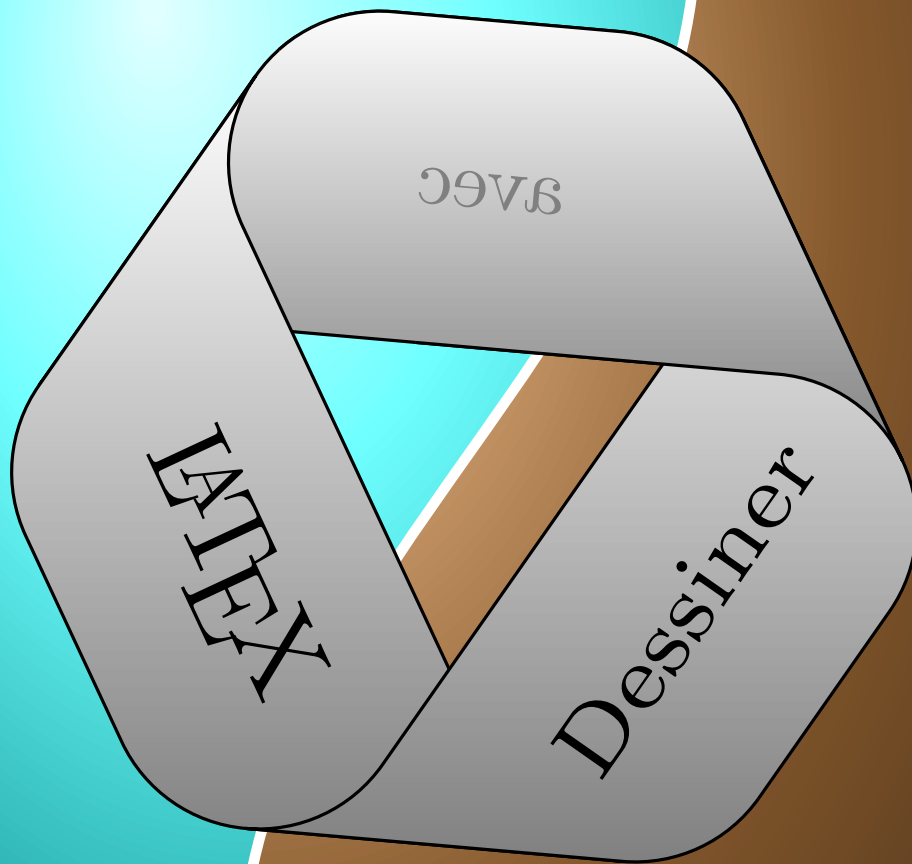


# TikZ

*pour l'impatient*

*G rard Tisseau et Jacques Duma*

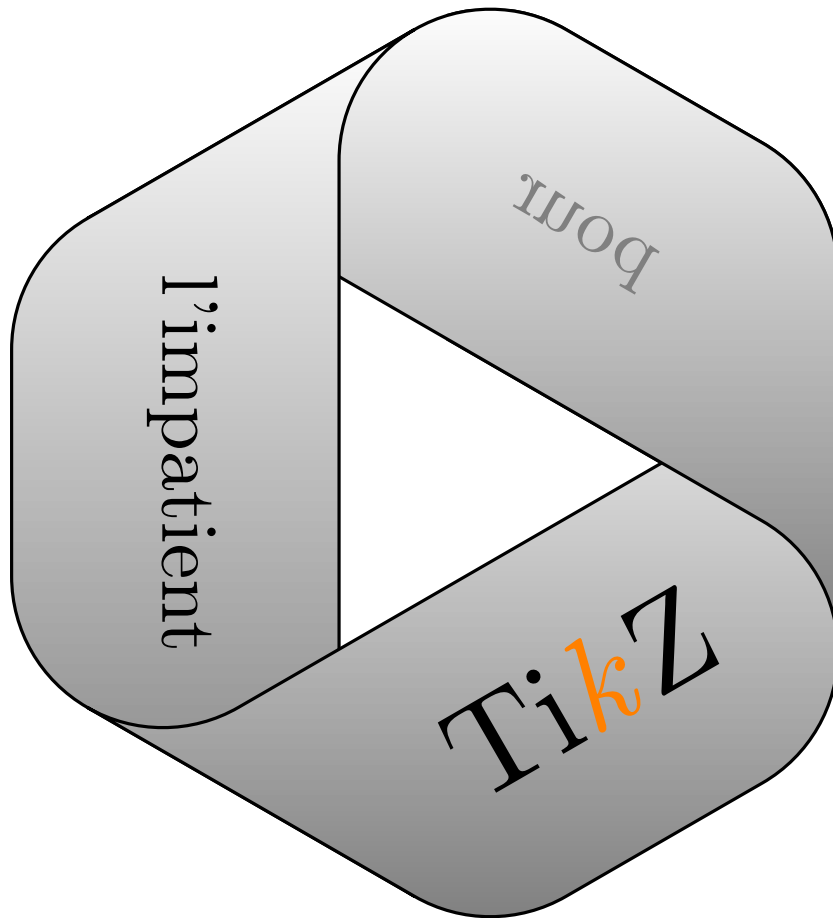




# Ti*k*Z pour l' impatient

G rard Tisseau          Jacques Duma

3 juillet 2015



```

\def\arete{3}   \def\epaisseur{5}   \def\rayon{2}

\newcommand{\ruban}{(0,0)
  ++(0:0.57735*\arete-0.57735*\epaisseur+2*\rayon)
  ++(-30:\epaisseur-1.73205*\rayon)
  arc (60:0:\rayon) -- ++(90:\epaisseur)
  arc (0:60:\rayon) -- ++(150:\arete)
  arc (60:120:\rayon) -- ++(210:\epaisseur)
  arc (120:60:\rayon) -- cycle}

\begin{tikzpicture}[very thick,top color=white,bottom color=gray]
  \shadedraw \ruban;
  \shadedraw [rotate=120] \ruban;
  \shadedraw [rotate=-120] \ruban;
  \draw (-60:4) node[scale=5,rotate=30]{Ti{\color{orange}\textit{k}}Z};
  \draw (180:4) node[scale=3,rotate=-90]{l'impatient};
  \clip (0,-6) rectangle (6,6); % pour croiser
  \shadedraw \ruban;
  \draw (60:4) node [gray,xscale=-3,yscale=3,rotate=30]{pour};
\end{tikzpicture}

```

# Table des matières

<b>Avant-propos</b>	<b>9</b>
Vous avez des documents à publier, avec des figures . . . . .	9
Vous avez essayé d'inclure des figures, sans grand succès . . . . .	9
Nous vous recommandons d'utiliser <i>TikZ</i> . . . . .	9
Ce livre vous aide à utiliser <i>TikZ</i> . . . . .	9
Chercher dans le livre : la table des matières . . . . .	9
Trouver une référence : le glossaire . . . . .	10
Le site compagnon . . . . .	10
Remerciements . . . . .	10
<b>1 Premières figures</b>	<b>11</b>
1.1 Utilisation de <i>TikZ</i> dans $\text{\LaTeX}$ . . . . .	11
1.1.1 <i>TikZ</i> est un package : <code>\usepackage{tikz}</code> . . . . .	11
1.1.2 Insérer une figure <i>TikZ</i> : <code>\begin{tikzpicture}</code> . . . . .	12
1.2 Le repérage des points . . . . .	13
1.2.1 Coordonnées cartésiennes : $(x,y)$ . . . . .	13
1.2.2 Coordonnées polaires : $(a:r)$ . . . . .	13
1.2.3 Échelle : <code>[scale=k]</code> . . . . .	14
1.3 Exemple : tracer un segment ou un cercle . . . . .	14
1.3.1 Énoncé : deux segments, un cercle . . . . .	14
1.3.2 Solution à la main . . . . .	14
1.3.3 Solution <i>TikZ</i> : <code>(a,b) -- (c,d)</code> et <code>(a,b) circle (r)</code> . . . . .	15
1.3.4 Écrire des textes : <code>(x,y) node [position] {texte}</code> . . . . .	15
1.3.5 Arc de cercle : <code>(x,y) arc (a:b:r)</code> . . . . .	16
1.3.6 Annotations : angle droit, segments égaux . . . . .	17
1.4 Figure géométrique : méthodes de base . . . . .	19
1.4.1 Problème principal : calculer les coordonnées . . . . .	19
1.4.2 Exemple : triangle de côtés 3, 4 et 5 . . . . .	19
1.4.3 Préparer la figure avec GeoGebra . . . . .	21
1.4.4 Faire engendrer le code <i>TikZ</i> par GeoGebra . . . . .	22
1.5 Exercices : figures géométriques . . . . .	22
1.5.1 Théorème de Thalès . . . . .	23
1.5.2 Parallélogramme . . . . .	23
1.5.3 Losange . . . . .	23
1.5.4 Centre de gravité . . . . .	24
1.5.5 Cercle circonscrit . . . . .	24
1.5.6 Orthocentre . . . . .	25
1.5.7 Centre du cercle inscrit . . . . .	25
1.6 Résumé . . . . .	26

<b>2</b>	<b>Chemins, options graphiques</b>	<b>27</b>
2.1	Simplifications, raccourcis, abstractions	27
2.1.1	Nommage des points : <code>\coordinate(nom) at (x,y)</code>	27
2.1.2	Enchaînement de traits : <code>chemin, position courante</code>	28
2.1.3	Rectangle : <code>(a,b) rectangle (c,d)</code>	29
2.1.4	Figures fermées : <code>cycle, fill</code>	29
2.1.5	Noeuds sur les traits : <code>midway, sloped</code>	30
2.1.6	Coordonnées relatives : <code>++(x,y)</code>	30
2.2	Décorations, styles, options graphiques	32
2.2.1	Options : <code>[ ]</code>	32
2.2.2	Épaisseur des traits : <code>thick, thin, line width=5pt</code>	32
2.2.3	Pointillés, styles des traits : <code>dotted, dashed, double</code>	32
2.2.4	Pointes de flèches : <code>-&gt;, &gt;= stealth</code>	33
2.2.5	Couleurs : <code>red, color=gray!20</code>	33
2.3	Axes, grille, fenêtre d'affichage	34
2.3.1	Axes	34
2.3.2	Quadrillage (grille) : <code>grid</code>	35
2.3.3	Fenêtre d'affichage : <code>clip</code>	36
2.4	Compléments : opacité, couleurs, styles	36
2.4.1	Ordre des tracés, transparence : <code>opacity</code>	36
2.4.2	Noms et calculs des couleurs, package <code>xcolor</code>	37
2.4.3	Définition de styles : <code>\tikzstyle, \tikzset</code>	38
2.5	Exercices : styles de traits, flèches, couleurs	38
2.5.1	Somme de deux vecteurs	38
2.5.2	Triangle rectangle inscrit dans un demi-cercle	39
2.5.3	Angle inscrit et angle au centre	39
2.5.4	Parallèles, aires égales	39
2.5.5	Composée de deux symétries centrales	39
2.5.6	Suite géométrique	40
<b>3</b>	<b>Courbes</b>	<b>41</b>
3.1	Tracer une courbe : <code>plot (...)</code>	41
3.1.1	Domaine : <code>[domain=a:b]</code>	42
	Le problème de <code>babel</code> français et de « : »	42
	Désactiver « : » avec <code>\shorthandoff{:}</code>	43
	Introduire une autre option <code>[domaine={a}{b}]</code>	43
	Utiliser le package <code>microtype</code>	43
	Exemples de domaines	43
3.1.2	Formules mathématiques disponibles	44
	Opérations	44
	Fonctions	44
	Fonctions trigonométriques	44
	Nombres aléatoires	45
	Opérations booléennes	45
3.2	Aspect du graphe	45
3.2.1	Nombre de points : <code>samples</code>	45
3.2.2	Lissage : <code>smooth, tension</code>	46
3.2.3	Discontinuités : on peut séparer les intervalles	46
3.2.4	Grandes valeurs : <code>scale, \clip</code>	47
3.3	Régions limitées par des courbes	48
3.3.1	Une courbe et des segments : <code>cycle, \fill, \filldraw</code>	48
3.3.2	Région entre deux courbes	49
3.3.3	Région non convexe : <i>interior rules</i>	49
3.4	Compléments techniques	50
3.4.1	Utilisation de Gnuplot : <code>plot function</code>	50
3.4.2	Automatisation de certaines configurations	51

3.5	Exercices . . . . .	52
3.5.1	Ellipse. Angles avec <code>circle</code> et <code>\clip</code> . . . . .	52
3.5.2	$a^b = b^a$ . <code>xscale</code> , <code>yscale</code> . . . . .	52
3.5.3	Fonction périodique : <code>\foreach</code> . . . . .	53
3.5.4	Fonctions réciproques, aires : <code>pattern</code> . . . . .	54
3.5.5	Lemniscate de Geronon. <code>\scope</code> , <code>xshift</code> , <code>\filldraw</code> . . . . .	55
3.6	Résumé . . . . .	56
<b>4</b>	<b>Géométrie dans l'espace</b> . . . . .	<b>59</b>
4.1	Coordonnées $(x, y, z)$ . . . . .	59
4.1.1	Représentation TikZ standard . . . . .	59
4.1.2	Autres représentations : <code>x=...</code> , <code>y=...</code> , <code>z=...</code> . . . . .	60
4.2	Quelques figures de géométrie . . . . .	60
4.2.1	Section d'un cube suivant un hexagone . . . . .	60
4.2.2	Grande diagonale d'un cube . . . . .	61
4.2.3	Droites et plans . . . . .	61
4.3	Courbes et surfaces . . . . .	62
4.3.1	Représentation paramétrique, <code>plot</code> , <code>\foreach</code> . . . . .	62
4.3.2	Hélice . . . . .	63
4.3.3	Cylindre $x^2 + y^2 = 1$ . . . . .	64
4.3.4	Sphère $x^2 + y^2 + z^2 = 1$ . . . . .	64
4.3.5	Paraboloïde $z = x^2 + y^2$ . . . . .	64
4.4	Résumé . . . . .	64
<b>5</b>	<b>Représentation de données</b> . . . . .	<b>65</b>
5.1	Notions de base . . . . .	65
5.1.1	Diagramme d'effectifs : <code>plot coordinates</code> . . . . .	65
5.1.2	Améliorer la lisibilité : <code>grid</code> , <code>node</code> , <code>\foreach</code> . . . . .	66
5.1.3	Marquer les points, étiqueter : <code>mark</code> , <code>node</code> , <code>rotate</code> . . . . .	67
5.1.4	Diagramme à barres : <code>xcomb</code> , <code>ycomb</code> , <code>polar comb</code> . . . . .	68
5.1.5	Histogramme : <code>xcomb</code> , <code>ycomb</code> , <code>line width</code> . . . . .	69
5.1.6	Affichage des données d'un fichier : <code>plot file</code> . . . . .	69
5.2	Diagramme à barres horizontales . . . . .	70
5.2.1	Le blé dans le monde : utilisation d'un tableur . . . . .	70
5.2.2	Barres horizontales : <code>plot file</code> , <code>xcomb</code> . . . . .	71
5.2.3	Installation d'une grille : <code>grid</code> , <code>xstep</code> , <code>ystep</code> . . . . .	73
5.2.4	Étiquetage du repère : <code>\foreach</code> , <code>node</code> . . . . .	73
5.2.5	Deux séries plus une légende : <code>plot</code> , <code>shift</code> , <code>node</code> . . . . .	74
5.3	Courbe des variations de données . . . . .	75
5.3.1	Production annuelle de riz : pré-traitement . . . . .	75
5.3.2	Courbe des variations : <code>plot file</code> . . . . .	76
5.3.3	Quadrillage : <code>grid</code> , <code>step</code> . . . . .	77
5.3.4	Annotations, décorations : <code>\foreach</code> , <code>node</code> , <code>mark</code> . . . . .	78
5.4	Diagramme à secteurs . . . . .	79
5.4.1	Répartition par catégories socioprofessionnelles . . . . .	79
5.4.2	Calcul des angles : pré-traitement avec un tableur . . . . .	79
5.4.3	Dessiner les secteurs : <code>\draw</code> , <code>arc</code> , <code>cycle</code> , <code>fill</code> , <code>\$</code> . . . . .	80
5.4.4	Diagramme complet : <code>\foreach</code> . . . . .	81
5.5	Résumé . . . . .	82
<b>6</b>	<b>Graphes : Introduction</b> . . . . .	<b>83</b>
6.1	Notions de base . . . . .	83
6.1.1	Nœuds et Arcs : <code>\draw</code> , <code>--</code> , <code>node</code> , et <code>\node</code> . . . . .	83
6.1.2	Chemin annoté : <code>\draw</code> avec opération <code>node</code> . . . . .	84
6.1.3	Graphe : <code>\node</code> puis <code>\draw</code> avec nom de nœud . . . . .	84
6.2	Styles des nœuds et des arcs . . . . .	84

6.2.1	Les arcs : <code>\draw</code> , <code>--</code> , <code> -</code> , <code>- </code> , <code>to</code> et options de flèches . . . . .	84
6.2.2	Extrémités des arcs : <code>[-&gt; </code> , <code>*-o</code> , <code>&gt;-&gt;&gt;</code> , <code>)-(</code> . . . . .	86
6.2.3	Frontières des nœuds : <code>circle</code> , <code>ellipse</code> , <code>diamond</code> . . . . .	86
6.2.4	Abstraction des styles : <code>\tikzstyle</code> , <code>\tikzset</code> . . . . .	87
6.2.5	Points d’ancrage des nœuds : <code>N.south</code> , <code>N.left</code> , <code>N.below</code> . . . . .	88
6.2.6	Flèches vers les ancres : <code>N.north</code> , <code>N.center</code> , <code>N.15</code> . . . . .	89
6.3	Techniques avancées . . . . .	90
6.3.1	Tracer un arc sans avancer : <code>edge</code> . . . . .	90
6.3.2	Étiquetage des arcs : <code>sloped</code> , <code>midway</code> , <code>pos</code> . . . . .	90
6.3.3	Inclinaison des étiquettes : <code>sloped</code> , <code>rotate</code> . . . . .	91
6.3.4	Modification de la taille des annotations : <code>scale</code> . . . . .	91
6.3.5	Insérer une sous-figure : <code>scope</code> , <code>shift</code> , <code>rotate</code> , <code>scale</code> . . . . .	91
6.3.6	Textes longs : <code>text width</code> , <code>justified</code> , <code>centered</code> . . . . .	93
6.3.7	Contournement d’un nœud . . . . .	94
6.4	Exercices . . . . .	95
6.4.1	Voyelle ou Consonne . . . . .	95
6.4.2	Les points cardinaux . . . . .	95
6.4.3	Orientations . . . . .	96
6.4.4	Pentagone . . . . .	96
6.4.5	Benzène . . . . .	97
6.4.6	Arbre généalogique . . . . .	98
6.5	Résumé . . . . .	98
<b>7</b>	<b>Graphes : Exemples</b> . . . . .	<b>99</b>
7.1	Graphe d’une relation . . . . .	99
7.1.1	Relations entre quadrilatères . . . . .	99
7.1.2	Des nœuds et des flèches : <code>node</code> et <code>-&gt;</code> . . . . .	99
7.1.3	Graphe final : courbure <code>bend</code> , ancrage <code>P.east</code> . . . . .	102
7.2	Organigramme informatique . . . . .	103
7.2.1	Somme des $N$ premiers nombres entiers . . . . .	103
7.2.2	Style des nœuds : <code>draw</code> , <code>ellipse</code> , <code>fill</code> , <code>text</code> . . . . .	103
7.2.3	Forme des flèches : <code>&gt;=</code> , <code>rounded corners</code> , <code> -</code> . . . . .	104
7.2.4	Organigramme final . . . . .	106
7.3	Diagrammes syntaxiques . . . . .	107
7.3.1	Grammaire des expressions mathématiques . . . . .	107
7.3.2	Alignement des nœuds, étiquetage . . . . .	107
7.3.3	Regroupement de figures : <code>scope</code> et <code>yshift</code> . . . . .	108
7.4	Graphe de preuve . . . . .	109
7.4.1	Résolution d’une équation : $2x + 3 = 7$ . . . . .	109
7.4.2	Placement des nœuds : <code>\node (a) at (x,y), below</code> . . . . .	110
7.4.3	Placement et étiquetage des flèches : <code>-&gt;</code> , <code>midway</code> . . . . .	111
7.4.4	Flèches courbes : <code>bend</code> , <code>to</code> . . . . .	112
7.4.5	Exercice d’amélioration . . . . .	113
7.5	Résumé . . . . .	114
<b>8</b>	<b>Des figures aux illustrations</b> . . . . .	<b>115</b>
8.1	Les anneaux olympiques . . . . .	115
8.1.1	Un anneau : <code>circle</code> , <code>fill</code> , <code>even odd rule</code> . . . . .	116
8.1.2	Entrelacer les anneaux : <code>\coordinate</code> , <code>fill</code> et <code>arc</code> . . . . .	116
8.1.3	La figure complète : <code>\newcommand</code> . . . . .	118
8.2	Diagrammes de Venn . . . . .	120
8.2.1	Ensembles $E$ , $A$ , $B$ : <code>rectangle</code> , <code>circle</code> , <code>\newcommand</code> . . . . .	120
8.2.2	Coloriage : <code>\fill</code> , <code>color</code> , <code>opacity</code> . . . . .	121
8.2.3	Méthode par superposition de couleurs . . . . .	121
	$A$ , $B$ et $A \cup B$ : <code>\draw</code> et <code>\fill</code> . . . . .	121
	$A \cap B$ , $A \setminus B$ , $B \setminus A$ et $A \Delta B$ : <code>\clip</code> et <code>scope</code> . . . . .	122



8.2.4	Méthode par coloriage entre les frontières . . . . .	123
	Définition des frontières : <code>rectangle</code> , <code>circle</code> et <code>arc</code> . . . . .	124
	Coloriage des régions : <code>\fill</code> , <code>even odd rule</code> . . . . .	125
8.3	Personnages et décors . . . . .	126
8.3.1	L'océan : <code>\shade</code> , <code>arc</code> , <code>top color</code> , <code>bottom color</code> . . . . .	126
8.3.2	Le quai : <code>\fill</code> , <code>rectangle</code> , <code>rotate</code> . . . . .	127
8.3.3	Les personnages : <code>\fill</code> , <code>ellipse</code> , <code>circle</code> . . . . .	127
8.3.4	Le cœur : <code>\draw</code> , .. <code>controls</code> and .. . . . .	127
8.3.5	Cœurs multicolores : <code>\shift</code> , <code>rotate</code> , <code>ball color</code> . . . . .	129
8.3.6	La figure complète : <code>scope</code> , <code>shift</code> , <code>rotate</code> . . . . .	130
8.3.7	La solution : <code>scope</code> , <code>shift</code> , <code>rotate</code> . . . . .	130
8.4	Résumé . . . . .	130
<b>9</b>	<b>Compléments techniques</b> . . . . .	<b>131</b>
9.1	Transformations avec <code>scope</code> . . . . .	131
9.1.1	Translations : <code>xshift</code> , <code>yshift</code> ou <code>shift</code> . . . . .	131
9.1.2	Combinaison de translation et rotation : <code>[xshift=6cm,rotate=45]</code> . . . . .	132
9.1.3	Translation et changement d'échelle : <code>[xshift=6cm,scale=0.5]</code> . . . . .	133
9.1.4	Épaisseur des traits : <code>\draw</code> et <code>line width</code> . . . . .	134
9.1.5	Taille et inclinaison de textes : <code>transform shape</code> . . . . .	136
9.1.6	Exercice . . . . .	137
9.2	Au sujet des arbres . . . . .	137
9.2.1	Définition : <code>\node node</code> et <code>child</code> . . . . .	138
9.2.2	Espacement des frères : <code>sibling distance</code> . . . . .	139
9.2.3	Forme globale : <code>level distance</code> et <code>grow</code> . . . . .	140
9.2.4	Étiquetage des arcs : <code>edge from parent</code> . . . . .	140
9.2.5	Style des arcs : <code>edge from parent path</code> . . . . .	141
9.3	Liaisons entre figures : <code>overlay</code> . . . . .	141
9.3.1	Définitions globales des noms : <code>remember picture</code> . . . . .	142
9.3.2	Dessiner d'une figure à l'autre : <code>overlay</code> . . . . .	142
9.3.3	La page courante est un nœud : <code>current page</code> . . . . .	143
9.4	Résumé . . . . .	143
<b>A</b>	<b>La syntaxe de TikZ</b> . . . . .	<b>145</b>
A.1	Les environnements : <code>{tikzpicture}</code> , <code>{scope}</code> . . . . .	145
A.2	Les commandes . . . . .	145
A.3	Les coordonnées . . . . .	146
A.3.1	Forme générale : (...) . . . . .	146
A.3.2	Calculs sur les nombres : package <code>pgfmath</code> . . . . .	146
A.3.3	Calculs sur les coordonnées : bibliothèque <code>calc</code> . . . . .	147
A.4	Les opérations de chemin . . . . .	147
A.5	Les options . . . . .	148
A.6	Utiliser des commandes L <sup>A</sup> T <sub>E</sub> X dans TikZ . . . . .	150
<b>B</b>	<b>Erreur ! Que faire ?</b> . . . . .	<b>151</b>
	Oubli du « ; » . . . . .	151
	Les nombres trop grands . . . . .	151
	Le « ! » dans la définitions des couleurs . . . . .	151
	Le problème de <code>babel</code> français et de « : » . . . . .	152
<b>C</b>	<b>Où trouver de l'aide ?</b> . . . . .	<b>153</b>
<b>D</b>	<b>Glossaire</b> . . . . .	<b>155</b>



# Avant-propos

## **Vous avez des documents à publier, avec des figures**

Vous avez régulièrement des documents à publier. Vous avez choisi  $\LaTeX$  pour sa grande qualité typographique, son ouverture et sa portabilité. Vous souhaiteriez maintenant inclure des figures et illustrations dans vos documents, mais sans avoir à sortir de l'environnement  $\LaTeX$ , et de manière à pouvoir produire directement des documents au format PDF, qui devient un format d'échange et de publication standard.

## **Vous avez essayé d'inclure des figures, sans grand succès**

Les différentes solutions que vous avez essayées ne vous ont pas semblées satisfaisantes.

Vous avez préparé un dessin dans un logiciel externe, puis vous avez utilisé la commande `\includegraphics`, mais vous n'avez pas trouvé pratique le fait de manipuler des fichiers externes et d'essayer d'adapter le style et le format à votre document.

Vous avez essayé `pstricks`, mais vous l'avez trouvé un peu trop complexe, même si `pstricks` et `TikZ` offrent à peu près les mêmes outils.

## **Nous vous recommandons d'utiliser `TikZ`**

`TikZ` est un package pour  $\LaTeX$  permettant d'inclure des figures au format PDF en restant dans l'environnement  $\LaTeX$ .

Il a été créé vers 2006 par Till Tantau. Il devient rapidement populaire, car il répond aux besoins précédents en évitant les inconvénients des autres solutions. La phase initiale d'apprentissage est rapide, et les figures simples peuvent être obtenues simplement. On sent que le langage a été conçu pour répondre à des besoins usuels de manière pratique. Il continue d'évoluer, et les extensions actuelles permettent de créer des illustrations très variées.

Utiliser `TikZ` est un plaisir car on obtient des figures précises et d'une grande qualité, avec une impression de maîtrise.

## **Ce livre vous aide à utiliser `TikZ`**

Dans ce livre, nous présentons `TikZ` de manière à vous rendre capable d'obtenir rapidement des figures incluses dans vos documents  $\LaTeX$ , en l'illustrant de différentes façons : géométrie, courbes, graphes, arbres, histogrammes, illustrations.

La lecture des deux premiers chapitres est indispensable. Vous pourrez créer vos figures dès le premier chapitre. Le deuxième fournit des compléments importants d'ordre général, et ensuite vous pourrez choisir en fonction de votre domaine d'application.

Les deux derniers chapitres présentent des exemples plus complexes et des compléments techniques. Il est préférable d'être à l'aise avec  $\LaTeX$  et `TikZ` pour les aborder.

## **Chercher dans le livre : la table des matières**

La table des matières est une sorte d'aide mémoire intégré.

Chaque fois que c'est possible, un titre est la description d'une tâche ou d'un problème suivis des mots-clés `TikZ` qui permettent de réaliser cette tâche ou de résoudre ce problème de façon standard.

Par exemple : Échelle : `[scale=k]`, Étiquetage des arcs : `sloped`, `midway`, `pos`, etc. D'un seul coup d'oeil, on devrait pouvoir retrouver une information utile et la situer dans le contexte d'un exemple.

### Trouver une référence : le glossaire

Ce livre contient un glossaire mais pas d'index. Le problème d'un index alphabétique est que l'on ne peut y trouver que des mots déjà connus.

Notre glossaire rassemble uniquement la liste des mots-clés du langage présentés dans le livre, avec pour chacun d'eux un court résumé et une référence au passage du Manuel de *TikZ & PGF* de Till Tantau présentant le concept.

Till Tantau signale lui-même que l'index de son manuel n'est pas très satisfaisant parce qu'il contient tout alors qu'il ne devrait contenir qu'une sélection. Il contient plus de 2000 entrées !

Nous avons justement fait une sélection (moins de 100 entrées). Le résumé devrait suffire pour l'usage indiqué dans cet ouvrage, et ensuite vous pourrez obtenir des compléments dans le manuel officiel.

### Le site compagnon

Un site Internet, créé par les auteurs, accompagne ce livre. On y trouvera le code de tous les exemples présentés ici, plus quelques compléments.

À partir du site, il est aussi possible de contacter les auteurs : Toutes les questions, toutes les critiques et toutes les suggestions sont les bienvenues.

<http://math.et.info.free.fr/TikZ/index.html>

La version papier de ce document est en noir et blanc, seule la version PDF disponible sur ce site est en couleurs.

### Remerciements

Merci à Emmanuel Collinet pour sa lecture très attentive de « *TikZ pour l' impatient* » et pour ses nombreuses remarques pertinentes. Ainsi, nous avons pu améliorer la qualité de ce document.

# Chapitre 1

## Premières figures

### 1.1 Utilisation de TikZ dans L<sup>A</sup>T<sub>E</sub>X

#### 1.1.1 TikZ est un package : `\usepackage{tikz}`

TikZ étant un package pour L<sup>A</sup>T<sub>E</sub>X, il s'utilise comme tout autre package, en déclarant `\usepackage{tikz}` dans le préambule.

Pour savoir quelle est la version de TikZ dont vous disposez, vous pouvez faire afficher le numéro de version par la commande `\pgfversion` (PGF est le nom du « moteur » de TikZ).

Par exemple, ce document a été composé le 3 juillet 2015 avec la version 3.0.0 de TikZ, mais la première version a été rédigée en février 2008 avec la version 2.00.

Si vous ne disposez pas de TikZ ou si vous avez une version plus ancienne que la version 2.00, le plus simple est sans doute de charger entièrement la plus récente version de la distribution T<sub>E</sub>X (TikZ est fourni avec les principales distributions de T<sub>E</sub>X).

Dans ce cas il faudra consulter la documentation de l'auteur si quelques différences apparaissent dans la composition des exemples.

[Version la plus récente de « TikZ & PGF Manual »](#)

Le document minimal utilisant TikZ est donc :

```
\documentclass{article}
\usepackage{tikz}
\begin{document}
La version de TikZ est : \pgfversion
\end{document}
```

Les standards sont TeX Live pour les systèmes Unix (y compris MacOS X, où elle est présente dans MacTeX) et MiKTeX pour les systèmes Windows. Ces distributions T<sub>E</sub>X sont assez souvent mises à jour (une fois par an environ).

La version 2 de TikZ est incluse dans la distribution TeX Live version 10.2 de 2008.

