}, 74
 x radius=, 41
 \x1, ..., 14
 x11names, 106
 x=, 11
 xcolor, 24, 105
 xscale=, 119
 xshift=, 11, 18, 60, 119
 xslant=, 119
 xstep=, 68

Y

y axis={.}, 72, 74
 y radius=, 41
 \y1, ..., 14
 y=, 11
 ybar, 70
 yscale=, 119
 yshift=, 60, 119
 yslant=, 119
 ystep=, 68