Pour Windows : <http://miktex.org>

Pour Mac : <http://www.tug.org/mactex/>

Pour Unix, Linux : <http://www.tug.org/texlive/>

Éventuellement, vous pouvez obtenir directement le package `pgf` et lui seul, mais il faut une certaine expertise pour l'installer correctement « à la main ». Cependant MikTeX dispose d'un utilitaire pour cela, accessible par un menu, et il existe une commande `tlmgr` sous Unix (TeX Live manager).

Package `pgf` : <http://sourceforge.net/projects/pgf/>  
ou <http://www.ctan.org/tex-archive/graphics/pgf/>

Comme dans toute utilisation de package, il faut être prudent : il peut exister des conflits avec d'autres packages. Le plus notable pour TikZ est le conflit avec `xcolor`, car TikZ redéfinit certaines des fonctions de `xcolor`. Cependant les deux packages peuvent rester compatibles si on déclare `xcolor` avant `tikz`.

### 1.1.2 Insérer une figure TikZ : `\begin{tikzpicture}`

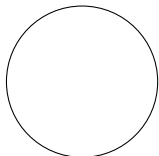
L'élément de base que permet de créer TikZ dans un document L<sup>A</sup>T<sub>E</sub>X est une *figure* (picture). Elle se matérialise dans le document L<sup>A</sup>T<sub>E</sub>X par un environnement `tikzpicture` :

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

À l'intérieur de cet environnement se trouve une zone de texte dans laquelle on écrit suivant la syntaxe spéciale de TikZ. Voici par exemple comment on trace un cercle de rayon 1 dont le centre a pour coordonnées cartésiennes (0,0) (dans le système de repérage de TikZ) :

```
\begin{tikzpicture}
\draw (0,0) circle (1) ;
\end{tikzpicture}
```

Ce qui donne :



La syntaxe `\draw (0,0) circle (1) ;` est analysée par TikZ suivant ses propres conventions, qui ne sont pas celles du L<sup>A</sup>T<sub>E</sub>X standard, et que nous détaillerons par la suite.

Comment une figure TikZ est-elle placée dans le document final? TikZ calcule d'abord l'encombrement total de la figure et fait en sorte que L<sup>A</sup>T<sub>E</sub>X considère cette figure comme un bloc rectangulaire. L<sup>A</sup>T<sub>E</sub>X insère alors ce bloc dans le flot normal, suivant les règles applicables aux blocs de type `mbox`.

Dans l'exemple qui suit, on a inséré une figure TikZ dans du texte et on l'a encadrée avec `\fbox` :

Voici une première ligne ...

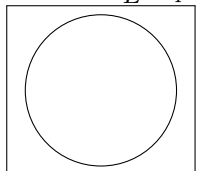
... et de rayon 1

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

}

inséré dans une ligne ...

Voici une première ligne de texte L<sup>A</sup>T<sub>E</sub>X. pour montrer l'insertion d'une figure. Voici le cercle de



centre (0,0) et de rayon 1 inséré dans une ligne L<sup>A</sup>T<sub>E</sub>X et encadré avec `fbox`. La figure est considérée comme un rectangle et est alignée sur la ligne de base.

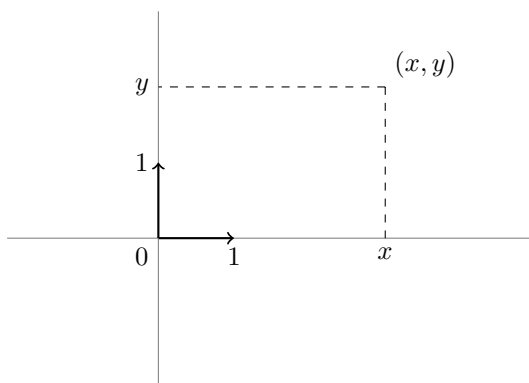
On peut ensuite gérer l'alignement et le placement sur la page avec les outils L<sup>A</sup>T<sub>E</sub>X habituels de mise en page. La plupart du temps, on isole la figure sur une ligne et on la centre avec `\begin{center} ... \end{center}`.

```
:newpage
```

## 1.2 Le repérage des points

### 1.2.1 Coordonnées cartésiennes : $(x, y)$

Dans le système par défaut, les points sont repérés à l'aide de deux axes perpendiculaires : l'axe des abscisses, horizontal et dirigé vers la droite et l'axe des ordonnées, vertical et dirigé vers le haut. Les vecteurs de base ont exactement pour longueur 1 cm (et quand on imprime, c'est très précisément 1 cm). La position d'un point est repérée par un couple de nombres  $(x, y)$

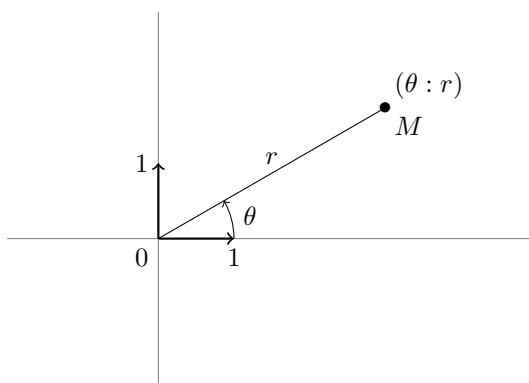


On peut se demander où est placée l'origine dans la figure engendrée. En fait, elle n'a pas de position prédéfinie. La figure occupe seulement la place minimale pour que tous les éléments tracés explicitement soient visibles. La position de l'origine par rapport à la figure dépend donc des points tracés. Si on trace juste un cercle de centre  $(2, 0)$  et de rayon 1, l'origine ne sera pas dans le cadre de la figure, qui ne montrera que les abscisses  $x$  entre 1 et 3 et les ordonnées  $y$  entre  $-1$  et 1.

TikZ possède d'autres systèmes de repérage que nous verrons au fur et à mesure. Signalons tout de suite les coordonnées polaires  $(\alpha:r)$  où  $\alpha$  est l'angle polaire en degrés et  $r$  le rayon polaire.

### 1.2.2 Coordonnées polaires : $(\alpha:r)$

On peut repérer la position d'un point  $M$  dans un système de coordonnées polaires, avec la syntaxe  $(\theta : r)$ , où  $\theta$  est l'angle orienté (en degrés) entre le vecteur de base des abscisses et le vecteur  $\overrightarrow{OM}$  et  $r$  est la distance  $OM$ .

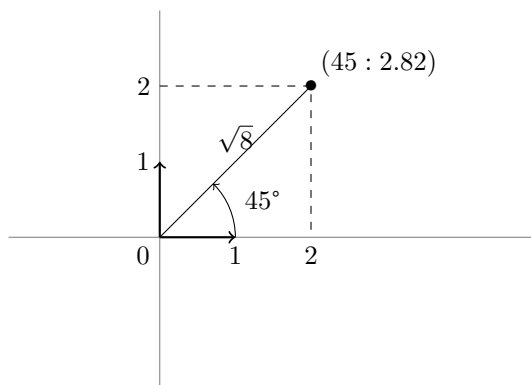


Les formules permettant de passer d'un système à un autre sont :

$$r = \sqrt{x^2 + y^2}, \quad \cos(\theta) = \frac{x}{r}, \quad \sin(\theta) = \frac{y}{r}$$

$$x = r \cos(\theta) \quad \text{et} \quad y = r \sin(\theta)$$

Par exemple, le point de coordonnées cartésiennes  $(2, 2)$  a pour coordonnées polaires approchées  $(45:2.82)$ , où 2.82 est la valeur approchée de  $\sqrt{8} = \sqrt{2^2 + 2^2}$ .



Dans la version de base, on doit spécifier les coordonnées sous forme numérique approchée. Cependant une extension de TikZ permet d'introduire quelques calculs formels. Nous verrons cela dans la suite, mais pour l'instant nous utiliserons uniquement les valeurs numériques.

### 1.2.3 Échelle : [scale=k]

Il peut arriver que la figure, telle qu'elle est définie avec les coordonnées, soit trop petite ou trop grande. On peut alors utiliser l'option [scale=k], qu'on écrit juste après `\begin{tikzpicture}`. Elle signifie que toutes les dimensions seront multipliées par  $k$ .

Pour avoir une figure deux fois plus grande, écrire

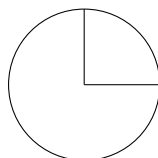
```
\begin{tikzpicture}[scale=2] ... \end{tikzpicture}
```

Le point de coordonnées  $(1,0)$  sera alors dessiné à 2 cm de l'origine au lieu de 1 cm.

## 1.3 Exemple : tracer un segment ou un cercle

### 1.3.1 Énoncé : deux segments, un cercle

Dans un repère orthonormal d'unité 1 cm, tracer les deux segments reliant l'origine aux points unités de coordonnées  $(1,0)$  et  $(0,1)$ , ainsi que le cercle trigonométrique.



### 1.3.2 Solution à la main

Pour cela, on procède en séparant les trois éléments de la figure :

1. Premier segment (horizontal)
  - (a) Positionner le crayon au-dessus de l'origine
  - (b) Abaisser le crayon et tracer le segment jusqu'au point de coordonnées  $(1,0)$
2. Deuxième segment (vertical)
  - (a) Relever le crayon, le positionner au-dessus de l'origine
  - (b) Abaisser le crayon et tracer le segment jusqu'au point de coordonnées  $(0,1)$
3. Cercle
  - (a) Prendre un compas, piquer sa pointe à l'origine
  - (b) Fixer un rayon de 1 cm, tracer le cercle.



### 1.3.3 Solution TikZ : (a,b) -- (c,d) et (a,b) circle (r)

Cette construction se traduit en TikZ de la manière suivante :

```
\begin{tikzpicture}
  \draw (0,0) -- (1,0) ;
  \draw (0,0) -- (0,1) ;
  \draw (0,0) circle (1) ;
\end{tikzpicture}
```

On y retrouve tous les éléments de la solution précédente.

La figure entière est délimitée par un *environnement* au sens de L<sup>A</sup>T<sub>E</sub>X :

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

À l'intérieur de cet environnement, TikZ introduit sa propre syntaxe, qui n'est plus celle de L<sup>A</sup>T<sub>E</sub>X (ce qui peut dérouter au début).

Chacune des sous-figures (segments, cercle) est délimitée par

```
\draw ... ;
```

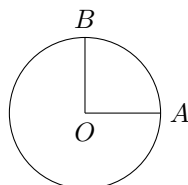
Chaque élément d'une sous-figure correspond à une opération à l'intérieur du `\draw` :

```
positionner : (x,y)
tracer un segment : --(x,y)
tracer un cercle : circle (r)
```

Il y a bien d'autres éléments de syntaxe dans TikZ, nous les verrons au fur et à mesure.

### 1.3.4 Écrire des textes : (x,y) node [position] {texte}

Pour que la figure soit plus parlante, on veut maintenant écrire le nom des points à côté des points : *O* pour l'origine, *A* pour (1,0), *B* pour (0,1)



Pour cela, on veut exprimer en gros les actions suivantes :

Écrire la lettre *O* en-dessous de l'origine

Écrire la lettre *A* à droite du point (1,0)

Écrire la lettre *B* au-dessus du point (0,1)

Ces trois instructions s'écrivent en TikZ de la manière suivante :

```
\draw (0,0) node[below]{$O$} ;
\draw (1,0) node[right]{$A$} ;
\draw (0,1) node[above]{$B$} ;
```

A chaque fois, on retrouve l'opération de positionnement : (0,0) pour se positionner à l'origine, etc.

Puis vient le mot-clé `node`. Cela fait référence à un concept de TikZ : le concept de nœud. Cela désigne en gros une boîte de texte. Il faut ensuite préciser où se place le centre de cette boîte par rapport à la position spécifiée. C'est l'objet de l'option entre crochets : `[below]` par exemple (il n'est pas nécessaire de donner une position très précise, TikZ fait des choix raisonnables). Puis on donne le texte à afficher, entre accolades, et dans ces accolades on écrit une expression L<sup>A</sup>T<sub>E</sub>X quelconque. C'est une des forces de TikZ : pouvoir annoter une figure avec des textes écrits en L<sup>A</sup>T<sub>E</sub>X.

Ces instructions se rajoutent à la figure précédente, qui devient :

```
\begin{tikzpicture}
  \draw (0,0) -- (1,0) ;
  \draw (0,0) -- (0,1) ;
  \draw (0,0) circle (1) ;
  \draw (0,0) node[below]{$O$} ;
  \draw (1,0) node[right]{$A$} ;
  \draw (0,1) node[above]{$B$} ;
\end{tikzpicture}
```

On peut ne pas spécifier du tout de position. Le centre du texte est alors placé au point de référence :

Par exemple, pour placer la lettre  $O$  précisément à l'origine, on peut écrire :

```
\draw (0,0) node{$O$};
```

Les positions possibles sont (avec les significations des mots anglais) :

```
above, below, right, left,
above left, above right, below left, below right
```

au-dessus, en-dessous, à droite, à gauche

au-dessus à gauche, au-dessus à droite, en-dessous à gauche, en-dessous à droite

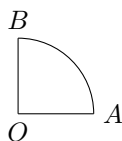
Parfois ces indications qualitatives ne sont pas assez précises. Le plus simple est souvent alors de placer le centre du nœud explicitement à un autre point proche du point principal, en contrôlant exactement les coordonnées. Il y a d'autres méthodes, mais elles introduisent de nouveaux concepts techniques que nous aborderons plus tard.

```

      •
      O
(0,0) node[below left] {$O$}
      •
      O
(-0.6,-0.3) node {$O$}
```

### 1.3.5 Arc de cercle : $(x,y)$ arc $(a:b:r)$

Comment dessiner seulement le quart de cercle de  $A$  à  $B$  ?



Pour spécifier mathématiquement un arc de cercle, on peut donner le centre du cercle, son rayon, l'angle polaire du point origine et l'angle polaire du point extrémité (le vecteur de référence des angles polaires étant comme d'habitude le vecteur  $(1,0)$ , ici  $\overrightarrow{OA}$ ).

Par exemple, ici, on pourrait dire : tracer l'arc de cercle de centre  $(0,0)$  et de rayon 1 dont l'origine a pour angle polaire  $0^\circ$  et donc l'extrémité a pour angle polaire  $90^\circ$ .

Malheureusement, TikZ n'a pas choisi cette façon de s'exprimer. Il faut dire quelque chose comme : tracer l'arc de cercle qui commence en  $A$ , de telle manière que  $A$  se trouve sur ce cercle avec un angle polaire de  $0^\circ$ , jusqu'au point d'angle polaire  $90^\circ$  sur le même cercle, sachant que le rayon du cercle vaut 1.

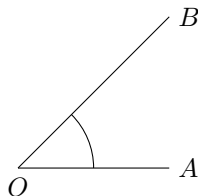
```
\draw (1,0) arc (0:90:1) ;
```



Bien sûr, ces informations suffisent pour retrouver le centre et pour reconstituer la figure, mais ce n'est sans doute pas la convention la plus naturelle possible (sauriez-vous construire géométriquement le centre?). L'avantage qu'on peut lui trouver, c'est justement de ne pas avoir à préciser le centre. Mais existe-t-il des cas où on trace un arc de cercle sans avoir aucune idée du centre?

Une application fréquente de cette construction est de marquer un angle sur une figure.

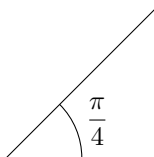
Soit le triangle  $OAB$  avec  $O(0,0)$ ,  $A(2,0)$ ,  $B(2,2)$ . Marquer sur le dessin l'angle en  $O$ .



```
\begin{tikzpicture}
  \draw (0,0) node[below] {$O$};
  \draw (2,0) node[right] {$A$};
  \draw (2,2) node[right] {$B$};
  \draw (0,0) -- (2,0) ;
  \draw (0,0) -- (2,2);
  \draw (1,0) arc (0:45:1) ;
\end{tikzpicture}
```

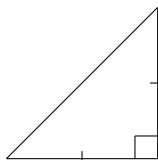
Et ensuite, on peut utiliser le concept de `node` pour écrire la valeur de l'angle à côté. Il faut alors placer la boîte de texte au bon endroit, près de l'arc. Ici on peut utiliser les coordonnées polaires, puisque l'angle en  $O$  est connu :  $45^\circ$ . Le nœud est alors placé dans la direction de l'angle moitié :  $22.5^\circ$ , à une distance un peu plus grande que le rayon de l'arc : 1.3. En fait, on ne peut indiquer qu'un nombre entier de degrés, donc on arrondit par exemple à 22.

```
\draw (22:1.3) node {$\dfrac{\pi}{4}$};
```



### 1.3.6 Annotations : angle droit, segments égaux

Annotons maintenant le triangle avec les signes habituels indiquant que c'est un triangle isocèle rectangle.



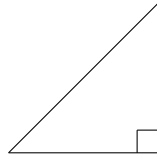
On marque l'angle droit avec deux petits segments perpendiculaires, l'égalité des côtés avec deux petits segments coupant les côtés. Bien que ces signes ne fassent pas vraiment partie de la figure et qu'ils ne vont pas faire l'objet d'une étude géométrique, il s'agit, après tout, quand même, d'éléments géométriques : des segments. On peut les tracer avec les outils vus jusqu'à maintenant, mais cela nécessite de calculer les coordonnées de leurs extrémités.

Cela peut paraître inhabituel, car on trace souvent ce genre de signes sans trop y faire attention et sans avoir à résoudre de nouveaux problèmes géométriques.

Ici les traits de la marque d'angle droit sont horizontaux et verticaux, donc il n'est pas difficile d'imaginer des coordonnées (leur longueur exacte résulte d'un choix arbitraire, disons 0.3).

Donc les nouveaux points à introduire sont  $(1.7, 0)$ ,  $(1.7, 0.3)$  et  $(1.7, 0.3)$ ,  $(2, 0.3)$

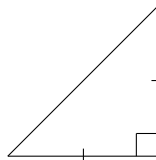
```
...
\draw (1.7, 0) -- (1.7, 0.3) ;
\draw (1.7, 0.3) -- (2, 0.3) ;
```



Les marques de segments égaux peuvent également être choisis horizontaux et verticaux, disons de longueur 0.2

Donc les nouveaux points à introduire sont  $(1.9, 1)$ ,  $(2.1, 1)$  et  $(1, -0.1)$ ,  $(1, 0.1)$ .

```
....
\draw (1.9,1) -- (2.1,1);
\draw (1,-0.1) -- (1,0.1);
```

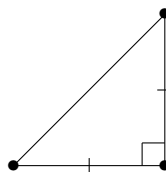


Et si on voulait marquer les points avec des points plus gros, comme  $\bullet$  ?

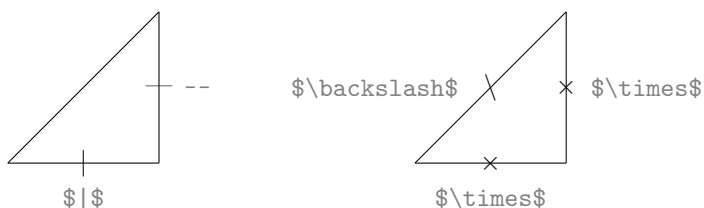
Eh bien, un gros point de ce type est simplement un caractère  $\LaTeX$  :  $\bullet$

Il suffit d'introduire un nœud (**node**), centré sur le point (il suffit de ne donner aucune option de position) et dont le contenu est  $\bullet$  :

```
\draw (0,0) node {$\bullet$} ;
\draw (2,0) node {$\bullet$} ;
\draw (0,2) node {$\bullet$} ;
```



On peut prolonger cette idée : un nœud de texte peut être rempli avec un symbole  $\LaTeX$ , et un symbole n'est rien d'autre après tout qu'un dessin. Écrire un symbole, c'est dessiner, et  $\LaTeX$  est bien fourni en symboles divers. On peut utiliser cette idée ici pour tracer les petits traits indiquant l'égalité des côtés : on peut utiliser les symboles  $|$  (barre verticale),  $---$  (tiret  $-$ ),  $\backslash$  (barre oblique  $\backslash$ ),  $/$  (barre oblique  $/$ ),  $\times$  (multiplication  $\times$ ).



## 1.4 Figure géométrique : méthodes de base

Nous venons de voir les éléments de base qui constituent une figure géométrique comme on en rencontre au collège et au lycée, et comment ils se traduisent en `TikZ` :

- des segments : `(a,b) -- (c,d)`
- des cercles : `(a,b) circle (r)`
- des arcs de cercle : `(a,b) arc (u:v:r)`
- des annotations de texte : `(a,b) node[position] {texte}`

Nous allons voir maintenant comment on peut créer de nombreuses figures de géométrie uniquement à l'aide de ces éléments de base. Autrement dit, vous pouvez être opérationnels dès maintenant avec les techniques vues jusqu'ici.

`TikZ` propose bien d'autres possibilités (son manuel comporte 560 pages), mais il est essentiel de se familiariser avec les bases pour comprendre l'intérêt des extensions et savoir quand les utiliser de manière pertinente. Et encore une fois, les bases peuvent vous suffire pour la plupart de vos figures.

Cela peut paraître surprenant de pouvoir se contenter de ces constructions, mais après tout, si vous observez une figure de géométrie usuelle, même compliquée, vous verrez bien qu'elle n'est formée que de ces éléments, du moins en ce qui concerne sa structure. Les figures usuelles sont bien de ce type : points, segments, triangles, parallélogrammes, polygones, cercles, droites parallèles et perpendiculaires, etc.

### 1.4.1 Problème principal : calculer les coordonnées

Construire une figure avec les techniques de base seulement suppose que vous devez calculer d'abord par vos propres moyens les coordonnées (cartésiennes ou polaires) de tous les points. Ce n'est pas ce qu'on fait d'habitude quand on trace une figure à la main : on dispose d'outils de dessin (règle, compas, rapporteur, papier quadrillé) et d'outils de calcul (calculatrice, logiciel mathématique).

La difficulté du calcul dépend de la figure que vous avez à construire et des contraintes sur cette figure. Si les positions des points sont imposées par un énoncé et si les points ne sont pas placés de manière pratique pour le dessin, alors il peut y avoir beaucoup de calculs. Mais souvent, vous avez une certaine marge de manœuvre : si votre but est d'illustrer une propriété géométrique, vous pouvez décider de placer certains points de telle manière que les calculs soient facilités.

Voici quelques stratégies, qui sont en fait déjà bien connues pour les tracés à la main :

- privilégier les points à coordonnées entières
- privilégier les directions verticales et horizontales
- privilégier les directions d'angles polaires simples et connus, ce qui permettra d'utiliser plus facilement les coordonnées polaires
- commencer par la fin : si on veut illustrer le cercle circonscrit à un triangle, commencer par placer le cercle

### 1.4.2 Exemple : triangle de côtés 3, 4 et 5

Voici un exemple : tracer un segment  $[AB]$  horizontal de longueur 5, puis tracer le triangle  $ABC$  direct tel que  $BC = 4$  et  $AC = 3$ .

On choisit des coordonnées simples pour  $A$  et  $B$  :  $A(0,0)$  et  $B(5,0)$ .

Il faut calculer les coordonnées de  $C$ , en traduisant  $AC = 3$  et  $BC = 4$ , soit  $x^2 + y^2 = 3^2$ ,  $(x - 5)^2 + y^2 = 4^2$ .

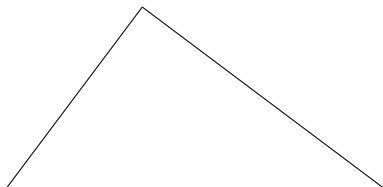
On résout ensuite le système par différence :  $10x - 25 = -7$ , soit  $x = \frac{9}{5} = 1.8$ , puis, en remplaçant,  $y = \frac{12}{5} = 2.4$

On peut alors écrire le code `TikZ`, qu'on peut annoter avec des commentaires  $\LaTeX$  pour plus de lisibilité :

```

\begin{tikzpicture}
\draw (0,0) -- (5,0);      % AB = 5
\draw (0,0) -- (1.8,2.4); % AC = 3
\draw (5,0) -- (1.8,2.4); % BC = 4
\end{tikzpicture}

```

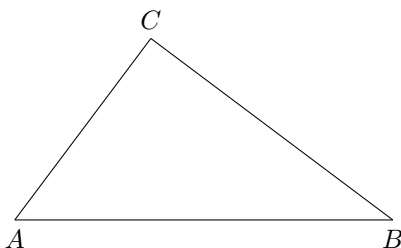


On peut rajouter des nœuds de texte pour montrer les noms Des points :

```

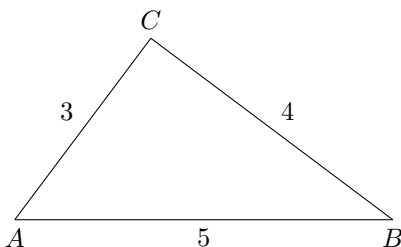
\draw (0,0) node [below] {$A$}; etc.

```

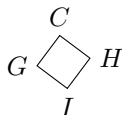


Et finalement, on peut indiquer les longueurs des côtés. Pour cela, il faut trois noeuds de texte, placés aux alentours des milieux des segments, milieux dont on calcule les coordonnées :  $\left(\frac{5}{2}, 0\right)$ ,  $\left(\frac{17}{5}, \frac{6}{5}\right)$ ,  $\left(\frac{9}{10}, \frac{6}{5}\right)$ .

Donc, par exemple `\draw (3.4,1.2) node [above right] {$4$}` ; etc.



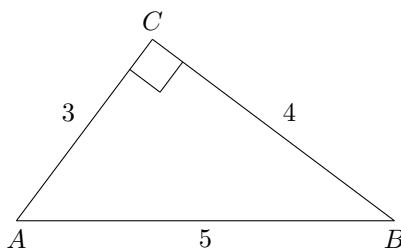
Il serait dommage de ne pas en profiter pour illustrer le théorème de Pythagore : l'angle en  $C$  est droit puisque  $3^2 + 4^2 = 5^2$ . Pour noter cela, il faut des petits traits dans l'angle en  $C$ , de longueur 0.5 par exemple, de manière à former un carré  $CGIH$  avec  $G$  sur  $[CA]$  et  $H$  sur  $[CB]$ .



On pose  $\overrightarrow{CG} = \frac{1}{2} \times \frac{1}{3} \overrightarrow{CA}$ ,  $\overrightarrow{CH} = \frac{1}{2} \times \frac{1}{4} \overrightarrow{CB}$ , puis  $\overrightarrow{CI} = \overrightarrow{CG} + \overrightarrow{CH}$ .

On trouve  $G(1.5, 2)$ ,  $H(2.2, 2.1)$  et  $I(1.9, 1.7)$ .

Sur le dessin on trace  $[GI]$  et  $[IH]$  : `\draw (1.5,2) -- (2.2,2.1)` ; etc.



Il faut quand même un certain temps pour concevoir tout cela, faire les calculs et réaliser la figure, et cela pour un simple triangle. Le plus long n'est d'ailleurs pas de tracer le triangle, mais de bien placer les annotations (noms des points, longueurs des côtés, marque d'angle droit). On conçoit alors que la réalisation d'une figure plus complexe peut demander beaucoup de temps et de calculs.

Une possibilité est alors de les faire exécuter d'abord par un autre logiciel spécialisé en géométrie et en mathématiques. Un exemple d'un tel logiciel est GeoGebra. Nous allons voir comment l'utiliser pour préparer cette figure.

### 1.4.3 Préparer la figure avec GeoGebra

Le site de GeoGebra est à l'adresse <http://www.geogebra.org/>

GeoGebra permet de construire interactivement et visuellement une figure avec des primitives mathématiques, comme tracer la perpendiculaire à une droite passant par un point, tracer l'intersection de deux lignes, tracer le milieu d'un segment, tracer l'image d'un point par une rotation donnée, tracer le cercle passant par trois points, etc. Il permet aussi de nommer les points, de tracer les droites par leurs équations, de déterminer le lieu d'un point quand un autre varie.

Autrement dit, il manipule directement des objets géométriques de manière visuelle et conceptuelle et il a des capacités de calcul mathématique. Ces possibilités peuvent être très avantageuses pour préparer une figure TikZ.

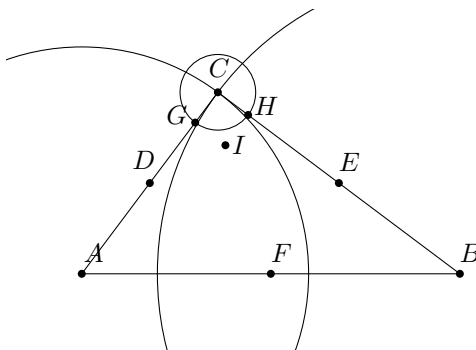
Un autre de ses avantages est que vous pouvez déplacer certains points pour que la figure soit bien mise en place sans avoir à tout redéfinir (les contraintes géométriques imposées restent vérifiées). Pour cette raison, on dit que c'est un logiciel de *géométrie dynamique* (un autre excellent logiciel bien connu de ce type est Cabri).

Voici comment réaliser la figure précédente avec GeoGebra :

Tracer le point  $A(0,0)$ , le point  $B(5,0)$ , le cercle de centre  $A$  et de rayon 3, le cercle de centre  $B$  et de rayon 4, puis définir  $C$  comme un des points d'intersection des deux cercles (le point d'ordonnée positive). Dans la colonne des coordonnées à gauche, vous pouvez lire les coordonnées de  $C$  :  $(1.8, 2.4)$ . Vous n'avez pas eu de calculs à faire pour cela.

De même on peut tracer les milieux des côtés et lire leurs coordonnées.

Pour la marque d'angle droit, on peut construire  $G$  et  $H$  comme les intersections du cercle de centre  $C$  et de rayon 0.5 avec les segments  $[CA]$  et  $[CB]$ , puis obtenir  $I$  comme image de  $C$  par la rotation de centre  $H$  et d'angle  $90^\circ$ .



On peut ainsi retrouver toutes les coordonnées utiles et reconstituer la figure.

### 1.4.4 Faire engendrer le code TikZ par GeoGebra

Au lieu de lire les coordonnées et de les utiliser pour écrire le code TikZ, vous pouvez laisser GeoGebra le faire pour vous ! En effet, depuis la préversion de développement 3.1 de septembre 2008, on trouve un menu "Fichier → Exporter → Export PGF/TikZ". Lorsqu'on demande un export, on obtient le code d'un document L<sup>A</sup>T<sub>E</sub>X contenant une figure TikZ. Si votre seul but est de produire un document avec juste une figure, cela suffit. La figure reproduit à l'identique (quasiment) celle que vous avez construite avec GeoGebra, avec la qualité TikZ. Autrement dit, dans ce cas, vous n'avez même pas à apprendre TikZ !

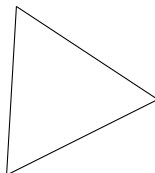
Il se peut que vous souhaitiez inclure cette figure dans un document L<sup>A</sup>T<sub>E</sub>X, ce que vous pouvez faire par copier-coller. Il se peut aussi que vous ne souhaitiez pas conserver tous les éléments de la figure (en particulier ce qui concerne les déclarations des couleurs, la grille, les axes). Il faut alors analyser précisément le code produit par GeoGebra pour extraire juste ce qui vous intéresse, par exemple les segments entre les points (il est préférable de cacher d'abord sous GeoGebra tout ce qu'on ne souhaite pas voir). En général, le code correspondant est identifiable et bien groupé.

Par exemple, après avoir tracé un triangle équilatéral avec la commande GeoGebra « polygone régulier », on peut extraire juste les lignes qui tracent le triangle :

```
\draw [color=zzttqq] (2,1)-- (4,2);
\draw [color=zzttqq] (4,2)-- (2.13,3.23);
\draw [color=zzttqq] (2.13,3.23)-- (2,1);
```

puis supprimer les indications de couleur. On obtient alors un tracé simple :

```
\begin{tikzpicture}
\draw (2,1)-- (4,2);
\draw (4,2)-- (2.13,3.23);
\draw (2.13,3.23)-- (2,1);
\end{tikzpicture}
```



On peut ensuite partir de ce code pour compléter la figure.

Ce procédé a été utilisé pour construire de nombreuses figures de géométrie de ce livre, en particulier pour les exercices qui suivent.

## 1.5 Exercices : figures géométriques

Dans chaque exercice, on demande de réaliser la figure spécifiée. Toutes ces figures peuvent être construites en TikZ uniquement à partir des éléments de base : coordonnées, segments, cercles, arcs, nœuds. Pour certaines figures, les coordonnées ont été obtenues à l'aide du logiciel GeoGebra.

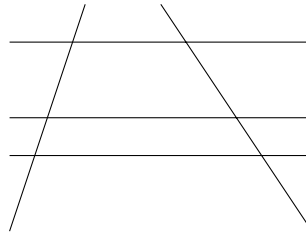
Comme éléments de solution, nous donnons les coordonnées utilisées (il est recommandé d'utiliser GeoGebra ou au minimum un papier quadrillé).

Il est parfois étonnant de s'apercevoir que, même pour des figures aussi classiques et relativement simples, le nombre de points peut être élevé (une vingtaine de points). C'est dû en partie aux points auxiliaires utilisés pour les annotations.



### 1.5.1 Théorème de Thalès

Figure : deux droites obliques coupées par trois parallèles horizontales.

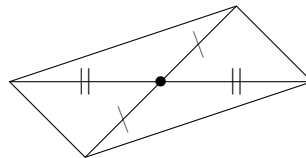


**Aide :**

droites obliques  $(0,0) -- (1,3)$  et  $(4,0) -- (2,3)$   
 droites parallèles  $(0,1) -- (4,1)$   $(0,1.5) -- (4,1.5)$  et  
 $(0,2.5) -- (4,2.5)$

### 1.5.2 Parallélogramme

Les diagonales d'un parallélogramme se coupent en leur milieu

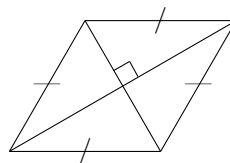


**Aide :**

Centre :  $\bullet (0,0)$   
 Diagonales :  $(-2,0) -- (2,0)$  et  $(1,1) -- (-1,-1)$   
 Marques  $\parallel$  :  $(-1,0)$  et  $(1,0)$   
 Marques  $\backslash$  :  $(-0.5,-0.5)$  et  $(0.5,0.5)$ ;

### 1.5.3 Losange

Les diagonales d'un losange sont perpendiculaires.  
 Ici le losange est formé de deux triangles équilatéraux.



**Aide :**

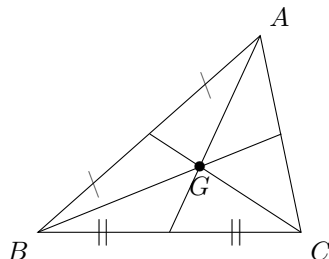
Losange :  $(0,0), (2,0), (3,1.73), (1,1.73)$   
 Diagonales  $(0,0) -- (3,1.73)$  ;  $(1,1.73) -- (2,0)$   
 Marques  $\$/\$$  :  $(2,1.73), (1,0)$   
 Marques  $---$  :  $(2.5,0.87), (0.5,0.87)$   
 Angle droit :  $(1.38,1.07), (1.59,1.19), (1.7,0.99)$

### 1.5.4 Centre de gravité

Le centre de gravité d'un triangle est l'intersection des médianes.

Les points ont été choisis sur le cercle trigonométrique.

$$A\left(\frac{3}{5}, \frac{4}{5}\right), B\left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right), C\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$



**Aide :**

Échelle : 2.

Triangle :  $A(0.6, 0.8), B(-0.87, -0.5), C(0.87, -0.5)$

Milieux :  $(0, -0.5), (0.74, 0.15), (-0.13, 0.15)$

Centre de gravité  $G(0.2, -0.07)$

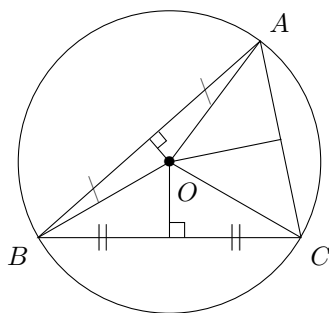
Marques  $\backslash$  :  $(0.24, 0.47), (-0.5, -0.18)$

Marques  $|$  :  $(-0.44, -0.5), (0.44, -0.5)$

### 1.5.5 Cercle circonscrit

Le centre du cercle circonscrit à un triangle est l'intersection des médiatrices. Même triangle :

$$A\left(\frac{3}{5}, \frac{4}{5}\right), B\left(-\frac{\sqrt{3}}{2}, -\frac{1}{2}\right), C\left(\frac{\sqrt{3}}{2}, -\frac{1}{2}\right)$$



**Aide :**

Échelle : 2.

Triangle :  $A(0.6, 0.8), B(-0.87, -0.5), C(0.87, -0.5)$

Milieux :  $(0, -0.5), (0.74, 0.15), (-0.13, 0.15)$

Cercle : centre  $O(0, 0)$ , rayon : 1 (tout ça pour ça!)

Marques  $\backslash$  :  $(0.24, 0.47), (-0.5, -0.18)$

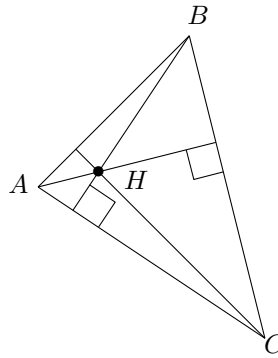
Marques  $|$  :  $(-0.44, -0.5), (0.44, -0.5)$

Angle droit sur  $[BC]$  :  $(0.1, -0.5), (0.1, -0.4), (0, -0.4)$

Angle droit sur  $[AB]$  :  $(-0.08, 0.09), (-0.02, 0.14), (-0.07, 0.2)$

### 1.5.6 Orthocentre

L'orthocentre est l'intersection des hauteurs.



**Aide :**

Triangle :  $A(0, 0), B(2, 2), C(3, -2)$

Pieds des hauteurs :

sur  $[AC]$  :  $(0.46, -0.31)$ , sur  $[BC]$  :  $(2.35, 0.59)$ , sur  $[AB]$  :  $(0.5, 0.5)$

Orthocentre :  $H(0.8, 0.2)$

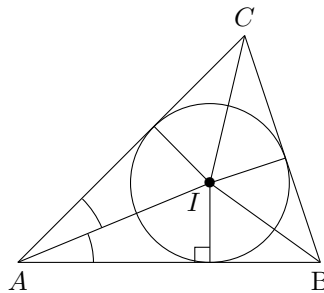
Angles droits :

sur  $[AC]$  :  $(0.8, -0.53), (1.02, -0.2), (0.68, 0.03)$

sur  $[BC]$  :  $(1.96, 0.49), (2.06, 0.1), (2.45, 0.2)$

### 1.5.7 Centre du cercle inscrit

Le centre du cercle inscrit est l'intersection des bissectrices.



**Aide :**

Triangle :  $A(0, 0), B(4, 0), C(3, 3)$

Cercle : centre  $I(2.54, 1.05)$ , rayon : 1,05

Points de contact :

sur  $[AB]$  :  $(2.54, 0)$ , sur  $[AC]$  :  $(1.8, 1.8)$ , sur  $[BC]$  :  $(3.54, 1.38)$

Marques d'angles (en  $A$ , l'angle est simple :  $45^\circ$ ) :

$BAI$  :  $(1, 0)$  arc  $(0:22:1)$ ,  $IAC$  :  $(1.11, 0.46)$  arc  $(23:45:1.2)$

Angle droit sur  $[AB]$  :  $(2.34, 0), (2.34, 0.2), (2.54, 0.2)$

## 1.6 Résumé

TikZ est un package pour L<sup>A</sup>T<sub>E</sub>X, déclaré par `\usepackage{tikz}`.

Une figure TikZ est engendrée par du code placé dans un environnement

`\begin{tikzpicture} ... \end{tikzpicture}`, suivant une syntaxe spéciale propre à TikZ.

Les points sont repérés par leurs coordonnées cartésiennes  $(x,y)$  ou par leurs coordonnées polaires  $(a:r)$

L'unité par défaut est 1 cm, mais on peut appliquer à toute la figure une échelle avec `[scale=k]` (l'unité devient  $k$  cm).

La principale commande de dessin est `\draw ... ;`

Tracer un segment : `\draw (a,b) -- (c,d) ;`

Tracer un cercle : `\draw (a,b) circle (r) ;`

Tracer un arc de cercle : `\draw (a,b) arc (u:v:r) ;`

Écrire un texte L<sup>A</sup>T<sub>E</sub>X : `\draw (a,b) node {texte} ;`

ou `\draw (a,b) node [position] {texte} ;` (position : `above`, `below`, etc.)

Le problème principal pour tracer une figure de géométrie est de déterminer les coordonnées des points, surtout pour les points auxiliaires servant aux annotations. Pour cela, il peut être pratique de s'aider d'un logiciel auxiliaire comme GeoGebra, qui est même capable d'engendrer du code TikZ.

# Chapitre 2

## Chemins, options graphiques

Dans ce chapitre, nous allons voir différentes façons de simplifier le tracé des figures par rapport aux méthodes de base vues dans le chapitre précédent. Nous introduirons la possibilité de *nommer* les points, de regrouper des traits élémentaires dans un *chemin*, de spécifier des positions par rapport à des traits et plus seulement à des points. D'autre part, nous présentons des options graphiques permettant de faire varier l'apparence des traits : pointillés, couleurs, remplissage. Et enfin nous introduirons des façons de préciser le cadre de la figure : axes, grille, fenêtre.

Ces outils permettent de réaliser de nombreuses figures de géométrie avec une présentation de qualité.

### 2.1 Simplifications, raccourcis, abstractions

Toutes les figures précédentes étaient construites point par point, trait par trait, et à partir de coordonnées qui devaient toutes être calculées au préalable. On peut trouver cela un peu fastidieux.

TikZ fournit différentes facilités pour simplifier les choses. Mais qui dit simplification dit nouveaux outils, nouvelle syntaxe, nouveaux concepts. Ces nouveautés ont un avantage et un inconvénient : l'avantage, c'est que le code à écrire sera un peu plus simple ou de plus haut niveau, l'inconvénient c'est que cela rajoute une charge mentale de mémoire, de compréhension, de décision. Il faut apprendre et comprendre les nouveautés et il faudra se demander lors de chaque figure quel va être le choix de l'outil le plus approprié. Ces nouveautés pourront même embrouiller rétrospectivement ce qu'on pensait avoir compris. Donc prudence : il n'est pas nécessaire de se précipiter sur toutes les nouveautés. Il faut que l'acquisition soit progressive.

#### 2.1.1 Nommage des points : `\coordinate(nom) at (x,y)`

Pour concevoir et vérifier une figure, il est plus agréable de pouvoir désigner les points par des noms plutôt que par des coordonnées.

Il y a plusieurs variantes syntaxiques. Nous conseillons la suivante, qui joue le rôle de déclaration, un peu comme dans un texte mathématique : « Soit  $A$  le point de coordonnées  $(1, 0)$  ». Cela s'écrit `\coordinate (A) at (1,0);`

```
\coordinate (nom) at (x,y) ;
```

Le nom peut être choisi librement, mais il ne doit pas contenir de signes de ponctuation ou de symboles spéciaux.

Si on reprend la figure du début, on peut déclarer les noms des points puis utiliser ensuite ces noms pour le tracé :

```
\begin{tikzpicture}
  \coordinate (O) at (0,0) ;
  \coordinate (A) at (1,0) ;
  \coordinate (B) at (0,1) ;
  \draw (O) -- (A) ;
```

```

\draw (O) -- (B) ;
\draw (O) circle (1) ;
\end{tikzpicture}

```

Apparemment, on n’y gagne pas en concision, puisqu’on rajoute des lignes. Mais on y gagne sur plusieurs points : en *lisibilité* d’abord (on voit clairement qu’on trace le segment  $[OA]$  par exemple), en *modularité* ensuite (on sépare nettement les positions particulières et la structure de la figure) et en *réutilisabilité* enfin (on peut utiliser plusieurs fois le même nom pour faire référence à la même position).

Cela facilite aussi les corrections : si on s’est trompé dans le calcul des coordonnées d’un point, il n’y a qu’un seul endroit à rectifier (la déclaration des coordonnées).

### 2.1.2 Enchaînement de traits : chemin, position courante

Jusqu’à maintenant, nous avons tracé les traits un par un, chacun dans une instruction `\draw`.

C’est un peu fastidieux quand il faut tracer des successions de traits qui s’enchaînent (des lignes brisées et polygones) car on est obligé d’écrire deux fois ou plus les points intermédiaires. `TikZ` introduit un raccourci pour cela : plutôt que d’écrire

```

\draw (0,0) -- (1,0) ;
\draw (1,0) -- (1,1) ;

```

On peut écrire en enchaînant les traits dans une même instruction :

```

\draw (0,0) -- (1,0) -- (1,1) ;

```

Cela s’appelle un *chemin* (path). Cette notion est introduite ici à propos d’une suite de segments, mais elle est beaucoup plus générale et forme une des bases de `TikZ`.

Syntaxiquement, elle se présente comme une suite d’*opérations de chemin* (path operations). Ici, il y en a trois :  $(0,0)$ , qui est une opération de positionnement explicite, puis  $-- (1,0)$  et  $-- (1,1)$ , qui sont des opérations de tracé de segment.

Ces dernières opérations ont un double rôle : un rôle de tracé d’abord, mais aussi un rôle de *positionnement implicite* après le tracé. En effet, le long d’un chemin, `TikZ` maintient une « position courante » : c’est la position qui résulte du tracé précédent et qui servira de référence (point de départ) pour le tracé suivant. D’une manière imagée, c’est la position du crayon.

Dans le cas d’un segment, cette position implicite est naturelle : après l’instruction de tracé  $-- (x,y)$ , la position courante devient  $(x,y)$ . Mais il y a d’autres instructions pour lesquelles la convention est moins évidente, ce qui peut être source de confusions.

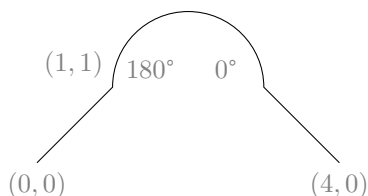
Par exemple, après le tracé d’un cercle par `circle (r)`, la position courante devient le centre du cercle.

Après le tracé d’un arc, la position courante devient l’extrémité de l’arc. Cette différence entre arc et cercle explique peut-être la convention inhabituelle choisie pour décrire un arc. Un arc peut être inclus dans une succession de lignes pour former une ligne continue :

```

\draw (0,0) -- (1,1) arc (180:0:1) -- (4,0);

```



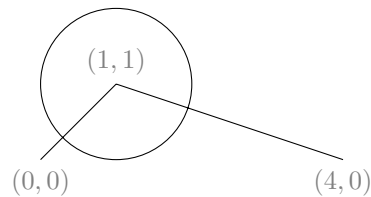
*Les indications données en gris ont été rajoutées pour bien montrer l’influence des paramètres, mais elles ne font pas partie du dessin proprement dit.*

Si on remplace l’arc par un cercle, on obtient :

```

\draw (0,0) -- (1,1) circle(1) -- (4,0);

```

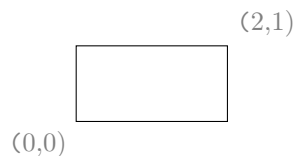


Après le tracé du cercle, le crayon est en son centre  $(1,1)$  et on repart de là pour joindre le point suivant  $(4,0)$ .

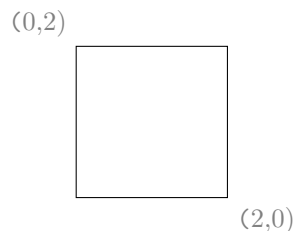
### 2.1.3 Rectangle : $(a,b)$ rectangle $(c,d)$

Il est fréquent d'avoir à tracer un rectangle dont les côtés sont parallèles aux axes. `TikZ` fournit une opération pour cela : `rectangle`, utilisée sous la forme  $(a,b)$  `rectangle`  $(c,d)$ , où  $(a,b)$  et  $(c,d)$  sont deux coins opposés du rectangle.

Exemple :  $(0,0)$  `rectangle`  $(2,1)$  ;



Autre exemple :  $(0,2)$  `rectangle`  $(2,0)$  ;



### 2.1.4 Figures fermées : `cycle`, `fill`

Pour tracer un triangle  $ABC$ , on peut toujours écrire :

```
\draw (a) -- (b) -- (c) -- (a);
```

Mais `TikZ` possède le concept de chemin fermé. Il est possible de dire : tracer de  $A$  à  $B$  puis à  $C$ , puis fermer la figure :

```
\draw (a) -- (b) -- (c) -- cycle;
```

Il y a deux intérêts à cela : d'abord dans la jonction des traits entre le départ du point initial et le retour au point initial (`TikZ` s'arrange pour que la jonction se fasse bien au pixel près) et ensuite dans le remplissage des figures fermées avec de la couleur ou des motifs.

Pour remplir une figure fermée, on utilise `fill` au lieu de `draw` :

```
\fill (0,0) -- (1,0) -- (1,1) -- cycle;
```



Le principal intérêt de la notion de chemin ne réside pas tellement dans l'économie d'écriture qu'elle permet (on économise quelques instructions `draw`), mais dans cette possibilité de définir des chemins fermés pour l'instruction `fill`. Elle a un autre avantage : elle permet de regrouper en un seul endroit les options graphiques qui s'appliquent sur tout le chemin (voir plus loin).

Nous allons voir plus loin comment remplir la figure avec des couleurs et des motifs.

### 2.1.5 Nœuds sur les traits : `midway`, `sloped`

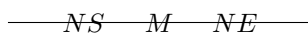
Dans ce qui précède, nous avons vu l'opération `node`, qui permet de placer une boîte de texte à une position donnée, avec éventuellement une option de positionnement :

```
\draw (0,0) node[below right] {$O$} node{$\bullet$};
```



On peut aussi placer un nœud pour annoter un trait, et non plus un point : pour cela on place l'opération `node` après l'extrémité du trait, avec une option de position relative au trait (`midway`, `near start`, `near end`). `TikZ` comprendra alors grâce à cette option que le nœud se rapporte au trait plutôt qu'au point.

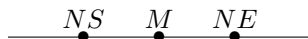
```
\draw (0,0) -- (4,0) node[midway] {$M$}
      node[near start] {$NS$}
      node[near end] {$NE$};
```



On peut ajouter les options usuelles de placement : `above`, `below`, etc. (ici : `above`), et ajouter d'autres nœuds (ici les points qui marquent les positions exactes) :

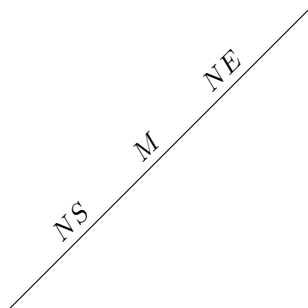
```
node[midway,above] {$M$} node[midway] {$\bullet$}
```

Cela permet par exemple de tracer rapidement le milieu  $M$  d'un segment.



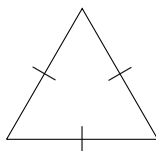
On peut aussi indiquer que le texte doit suivre l'orientation du trait : option `sloped`  

```
node[midway,above,sloped] {$M$}
```



Cette dernière possibilité est pratique pour noter des segments égaux par de petits traits. Au lieu de tracer ces petits traits explicitement à l'aide de coordonnées, on peut les considérer comme des nœuds associés aux segments (option `midway`), avec un contenu symbolique comme `$|` (un trait vertical `|`) et l'option `sloped` pour que le petit trait reste perpendiculaire au segment :  

```
node[midway, sloped]{$|}
```



### 2.1.6 Coordonnées relatives : `++(x,y)`

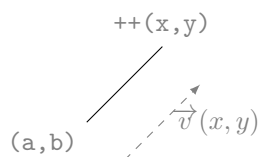
Il peut être pratique de désigner un point non pas par ses coordonnées absolues mais par un déplacement à partir du point courant.

La notation `\draw (a,b) -- ++(x,y);` est un raccourci qui remplace `\draw (a,b) -- (c,d)` par  $c = a + x$  et  $d = b + y$ .



Autrement dit, on joint le point  $(a, b)$  au point qui s'en déduit par la translation de vecteur  $(x, y)$ .

Ainsi, `\draw (1,2) -- ++(1,1)`; équivalent à `\draw (1,2)--(2,3)`;

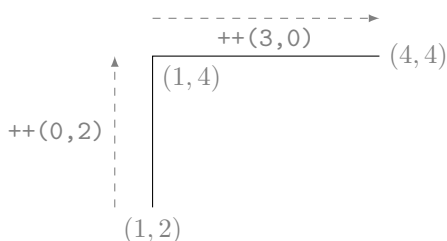


*Ici encore, les annotations en gris ne font pas partie de la figure proprement dite.*

Cette opération est intéressante pour décrire un chemin par étapes locales : pour trouver le trésor, partez du grand chêne en  $(1, 2)$ , puis faites 2 pas vers le nord, puis 3 pas vers l'est.

`\draw (1,2) -- ++(0,2) -- ++(3,0)`;

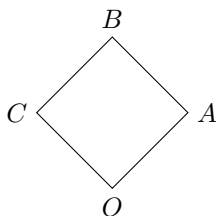
C'est équivalent à `\draw (1,2) -- (1,4) -- (4,4)`;



Cela peut être pratique pour tracer des polygones dont on connaît les vecteurs des côtés, par exemple un carré  $OABC$  :

`\draw (0,0) -- ++(1,1) -- ++(-1,1) --++ (-1,-1) -- cycle`;

Les coordonnées des vecteurs sont :  $\vec{OA}(1, 1)$ ,  $\vec{AB}(-1, 1)$ ,  $\vec{BC}(-1, -1)$



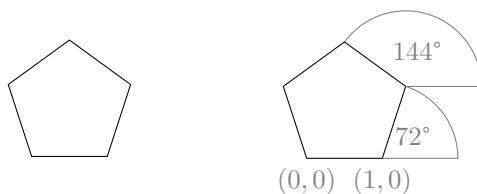
Cette dernière technique est particulièrement intéressante en utilisant les coordonnées polaires, par exemple pour tracer un pentagone régulier. L'angle entre deux côtés successifs d'un pentagone régulier est  $72^\circ$ , et les côtés ont tous la même longueur, prenons 1 par exemple.

Le premier côté est  $(0,0) -- (1,0)$ .

Pour tracer le deuxième côté, on peut écrire : `\draw -- ++(72:1)`. Cela revient à dire : tracer un segment dans la direction d'angle polaire  $72^\circ$ , sur une distance 1. Pour le troisième côté, il suffira de dire `-- ++(144:1)`, sachant que 144 est le double de 72, c'est l'angle polaire du troisième côté. Et ainsi de suite avec les multiples de 72 :

`\draw (0,0) -- (1,0)`

`-- ++(72:1) -- ++(144:1) -- ++(216:1) -- cycle`;



## 2.2 Décorations, styles, options graphiques

### 2.2.1 Options : [ ]

Une figure n'est pas constituée que de traits pleins entre des points. Il devient très vite nécessaire de modifier l'aspect des lignes : pointillés au lieu de traits pleins, couleur, épaisseur des traits, pointes de flèches, etc. Non pas pour le simple plaisir de faire joli, mais pour communiquer des informations à propos de la figure.

TikZ permet cela par l'intermédiaire d'*options graphiques*. Le concept et la syntaxe des options est analogue à ce qu'on trouve dans L<sup>A</sup>T<sub>E</sub>X : les options s'écrivent entre crochets, séparées par des virgules. Par exemple `[thick, red]` signifiera qu'on veut des traits épais et rouges. Remarquez qu'on a déjà utilisé cette syntaxe pour le positionnement des noeuds : dans l'expression `node[below]{}`, `below` était une option au sens indiqué ici.

Les questions qui se posent à propos de cette nouvelle construction syntaxique sont les suivantes :


- où peut-on (doit-on) écrire ces options ?
- quelles sont les options disponibles, avec quelle syntaxe ?
- quels sont les éléments de la figure affectés par l'option ?

Nous ne pouvons pas répondre d'un seul coup ni complètement à toutes ces questions. En gros, les options s'écrivent immédiatement après les différentes commandes comme `\tikzpicture` (pour une option s'appliquant à toute la figure), `\draw` (pour une option s'appliquant uniquement au chemin), et dans les opérations de chemin, comme `node`. Elles sont ... optionnelles, c'est-à-dire qu'on peut ne pas les faire figurer (dans ce cas TikZ choisira alors automatiquement des valeurs par défaut). En fait, chaque fois que nous introduirons une nouvelle possibilité de TikZ, nous indiquerons les principales options associées.

### 2.2.2 Épaisseur des traits : `thick`, `thin`, `line width=5pt`

À chaque opération `draw`, on peut spécifier l'épaisseur des traits, avec les options suivantes (fin ou épais) :

`thin`, `very thin`, `ultra thin`  
`thick`, `very thick`, `ultra thick`

Par exemple `\draw [very thick] (0,0) -- (1,0);` 

Remarquez que TikZ permet de s'exprimer de manière qualitative (en anglais), sans trop se préoccuper de précision absolue. C'est souvent suffisant, mais si on veut on peut aussi spécifier la largeur exacte du trait, avec des unités de longueur :

`[line width=5pt]`

On peut remarquer deux choses dans cette syntaxe : d'abord les noms des options peuvent contenir des espaces, et ensuite il y a deux formes possibles, l'une courte, comme `[thin]`, et l'autre longue comme `[line width=5pt]`. On différencie en fait le *nom* de l'option (`line width`) et la *valeur* de l'option (`5pt`). La plupart du temps, TikZ se débrouille tout seul pour trouver de quelle option on parle quand on donne une valeur seulement (quand on donne `[red]`, TikZ reconstitue la syntaxe complète `[color=red]`). En cas d'ambiguïté il faut préciser le nom de l'option.

### 2.2.3 Pointillés, styles des traits : `dotted`, `dashed`, `double`

On peut spécifier qu'un trait doit être en *pointillés* :

`dotted` 

Options possibles : `dotted`, `loosely dotted`, `densely dotted`

ou en *traitillés* :


`dashed` 

Options possibles : `dashed`, `loosely dashed`, `densely dashed`

On peut aussi spécifier qu'un trait doit être double :

`\draw [double] (0,0) -- (1,0) =====`

On peut même préciser l'écartement entre les deux traits :

`\draw [double distance = 5pt] (0,0) -- (1,0) `

### 2.2.4 Pointes de flèches : `->`, `>= stealth`

On peut spécifier que l'extrémité d'un trait doit être une pointe de flèche avec l'option `[->]`.

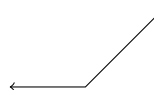
Si la flèche doit être à l'origine plutôt qu'à l'extrémité : `[<-]`.

Et on peut combiner les deux : `[<->]`.



Pour un chemin formé de traits contigus dans une même instruction `\draw`, la flèche s'applique à tout le chemin considéré comme un seul trait.

Exemple : `\draw [<->] (0,0) -- (1,0) -- (1,1);`



Les pointes de flèches par défaut sont un peu petites, et on peut en préférer d'autres. On peut alors spécifier le type de pointe qu'on veut avec l'option `>`, par exemple : `[>=stealth]`. Rappelons que les options s'accumulent entre crochets, séparées par des virgules : `[>=stealth,->]`

Le mot *stealth* évoque la forme de l'avion furtif appelé *stealth-fighter* (*stealth* = ruse).



Autre forme de flèche, plus triangulaire : `[>=latex]`



Une extension de *TikZ* permet d'avoir des pointes de flèches variées (library `arrows`)

### 2.2.5 Couleurs : `red`, `color=gray!20`

On peut indiquer qu'un trait ou une région doit être d'une couleur donnée en indiquant seulement le nom de la couleur comme option : `[red]` ou avec le nom d'option : `[color=red]`.

On peut spécifier une couleur pour toute la figure :

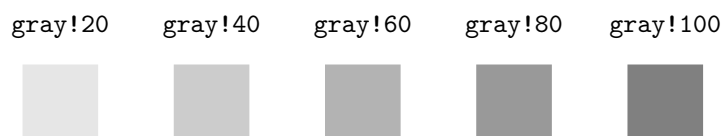
`\begin{tikzpicture}[red] ...` ou juste un trait `\draw[red]...`

Les noms de couleur autorisés sont les noms de couleur standard en *L<sup>A</sup>T<sub>E</sub>X*. Les couleurs de base sont :

`red`, `green`, `blue`, `cyan`, `yellow`, `magenta`, `black`, `white`, `gray`

On peut de plus ajouter une nuance à la couleur, le procédé consistant à suffixer le nom de couleur par un point d'exclamation suivi d'un nombre de 0 à 100.

Exemple : `[color=gray!20]`, signifie que la couleur est grise à 20% (plus le nombre est petit, plus la couleur est claire). Mais attention, syntaxiquement il faut alors indiquer obligatoirement le nom de l'option : `color=gray!20`



On peut distinguer la couleur de remplissage et la couleur du contour :

```
\draw [black,fill=gray!20]
```



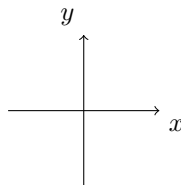
## 2.3 Axes, grille, fenêtre d'affichage

Dans les figures de géométrie la position précise des points est importante, et il est pratique de l'indiquer visuellement en traçant les axes ainsi qu'un quadrillage (en anglais *grid*, souvent traduit par *grille*).

### 2.3.1 Axes

*TikZ* ne possède pas d'instruction particulière pour tracer les axes. Un axe n'est rien d'autre qu'un segment qu'on trace entre deux points, comme tous les autres segments, en lui appliquant éventuellement des options graphiques (flèches, épaisseur, pointillés, couleur) et en ajoutant des annotations :

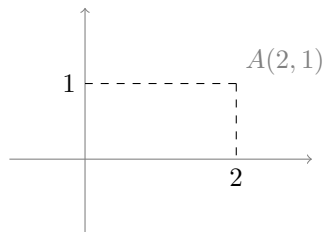
```
\begin{center}
\begin{tikzpicture}
\draw[->] (-1,0) -- (1,0);
\draw (1,0) node[right] {$x$};
\draw [->] (0,-1) -- (0,1);
\draw (0,1) node[above] {$y$};
\end{tikzpicture}
\end{center}
```



Une figure classique consiste à indiquer les coordonnées d'un point dans les axes.

Par exemple, pour le point  $A(2, 1)$ , il faut joindre le point  $A$  à ses projetés sur les axes  $(2, 0)$  et  $(0, 1)$ , et indiquer à côté de ces projetés l'abscisse 2 et l'ordonnée 1.

```
\draw [dashed] (2,1) -- (2,0) node[below] {$2$};
\draw [dashed] (2,1) -- (0,1) node[left] {$1$};
```



Plutôt que de tracer séparément les deux segments, il est possible de faire le dessin en une seule opération. On veut joindre le point  $(0, 1)$  au point  $(2, 0)$  par une ligne d'abord horizontale, puis verticale. *TikZ* possède une opération pour cela, notée  $-|$  :

```
\draw [dashed] (0,1) -| (2,0);
```

Ou, dans l'autre sens, on joint le point  $(2, 0)$  au point  $(0, 1)$  par une ligne d'abord verticale puis horizontale : `\draw [dashed] (2,0) |- (0,1);`

### 2.3.2 Quadrillage (grille) : grid

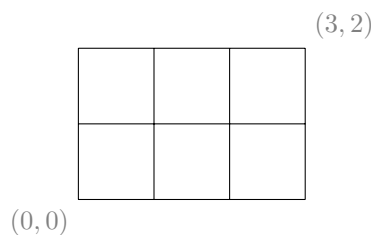
Pour rendre les coordonnées directement lisibles, il est souvent pratique d’afficher un quadrillage (une *grille*). TikZ fournit l’instruction `grid` pour cela.

La syntaxe est `\draw (a,b) grid (c,d);`

Cela dessine un quadrillage dont les points sont les points de coordonnées entières à l’intérieur du rectangle défini par deux coins opposés  $(a, b)$  et  $(c, d)$ .

Par défaut, les traits sont espacés de 1 cm.

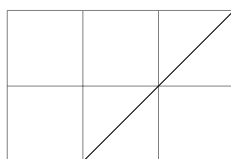
Par exemple `\draw (0,0) grid (3,2);`



Le problème d’un quadrillage, c’est qu’il risque d’embrouiller la figure. Il vaut donc mieux le dessiner discrètement. Pour cela, on peut utiliser les options `very thin` et `gray` (trait très fin et gris).

```
\draw [very thin, gray] (0,0) grid (3,2);
```

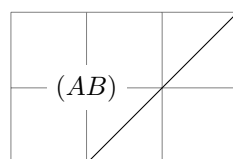
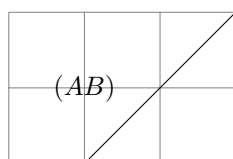
```
\draw (1,0) -- (3,2);
```



Cela peut ne pas suffire pour les nœuds de texte. On peut alors affecter à ces noeuds l’option `fill=white`. Cela colorie le fond du nœud en blanc (non transparent). Il faut alors tracer le noeud après la grille.

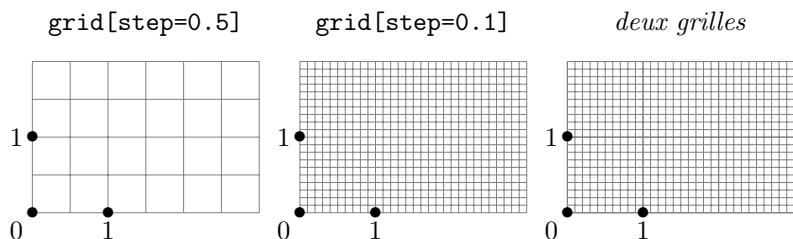
node

grid puis node[fill=white]



La distance par défaut de 1 cm peut parfois être insuffisante. On peut alors utiliser l’option `step` appliquée à l’opération `grid`.

Par exemple, il est possible d’obtenir un quadrillage de cahier d’écolier avec : `grid [step=0.5]`, et pour un papier millimétré : `grid [step=0.1]`. Et si on veut renforcer les traits des unités, on peut tracer une autre grille standard par-dessus, avec des traits un peu plus gros.



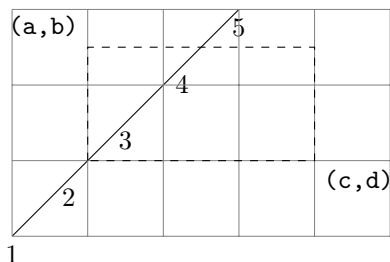
### 2.3.3 Fenêtre d'affichage : clip

Il se peut que, pour différentes raisons, on doive réduire la fenêtre d'affichage, c'est-à-dire la région qui sera effectivement affichée. On utilise alors l'instruction `\clip`.

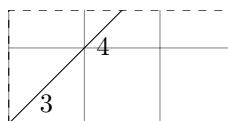
La syntaxe est `\clip (a,b) rectangle (c,d);`

La région effectivement affichée sera seulement l'intérieur du rectangle dont les coins opposés sont  $(a, b)$  et  $(c, d)$ . Cela affectera uniquement les dessins placés après l'instruction `\clip`.

Les pointillés ont été rajoutés pour l'exemple, ils ne font pas partie de l'instruction `\clip`.

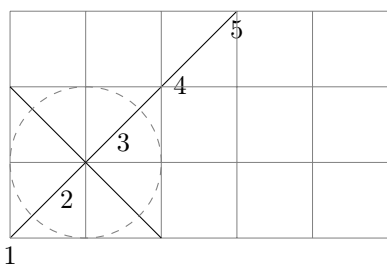


`\clip (a,b) rectangle (c,d)`

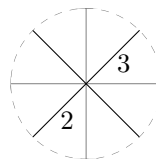


En fait, la fenêtre n'a pas forcément toujours une forme de rectangle. Elle a la forme qu'on définit à l'intérieur de l'instruction `\clip`. En particulier, elle peut avoir une forme circulaire.

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



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



## 2.4 Compléments : opacité, couleurs, styles

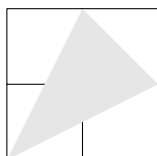
### 2.4.1 Ordre des tracés, transparence : opacity

Dans une figure, les traits se superposent dans l'ordre du tracé. Or certains tracés sont transparents et d'autres pas. Donc les tracés non transparents vont cacher les traits qui ont été tracés avant.

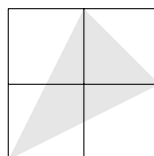
C'est le cas en particulier avec la grille `grid` et des commandes `\fill`.

La grille est transparente, mais un triangle rempli avec `\fill` ne l'est pas :

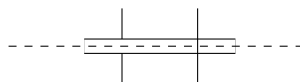
`grid, puis \fill`



`\fill, puis grid`



Autre exemple, dans l'option `double`, qui trace un trait double, en fait le trait est un rectangle et la région entre les deux traits n'est pas transparente. Il faut éventuellement y penser pour l'ordre de tracé des traits, sachant que les différents traits se dessinent l'un après l'autre, c'est-à-dire l'un sur l'autre. Dans l'exemple suivant, le trait vertical de gauche est tracé avant le trait horizontal double, et le trait vertical de droite est tracé après. Le trait horizontal en traitillé est tracé après, et montre comment le trait double s'aligne : il est centré sur la ligne du trait simple.



Il existe une option pour gérer la transparence ou plutôt l'opacité : `opacity`, avec des valeurs allant de 0 (transparent) à 1 (opaque).

On peut reprendre le premier dessin (la grille suivie d'un triangle rempli de gris) en appliquant `[gray,opacity = 0.5]` au triangle. On obtient :

`grid, puis \fill[gray,opacity=0.5]`      `opacity=0.2`



Le triangle est bien tracé par-dessus la grille, mais il est en partie transparent. Plus précisément, on peut distinguer `draw opacity` l'opacité pour le contour et `fill opacity` l'opacité pour le remplissage, sachant que `opacity` fixe les deux à la fois.

## 2.4.2 Noms et calculs des couleurs, package xcolor

Si on veut disposer de noms de couleur plus variés qu'en L<sup>A</sup>T<sub>E</sub>X standard, on peut charger le package `xcolor`. Il est déclaré par `\usepackage[usenames, dvipsnames]{xcolor}`.

Attention, il faut le déclarer avant `\usepackage{tikz}`, sinon il y a incompatibilité.

On dispose alors des couleurs suivantes :

- *De jaune à orange* : GreenYellow, Yellow, Goldenrod, Dandelion, Apricot, Peach, Melon, YellowOrange;
- *De orange à rouge* : Orange, BurntOrange, Bittersweet, RedOrange, Mahogany, Maroon, BrickRed, Red;
- *De rouge à rose* : OrangeRed, RubineRed, WildStrawberry, Salmon, CarnationPink, Magenta, VioletRed, Rhodamine;
- *De rose à violet* : Mulberry, RedViolet, Fuchsia, Lavender, Thistle, Orchid, DarkOrchid, Purple;
- *De violet à bleu* : Plum, Violet, RoyalPurple, BlueViolet, Periwinkle, CadetBlue, CornflowerBlue, MidnightBlue;
- *De bleu à bleu clair* : NavyBlue, RoyalBlue, Blue, Cerulean, Cyan, ProcessBlue, SkyBlue, Turquoise;
- *De bleu clair à vert* : TealBlue, Aquamarine, BlueGreen, Emerald, JungleGreen, SeaGreen, Green, ForestGreen;
- *De vert à brun* : PineGreen, LimeGreen, YellowGreen, SpringGreen, OliveGreen, RawSienna, Sepia, Brown, Tan.

Sinon, on peut fabriquer soi-même ses propres couleurs en faisant des mélanges :

`[color=blue!30!red]` désigne une couleur avec 30% de bleu et 70% de rouge. C'est donc une couleur plutôt rouge, légèrement violette.

De même `[color=blue!30!red!40!green]` désigne le mélange de la couleur précédente en proportion de 40%, avec 60% de vert. Cela donne une définition globale dans le système RVB, avec des valeurs de 0 à 255 :  $R = 31, B = 71, V = 153$  (calcul de barycentres). Le résultat est plutôt vert.

Cela donne une grande liberté, mais il n'est peut-être pas très raisonnable de vouloir à tout prix inventer ses propres couleurs. C'est un exercice difficile, et les coloristes ont établi des palettes soigneusement étalonnées qu'il est préférable d'utiliser. Les 64 couleurs de `xcolor` plus les couleurs standard devraient suffire.

### 2.4.3 Définition de styles : `\tikzstyle`, `\tikzset`

Il est fréquent que plusieurs éléments d'une figure aient les mêmes options graphiques, et que ces options soient associées à une convention particulière de représentation, qu'on appelle un *style*. Il est alors pratique de séparer la définition du style, qui est commune, et ses applications aux divers éléments.

Par exemple, pour que les traits soient en pointillés épais et que la couleur de remplissage soit en gris, on écrit habituellement :

```
\draw[thick, dashed, fill=gray] ...
```

On peut alors désigner ce style par un nom, par exemple `grisEncadre`. On déclare alors ce nom à TikZ de la façon suivante :

```
\tikzstyle{grisEncadre}=[thick, dashed, fill=gray!20]
```

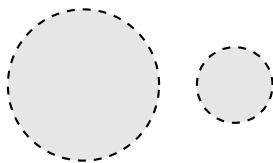
On peut écrire cette commande n'importe où dans le document L<sup>A</sup>T<sub>E</sub>X, avant la figure.

Ensuite, on peut utiliser ce nom dans la figure de la manière suivante, en écrivant seulement le nom comme option, et cela jouera le rôle d'une abréviation :

```
\draw [grisEncadre] (0,0) circle (1);
```

Et on peut utiliser ce même style plusieurs fois :

```
\tikzstyle{grisEncadre}=[thick, dashed, fill=gray!20]
\begin{tikzpicture}
\draw [grisEncadre] (0,0) circle (1);
\draw [grisEncadre] (2,0) circle(0.5);
\end{tikzpicture}
```



Cette façon de faire augmente la lisibilité et facilite les corrections : en corrigeant seulement la définition du style, toutes les figures utilisant ce style seront modifiées automatiquement.

En fait, la commande `\tikzstyle` semble tombée en désuétude dans la version 2 (même si elle est toujours correcte et utilisable).

La méthode utilisée par la version 2 utilise la commande `\tikzset` pour définir un style, avec une syntaxe un peu plus compliquée :

```
\tikzset{grisEncadre/.style={thick, dashed, fill=gray!20}}
```

Cette légère complication permet de profiter des possibilités du package `pgfkeys`, que nous ne détaillons pas dans cet ouvrage.

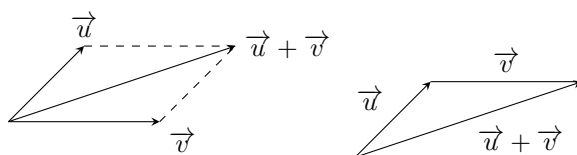
## 2.5 Exercices : styles de traits, flèches, couleurs

Voici des exercices à réaliser avec TikZ. Pour chacun d'eux, on donne une figure et il faut écrire le code TikZ qui trace cette figure. On donne comme indications les techniques utilisées, c'est-à-dire les mots-clés TikZ utilisés.

Les figures sont des figures classiques de géométrie.

### 2.5.1 Somme de deux vecteurs

La somme de deux vecteurs s'obtient en traçant la diagonale d'un parallélogramme, ou en plaçant les vecteurs bout à bout.

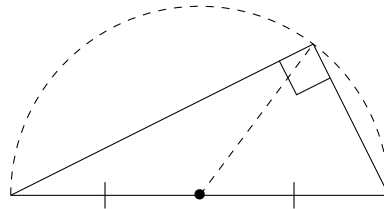




Techniques utilisées : `->`, `>= stealth`, `--`, `node`, `above`  
`below right`, `dashed`, `midway`

### 2.5.2 Triangle rectangle inscrit dans un demi-cercle

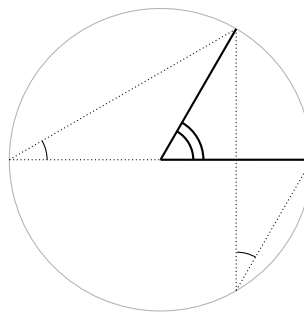
Tout triangle rectangle est inscrit dans un demi-cercle dont le diamètre est l'hypoténuse.



Techniques utilisées : `dashed`, `arc`, `node`, `--`

### 2.5.3 Angle inscrit et angle au centre

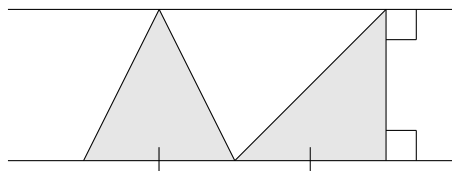
L'angle au centre est le double de l'angle inscrit.



Techniques utilisées : `double distance`, `thick`, `arc`,  
`color=gray!20`, `circle`, `--`, `densely dotted`

### 2.5.4 Parallèles, aires égales

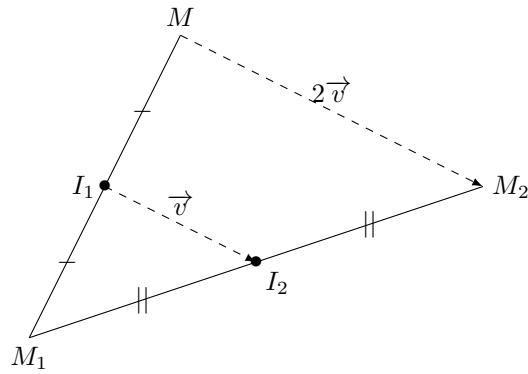
Deux triangles, de bases égales et de hauteurs égales, ont des aires égales.



Techniques utilisées :  
`color=gray!20`, `--`, `cycle`, `node`, `\fill`

### 2.5.5 Composée de deux symétries centrales

La composée  $s_2 \circ s_1$  de deux symétries centrales de centres  $I_1$  et  $I_2$  est la translation de vecteur  $2\overrightarrow{I_1I_2}$ . Si  $M_1 = s_1(M)$  et  $M_2 = s_2(M_1)$ , alors  $\overrightarrow{MM_2} = 2\overrightarrow{I_1I_2}$ .



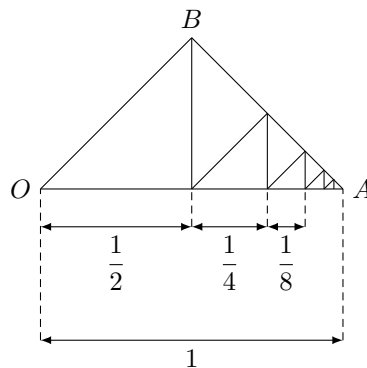
Techniques utilisées : `>=latex`, `node`, `below`, `midway`, `--`, `above`, `dashed`, `left`, `below right`, `->`

### 2.5.6 Suite géométrique

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = \lim_{n \rightarrow +\infty} \sum_{k=1}^n \frac{1}{2^k} = 1$$

Le triangle  $OAB$  est isocèle rectangle en  $B$ , direct. À partir de  $B$ , on effectue une suite de projections orthogonales : sur  $[OA]$  d'abord, puis sur  $[AB]$ , puis sur  $[OA]$ , puis sur  $[AB]$ , etc.

Les longueurs des segments sur  $[OA]$  forment une suite géométrique de raison  $\frac{1}{2}$ .



Techniques utilisées :

`>=latex`, `<->`, `node`, `left`, `above`, `--`, `right`, `midway`, `below`, `densely dashed`

# Chapitre 3

## Courbes

### 3.1 Tracer une courbe : plot (...)

Dans ce chapitre, nous allons voir comment tracer des courbes définies à l'aide d'équations mathématiques, comme l'équation cartésienne  $y = \sin(x)$ , l'équation polaire  $r = f(\theta)$  ou les

équations paramétriques  $\begin{cases} x = \cos(t) \\ y = \sin(t) \end{cases}$ .

L'opération principale pour cela est `plot`, à insérer dans une commande `\draw` par exemple :

```
\draw plot ... ;
```

Nous décrivons ici la syntaxe de cette opération dans la version 2 de `TikZ`.

Cette opération permet de tracer des courbes paramétrées, le nom du paramètre par défaut étant `\x`.

Après le mot `plot`, on écrit :

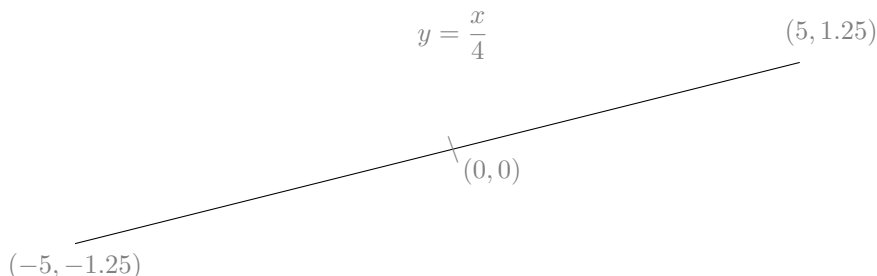
- ou bien  $(X, Y)$ , où  $X$  et  $Y$  sont les formules de l'abscisse et de l'ordonnée en fonction du paramètre, Si ces formules contiennent des parenthèses ou des virgules, il faut les écrire entre accolades. C'est très souvent le cas pour  $Y$ . Par exemple, on écrit `(\x, {\exp(\x)})` ;
- ou bien  $(\theta : r)$ , où  $\theta$  et  $r$  sont les formules de l'angle polaire (en degrés) et du rayon en fonction du paramètre (rayon algébrique, qui peut être négatif). Les accolades peuvent être nécessaires également.

Par exemple, pour tracer la courbe de la fonction d'équation cartésienne  $y = \frac{x}{4}$ , il faut d'abord

la traduire en équations paramétriques :  $\begin{cases} x = x \\ y = \frac{x}{4} \end{cases}$ , et on écrit :

```
\draw plot (\x, \x/4);
```

ce qui donne :



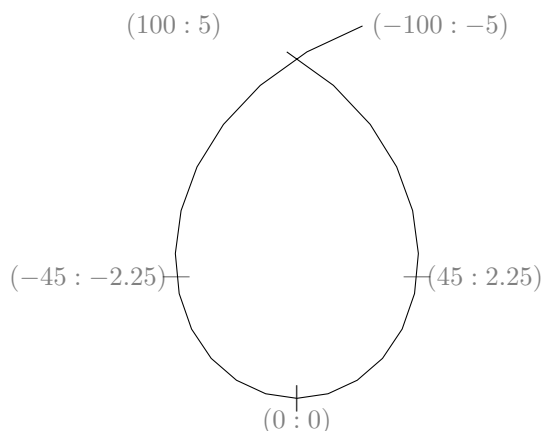
On a ajouté des annotations en gris pour montrer comment la figure se place.

Ici, `TikZ` a utilisé ses options par défaut pour choisir le domaine du paramètre :  $-5 \leq x \leq 5$ .

Autre exemple : la courbe d'équation polaire  $\theta = 20 \times r$ . On peut la représenter en choisissant  $r$  comme paramètre, ce qui s'écrit :

```
\draw plot (20*\x:\x);
```

La courbe part du point ( $\theta = -100^\circ : r = -5$ ) et arrive au point ( $\theta = 100^\circ : r = 5$ ) car  $\backslash x$  varie par défaut de  $-5$  à  $5$ . On obtient :



Si on ne souhaite pas conserver le nom par défaut  $\backslash x$  pour le paramètre, on peut lui attribuer un nom, par exemple  $\backslash t$ , avec l'option `[variable= $\backslash t$ ]` appliquée à l'opération `plot`.

Exemple : `\draw plot [variable= $\backslash t$ ] ( $\backslash t, 2*\backslash t$ );`

Il reste différentes questions, que nous allons étudier dans la suite :

- Comment définir le domaine du paramètre ?
- Quelles sont les formules mathématiques disponibles ?
- Comment spécifier le nombre de points calculés, le lissage ?
- Comment sont traités les points singuliers (discontinuités, valeurs trop grandes) ?

### 3.1.1 Domaine : `[domain=a:b]`

Pour dire que le paramètre ( $\backslash x$  par défaut) varie de  $a$  à  $b$ , *TikZ* a prévu la syntaxe `[domain=a:b]`.

Malheureusement, cette syntaxe pose un problème pour les utilisateurs français qui ont déclaré le package `babel` avec une option `french`, `frenchb` ou `français`.

#### Le problème de `babel` français et de « : »

**Remarque :** Dans la dernière version de *TikZ* & *PGF* (la version 2.10), qui est actuellement fournie avec la distribution *TeXlive* 2011, ce problème de `babel` du « : » est corrigé.

Si on dispose d'une version plus ancienne, lire la suite.

Avec le package `babel` français (`french`, `frenchb` ou `français`), le code interne *TeX* du caractère « : » (son *catcode*) est modifié pour permettre un traitement spécial adapté aux règles typographiques de la langue française. Cela ne pose pas de problèmes en général dans *L<sup>A</sup>TeX*, mais comme *TikZ* utilise une syntaxe spéciale et donc une procédure de lecture spéciale, cela interfère avec ses règles syntaxiques et provoque une erreur, en particulier dans l'option `[domain=a:b]`. On obtient en général dans la fenêtre de console le message suivant, difficile à comprendre, mais qui peut servir d'indice pour repérer l'erreur :

```
Paragraph ended before \tikz@plot@samples@recalc was complete
```

Nous proposons deux solutions différentes :

- dire à *L<sup>A</sup>TeX* de ne plus considérer le caractère « : » comme un caractère spécial dans les passages où il doit être interprété par *TikZ*, en utilisant la commande fournie par `babel` français : `\shorthandoff{:}` ;
- configurer *TikZ* pour modifier la syntaxe de l'option `domain=a:b`, ou plutôt introduire une autre option équivalente `domaine={a}-{b}` qui n'utilise pas les deux-points.

Aucune des deux solutions n'est parfaite, nous les développons dans ce qui suit.

**Désactiver « : » avec `\shorthandoff{:}`**

On peut dire à  $\text{\LaTeX}$  de ne plus considérer le caractère « : » comme un caractère spécial dans les passages où il doit être interprété par  $\text{\TikZ}$ . La commande pour cela est :

```
\shorthandoff{:}
```

Où exactement placer cette commande ? Répondre précisément et complètement à cette question est difficile, parce qu'il s'agit d'interactions à un niveau de base de  $\text{\TeX}$  (la lecture des caractères et l'exécution des commandes), et que des effets inattendus peuvent se produire.

La plupart du temps il suffit d'insérer la commande à l'intérieur de chaque environnement `tikzpicture` :

```
\begin{tikzpicture} \shorthandoff{:} ... \end{tikzpicture}
```

Cependant, cela ne suffit pas si la `tikzpicture` est elle-même à l'intérieur d'une autre commande, comme quand on veut encadrer une `tikzpicture` par `\fbox`. On obtient le message d'erreur suivant :

```
Argument of \tikz@plot@samples@recalc has an extra }
```

Dans ce cas-là, on peut essayer d'écrire :

```
{ \shorthandoff{:}
 \fbox{ \begin{tikzpicture} ... \end {tikzpicture} } }
```

**Introduire une autre option `[domaine={a}{b}]`**

Nous proposons d'introduire l'option `domaine` (c'est un mot français rappelant que le problème se pose pour les utilisateurs de `babel` en français).

Pour la définir, il suffit d'écrire la ligne suivante à la fin du préambule  $\text{\LaTeX}$  du document :

```
\tikzset { domaine/.style 2 args={domain=#1:#2} }
```

Alors, dans une `tikzpicture`, au lieu d'écrire `[domain=a:b]`, on écrira

```
[domaine={a}{b}]
```

Cette solution est pratique quand on écrit soi-même le code de ses figures  $\text{\TikZ}$ . Elle ne règle cependant pas tous les problèmes quand on veut récupérer un code  $\text{\TikZ}$  qui utilise directement l'option `domain` originale. Dans ce cas-là, il vaut mieux utiliser la solution avec `\shorthandoff{:}`.

**Utiliser le package `microtype`**

Un package récent, `microtype`, permet de résoudre le problème globalement de façon simple.

Il suffit d'ajouter :

```
\usepackage [babel=true,kerning=true]{microtype} après la déclaration d'utilisation du pa-
```

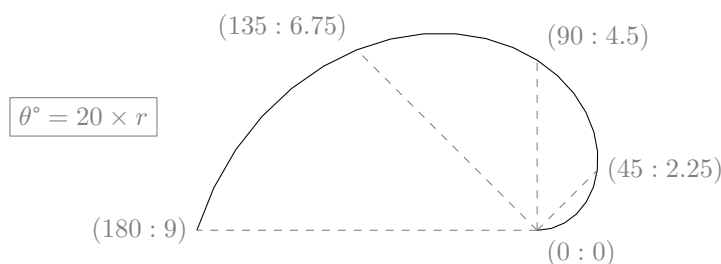
ckage `babel`. Ainsi le caractère « : » ne semble plus poser de problème.

*Attention* : nous supposons, à partir de maintenant, que toutes ces précautions sont prises quand nous utilisons `[domain=a:b]` dans les exemples.

**Exemples de domaines**

Avec les précautions concernant « : », voici comment on peut modifier le domaine de la courbe polaire précédente (d'équation  $\theta = 20 \times r$ ), par exemple en le fixant à  $0 \leq x \leq 90$ . On peut également en profiter pour modifier l'échelle (diviser les dimensions par 2 avec `scale=0.5`).

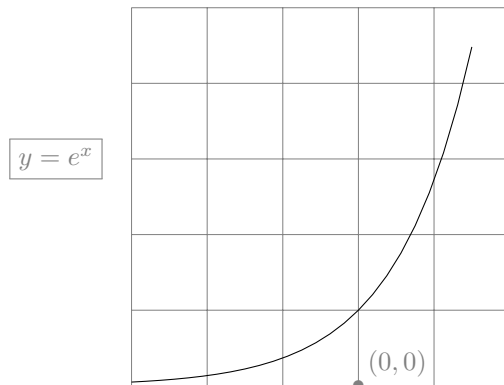
```
\draw [domain=0:9, scale=0.5] plot (20*\x:\x);
```



Le bon réglage du domaine est important pour les fonctions qui prennent de très grandes valeurs, par exemple l'exponentielle.

Avec `\draw plot(\x,{exp(\x)});`, on obtient un trait quasiment vertical qui prend toute la page, car  $e^5 \approx 148$ .

Il vaut mieux écrire `\draw [domain=-3:1.5] plot(\x,{exp(\x)});`



### 3.1.2 Formules mathématiques disponibles

Depuis la version 2, *TikZ* permet d'écrire des formules mathématiques avec une syntaxe usuelle analogue à celle des calculatrices, des logiciels mathématiques ou des langages de programmation.

Cependant, la gamme de fonctions disponibles n'est pas très grande. *TikZ* n'est pas un logiciel de calcul, et il ne faut pas lui en demander trop.

Si les fonctions disponibles ne suffisent pas, il y a deux grandes solutions :

- utiliser l'opération `plot function` qui fait appel (de manière transparente) au logiciel libre Gnuplot, qu'il faut avoir préalablement installé et connecté à *TikZ*. Voir plus loin « compléments techniques » pour des précisions sur Gnuplot ;
- préparer les données (coordonnées des points de la courbe) à l'aide d'un logiciel extérieur, les sauvegarder sur fichier sous un format convenable, puis utiliser l'opération `plot file` (voir chapitre suivant).

#### Opérations

Addition, soustraction, multiplication, division, élévation à une puissance :

`x+y`, `x-y`, `x*y`, `x/y`, `x^y`

Modulo, maximum, minimum :

`mod(x,y)`, `max(x,y)`, `min(x,y)`

#### Fonctions

Valeur absolue, exponentielle, logarithme népérien, racine carrée :

`abs(x)`, `exp(x)`, `ln(x)`, `sqrt(x)`

Arrondi, partie entière (plancher), entier immédiatement supérieur (plafond) :

`round(x)`, `floor(x)`, `ceil(x)`

#### Fonctions trigonométriques

Les angles sont en degrés.

`sin(x)`, `cos(x)`, `tan(x)`, `cot(x)`, `sec(x)`, `cosec(x)`

Fonctions réciproques :

`asin(x)`, `acos(x)`, `atan(x)`

Constante  $\pi$ , conversions

`pi`, `x r`, `deg(x)`, `rad(x)`

En TikZ, les fonctions trigonométriques attendent des angles en degrés. Pour retrouver les fonctions mathématiques habituelles qui attendent des angles en radians, il faut convertir les radians en degrés.

Appelons  $s$  la fonction notée en TikZ par `sin` et continuons à appeler `sin` la fonction mathématique habituelle en radians. On a alors les relations :

$$\sin(x) = s\left(\frac{180}{\pi}x\right) \text{ et } s(x) = \sin\left(\frac{\pi}{180}x\right)$$

Multiplier  $x$  par  $\frac{180}{\pi}$  (c'est-à-dire convertir les radians en degrés) peut se faire de deux façons en TikZ : `deg(x)` ou `x r`.

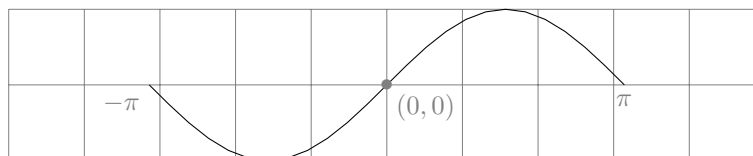
Par exemple `sin(pi/2 r)` vaut 1, ainsi que `sin(deg(pi/2))`

Mais attention au niveau de priorité de la notation `r` : elle est de même priorité qu'une multiplication. Elle représente en effet la multiplication par  $\frac{180}{\pi}$ .

Autrement dit `a*b r` représente  $ab \times \frac{180}{\pi}$  mais `a+b r` représente  $a + \left(b \times \frac{180}{\pi}\right)$ .

La courbe habituelle de la fonction sinus sur  $[-\pi; \pi]$  est obtenue par

```
\draw [domain=-pi:pi] plot (\x,{sin(\x r)});
```



La fonction sinus en radians sur  $[-\pi; \pi]$

Inversement, pour convertir de degrés en radians (c'est-à-dire multiplier par  $\frac{\pi}{180}$ ), utiliser `rad(x)`

### Nombres aléatoires

Un réel aléatoire entre 0 et 1 : `rnd`

Un réel aléatoire entre  $-1$  et 1 : `rand`

### Opérations booléennes

Tests d'égalité et d'inégalités :

`x == y`, `x < y`, `x > y`

Le résultat est 1 pour *vrai* et 0 pour *faux*.

La manuel ne précise pas avec quelle approximation numérique ces tests sont effectués.

## 3.2 Aspect du graphe

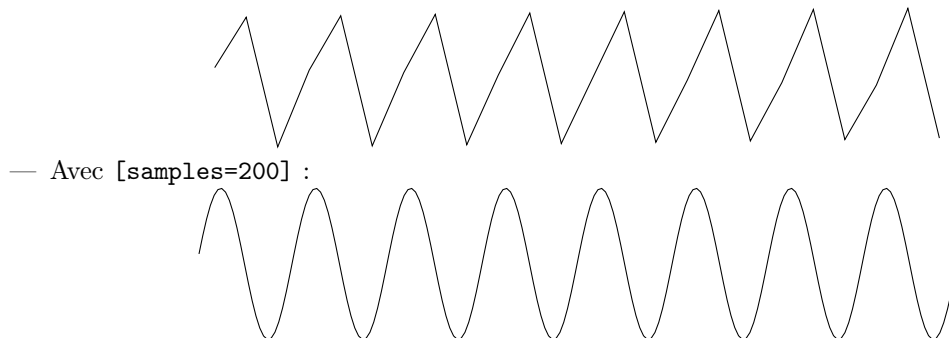
### 3.2.1 Nombre de points : samples

Par défaut, lorsqu'on demande à TikZ de tracer une courbe avec `plot`, il calcule un certain nombre de points de la courbe (25 points) puis il les joint par des segments. On peut changer ces choix par des options.

On peut changer le nombre de points avec l'option `samples`.

Par exemple, pour la courbe dont la formule TikZ est `(\x,{sin(5*\x r)})`, c'est-à-dire la courbe de la fonction mathématique  $x \mapsto \sin(5x)$

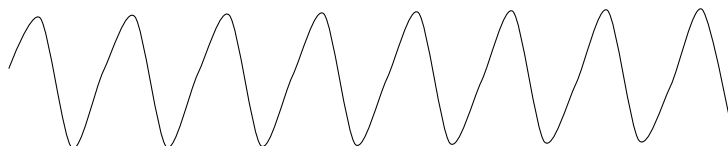
— Avec la valeur de `samples` par défaut (25) :



### 3.2.2 Lissage : `smooth`, `tension`

On peut demander à TikZ de ne pas joindre les points par des segments mais par des courbes (calculées avec un algorithme de lissage). On obtient alors une courbe lissée.

La courbe de la fonction précédente  $x \mapsto \sin(5x)$  avec 25 points, mais lissée avec l'option `[smooth]` est moins cahotique, mais ne compense pas l'effet dû au faible nombre de points :



Le degré de lissage (la courbure des arcs de courbes) peut être configuré avec l'option `tension`. Une valeur de 0 donne un segment (comme si on n'avait pas mis l'option `smooth`), la valeur par défaut est 0.55, et le maximum est la valeur 1 (plus la valeur est grande, plus les arcs sont courbés).

Avec `[smooth, tension=1]` :



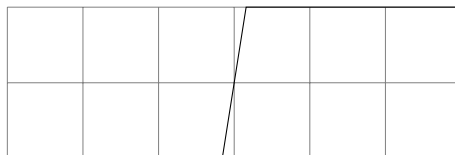
On voit que, de toutes façons, le meilleur effet de lissage est obtenu en augmentant le nombre de points avec `samples`. Cependant, l'option `smooth` peut quand même être efficace pour des courbes simples et régulières.

### 3.2.3 Discontinuités : on peut séparer les intervalles

Exemple : tracer la courbe de la fonction  $x \mapsto \frac{x}{|x|}$ . La fonction n'est pas définie en 0, et pas non plus prolongeable par continuité en 0. La courbe est formée de deux demi-droites :  $y = -1$  pour  $x < 0$  et  $y = 1$  pour  $x > 0$ .

Quand on écrit l'équation directement, TikZ ne donne pas un résultat correct (contrairement par exemple à GeoGebra) :

```
\draw[domain=-3:3] plot(\x,{\x/abs(\x)});
```

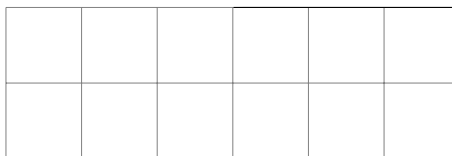


TikZ joint systématiquement les points qu'il calcule.

La solution usuelle est de tracer séparément deux morceaux de courbes, sur deux intervalles qui évitent le point de discontinuité, mais quand même très proches de ce point, par exemple  $[-3, -0.01]$  et  $[0.01, 3]$ .



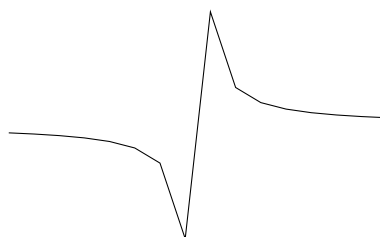
```
\draw[domain=-3:-0.01] plot(\x,{\x/abs(\x)});
\draw[domain=0.01:3] plot(\x,{\x/abs(\x)});
```



### 3.2.4 Grandes valeurs : `scale`, `\clip`

Prenons l'exemple de la fonction  $x \mapsto \frac{1}{x}$ , tracée à l'échelle 0.5, avec le nombre de points par défaut (25).

```
\draw [scale=0.5] plot (\x, {1/\x});
```



Le « dernier » point à gauche de 0 ne devrait pas être relié au « premier » point à droite de 0 parce que le domaine de définition de la fonction n'est pas connexe.

Si on voulait augmenter le nombre de points (`[samples=100]`), ce serait encore pire : on obtiendrait un message d'erreur : *Dimension too large*. Les valeurs au voisinage de 0 sont en effet trop grandes pour les capacités de calcul de *TikZ* (sans compter le problème de la valeur en 0).

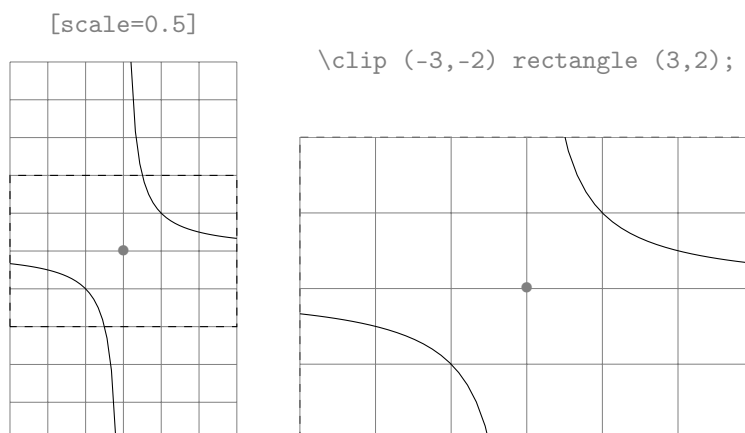
La seule façon de résoudre le problème est d'étudier la fonction avant, d'analyser les problèmes de discontinuité et de valeurs trop grandes, et de tracer séparément deux morceaux de courbe sur des domaines différents, en prenant soin d'éviter les abscisses trop proches de 0.

On peut alors essayer `[domain=-3:-0.2]` pour l'une et `[domain=0.2:3]` pour l'autre. Les courbes se tracent mais les valeurs extrêmes des ordonnées restent un peu grandes (−5 et 5), ce qui donne une image de 10 cm de hauteur.

Il y a deux sortes de solutions à cela : l'option `scale` ou la commande `\clip`.

Avec `scale`, on peut réduire l'échelle, par exemple `[scale=0.5]`.

Avec `\clip`, on garde l'échelle mais on peut réduire la fenêtre d'affichage, par exemple : `\clip (-3,-2) rectangle (3,2);` (commande à placer au début de la figure).



Il peut paraître un peu décevant que *TikZ* ne prenne pas en charge ces problèmes de discontinuité et de valeurs trop grandes, contrairement à certaines calculatrices ou logiciels. Mais encore

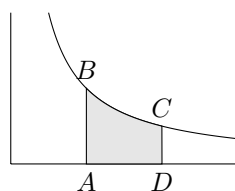
une fois, ce n'est qu'un module pour  $\text{T}_\text{E}_\text{X}$ , qui n'est après tout qu'un logiciel de composition typographique et n'est pas du tout prévu pour être un logiciel de calcul. Il est même remarquable que  $\text{T}_\text{E}_\text{X}$  arrive à supporter toutes ces extensions de manière robuste.

### 3.3 Régions limitées par des courbes

Il arrive souvent qu'on doive représenter une région limitée par des courbes, par exemple pour des calculs d'aires. La technique de base est de définir le contour à l'aide d'un chemin, en enchaînant les courbes, puis de demander à remplir ce chemin avec `fill`.

#### 3.3.1 Une courbe et des segments : `cycle`, `\fill`, `\filldraw`

Exemple : remplir la région comprise entre l'axe des abscisses, la courbe de la fonction  $x \mapsto \frac{1}{x}$  et les droites d'équation  $x = 1$  et  $x = 2$ .



Il s'agit de remplir la région  $ABCD$ , avec  $A(1, 0)$ ,  $B(1, 1)$ ,  $C(2, 0.5)$ ,  $D(2, 0)$ .

Pour cela on définit un chemin fermé avec dans l'ordre : le segment de  $A$  à  $B$ , la courbe de  $B$  à  $C$  et les segments de  $C$  à  $D$ , puis de  $D$  à  $A$ .

Le code suivant doit être inclus dans un environnement `tikzpicture` en n'oubliant pas d'éviter le problème des deux-points grâce à `\shorthandoff{.}`. Notez bien que tous les éléments du chemin sont connectés les uns aux autres (y compris la courbe) par des tirets `--` pour indiquer un chemin connexe, et que le chemin est fermé par `-- cycle`.

```
\fill[color=gray!20]
(1,0) -- (1,1)           % segment de A à B
-- plot [domain=1:2] (\x,1/\x) % courbe de B à C
-- (2,0) -- cycle;      % segment de C à D puis fermer
```



Il y a un sous-entendu concernant la position implicite du crayon après la tracé d'une courbe par `plot` : le crayon est placé au dernier point tracé par cette opération, ici le point  $C(2, 0.5)$ , sachant que les points sont tracés dans l'ordre du domaine `[domain=1:2]`, c'est-à-dire de 1 à 2. Le même morceau de courbe aurait pu être tracé de 2 à 1 par `[domain=2:1]`, mais alors il aurait été parcouru dans l'autre sens et la position finale du crayon aurait été  $B(1, 1)$ .

On souhaite souvent tracer le contour également. Au lieu de `\fill`, on peut utiliser `\filldraw`, qui remplit et trace le contour. Mais il faut spécifier deux couleurs différentes, avec les options `fill` et `draw` :

```
\filldraw [fill=gray!20,draw=black]
```



Cette région n'est qu'un élément de la figure complète, avec les axes et la courbe. Pour des raisons d'ordre du tracé et de recouvrement, il est préférable de tracer la région avant.

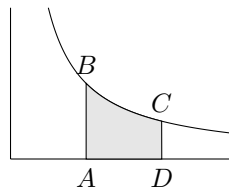
La structure de la figure complète est donc :

```

\begin{tikzpicture}
%\shorthandoff{:}      % protection des deux-points pour domain
\filldraw ... ;      % la région ABCD
\draw ... ;          % l'axe des abscisses de 0 à 3
\draw ... ;          % l'axe des ordonnées de 0 à 2
\draw plot ... ;    % la courbe complète, de 0.5 à 3
\draw ... ;          % les noeuds de texte A, B, C, D
\end{tikzpicture}

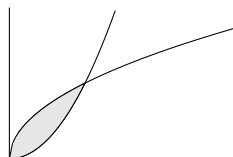
```

$$\text{Aire}(ABCD) = \int_1^2 \frac{1}{x} dx = \ln(2)$$



### 3.3.2 Région entre deux courbes

Exemple : remplir la région comprise entre la courbe de  $x \mapsto x^2$  et celle de  $x \mapsto \sqrt{x}$  sur  $[0; 1]$ .



Le principe est le même : on définit un contour, puis on remplit la région intérieure. Il faut juste faire attention aux sens des tracés des courbes. Il faut tracer une première courbe de 0 à 1, puis tracer l'autre de 1 à 0 pour revenir à l'origine.

```

\filldraw[draw=black,fill=gray!20]
  plot [smooth, domain=0:1] (\x, {\sqrt{\x}})
-- plot [smooth, domain=1:0] (\x, \x^2)
-- cycle;

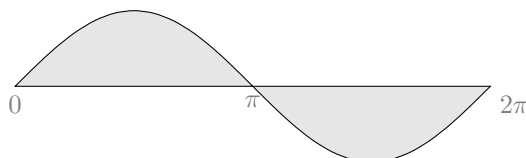
```



### 3.3.3 Région non convexe : *interior rules*

Pour l'instant, nous avons défini un contour et utilisé la commande `\fill` pour remplir son intérieur. Mais comment est défini exactement l'*intérieur* d'un contour ? La question devient délicate lorsque la frontière de la région se recoupe et se traverse elle-même.

Exemple : remplir la région limitée par la courbe de  $x \mapsto \sin(x)$  et l'axe des abscisses sur  $[0; 2\pi]$ .



La frontière se recoupe et se traverse au point  $(\pi, 0)$ .

Pour remplir la région grisée, on peut définir un chemin fermé comme précédemment :

```

\filldraw[fill=gray!20,draw=black]
  (0,0) -- plot [domain=0:2*pi] (\x, {\sin(\x r)}) -- cycle;

```

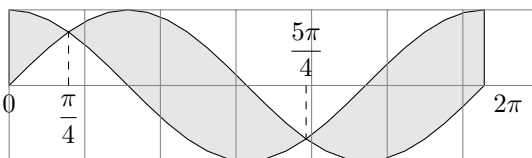
On obtient ce qu'on veut, mais il faut bien comprendre ce qu'a fait *TikZ*. Il dispose de deux modes de calcul pour déterminer l'intérieur d'un chemin. Dans le cas de l'exemple précédent, les deux modes donnent le même résultat, mais ce ne sera pas toujours le cas.

Il a utilisé ici son mode par défaut, qu'on aurait pu demander explicitement par l'option `nonzero rule`, l'autre mode étant `even odd rule`.

La définition de ces modes n'est pas simple (voir le manuel : *interior rules*). Comme le dit lui-même le manuel : « Vous pensez que c'est compliqué ? Eh bien, oui, ça l'est ! ». Le plus compliqué est le mode par défaut, l'autre est un peu plus naturel.

Cependant, pour les cas usuels étudiés ici (région entre deux courbes de fonctions continues), les deux modes donnent le même résultat. Par exemple : tracer la région délimitée par la courbe du sinus et la courbe du cosinus sur  $[0; 2\pi]$ . On trace d'abord la courbe du sinus de 0 à  $2\pi$ , puis la courbe du cosinus de  $2\pi$  à 0.

```
\filldraw [draw=black,fill=gray!20]
  plot [domain=0:2*pi] (\x,{sin(\x r)})
-- plot [domain=2*pi:0] (\x,{cos(\x r)})
-- cycle;
```



Notez que le quadrillage `grid` doit être dessiné après les courbes pour qu'il soit entièrement visible. On peut aussi dessiner le quadrillage avant, mais alors si on veut qu'il reste visible, il faut que les dessins tracés par-dessus soient transparents, par exemple avec l'option `[opacity=0.5]`.

## 3.4 Compléments techniques

### 3.4.1 Utilisation de Gnuplot : plot function

Il se peut que les formules mathématiques intégrées dans *TikZ* ne vous suffisent pas. Il faut alors faire appel à un programme extérieur. Une possibilité prévue par *TikZ* est d'utiliser Gnuplot, logiciel permettant de tracer toutes sortes de courbes et diagrammes.

Le mécanisme est le suivant : quand on donne la commande `\draw` avec une opération `plot function`, comme dans l'exemple suivant :

```
\draw plot function {sin(x)};
```

*TikZ* fait appel au logiciel Gnuplot (qui doit avoir été installé et connecté à *TikZ*), qui se charge de calculer une liste de points à tracer, les enregistre dans un fichier, puis *TikZ* va lire ce fichier et s'en sert pour tracer la courbe.

Plus précisément, si le nom du fichier  $\LaTeX$  est `doc.tex`, *TikZ* engendre un fichier : `doc.pgf-plot.gnuplot`, puis ce fichier est exécuté par Gnuplot, qui fournit en retour un fichier : `doc.pgf-plot.table`, qui est ensuite utilisé par *TikZ* pour tracer la courbe, exactement comme si on avait écrit : `\draw plot file {doc.pgf-plot.table}`;

Pour que cela soit possible, il faut certaines manipulations préalables. Il faut d'abord charger et installer Gnuplot, ce qui est plus ou moins facile suivant les systèmes d'exploitation.

La page d'accueil de Gnuplot est <http://www.gnuplot.info/>

Il faut s'assurer de la compatibilité entre les versions de Gnuplot, du système et de  $\TeX$ .

Il faut ensuite configurer le système  $\TeX$  pour qu'il puisse se connecter à Gnuplot. Là aussi, la manoeuvre à suivre dépend des systèmes.

Quand tout fonctionne, on peut utiliser `\draw plot function {...}`, en notant que la syntaxe de représentation des fonctions n'est pas celle de *TikZ*, c'est celle de Gnuplot.

Cette opération admet différentes options permettant de contrôler Gnuplot, en particulier l'option `parametric` pour tracer des courbes paramétrées.

Depuis la version 2 de TikZ, l'usage de Gnuplot devient moins nécessaire car on peut maintenant écrire directement des formules mathématiques. Il n'est donc pas systématiquement conseillé d'utiliser Gnuplot, d'autant que son installation peut être délicate.

### 3.4.2 Automatisation de certaines configurations

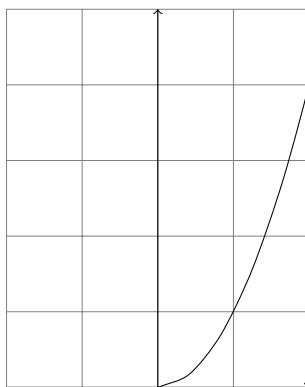
Dans les calculatrices graphiques et les logiciels qui tracent des courbes, il y a souvent un couplage étroit entre différentes options : la fenêtre de tracé, le domaine d'étude de la fonction, le tracé des axes et du quadrillage. En général, on fixe les bornes pour l'abscisse et l'ordonnée, et tout le reste en découle.

Il est possible de réaliser cela avec TikZ, en utilisant ses possibilités de gestion d'options (le package `pgfkeys`) et un peu de programmation L<sup>A</sup>T<sub>E</sub>X. Ce package nécessite la version 2 de TikZ.

Le but recherché est le suivant : introduire quatre nouvelles options `xmin`, `xmax`, `ymin`, `ymax` permettant de fixer les bornes, et trois commandes L<sup>A</sup>T<sub>E</sub>X `\axes`, `\grille`, `\fenetre` utilisant ces bornes pour tracer les axes et la grille (`grid`), et fixer la fenêtre (`\clip`).

Exemple d'utilisation :

```
\begin{center}
\begin{tikzpicture} [xmin=-2,xmax=2,ymin=0,ymax=5]
\grille \axes \fenetre
\draw plot[smooth] (\x,\x^2);
\end{tikzpicture}
\end{center}
```



L'ordre des tracés est délicat : si on place `\axes` avant `\grille`, les axes sont recouverts par la grille. Si on place `\fenetre` avant `\axes`, la pointe de flèche des abscisses est rognée (ce problème peut être évité en élargissant la fenêtre vers le bas).

Pour obtenir ces nouvelles options et commandes, il suffit d'insérer à la fin du préambule du document L<sup>A</sup>T<sub>E</sub>X les quelques lignes suivantes :

```
% Définition des nouvelles options xmin, xmax, ymin, ymax
% Valeurs par défaut : -3, 3, -3, 3
\tikzset{
  xmin/.store in=\xmin, xmin/.default=-3, xmin=-3,
  xmax/.store in=\xmax, xmax/.default=3, xmax=3,
  ymin/.store in=\ymin, ymin/.default=-3, ymin=-3,
  ymax/.store in=\ymax, ymax/.default=3, ymax=3,
}

% Commande qui trace la grille entre (xmin,ymin) et (xmax,ymax)
\newcommand {\grille}
  {\draw[help lines] (\xmin,\ymin) grid (\xmax,\ymax);}

% Commande \axes
```

```

\newcommand {\axes} {
  \draw[->] (\xmin,0) -- (\xmax,0);
  \draw[->] (0,\ymin) -- (0,\ymax);
}

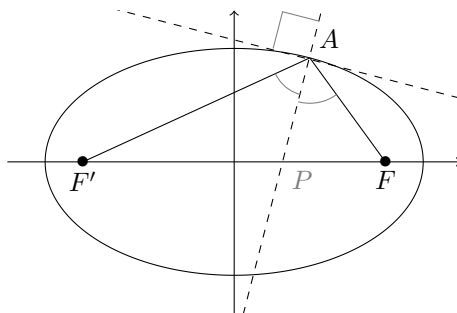
% Commande qui limite l'affichage à (xmin,ymin) et (xmax,ymax)
\newcommand {\fenetre}
  {\clip (\xmin,\ymin) rectangle (\xmax,\ymax);}

```

## 3.5 Exercices

### 3.5.1 Ellipse. Angles avec circle et \clip

Illustrer la propriété suivante : la tangente et la normale en un point  $A$  à une ellipse sont les bissectrices de  $(AF, AF')$ , où  $F$  et  $F'$  sont les foyers de l'ellipse.



Echelle 0.5

Ellipse (représentation paramétrique) : 
$$\begin{cases} x = 5 \cos(\theta) \\ y = 3 \sin(\theta) \end{cases}$$

Foyers  $F(4,0)$  et  $F'(-4,0)$  car  $3^2 + 4^2 = 5^2$ .

$A(2, 2.75)$

Tangente :  $y = -0.26x + 3.27$ , normale :  $y = 3.82x - 4.89$

Angle droit :  $(2.25, 3.72) \text{ -- } (1.28, 3.97) \text{ -- } (1.03, 3)$

Les coordonnées ont été obtenues avec GeoGebra.

Pour les marques d'angle, on a tracé des cercles de centre  $A$  (rayons 1 et 1.2), et on n'en a montré que la partie intérieure aux triangles  $AF'P$  et  $APF$ ,  $P$  étant le point de la normale sur l'axe  $Ox$  :  $P(1.28, 0)$ . Pour ne montrer qu'une partie, on utilise `\clip`, et pour limiter la portée de ce clip, on utilise un environnement `\begin{scope} ... \end{scope}`

```

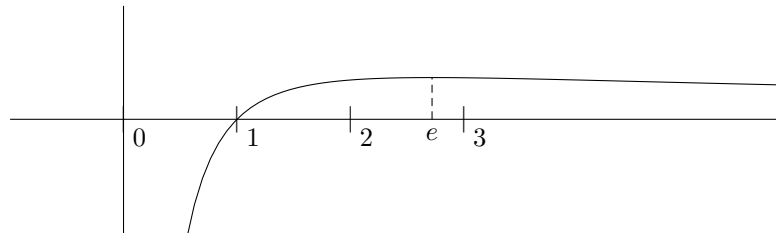
\begin{scope}
  % pour limiter la portée du \clip
  \clip (a) -- (p) -- (f2) -- cycle; % triangle APF'
  \draw [gray] (a) circle (1); % cercle pour arc
\end{scope}

```

### 3.5.2 $a^b = b^a$ . xscale, yscale

Illustrer la propriété suivante : l'équation  $a^b = b^a$  n'a que deux solutions entières  $(a, b)$  avec  $a \leq b$  :  $(2, 2)$  et  $(2, 4)$

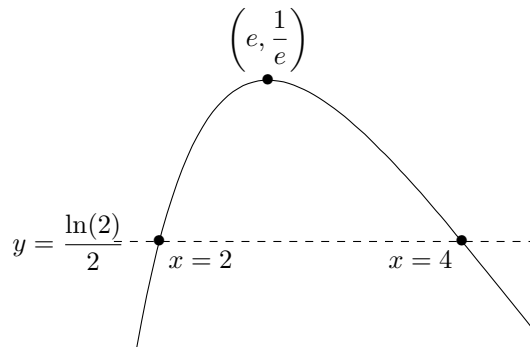
L'équation est équivalente à  $\frac{\ln(a)}{a} = \frac{\ln(b)}{b}$ . Pour illustrer cela, on trace la courbe de  $x \mapsto \frac{\ln(x)}{x}$  et on étudie ses intersections avec des droites horizontales  $y = k$ . L'étude du sens de variation montre un maximum en  $e$  et une limite de 0 en  $+\infty$ . Donc la seule solution envisageable est  $a = 2$ .



Pour mieux voir le sens de variation au voisinage de  $e$ , on peut choisir une échelle différente pour les abscisses et pour les ordonnées, par exemple multiplier les abscisses par 2 et les ordonnées par 100 : ici `[xscale=2,yscale=100]`.

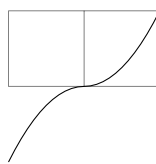
Pour les points d'ordonnée  $\frac{\ln(2)}{2}$ , on peut utiliser directement une formule de calcul dans les coordonnées, par exemple `(2, {\ln(2)/2})`.

On limite également le domaine de la fonction, avec `[domain=1.85:4.5]`. Attention aux deux-points : protéger avec `\sortheadhoff(:)` ou utiliser `[domaine={1.85}{4.5}]`

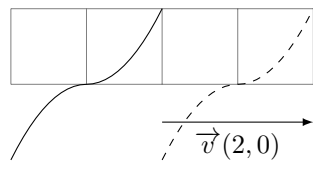


### 3.5.3 Fonction périodique : `\foreach`

Tracer la courbe de la fonction périodique de période 2 dont la formule sur  $[-1; 1]$  est  $y = x^2$ . L'idée est de tracer d'abord la courbe sur une période avec `\draw plot [domain=-1:1] (\x, \x^2);`



puis de reproduire ce morceau par des translations de vecteur  $(2, 0)$ .



On ajoute pour cela `\draw plot [domaine={-1}{1}] (\x+2, \x^2); .`

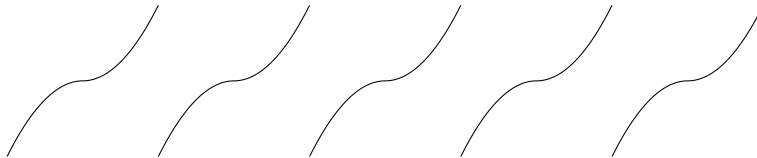
On peut rajouter de même différents morceaux, en factorisant les options au début :

```
\begin{tikzpicture} [domaine={-1}{1}, samples=80]
\draw plot (\x, \x^2);
\draw plot (\x+2, \x^2);
\draw plot (\x+4, \x^2);
```

```
\draw plot (\x+6,\x^2);
\draw plot (\x+8,\x^2);
\end{tikzpicture},
```

Cette répétition d'écritures peut être résumée par l'utilisation de `\foreach` :

```
\foreach \k in {0,2,...,8}
  {\draw plot (\x + \k, \x^2);}
```

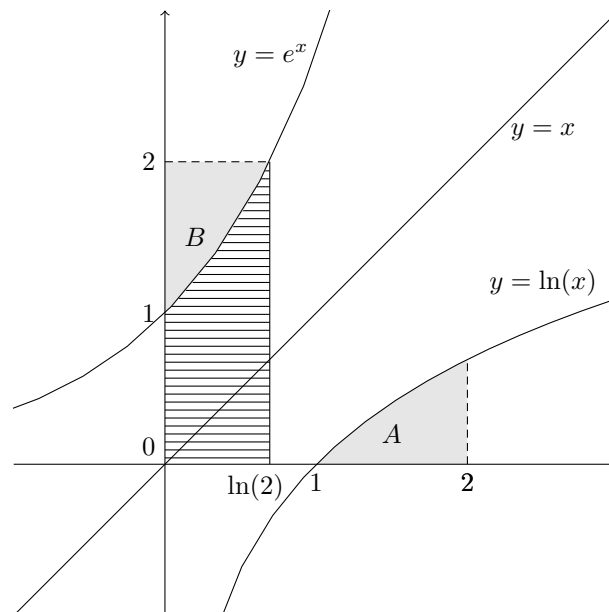


`\foreach` est une instruction de *boucle* au sens usuel de l'informatique. Nous en verrons d'autres exemples plus loin.

### 3.5.4 Fonctions réciproques, aires : pattern

Illustrer la formule  $\int_1^2 \ln(x) dx + \int_0^{\ln(2)} e^x dx = 2 \ln(2)$  en l'interprétant en termes d'aires.

L'intégrale du logarithme est représentée par l'aire grise  $A$ , l'intégrale de l'exponentielle par l'aire hachurée. Les fonctions  $\ln$  et  $\exp$  sont réciproques, donc les deux aires grises  $A$  et  $B$  sont égales, par symétrie par rapport à la droite d'équation  $y = x$ . L'aire (hachurée +  $B$ ) est l'aire d'un rectangle et vaut  $2 \ln(2)$ ,



On trace les régions avant le reste. Pour les hachures, il faut utiliser la bibliothèque `patterns`, en déclarant dans le préambule :

```
\usepackage{tikz}
\usetikzlibrary{patterns}
```

On l'utilise alors avec l'option `pattern` (voir le manuel pour les différents motifs possibles : `vertical lines`, `north east lines`, `dots`, etc.) :



```

\fill [pattern=horizontal lines]
(0,0)
-- plot [domaine={0}{ln(2)}] (\x,{exp(\x)})
-- ({ln(2)},0)
-- cycle;

```

Attention au domaine de définition du logarithme, et aux grandes valeurs au voisinage de 0. Il faut choisir un domaine convenable. De même pour les grandes valeur de l'exponentielle. De plus, on peut fixer la fenêtre pour bien cadrer le dessin. Attention à l'ordre des différentes commandes.

La structure du code est :

```

\begin{tikzpicture}[scale = 2,xmin=-1,xmax=3,ymin=-1,ymax=3]
\fenetre
\fill ... ;           % aire grise avec logarithme
\fill ... ;           % aire grise avec exponentielle
\fill ... ;           % aire hachurée
\axes
\draw plot ... ;      % courbe de ln
\draw plot ...;      % courbe de exp
\draw plot ... ;      % courbe y = x
\draw [dashed] ... ; % les lignes de rappel en pointillés
...                   % les noeuds de texte
\end{tikzpicture}

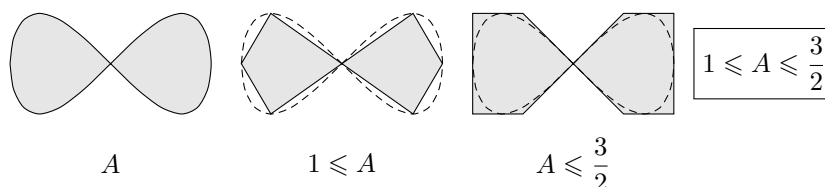
```

### 3.5.5 Lemniscate de Geron. `\scope`, `xshift`, `\filldraw`

La lemniscate de Geron est la courbe de représentation paramétrique

$$\begin{cases} x = \cos(\theta) \\ y = \sin(\theta) \cos(\theta) \end{cases}$$

Son équation cartésienne est  $x^4 + y^2 = x^2$ , soit  $y = x\sqrt{1-x^2}$  ou  $y = -x\sqrt{1-x^2}$ . Le but est de représenter l'aire de la région intérieure délimitée par la courbe, puis de représenter un encadrement de cette aire.



Pour représenter l'aire  $A$ , utiliser `\filldraw`. TikZ détermine correctement la région intérieure. Pour représenter l'encadrement d'aires, il faut calculer différents éléments de la fonction  $x \mapsto x\sqrt{1-x^2}$  sur  $[0; 1]$  :

la tangente en  $O$  a pour équation  $y = x$ , le maximum est  $\left(\frac{\sqrt{2}}{2}, \frac{1}{2}\right)$ .

Mettre au point d'abord 4 figures séparées (les trois aires puis le résultat). Ne pas oublier que les angles de la formule mathématique sont en radians.

Ensuite, regrouper ces figures en une seule en utilisant `scope` et `xshift`.

Ainsi, `\begin{scope}[xshift=1.5 cm] ... \end{scope}` effectue une translation de 1.5 cm vers la droite et limite l'effet de cette translation à l'environnement `scope`.

En remplissant les régions, attention à l'ordre du tracé : tracer les régions remplies avant les autres éléments.

La structure du code est donc :

```

\begin{tikzpicture}[scale=...]           % choisir une échelle
  \begin{scope}                         % première figure
    \filldraw[fill=...] plot ...;      % la courbe et l'aire
    \draw (0,-1) node {$A$};          % commentaire
  \end{scope}

  \begin{scope}[xshift= 2.3cm]          % translation
    \filldraw ... ;                   % aire minorante
    \draw[densely dashed] plot ...;    % la courbe
    \draw (0,-1) node ... ;           % commentaire
  \end{scope}

  \begin{scope}[xshift= 4.6cm]          % translation
    \filldraw ... ;                   % aire majorante
    \draw[densely dashed] plot ... ;  % la courbe
    \draw (0,-1) node ... ;           % commentaire
  \end{scope}

  \begin{scope}[xshift= ...cm]         % translation
    \draw (0,0) node ... ;            % le résultat
  \end{scope}
\end{tikzpicture}

```

### 3.6 Résumé

Pour tracer une courbe définie par des équations mathématiques utiliser :

```
\draw plot (abscisse,ordonnée); ou \draw plot (angle:rayon);
```

Les formules sont données en fonction d'un paramètre nommé `\x` par défaut, ou qu'on peut nommer comme on veut avec l'option `variable=`.

Avec l'option `[domain=a:b]`, on peut fixer le domaine de variation de ce paramètre, mais cette syntaxe pose des problèmes quand on utilise `babel` français. Ces problèmes peuvent être résolus par l'emploi de `shorthandoff(:)` ou par la programmation d'une option `[domaine={a}{b}]`.

TikZ fournit une syntaxe pour les formules mathématiques usuelles. Les fonctions trigonométriques utilisent des angles en degrés, mais TikZ fournit des possibilités de conversion de radians en degrés : `r` ou `deg()`.

L'aspect des courbes peut être contrôlé par des options : `samples` pour le nombre de points, `smooth` pour le lissage.

Le traitement des discontinuités ou des grandes valeurs n'est pas automatique. Il faut alors séparer soi-même les intervalles, fixer explicitement l'échelle avec `scale`, `xscale`, `yscale` et la fenêtre d'affichage avec `\clip`.

Les courbes formées de morceaux qu'on reproduit par des transformations (fonction périodique par exemple) peuvent être construites avec la commande `\foreach`.

On peut remplir des régions délimitées par des courbes. Pour cela il faut définir un chemin fermé (avec `--` et `cycle`) et utiliser `\fill` ou `\filldraw`. Certaines portions du chemin peuvent être des courbes tracées avec `plot`, sachant qu'elles sont parcourues dans le sens défini par l'ordre des valeurs extrêmes du paramètre dans `domain=a:b` (de  $a$  vers  $b$ , même si  $a > b$ ).

On peut remplir une région avec de la couleur ou des motifs, comme par exemple : `[pattern= horizontal lines]` (voir la bibliothèque `patterns`).

Le remplissage des régions est régi par des règles techniques à consulter dans le manuel (*interior rules* : `nonzero rule` par défaut, ou

`even odd rule`). La règle par défaut donne en général le résultat attendu.

Une autre possibilité pour tracer des courbes est d'utiliser `plot function`, qui fait appel au logiciel extérieur Gnuplot. Cela peut être utile quand les fonctions mathématiques de TikZ se révèlent insuffisantes, ou pour récupérer des figures chargées sur Internet. Mais cela exige une

bonne installation et configuration de Gnuplot et de T<sub>E</sub>X. Se référer aux manuels de TikZ et de Gnuplot.

Une autre possibilité encore est d'utiliser `plot file {fichier}`, où *fichier* contient la liste des coordonnées des points avec un point par ligne.

On peut retrouver les options de certaines calculatrices graphiques avec un peu de programmation :

`xmin, xmax, ymin, ymax, \axes, \grille, \fenetre.`

Pour construire une figure composée de plusieurs sous-figures indépendantes, on peut utiliser l'environnement `scope` pour séparer les sous-figures et les options `shift`, `xshift`, ou `yshift` pour les décaler.



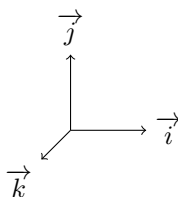
# Chapitre 4

## Géométrie dans l'espace

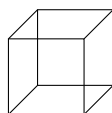
### 4.1 Coordonnées $(x, y, z)$

#### 4.1.1 Représentation TikZ standard

TikZ prévoit une syntaxe  $(x, y, z)$  pour représenter des points dans l'espace. Mais ce n'est pas un logiciel de modélisation 3D : il ne gère pas les surfaces cachées, ne permet pas de voir les figures sous différents points de vue. Il se contente de tracer des figures planes qui sont les projetées de figures de l'espace, sachant que le point  $M(x, y, z)$  est défini par  $\overrightarrow{OM} = x\vec{i} + y\vec{j} + z\vec{k}$ , où  $(\vec{i}, \vec{j}, \vec{k})$  sont des vecteurs du plan, avec par défaut  $\vec{k} = -\frac{1}{2\sqrt{2}}\vec{i} - \frac{1}{2\sqrt{2}}\vec{j}$  (ce n'est pas ce que dit le manuel, mais on peut vérifier sur la figure suivante).

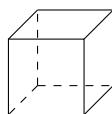


Le cube unité est alors obtenu par :

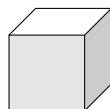


```
\draw (0,0,0)--(1,0,0)--(1,1,0)--(0,1,0)--cycle; % face arrière
\draw (0,0,1)--(1,0,1)--(1,1,1)--(0,1,1)--cycle; % face avant
% arêtes horizontales, de l'arrière vers l'avant
\draw (0,0,0) -- (0,0,1); % bas gauche
\draw (1,0,0) -- (1,0,1); % bas droit
\draw (1,1,0) -- (1,1,1); % haut droit
\draw (0,1,0) -- (0,1,1); % haut gauche
```

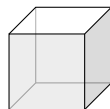
Si l'on considère le cube comme un solide plein, on peut tracer en pointillés (`[dashed]`) les arêtes cachées (celles qui partent de l'origine) :



Ou bien on peut ombrer les faces avec `\fill` (le soleil est en haut à gauche) :



On peut de plus rendre les faces transparentes avec `opacity` :



Tout cela est possible avec les options habituelles de `TikZ`, mais ce n'est pas automatique. Il faut décider soi-même ce qui sera en pointillés [`dashed`], en gris [`fill=...`], transparent [`opacity=0.7`], et bien dessiner les éléments dans le bon ordre.

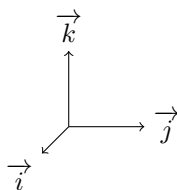
### 4.1.2 Autres représentations : $x=...$ , $y=...$ , $z=...$

La représentation de `TikZ` est une convention comme une autre, et une fois qu'on l'a comprise, on peut l'utiliser. Cependant, l'enseignement mathématique français utilise une autre convention à laquelle sont attachés des réflexes et des images mentales, et changer ces réflexes peut induire une charge mentale inutile.

`TikZ` possède des options `x`, `y` et `z`, qui permettent de redéfinir les vecteurs de base en fonction de coordonnées planes absolues (abscisses horizontales vers la droite, ordonnées verticales vers le haut).

La représentation mathématique française usuelle peut être obtenue par :

```
[x= {(-0.353cm,-0.353cm)}, z={0cm,1cm}, y={1cm,0cm}]
```



Si l'on souhaite utiliser souvent cette représentation, il peut être pratique d'en faire un style (par exemple `math3d`), qu'on définit une fois pour toutes au début du document :

```
\tikzset{math3d/.style=
  {x= {(-0.353cm,-0.353cm)}, z={0cm,1cm},y={1cm,0cm}}}
```

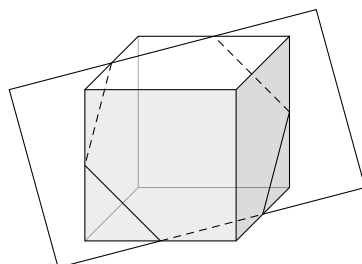
On peut d'ailleurs choisir une autre valeur de `x` pour les besoins de la perspective. Pour chaque figure où l'on veut utiliser cette convention, il suffira d'ajouter l'option `math3d` :

```
\begin{tikzpicture} [math3d] ... \end{tikzpicture}
```

## 4.2 Quelques figures de géométrie

### 4.2.1 Section d'un cube suivant un hexagone

En joignant les milieux de certains côtés, on obtient un hexagone régulier, section du cube par le plan médiateur de la grande diagonale.



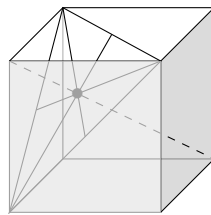
En soi, il n'y a pas de difficulté particulière concernant TikZ. Il faut préalablement calculer les coordonnées de tous les points, mais c'est plus un problème mathématique qu'un problème de dessin. Pour le dessin, il faut tracer des segments entre les points, en repérant ce qui doit être ombré, mis en pointillés ou transparent.

Une décision à prendre est de choisir un parallélogramme pour représenter le plan de l'hexagone. On a choisi des points dont la position est facilement repérable par rapport aux autres éléments du dessin, le but étant de favoriser la vision dans l'espace.

$$(3/2, 0, 0) \quad (0, 3/2, 0) \quad (-1/2, 1, 1) \quad (1, -1/2, 1)$$

### 4.2.2 Grande diagonale d'un cube

La grande diagonale du cube coupe un certain triangle équilatéral en son centre de gravité, perpendiculairement.



Là encore, il faut calculer les coordonnées de tous les points, repérer ce qui est caché et le traduire par des couleurs, des pointillés, une opacité, un ordre de tracé.

### 4.2.3 Droites et plans

Il est fréquent d'avoir à illustrer un problème de géométrie qui demande de dessiner des droites et des plans quand on connaît leurs équations.

Par exemple : construire la perpendiculaire commune aux deux droites  $d$  et  $d'$  dont les représentations paramétriques sont :

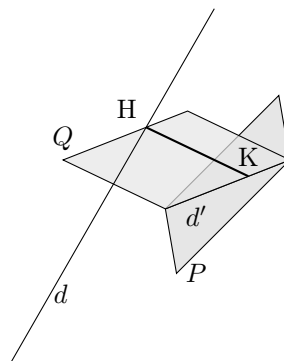
$$d \begin{cases} x = 1 - t \\ y = t \\ z = 2t \end{cases} \quad \text{et} \quad d' \begin{cases} x = t \\ y = 2 + 2t \\ z = t \end{cases}$$

Une solution mathématique consiste à construire le plan  $P$  parallèle à  $d$  et contenant  $d'$  (d'équation  $x - y + z - 1 = 0$ ), puis le plan  $Q$  perpendiculaire à  $P$  et contenant  $d'$  (d'équation  $x - z = 0$ ). Ce plan  $Q$  coupe  $d$  en  $H \left( \frac{2}{3}, \frac{1}{3}, \frac{2}{3} \right)$ .

Soit  $K$  le projeté de  $H$  sur  $P$  :  $K \left( -\frac{1}{3}, \frac{4}{3}, -\frac{1}{3} \right)$ .

Alors la perpendiculaire commune à  $d$  et  $d'$  est la droite  $(HK)$ .

Le problème est mathématiquement résolu, mais il reste à faire la figure suivante pour l'illustrer :



Pour cela, il reste de nombreux choix supplémentaires à faire et de problèmes auxiliaires à résoudre.

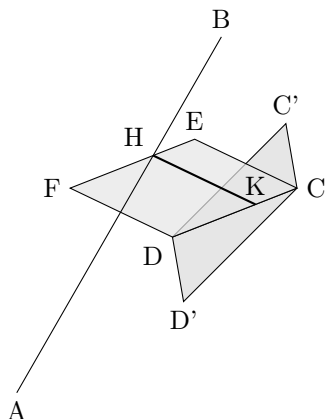
On choisit d'abord deux points  $A$  et  $B$  de  $d$  et on trace le segment  $[AB]$ . De même, on choisit  $C$  et  $D$  sur  $d'$  et on trace le segment  $[CD]$ . Ce n'est pas grand chose, mais rien que cela peut demander plusieurs essais pour que la figure ne soit pas trop grande et que la vue en perspective soit interprétable. On choisit  $A(2, -1, -2); B(0, 1, 2); C(0, 2, 0); D(-1, 0, -1)$ .

Ensuite il faut représenter le plan  $P$  par un parallélogramme. On décide de choisir  $[CD]$  comme diagonale de ce parallélogramme. On choisit l'autre diagonale  $[C'D']$  parallèle à  $d$  pour bien suggérer le parallélisme. Mais pour que le parallélogramme ne soit pas trop grand, on prend pour longueur  $C'D'$  la moitié de  $AB$ .

On obtient  $C' \left(-1, \frac{3}{2}, \frac{1}{2}\right); D' \left(0, \frac{1}{2}, -\frac{3}{2}\right)$ .

Il faut ensuite représenter le plan  $Q$  par un parallélogramme. On choisit  $[CD]$  comme un des côtés et on s'arrange pour que le côté parallèle à  $[CD]$  contienne  $H$ . Les autres sommets sont alors  $E(1, 1, 1)$  et  $F(0, -1, 0)$ .

Pour le dessin, il faut faire attention à l'ordre du tracé pour suggérer la perspective : le contour et le remplissage du plan  $P$  doivent être faits d'abord (le plan  $P$  est placé derrière), puis celui du plan  $Q$  avec une certaine transparence [`opacity = 0.7`].



On voit que, pour une structure mathématique relativement simple (deux plans et trois droites), les contraintes du tracé en perspective rendent la réalisation assez délicate : il faut matérialiser les droites par des segments et les plans par des parallélogrammes, suggérer les propriétés de la figure (parallélisme), rendre la figure lisible et pas trop grande. Et tout cela en respectant les équations données par l'énoncé et les règles de la représentation en perspective.

Une astuce consisterait à considérer que la figure finale n'est après tout qu'une figure plane comportant essentiellement deux parallélogrammes, et de tracer ces parallélogrammes sans se soucier d'un quelconque repère de l'espace. On peut alors choisir les parallélogrammes de formes arbitraires, à condition de respecter la possibilité d'interpréter la figure comme une projection d'une figure de l'espace.

Cependant, il ne serait plus possible de les choisir arbitrairement si on devait de plus représenter le repère de l'espace de manière cohérente avec les équations (ou alors dans ce cas il faudrait calculer la bonne position du repère).

## 4.3 Courbes et surfaces

### 4.3.1 Représentation paramétrique, `plot`, `\foreach`

Une *courbe* dans l'espace peut être définie par une représentation paramétrique à un paramètre. On peut donc la dessiner avec l'opération `plot (X, Y, Z)` en exprimant  $X, Y$  et  $Z$  en fonction d'un paramètre (nommé `\x` par défaut).



Une *surface* peut être définie mathématiquement par une représentation paramétrique à deux paramètres  $t$  et  $u$ . TikZ ne prévoit pas cette possibilité, mais on peut cependant la mettre en oeuvre.

Lorsqu'on fixe  $t$  et qu'on fait varier  $u$ , on obtient une courbe dans l'espace, tracée sur la surface. L'idée est de tracer chacune de ces courbes avec l'opération `plot`, et cela pour différentes valeurs de  $t$ , qu'on peut obtenir à l'aide d'une boucle `\foreach`.

Le schéma est donc :

```
\foreach \t in {...} { plot ( ..., ..., ... ) }
```

Prenons comme exemple une surface de révolution d'axe  $Oz$  (comme un cylindre, un cône, une sphère). Elle est obtenue en faisant tourner la courbe plane d'équation  $x = f(z)$  autour de l'axe  $Oz$  et a pour représentation paramétrique (avec deux paramètres  $\theta$  et  $t$ ,  $\theta$  variant de 0 à  $2\pi$ ) :

$$\begin{cases} x = f(t) \cos(\theta) \\ y = f(t) \sin(\theta) \\ z = t \end{cases}$$

On peut alors représenter la surface par empilement de disques horizontaux (un disque est obtenu pour  $t$  constant), en les dessinant dans l'ordre croissant des valeurs de  $t$ . On matérialise ainsi les sections de la surface par des plans horizontaux (les courbes de niveau).

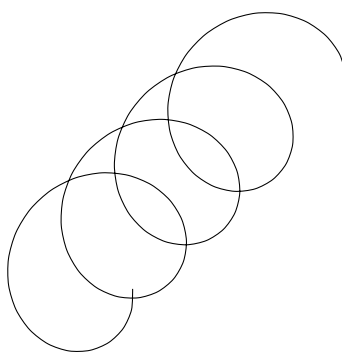
Avec TikZ, les noms des paramètres vont être `\x` pour  $\theta$  et `\t` pour  $t$ . On fera varier `\t` grâce à une boucle `\foreach`, et `\x` sera le paramètre d'une instruction `plot`. Suivant la surface, il faut choisir correctement les valeurs de `\t` et le domaine de `\x`

### 4.3.2 Hélice

Une hélice circulaire (un ressort) est définie par :

$$\begin{cases} x = \frac{\theta}{\pi} \\ y = \cos(\theta) \\ z = \sin(\theta) \end{cases}$$

```
\draw [domain=-4*pi:4*pi, samples=80, smooth]
  plot (\x/pi, {cos(\x r)}, {sin(\x r)}) ;
```



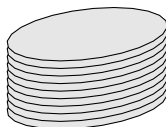
TikZ n'a pas de mécanisme automatique pour mettre en évidence les parties cachées. Sur cette figure, il n'est pas facile de le faire, contrairement à ce qui peut se passer pour les faces d'un cube. L'axe de l'hélice est l'axe des abscisses. Dans le sens des variations du paramètre, de  $-4\pi$  à  $4\pi$ , les spires vont de l'arrière vers l'avant, en tournant dans le sens inverse des aiguilles d'une montre.

On constate les limitations de TikZ en ce qui concerne les figures dans l'espace. Il fournit une façon simple de repérer précisément les points dans l'espace avec trois coordonnées, ce qui permet de faire des figures exactes, mais il ne permet pas d'analyser les figures obtenues pour obtenir un rendu analogue à la vision humaine.

### 4.3.3 Cylindre $x^2 + y^2 = 1$

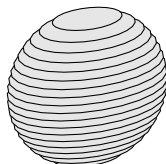
C'est une surface de révolution de représentation paramétrique :  $(\cos(\theta), \sin(\theta), t)$

```
\foreach \t in {-1,-0.9,...,1} {
  \filldraw plot[domain=0:2*pi]
    ({cos(\x r)}, {sin(\x r)}, \t);
}
```



### 4.3.4 Sphère $x^2 + y^2 + z^2 = 1$

C'est une surface de révolution de représentation paramétrique :  $(\sqrt{1-t^2} \cos(\theta), \sqrt{1-t^2} \sin(\theta), t)$

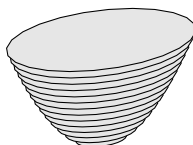


On n'obtient pas tout à fait l'aspect visuel habituel d'une sphère, mais cela s'explique par le type de perspective utilisé : perspective cavalière (c'est-à-dire projection sur un plan) et non pas perspective centrale (qui correspond à notre mode de vision).

Il y a d'autres façons de représenter une sphère, mais ici la représentation par empilement de disques horizontaux permet de suggérer le relief et de cacher les parties qui sont derrière.

### 4.3.5 Paraboloïde $z = x^2 + y^2$

C'est une surface de révolution obtenue en faisant tourner une parabole autour de  $Oz$ . Une représentation paramétrique est  $(\sqrt{t} \cos(\theta), \sqrt{t} \sin(\theta), t)$



## 4.4 Résumé

TikZ permet d'exprimer les coordonnées  $(x, y, z)$  d'un point de l'espace, qu'il représente par projection sur le plan de dessin.

Le repère correspondant de l'espace a une position prédéfinie, qu'on peut modifier par les options `x`, `y` et `z`, qui fixent les projetés des vecteurs de l'espace. Pour une représentation française standard, on peut choisir

```
[x={(-0.353cm,-0.353cm)}, z={(0cm,1cm)}, y={(1cm,0cm)}]
```

TikZ n'est pas un logiciel de modélisation 3D, et il ne fournit aucune autre fonctionnalité automatique à part la tracé d'un point par  $(x, y, z)$ . Il faut donc décider soi-même des représentations du relief, de la perspective, des parties cachées. À part les cas simples (faces planes, segments), ce n'est pas forcément possible. Il est alors conseillé de se tourner vers des logiciels spécialisés.

Les courbes peuvent être tracées à partir de leurs représentations paramétriques, en utilisant l'opération `plot`.

Les surfaces peuvent être représentées par leurs courbes de niveau, en combinant l'utilisation de l'instruction `\foreach` et celle de l'opération `plot`.

# Chapitre 5

## Représentation de données

### 5.1 Notions de base

On parle de représentation graphique de données, lorsque des informations relatives à un certain phénomène et qui sont fournies comme une liste de nombres, sont affichées sous forme d'un diagramme significatif.

On va montrer, dans ce chapitre, comment réaliser les diagrammes utilisés de façon usuelle dans ce type de situation : les courbes, les diagrammes à barres, les histogrammes, les diagrammes à secteurs.

#### 5.1.1 Diagramme d'effectifs : plot coordinates

Dans une classe, un professeur a rendu un devoir noté sur 10, en points entiers. Il a compté combien il y avait de copies pour chaque note. Il désire faire une représentation graphique du nombre de copies par note.

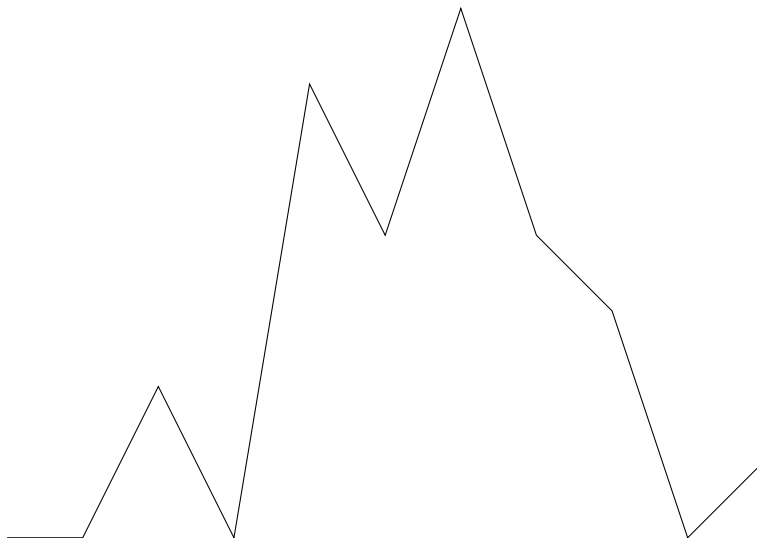
Note	0	1	2	3	4	5	6	7	8	9	10
Nombre de copies	0	0	2	0	6	4	7	4	3	0	1

Pour être traitées, ces données numériques vont être représentées sous forme d'une liste de couples de coordonnées, avec en abscisse la note et en ordonnée le nombre des copies ayant obtenu cette note.

Ainsi en utilisant l'opération `plot coordinates` :

```
\begin{tikzpicture}
  \draw plot coordinates {(0,0) (1,0) (2,2) (3,0)
    (4,6) (5,4) (6,7) (7,4) (8,3) (9,0) (10,1)};
\end{tikzpicture}
```

on va dessiner le diagramme suivant :



Comme on le voit, la commande :

```
\draw plot coordinates {(0,0) (1,0) (2,2) ... (10,1)};
```

produit le même tracé que : `\draw (0,0)--(1,0)--(2,2) ... --(10,1);`

On peut alors se demander quel intérêt il y a à utiliser `plot coordinates`.

Sur ce *petit exemple*, ce n'est pas évident. Mais les séries statistiques comportent parfois une très grande quantité de données, et nous verrons plus loin comment traiter ces données facilement à l'aide d'applications auxiliaires (tableur ou outils statistique) et créer des fichiers de données à associer au code source du document.

Nous allons maintenant présenter plusieurs exemples avec ces données. Pour éviter d'avoir à les recopier plusieurs fois, nous allons les définir sous forme d'une commande  $\LaTeX$  réutilisable dans les exemples qui vont suivre. En effet, bien que  $TikZ$  définisse ses propres règles syntaxiques, il peut quand même bénéficier de certains mécanismes de  $\LaTeX$ , en particulier l'utilisation de commandes vues comme des macros de remplacement de texte.

Voici la commande `\nombresCopiesParNote` :

```
\newcommand{\nombresCopiesParNote}
{(0,0) (1,0) (2,2) (3,0) (4,6) (5,4) (6,7) (7,4) (8,3) (9,0) (10,1)}
```

Maintenant, l'exemple précédent peut être obtenu plus simplement avec :

```
\draw plot coordinates {\nombresCopiesParNote};
```

### 5.1.2 Améliorer la lisibilité : `grid`, `node`, `\foreach`

Nous allons maintenant montrer comment améliorer ce diagramme pour rendre compréhensibles les informations fournies.

Il est nécessaire de placer ces données sur une grille pour rendre les valeurs plus facilement lisibles : `\draw (-1,0) grid (11,7);`

On peut ensuite étiqueter les abscisses avec des commandes de ce type :

```
\draw (\x,0) node[below]{\x} en faisant varier \x entre 0 et 10.
```

ou les ordonnées avec : `\draw (-1,\y) node[left]{\y};`

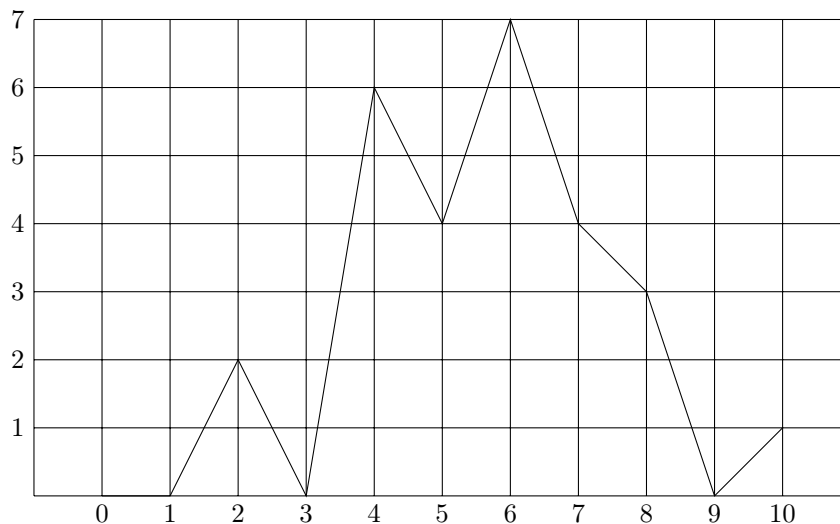
Pour cela on peut utiliser la commande `\foreach` qui permet en deux instructions d'effectuer l'étiquetage des abscisses et des ordonnées :

```
\foreach \y in {1,2,...,7} \draw(-1,\y)node[left]{\y};
\foreach \x in {0,1,...,10} \draw(\x,0)node[below]{\x};
```

Dans la commande `\foreach`, la variable `\x` va prendre successivement les valeurs définies par  $\{0, 1, \dots, 10\}$  c'est-à-dire  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

Les trois points signifient que la liste doit être prolongée en utilisant comme modèles les 2 premières valeurs et jusqu'à la valeur finale. Pour cela, on ajoute à la seconde valeur de la différence d'elle même avec la première et ainsi de suite...

Par exemple :  $\{-1, 4, \dots, 19\}$  représente  $\{-1, 4, 9, 14, 19\}$



### 5.1.3 Marquer les points, étiqueter : `mark`, `node`, `rotate`

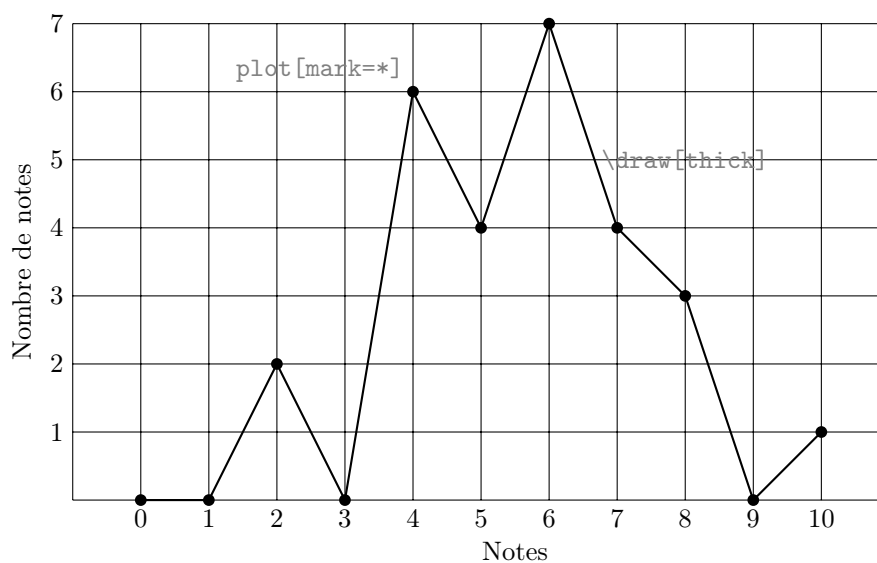
On peut placer un texte sur chaque axe pour préciser ce qu'il représente. On utilise pour cela un nœud (`node`) correctement placé :

```
\draw (5.5,-0.75) node{Notes};
\draw (-1.75,3.5) node[rotate=90]{Nombre de notes};
```

On peut aussi marquer les points significatifs de la courbe à l'aide de l'option `mark` affectée à l'opération `plot` :

```
\draw[thick] plot[mark=*] coordinates {\nombresCopiesParNote};
```

On obtient alors cette nouvelle version du diagramme :



Le marquage des points est fait à l'aide de l'option `mark=` suivie d'une des 3 valeurs `*` ou `+` ou `x` qui placent respectivement  $\bullet$  ou  $\times$  sur chaque point de la liste. L'option `mark size=`, suivie d'une dimension, permet de modifier la taille de la marque. Par exemple : `[mark=*,mark size=5mm]`

L'option `ball` permet aussi de placer une marque comme celle-ci  $\bullet$

D'autres marques sont disponibles dans la bibliothèque (library) `plotmarks`. Pour cela, dans le préambule, lors du chargement de `TikZ` il faut ajouter :

```
\usetikzlibrary{plotmarks} après \usepackage{tikz}
```

Pour un diagramme d'effectifs, la courbe qui relie les points n'est pas très significative, et ce n'est donc pas une bonne idée de tracer cette courbe.

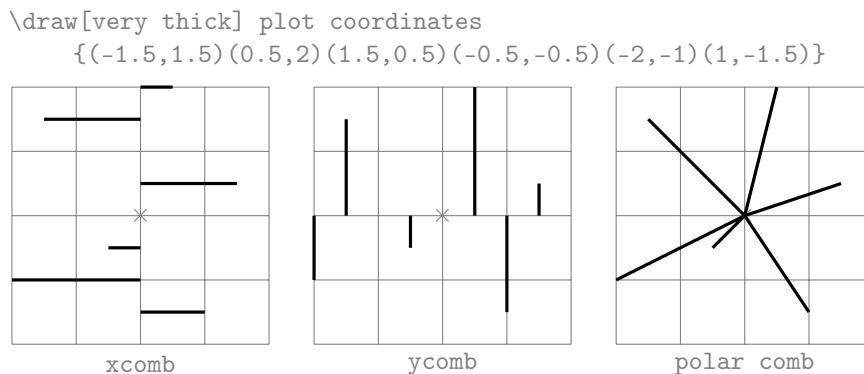
#### 5.1.4 Diagramme à barres : `xcomb`, `ycomb`, `polar comb`

Par défaut, l'opération `plot` relie les points donnés par une ligne polygonale, mais on peut remplacer cette ligne par des barres :

- horizontales avec l'option `xcomb` ;
- verticales avec l'option `ycomb` ;
- joignant les points à l'origine avec l'option `polar comb`.

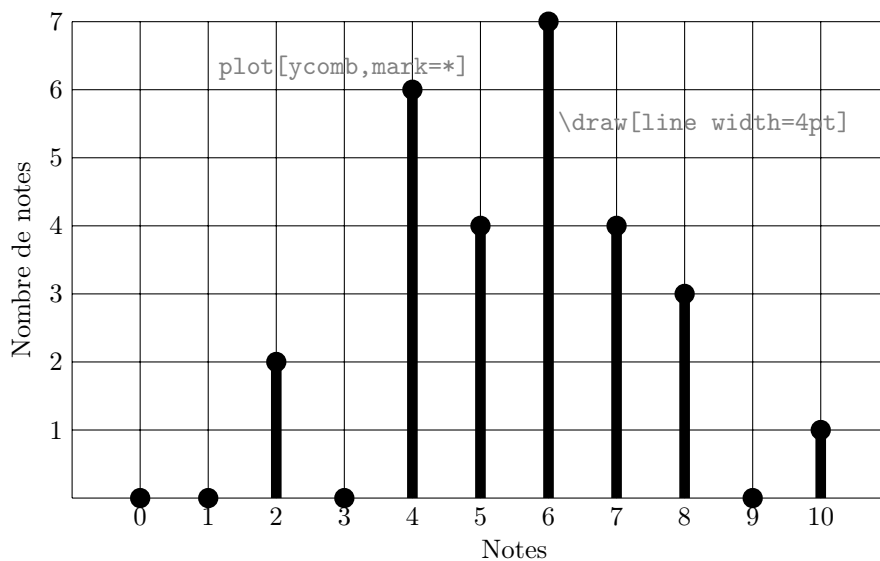
Voici trois petits exemples où la même liste de coordonnées est affichée avec les options `xcomb`, `ycomb` et `polar comb`.

On pourra ainsi comparer les différents modes d'affichage :



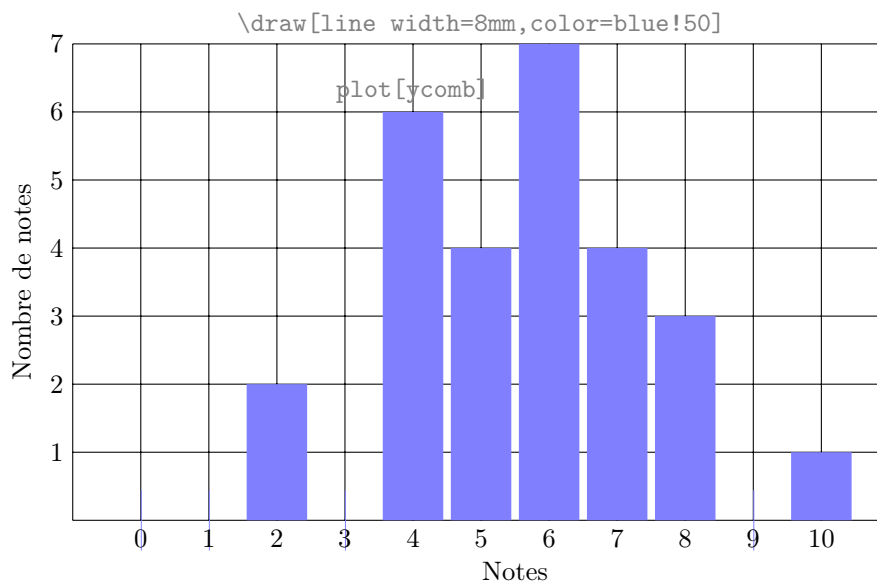
Pour le diagramme actuel, la meilleure option est donc `ycomb`.

On peut aussi rendre les barres bien visibles en dessinant des traits plus larges et en marquant le point avec, par exemple :



### 5.1.5 Histogramme : `xcomb`, `ycomb`, `line width`

Un histogramme est aussi un diagramme à barre, en effet il suffit de supprimer les marques du diagramme précédent et d'augmenter la largeur du trait pour obtenir un histogramme traditionnel :



### 5.1.6 Affichage des données d'un fichier : `plot file`

Dans les exemples précédents, les effectifs par note ont été donnés sous forme d'une liste de coordonnées :

```
{(0,0) (1,0) (2,2) (3,0) (4,6) (5,4) (6,7) (7,4) (8,3) (9,0) (10,1)}
```

Cependant les informations statistiques sont souvent disponibles sous forme de fichier de texte contenant les données numériques. Voici par exemple le fichier « `nombresCopiesParNote.txt` » contenant les mêmes informations que ci-dessus sous un format différent :

```
0 0
1 0
2 2
3 0
4 6
5 4
6 7
7 4
8 3
9 0
10 1
```

Il y a un couple de coordonnées par ligne, et il y a au moins un séparateur (espace ou tabulation) entre l'abscisse et l'ordonnée. Il est facile de créer un tel fichier avec n'importe quel éditeur de texte ordinaire.

En général si on copie deux colonnes d'un tableur et qu'on les colle dans l'éditeur de texte, on obtient exactement ce que l'on désire.

Une fois ce fichier « `nombresCopiesParNote.txt` » placé dans le dossier contenant le source `.tex` du document en cours de rédaction, on obtiendra exactement le même résultat en remplaçant la commande :

```
\draw plot coordinates {\nombresCopiesParNote};
```

ou la commande :

```
\draw plot coordinates {(0,0) (1,0) (2,2) (3,0)
(4,6) (5,4) (6,7) (7,4) (8,3) (9,0) (10,1)};
```

par la commande :

```
\draw plot file {nombresCopiesParNote.txt};
```

Le principal avantage de cette approche est qu'il est, dans ce cas, facile de changer les données dans le fichier texte auxiliaire, sans modifier le code source `.tex` du document. Une simple compilation permet alors une mise à jour rapide de l'affichage des données.

Un autre avantage concerne le traitement des données statistiques de grande taille (des centaines de points). Dans ce cas l'inclusion des données dans le source serait trop pénible et provoquerait des risques d'erreurs.

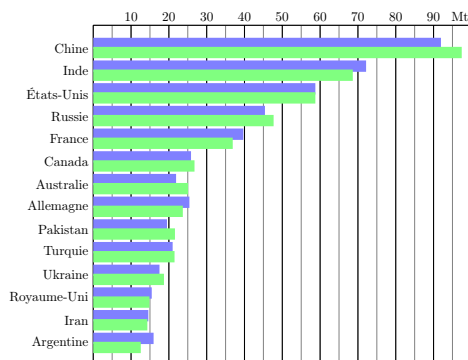
## 5.2 Diagramme à barres horizontales

### 5.2.1 Le blé dans le monde : utilisation d'un tableur

On désire présenter sur un diagramme, de façon significative, les deux séries statistiques suivantes. Ces données représentent la production de blé pour les années 2004 et 2005 (en millions de tonnes), pour les principaux pays producteurs dans le monde :

Pays	Production 2004	Production 2005
Chine	91,96	97,45
Inde	72,16	68,64
États-Unis	58,74	58,74
Russie	45,41	47,70
France	39,69	36,89
Canada	25,86	26,78
Australie	21,91	25,09
Allemagne	25,43	23,69
Pakistan	19,50	21,61
Turquie	21,00	21,50
Ukraine	17,52	18,70
Royaume-Uni	15,47	14,86
Iran	14,57	14,31
Argentine	15,96	12,57

On veut obtenir une figure de ce type :



La solution la plus rapide est de réaliser le diagramme à l'aide d'un logiciel adapté, comme un tableur, par exemple. On peut alors exporter la figure et l'inclure dans le document. Hélas, dans ce cas, le style de la figure ne sera pas parfaitement assorti à la typographie du document, et on risque de rencontrer des difficultés pour la mise à l'échelle de la figure.

Faire la figure avec `TikZ` réclame évidemment un petit effort qui sera largement validé par la qualité du résultat obtenu.

Comme nous l'avons vu précédemment, l'opération `plot` effectue un tracé à l'aide de coordonnées de points.

Nous pouvons décider, par exemple, de représenter chaque valeur par une barre horizontale. Chaque valeur est donc l'abscisse d'un point dont il faut fixer l'ordonnée.



Pour cela, nous allons procéder, à l'aide du tableur, à un pré-traitement des données pour préparer les fichiers textes nécessaires.

On va produire les deux fichiers suivants :

"producBle2004.txt"	"producBle2005.txt"
91.96 14	97.45 14
72.16 13	68.64 13
58.74 12	58.74 12
45.41 11	47.70 11
39.69 10	36.89 10
25.86 9	26.78 9
21.91 8	25.09 8
25.43 7	23.69 7
19.50 6	21.61 6
21.00 5	21.50 5
17.52 4	18.70 4
15.47 3	14.86 3
14.57 2	14.31 2
15.96 1	12.57 1

On a placé les données en abscisse et on a numéroté les ordonnées par valeurs décroissantes pour que les barres horizontales se placent les unes en dessous des autres, par pays. On utilisera l'option `xcomb` de l'opération `plot` qui permet de relier chaque point à l'axe des ordonnées par une barre horizontale, comme on a vu précédemment.

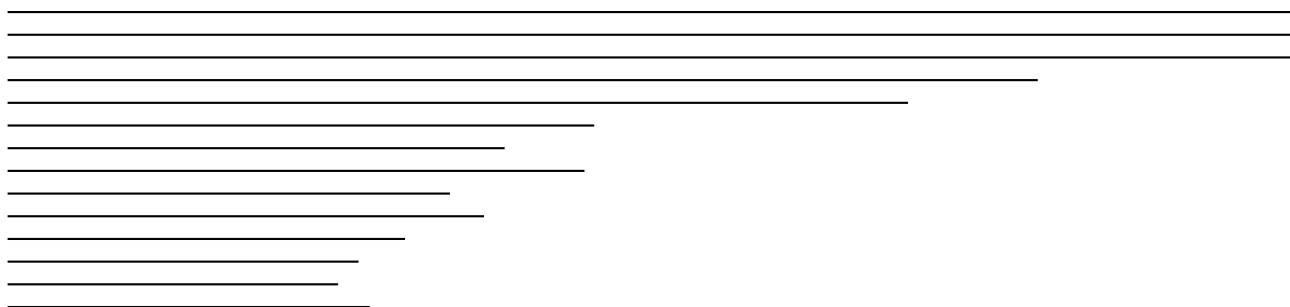
**Attention :** en français le séparateur décimal est la virgule. On a utilisé la fonction de recherche-remplacement de notre éditeur de texte pour mettre des points décimaux à la place des virgules.

### 5.2.2 Barres horizontales : `plot file, xcomb`

Maintenant, si on utilise la commande :

```
\draw[thick] plot[xcomb] file {producBle2004.txt};
```

Le résultat donne quelque chose de ce type :



On constate que la figure déborde dans la marge droite de la feuille. En effet la Chine produit 91,96 Mt de blé, or l'unité par défaut de TikZ est le centimètre et on obtient donc une barre de presque un mètre.

Il va falloir jouer avec l'échelle de la figure pour obtenir un affichage de taille correcte en ajoutant des options à l'environnement `tikzpicture`.

Un facteur multiplicatif pour les abscisses peut être défini à la suite de l'option `xscale=`.

Pour que la figure tienne sur la largeur de la feuille, ce facteur doit être inférieur à 0,1 et il sera nécessaire de faire quelques essais pour parvenir à un résultat satisfaisant.

Comme il y a 14 pays dans la liste, le diagramme fera au moins 14 cm de haut, ce qui est beaucoup.

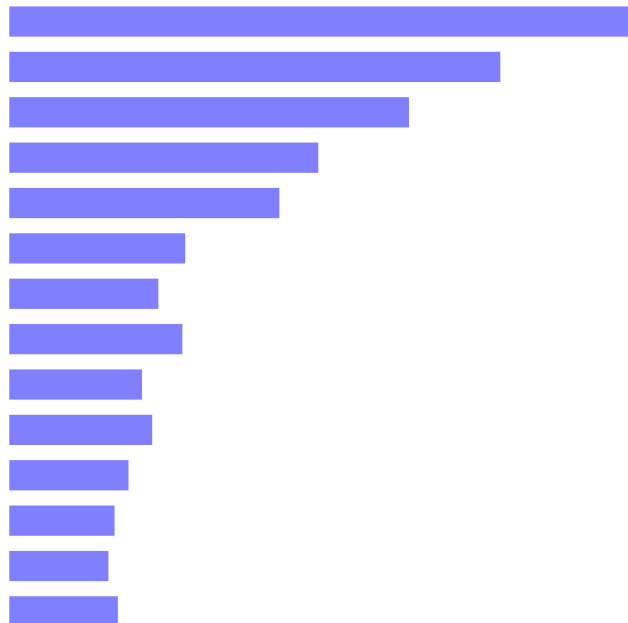
On pourra, de même, réduire cette hauteur en fixant un facteur multiplicatif pour les ordonnées à la suite de l'option `yscale=`.

Voici ce qu'on obtient avec, par exemple : `[xscale=0.09,yscale=0.6]`



On peut maintenant se préoccuper du style des barres. Si on veut une présentation en histogramme on peut écrire par exemple :

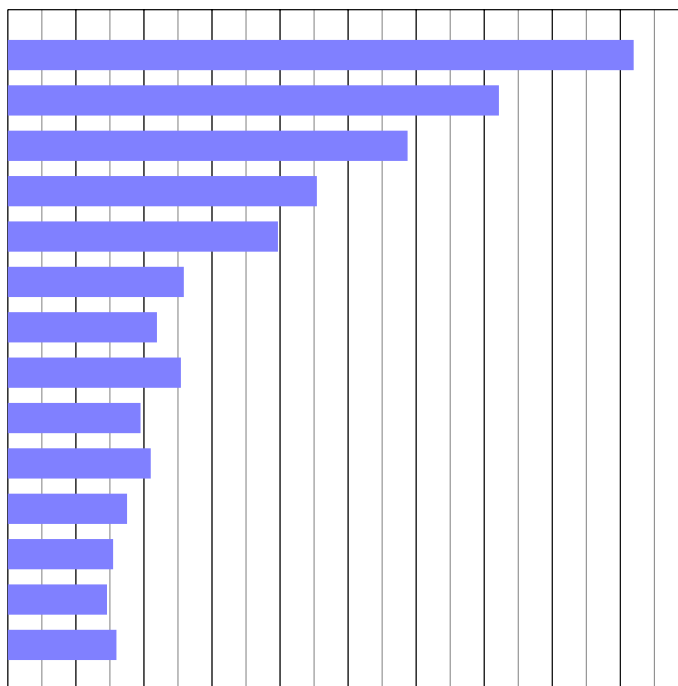
```
\draw[line width=4mm,color=blue!50]  
plot[xcomb] file {producBle2004.txt};
```



### 5.2.3 Installation d'une grille : `grid`, `xstep`, `ystep`

On va placer des lignes verticales pour faciliter la lecture des informations. On utilise pour cela une grille dont le pas sera astucieusement choisi :

```
\draw (0,0) grid[xstep=10,ystep=15] (100,15);
```



L'option `xstep=10` place les lignes verticales tous les 10 millions de tonnes et l'option `ystep=15` place les lignes horizontales toutes les 15 unités, ce qui représente en réalité uniquement la ligne du bas et la ligne du haut.

On a aussi ajouté une sous-grille en gris de pas plus petit (5 Mt) :

```
\draw[gray,very thin] (0,0) grid[xstep=5,ystep=15] (100,15);
```

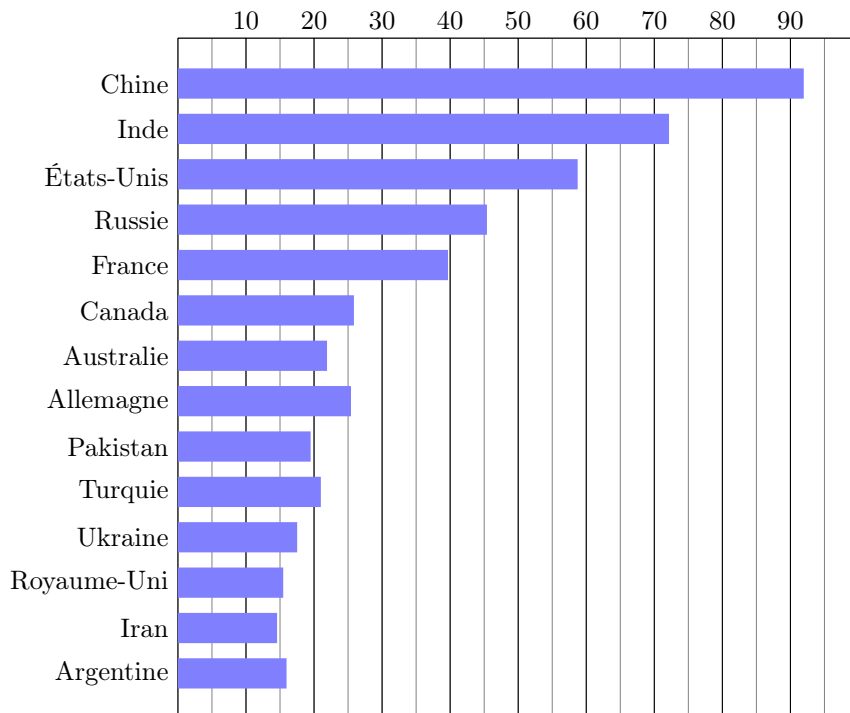
Cette commande a été placée avant la commande précédente dans l'environnement `tikzpicture` de façon à ce que le reste de la figure soit dessiné par dessus.

### 5.2.4 Étiquetage du repère : `\foreach`, `node`

On utilisera ensuite des commandes `\foreach` pour graduer les abscisses de la grille de repérage et pour placer les noms de pays en ordonnées.

```
\foreach \x in {10,20,...,90} \draw(\x,15)node[above]{\x};
```

Cette commande place les nombres en haut du diagramme.



```
\foreach \n/\y in {Chine/14,Inde/13,États-Unis/12,Russie/11,
  France/10,Canada/9,Australie/8,Allemagne/7,Pakistan/6,
  Turquie/5,Ukraine/4,Royaume-Uni/3,Iran/2,Argentine/1}
  \draw (0,\y) node [left] {\n};
```

Nous utilisons ici une nouvelle forme de la commande `\foreach` qui permet de contrôler plusieurs variables simultanément. Les valeurs proposées entre accolades sont, dans ce cas, une liste de valeurs multiples.

Dans cette forme de la commande `\foreach`, il y a ici deux variables :

`\n` qui va prendre pour valeurs les noms de ville

`\y` qui va prendre pour valeurs les ordonnées du point d'étiquetage

De même que les deux variables sont séparées par un `/` dans la déclaration `\n/\y` leurs valeurs respectives dans la liste qui suit sont aussi séparées par un `/` comme pour `Chine/14` par exemple.

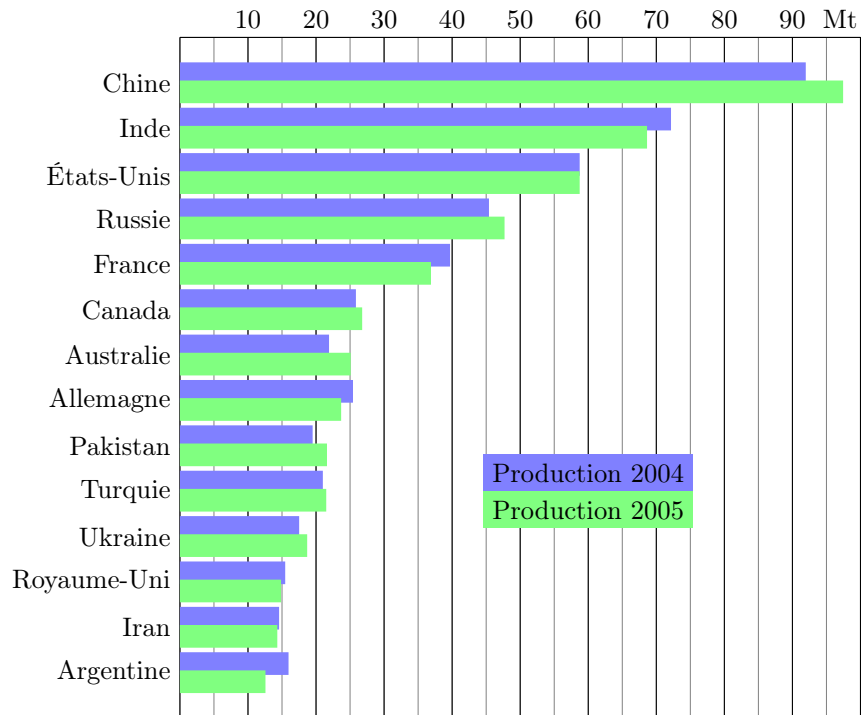
### 5.2.5 Deux séries plus une légende : `plot`, `shift`, `node`

Il est possible de placer la seconde série statistique (la production de blé 2005) sur le même diagramme en utilisant une couleur différente et en décalant légèrement les barres des deux séries verticalement avec `yshift`. La première série est décalée de 2 mm vers le haut avec `yshift=2mm`, et la seconde série est décalée de 2 mm vers le bas avec `yshift=-2mm` :

```
\draw [line width=3mm,color=blue!50,yshift=2mm]
  plot[xcomb] file {producBle2004.txt};
\draw [line width=3mm,color=green!50,yshift=-2mm]
  plot[xcomb] file {producBle2005.txt};
```

C'est une bonne idée d'ajouter par la même occasion une légende, dans deux nœuds correctement placés :

```
\draw(60,5)node[fill=blue!50,above] {Production 2004};
\draw(60,5)node[fill=green!50,below] {Production 2005};
```



Les principaux pays producteurs de blé (en millions de tonnes)

### 5.3 Courbe des variations de données

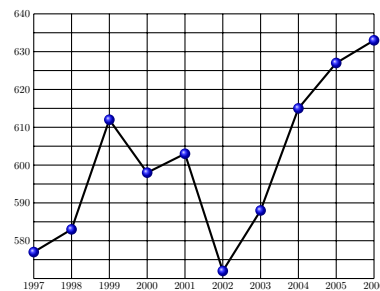
#### 5.3.1 Production annuelle de riz : pré-traitement

On désire présenter par une courbe l'évolution de la production mondiale de riz durant les années 1997 à 2006.

Les données sont présentées dans la table suivante :

Année	Production de riz en Mt
1997	577
1998	583
1999	612
2000	598
2001	603
2002	572
2003	588
2004	615
2005	627
2006	633

On désire présenter les variations ainsi :



Il semble tout à fait naturel de penser à procéder de la même façon que dans le paragraphe précédent :

- créer un fichier de texte contenant ces données ;
- utiliser la commande `\draw plot file {nomDuFichier.txt}`.

Hélas, ce n'est pas aussi simple : *TikZ* supporte assez mal les points dont les coordonnées sont grandes. L'unité par défaut du dessin est le centimètre. Avec 2006 on obtient donc une figure de plus de 20 m de large.

On peut alors espérer utiliser l'option `scale` pour modifier l'échelle du dessin. Hélas, ça ne marche pas non plus, car *TikZ* utilise aussi pour ses calculs internes des nombres d'amplitude limitée.

On découvre expérimentalement (ça ne semble pas clairement documenté dans le manuel de l'auteur) qu'il vaut mieux limiter les coordonnées des points à des valeurs comprise entre  $-500$  cm et  $+500$  cm. On peut malgré tout utiliser des nombres plus grands dans les calcul, *TikZ* est simplement incapable d'utiliser des points dont les coordonnées sortent de cet intervalle. Il est dans ce cas impossible de dessiner, même en réduisant la fenêtre d'affichage avec la commande `\clip` ou en changeant d'échelle avec l'option `scale`.

Si une erreur de compilation survient avec un message de ce type :

```
! dimension too large. <recently read> \pgf@yy
```

on doit penser à vérifier que certaines coordonnées ne sont pas trop grandes.

Nos données ne sont pas du tout comprises dans cet intervalle. Nous allons donc effectuer un pré-traitement, à l'aide d'un tableur par exemple, pour obtenir des données plus adaptées au dessin. On peut décider de :

- soustraire 1990 aux années ;
- diviser par 10 les nombres de millions de tonnes.

On produit ainsi le fichier : "producRiz.txt"	7	57.7
	8	58.3
	9	61.2
	10	59.8
	11	60.3
	12	57.2
	13	58.8
	14	61.5
	15	62.7
	16	63.3

### 5.3.2 Courbe des variations : plot file

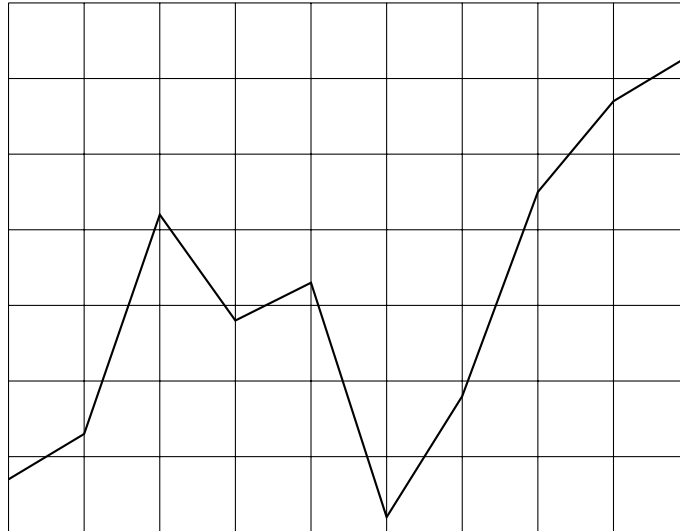
La commande `\draw plot file {producRiz.txt}`; permet d'obtenir immédiatement un résultat visible :



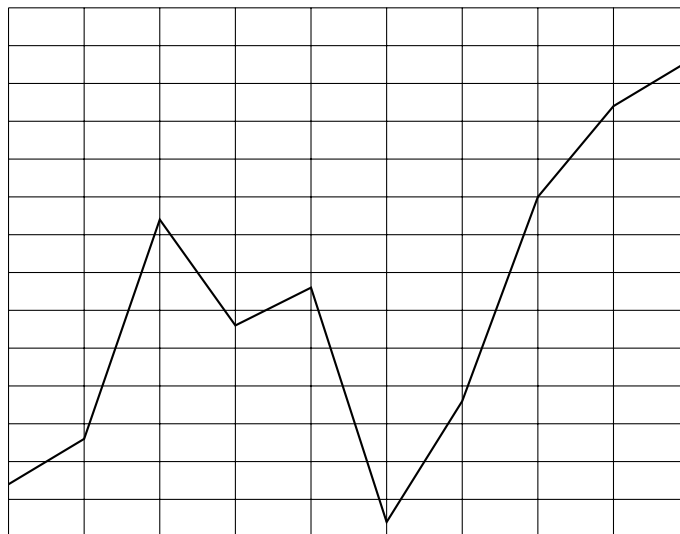
### 5.3.3 Quadrillage : grid, step

On va ajouter une grille en faisant varier les abscisses entre 7 (pour 1997) et 16 (pour 2006), les ordonnées entre 57 (pour un minimum de 57,2) et 64 (pour un maximum de 63,3) :

```
\draw (7,57) grid (16,64);
```



Pour faciliter la lecture, on peut quadriller plus finement selon les ordonnées en fixant pour l'opération `grid` l'option `[ystep=0.5]` :



On constate que la ligne horizontale inférieure du quadrillage est absente. En effet les erreurs d'arrondi dans le calcul du quadrillage produisent parfois des problèmes de ce type.

Pour corriger ce problème, il suffit de faire quelques essais en modifiant progressivement la valeur de l'option `ystep` qui fixe le pas vertical de la grille du quadrillage. Dans le cas présent, on obtient un résultat satisfaisant en augmentant très légèrement le pas, avec :

```
\draw (7,57) grid [ystep=5.00001mm] (16,64);
```

### 5.3.4 Annotations, décorations : `\foreach`, `node`, `mark`

On peut maintenant étiqueter les axes du quadrillage pour faciliter la lecture des données qui sont affichées.

On commence par l'axe des ordonnées avec la commande `\foreach` :

```
\foreach \y in {58,59,...,64}
  \draw (7,\y) node[left,scale=0.8]{\y0};
```

Pour l'étiquetage des ordonnées, on remarquera que la variable `\y` prend les valeurs de 58 à 64, mais que le contenu du nœud est `{\y0}`, c'est-à-dire la valeur de `\y` suivie du caractère 0, ce qui fera donc : 580, 590, ... jusqu'à 640.

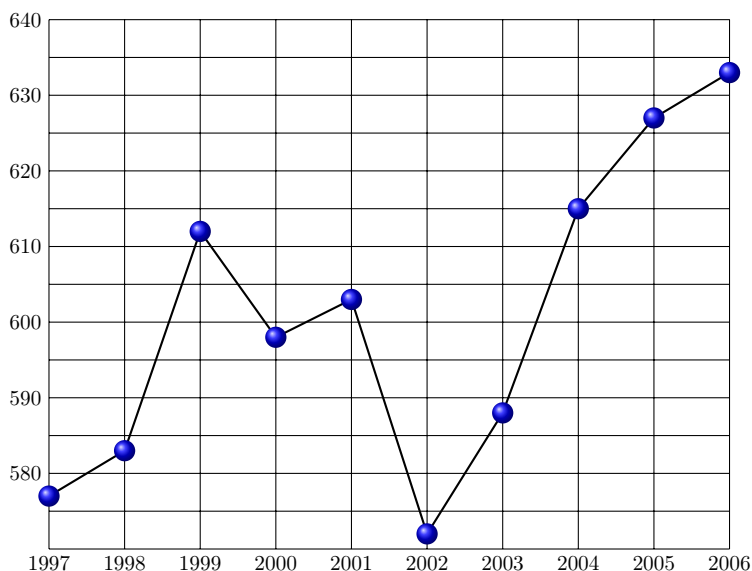
```
\foreach \x in {1997,1998,...,2006}
  \draw (\x-1990,57) node [below,scale=0.8] {\x};
```

Pour l'étiquetage des abscisses, on utilise les capacités de calcul de la version 2 de PGF. La variable `\x` prend les valeurs successives de 1997 à 2006, mais les nœuds contenant les années doivent être placés aux points d'abscisses successives 7, 8, ... jusqu'à 16. On place donc chaque nœud en `(\x-1990,57)`

L'abscisse `\x-1990` est la valeur de `\x` à laquelle on soustrait 1990. Ainsi les abscisses seront bien les valeurs 7, 8, ... jusqu'à 16 (valeurs inférieures à 500).

Cette syntaxe, qui accepte des formules de calcul dans les commandes `TikZ`, est réservée à la version 2 de `TikZ & PGF`. De plus elle ne fonctionne que si le bibliothèque `calc` a été chargée dans le préambule, dans lequel il faut donc placer `\usetikzlibrary{calc}` juste après `\usepackage{tikz}`.

Pour terminer, on va effectuer une petite amélioration esthétique de la figure en accentuant la visibilité des points significatifs. Pour cela, on ajoute, à l'opération `plot`, l'option `[mark=ball,mark size=3pt]` pour obtenir :



Production annuelle mondiale de riz en million de tonnes



## 5.4 Diagramme à secteurs

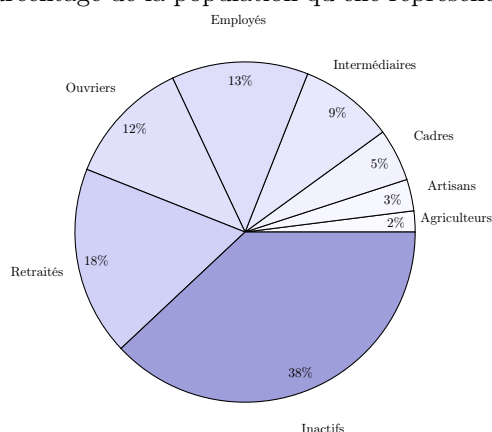
### 5.4.1 Répartition par catégories socioprofessionnelles

Voici la répartition, en pourcentage, de la population française, par catégories socioprofessionnelles, pour l'année 1999 :

Catégorie	%
Agriculteurs exploitants	2
Artisans, commerçants, chefs d'entreprise	3
Cadres, professions intellectuelles supérieures	5
Professions intermédiaires	9
Employés	13
Ouvriers	12
Retraités	18
Autres sans activité professionnelle	38

Dans ce cas, il est assez naturel de représenter ces données à l'aide d'un diagramme à secteurs. Le disque complet représentera la population française. Chaque catégorie socioprofessionnelle sera associée à un secteur d'angle proportionnel au pourcentage de la population qu'elle représente.

Voici l'aspect de la figure que l'on désire réaliser :

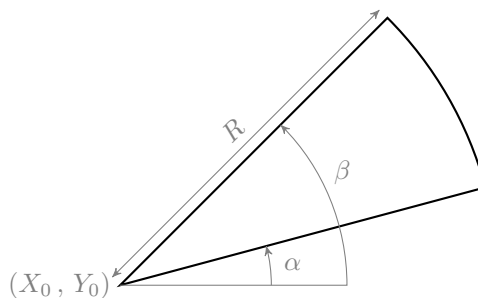


### 5.4.2 Calcul des angles : pré-traitement avec un tableur

Dans *TikZ*, il n'y a pas de commande spécifique prévue pour construire automatiquement des diagrammes à secteurs.

Il va donc falloir construire chaque secteur à l'aide de l'opération `arc` écrite sous la forme suivante :

```
\draw (X_0, Y_0) -- (\alpha : R) arc (\alpha : \beta : R) -- cycle;
```



On constate qu'il est nécessaire, pour tracer un secteur, de calculer au préalable les angles qui définissent sa forme, l'angle du secteur, mais aussi l'angle que fait le rayon du début du secteur avec l'horizontale, puis celui que fait le rayon de la fin du secteur avec l'horizontale :

Catégorie	%	angle en °	début du secteur	fin du secteur
Agriculteurs	2	7,2	0	7,2
Artisans	3	10,8	7,2	18
Cadres	5	18	18	36
Intermédiaires	9	32,4	36	68,4
Employés	13	46,8	68,4	115,2
Ouvriers	12	43,2	115,2	158,4
Retraités	18	64,8	158,4	223,2
inactifs	38	136,8	223,2	360

Ces résultats ont été obtenus à l'aide d'un tableur.

### 5.4.3 Dessiner les secteurs : `\draw`, `arc`, `cycle`, `fill`, `$`

Pour dessiner un secteur on utilise son angle de début et son angle de fin.

On peut aussi étiqueter ce secteur, par exemple, en notant la catégorie à l'extérieur et en plaçant le pourcentage à l'intérieur.

Expliquons en détail deux exemples :

- Agriculteurs (2% secteur de  $0^\circ$  à  $7,2^\circ$ );
- Employés (13% secteur de  $68,4^\circ$  à  $115,2^\circ$ ).

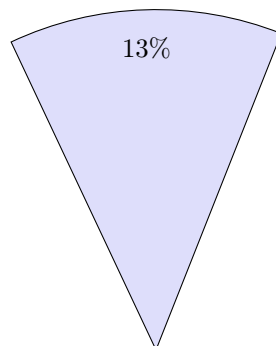


```
\draw[fill=black!2!blue!2] (0,0)--(0:4.5)arc(0:7.2:4.5)--cycle;
\draw (3.6:4) node {2\%};
\draw (3.6:5.6) node {Agriculteurs};
```

Le secteur est centré à l'origine  $(X_0, Y_0) = (0, 0)$ , le rayon du cercle est  $R = 4,5$  cm et le secteur de  $\alpha = 0^\circ$  à  $\beta = 7,2^\circ$  est colorié avec un mélange de 2% de noir et de bleu [`fill=black!2!blue!2`], c'est-à-dire proportionnellement au pourcentage de la population qu'il représente. Ainsi le secteur sera d'autant plus sombre que son angle sera grand.

L'étiquette « 2% » est placée à 4 cm du centre, à l'intérieur du secteur, avec un angle de  $3,6^\circ$  (la moitié de  $7,2^\circ$ ) c'est-à-dire sur la bissectrice de l'angle du secteur. L'étiquette « Agriculteurs » est placée, avec le même angle à 5,6 cm du centre, c'est-à-dire à l'extérieur du secteur.

Employés



```
\draw[fill=black!13!blue!13]
(0,0)--(68.4:4.5)arc(68.4:115.2:4.5)--cycle;
\draw ({(68.4+ 115.2)/2}:4) node {13\%};
\draw ({(68.4+ 115.2)/2}:5.6) node {Employés};
```

Cette fois-ci, le secteur de,  $\alpha = 68,4^\circ$  à  $\beta = 115,2^\circ$ , est colorié avec un mélange de 13% de noir et de bleu. Il est donc plus sombre que le précédent.

Pour aligner les étiquettes, l'angle de la bissectrice  $\frac{\alpha + \beta}{2}$  a été calculé comme moyenne des angles  $\alpha$  et  $\beta$  qui délimitent le secteur.

On a utilisé pour cela les capacités de calcul de la version 2 de PGF :

ainsi,  $\{(68.4+115.2)/2\}:4$  sera évalué à  $(91.8:4)$

On remarque ici que pour effectuer le calcul, l'expression à évaluer doit être placée entre accolades  $\{(68.4+115.2)/2\}$ , pour éviter les conflits de syntaxe.

En résumé, sur ces deux exemples, on remarque que les trois commandes utilisées pour la construction de chaque secteur colorié et étiqueté, ont la structure générale suivante :

```
\draw[fill=black!\p!blue!\p]
  (0,0) -- (\a:4.5) arc (\a:\b:4.5) -- cycle;
\draw (\{(\a+\b)/2\}:4) node {\p\%};
\draw (\{(\a+\b)/2\}:5.6) node {\c};
```

- $\backslash a$  est l'angle  $\alpha$ 
  - utilisé pour définir de début du secteur.
- $\backslash b$  est l'angle  $\beta$ 
  - utilisé pour définir de fin du secteur.
- $\backslash p$  est le pourcentage de la population représentée
  - utilisé pour fixer l'intensité la couleur ;
  - utilisé pour définir le texte l'étiquette intérieure.
- $\backslash c$  est la catégorie socioprofessionnelle
  - utilisée pour définir le texte l'étiquette extérieure.

De plus, la taille du diagramme complet est contrôlée par les valeurs des rayons de trois cercles :

- le cercle des secteurs : de rayon  $R = 4,5$  (en centimètres)
- le cercle des étiquettes intérieures de pourcentage : de rayon 4
- le cercle des étiquettes extérieures de catégorie : de rayon 5,6

#### 5.4.4 Diagramme complet : $\backslash foreach$

Pour dessiner tous les secteurs du diagramme, les commandes ci-dessus doivent être exécutées en faisant varier les valeurs de  $\backslash a$ ,  $\backslash b$ ,  $\backslash p$  et  $\backslash c$ .

On va utiliser pour cela quatre variables  $\backslash a/\backslash b/\backslash p/\backslash c$  dans la commande  $\backslash foreach$  et créer la liste des valeurs que doivent prendre ces variables à l'aide du tableau donné au début.

Par exemple, les trois premières lignes du tableau :

Catégorie	%	angle en °	début du secteur	fin du secteur
Agriculteurs	2	7,2	0	7,2
Artisans	3	10,8	7,2	18
Cadres	5	18	18	36

vont fournir les valeurs respectives des variables  $\backslash a/\backslash b/\backslash p/\backslash c$  sous la forme :

$\{0/7.2/2/Agriculteurs, 7.2/18/3/Artisans, 18/36/5/Cadres, \dots\}$

On obtiendra ainsi avec une commande  $\backslash foreach$  unique le diagramme complet avec tous les secteurs coloriés et étiquetés :

```
\begin{tikzpicture}
\foreach \a/\b/\p/\c in
{
0/7.2/2/Agriculteurs, 7.2/18/3/Artisans,
18/36/5/Cadres, 36/68.4/9/Intermédiaires,
68.4/115.2/13/Employés, 15.2/158.4/12/Ouvriers,
158.4/223.2/18/Retraités, 223.2/360/38/Inactifs
```

```

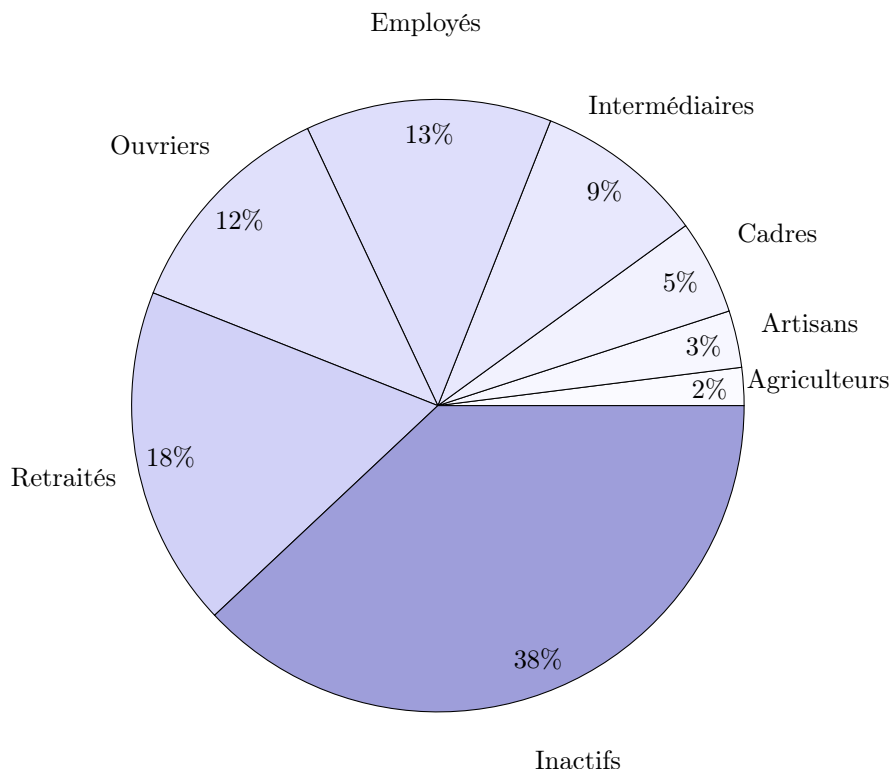
}
{
\draw[fill=black!\p!blue!\p]
(0,0) -- (\a:4.5) arc (\a:\b:4.5) -- cycle;
\draw ({(\a+\b)/2}:4) node {\p\%};
\draw ({(\a+\b)/2}:5.6) node {\c};
}
\end{tikzpicture}

```

On remarque ci-dessus :

- la liste des données, entre accolades, après `\foreach \a/\b/\p/\c in`
- les trois commandes de construction d'un secteur qui sont insérées entre des accolades et forme donc le bloc des instructions exécutées à chaque étape par la commande `\foreach`

Et voici le diagramme obtenu :



Répartition par catégorie socioprofessionnelle en France en 1999

## 5.5 Résumé

On a vu ici des exemples de représentations graphiques de données.

À partir de séries statistiques variées enregistrées sous forme de fichiers de texte, on a dessiné, des courbes, des diagrammes à barres horizontales ou verticales, des histogrammes et des diagrammes à secteurs.

On a montré qu'il est souvent nécessaire d'effectuer un traitement des données au préalable à l'aide d'une application auxiliaire comme un tableur par exemple, pour obtenir les coordonnées des points significatifs du diagramme.

On a aussi constaté qu'un diagramme se construit par essais successifs, en construisant d'abord une première figure, puis en procédant à des modifications progressives pour obtenir une figure satisfaisante.

**Attention :** Ne jamais utiliser de coordonnées de points en dehors de l'intervalle  $[-500; 500]$  en centimètres.

# Chapitre 6

## Graphes : Introduction

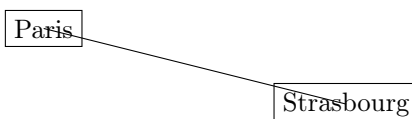
### 6.1 Notions de base

On va montrer ici, à l'aide d'exemples simples, comment construire des *diagrammes de graphes* constitués de nœuds reliés par des arcs.

#### 6.1.1 Nœuds et Arcs : `\draw`, `--`, `node`, et `\node`

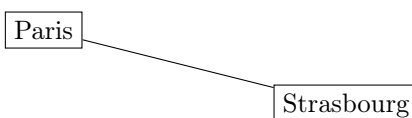
Allons d'abord de Paris à Strasbourg, le plus simple est d'écrire :

```
\draw (0,0)node[draw]{Paris} -- (4,-1)node[draw]{Strasbourg};
```



Mais on peut aussi définir d'abord les villes puis les relier ensuite :

```
\node[draw] (P) at (0,0) {Paris};  
\node[draw] (S) at (4,-1) {Strasbourg};  
\draw (P) -- (S);
```



Si l'on regarde attentivement, on constate que les deux écritures donnent des résultats légèrement différents :

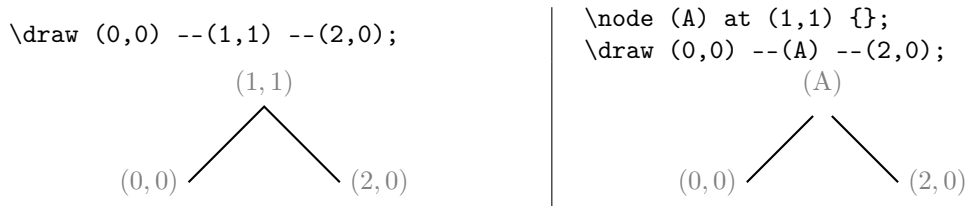
- Dans le premier cas, `node` est une opération exécutée sur un chemin. On peut considérer chaque nœud comme une décoration du point auquel il est associé. La ligne tracée par la commande `\draw` joint deux points, les nœuds sont des objets ajoutés ensuite et centrés sur les points. L'option `draw` de l'opération `node` trace le contour du nœud.
- Dans le second cas, `\node` est une commande TikZ qui permet de définir un nœud, de le nommer et de le dessiner. On peut alors considérer les nœuds comme des objets préexistants que l'on va ensuite relier avec la commande `\draw`. Ces nœuds ne sont pas des points, ils ont une certaine dimension, la ligne tracée joint les bords de ces objets.

La syntaxe de définition d'un nœud avec la commande `\node` est :

```
\node[<options>] (<nom>) at (<position>) {<contenu>;}
```

Par défaut, un nœud a la forme d'un rectangle qui englobe son contenu comme on le voit dans les deux cas de figure ci-dessus.

S'il n'y a pas l'option `draw`, le contour n'est pas tracé, mais le rectangle a des dimensions non nulles *même si le contenu est vide* comme on peut le constater en comparant les deux exemples ci-dessous :



Dans la figure de gauche, le trait du chemin passe par les trois points de référence, il n'y a pas de nœud.

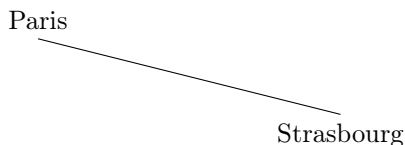
Dans celle de droite, le nœud, nommé (A), qui a été dessiné est visible, bien qu'il soit vide, car le trait du chemin s'interrompt à la frontière du nœud, alors que celle-ci est elle-même invisible.

### 6.1.2 Chemin annoté : `\draw` avec opération `node`

La première méthode (ajouter des nœuds sur un chemin) a déjà été utilisée précédemment, par exemple pour annoter les figures de géométrie.

On peut préciser la position des nœuds autour des points du chemin en ajoutant à l'opération `node` des options de position comme `above` ou `below` par exemple.

```
\draw (0,0)node[above]{Paris}--(4,-1)node[below]{Strasbourg};
```

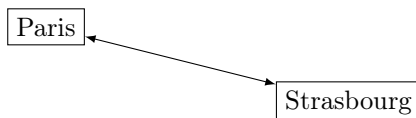


### 6.1.3 Graphe : `\node` puis `\draw` avec nom de nœud

La seconde méthode (relier par un chemin des nœuds nommés précédemment) nous intéresse davantage dans ce chapitre, car elle sera plutôt utilisée pour construire des diagrammes de graphes.

Dans ce cas, on modifiera le style des traits, à l'aide d'options de la commande `\draw`, comme par exemple `[<->,>=latex]` pour obtenir des flèches :

```
\draw[<->,>=latex] (P) -- (S);
```



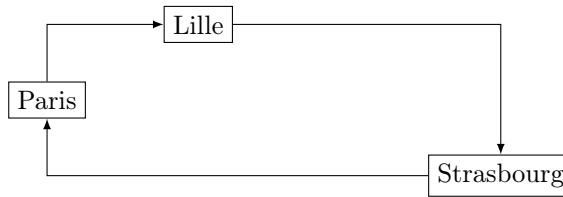
## 6.2 Styles des nœuds et des arcs

### 6.2.1 Les arcs : `\draw`, `--`, `|-`, `-|`, `to` et options de flèches

Nous allons maintenant étudier comment construire des graphes constitués de nœuds et d'arcs. On présentera les différentes options qui permettent de préciser le style des arcs et des nœuds.

Nous avons constaté sur l'exemple précédent que l'arc entre deux nœuds est une ligne droite qui relie les centres des nœuds, mais s'arrête à la périphérie, sur la frontière rectangulaire du nœud.

On peut remplacer la liaison `--` par une liaison `|-` (départ vertical, arrivée horizontale) ou `-|` (départ horizontal, arrivée verticale) :

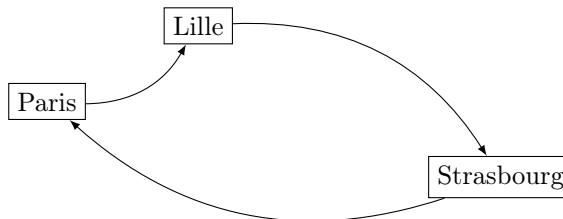


```

\draw[->,>=latex] (P) |- (L);
\draw[->,>=latex] (L) -| (S);
\draw[->,>=latex] (S) -| (P);
  
```

On peut aussi utiliser une liaison `to`, suivie d'une option de courbure.

Pour cette option de courbure, le plus simple est d'indiquer vaguement ce que l'on veut, soit une *courbure vers la droite* avec `[bend right]`, soit une *courbure vers la gauche* avec `[bend left]` :



```

\draw[->,>=latex] (P) to[bend right] (L);
\draw[->,>=latex] (L) to[bend left] (S);
\draw[->,>=latex] (S) to[bend left] (P);
  
```

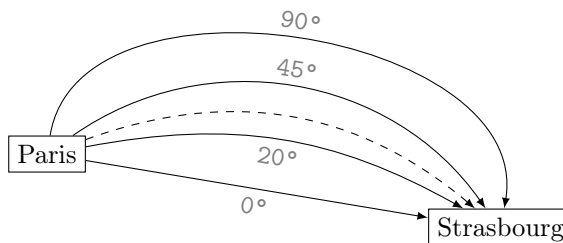
Pour l'option `bend`, la courbure est relative à la direction de parcours du chemin en ligne droite défini par deux points successifs.

Attention : Le mot courbure prête ici à confusion. Pour l'option `bend left` par exemple, cela signifie que le chemin subit une déformation courbe qui le place sur la gauche, dans le sens du parcours du chemin en ligne droite directe.

Pour que tout soit parfaitement clair, voici quelque exemples de flèches tracées avec `to[bend left]`, le chemin direct étant visualisé en gris :



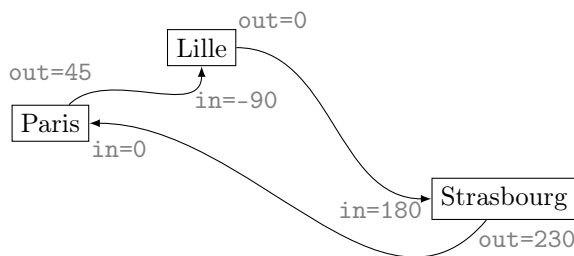
L'importance de la courbure de l'option `bend` peut être modifiée en précisant après un signe = un angle en degrés ( $0^\circ$  correspond à la ligne droite) :



```

\draw[->,>=latex] (P) to[bend left=0] (S);
\draw[->,>=latex] (P) to[bend left=20] (S);
\draw[->,>=latex,dashed] (P) to[bend left] (S);
\draw[->,>=latex] (P) to[bend left=45] (S);
\draw[->,>=latex] (P) to[bend left=90] (S);
  
```

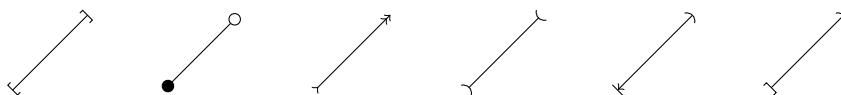
Avec l'opération de liaison `to`, on peut aussi être encore plus précis et utiliser l'option `out=` qui définit l'angle polaire en degrés de sortie du nœud origine et l'option `in=` pour l'angle d'entrée dans le nœud extrémité :



```
\draw[->,>=latex] (P) to[out=45,in=-90] (L);
\draw[->,>=latex] (L) to[out=0,in=180] (S);
\draw[->,>=latex] (S) to[out=230,in=0] (P);
```

### 6.2.2 Extrémités des arcs : [->|, \*-o, >->>, )-(

De nombreuses variations sont possibles pour définir les extrémités des arcs. On se reportera à la documentation générale pour plus d'informations sur ce sujet. Voici quelques exemples :



```
\draw[[-]] (0,0) -- (1,1); \draw[*-o] (2,0) -- (3,1);
\draw[>->>] (4,0) -- (5,1); \draw[)-()] (6,0) -- (7,1);
\draw[|<-)] (8,0) -- (9,1); \draw[{}-)] (10,0) -- (11,1);
```

On remarquera la nécessité d'utiliser des accolades dans le cas où on utilise le crochet. Ceci évite, lors de la lecture de l'option, une erreur de syntaxe.

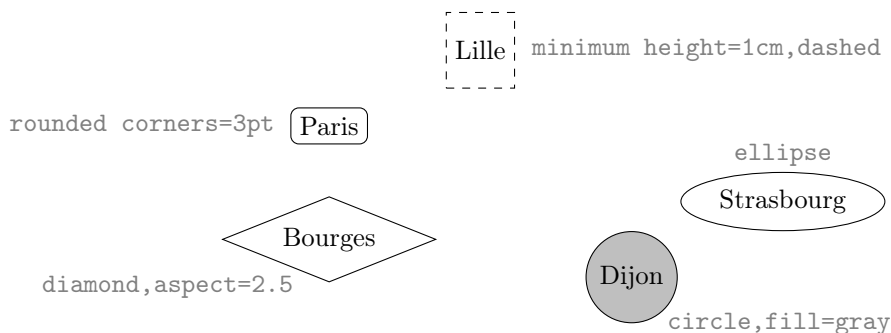
### 6.2.3 Frontières des nœuds : circle, ellipse, diamond

Nous avons vu l'option `draw` de `\node` qui trace la frontière du nœud.

Cette frontière peut être modifiée et décorée à l'aide de différentes options :

- de forme : `rectangle` (défaut), `circle`, `ellipse`, `diamond`;
- de style : `dashed`, `dotted`, `thick`, `red`, `fill`;
- de modification : `rounded corners`, `minimum width` ou `height`.

Pour que certaines options de forme soient disponibles, le chargement d'une bibliothèque (`library`) dans le préambule est nécessaire. Après le chargement de `\usepackage{tikz}` il faut ajouter `\usetikzlibrary{shapes}`





```

\node[draw,rectangle,rounded corners=3pt] (P)at(0,0){Paris};
\node[draw,minimum height=1cm,dashed] (L)at(2,1) {Lille};
\node[draw,ellipse] (S)at(6,-1) {Strasbourg};
\node[draw,diamond,aspect=2.5] (B)at(0,-1.5) {Bourges};
\node[draw,circle,fill=gray!50] (D)at(4,-2) {Dijon};

```

Remarque : L'option `diamond` est accompagnée de son option `aspect` qui fixe le rapport entre la largeur et la hauteur du nœud.

#### 6.2.4 Abstraction des styles : `\tikzstyle`, `\tikzset`

Pour les graphes un peu complexes dans lesquels figurent des nœuds et des arcs de styles variés, il est parfois utile de nommer les styles. Plutôt que de procéder aux copier-coller des options, on définira des styles à l'aide de la commande `\tikzstyle` (pour PGF version 1) ou de la commande `\tikzset` (pour PGF version 2).

On définit, par exemple, un style pour `ville` et le style de Paris sera, par exemple, `capitale` :

```

\tikzstyle{ville}=[draw,rectangle,rounded corners=3pt]
\tikzstyle{capitale}=[draw,ellipse,very thick,fill=black!25]

```

ou avec la commande `\tikzset` de PGF version 2 :

```

\tikzset{ville/.style={draw,rectangle,rounded corners=3pt}}
\tikzset{capitale/.style={draw,ellipse,very thick,fill=black!25}}

```

Il suffira alors d'écrire dans le code :

```

\node[capitale] (P)at(0,0){Paris};
\node[ville] (L)at(2,1){Lille}; . . .

```

On définit de même des styles `radial` et `transversal` pour les arcs :

```

\tikzset{radial/.style={very thick,->,>=stealth'}}
\tikzset{transversal/.style={<->,>=stealth',thick,dashed}}

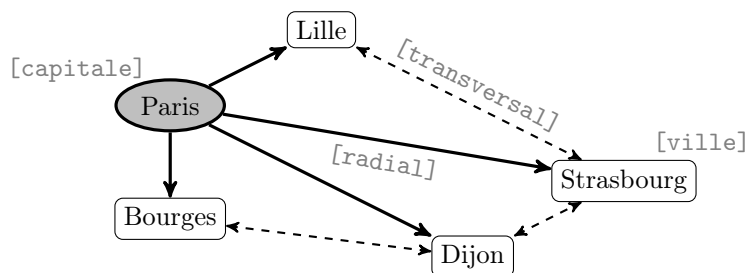
```

On peut alors effectuer les liaisons :

```

\draw[radial] (P)--(L);
\draw[transversal] (D)--(B); . . .

```



Remarque : Un des avantages principaux de la définition de style est de rendre le code beaucoup plus lisible.

Mais il y a plus important encore, les commandes `\tikzstyle` ou `\tikzset` peuvent être définies à l'extérieur de l'environnement `{tikzpicture}`. Dans ce cas, toutes les figures qui suivent pourront utiliser les styles définis. Ceci permet ainsi d'uniformiser le style de plusieurs figures distinctes. Par contre, si les commandes `\tikzstyle` ou `\tikzset` sont définies à l'intérieur d'un environnement `{tikzpicture}`, la portée est limitée à cet environnement.

**Attention :** La commande `\tikzstyle` de TikZ & PGF version 1 est toujours valable dans la version 2, mais, on lui préférera la nouvelle commande `\tikzset` qui permet de définir des styles, mais qui offre aussi des possibilités supplémentaires que l'on utilise par ailleurs.

Les définitions de style précédentes peuvent ainsi être définies globalement dans une unique commande `\tikzset`.

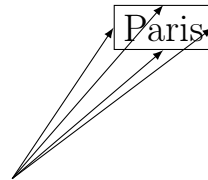
```
\tikzset{ville/.style={draw,rectangle,rounded corners=3pt},
  capitale/.style={draw,ellipse,very thick,fill=black!25},
  radial/.style={very thick,->,>=stealth'},
  transversal/.style={<->,>=stealth',thick,dashed}}
```

Les styles ainsi définis pourront être ensuite utilisés dans tous les environnements `{tikzpicture}` qui suivent.

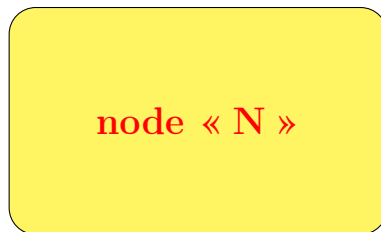
### 6.2.5 Points d'ancrage des nœuds : N.south, N.left, N.below

On a vu en géométrie comment placer un nœud autour d'un point. On montre ici comment atteindre certains points d'ancrage d'un nœud :

```
\node[draw] (N) at (2,2) {\Large Paris};
\draw[->,>=latex] (0,0) -- (N.north);
\draw[->,>=latex] (0,0) -- (N.south);
\draw[->,>=latex] (0,0) -- (N.west);
\draw[->,>=latex] (0,0) -- (N.east);
```

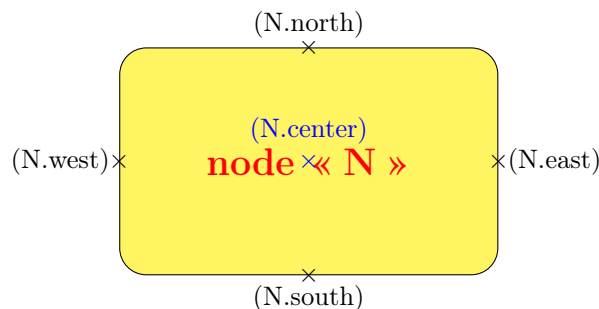


Pour visualiser les différents points d'ancrage d'un nœud, nous allons d'abord créer un nœud de grande taille en utilisant les options de décoration vues dans le paragraphe précédent pour définir son style :



```
\tikzstyle{noeud}=[minimum width=5cm,minimum height=3cm,
  rectangle,rounded corners=10pt,draw,
  fill=yellow!75,text=red,font=\bfseries]
\node[noeud] (N) at (0,0) {\Large node \og{}N\fg{}};
```

Une fois le nœud « N » placé, on dessine sur la figure les principaux points d'ancrage, avec leurs noms :

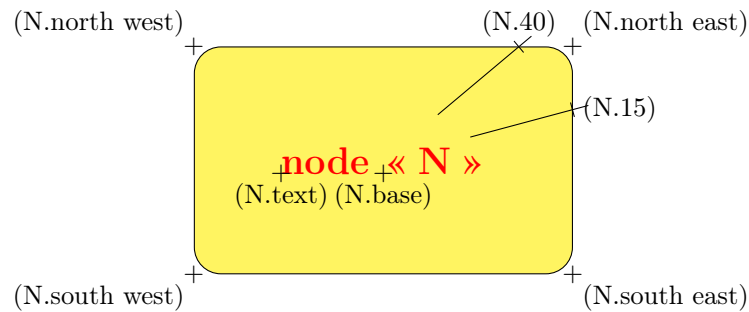


```

\draw (N.north) node[{$\times$}] node[above]{(N.north)};
\draw (N.south) node[{$\times$}] node[below]{(N.south)};
\draw (N.west) node[{$\times$}] node[left]{(N.west)};
\draw (N.east) node[{$\times$}] node[right]{(N.east)};
\draw[blue] (N.center) node[{$\times$}] node[above=3pt]{(N.center)};

```

En réalité, il y a de très nombreux points d'ancrage, ce qui permet, comme toujours avec TikZ d'être extrêmement précis dans la disposition des objets. Voici quelques autres des principaux points d'ancrage :



Les noms de tous ces points sont assez compréhensibles. On remarquera en particulier les points (N.15) et (N.40) qui font référence aux points d'intersection de la frontière du nœud avec la droite passant par le centre et qui fait un angle de 15° (respectivement 40°) avec l'horizontale.

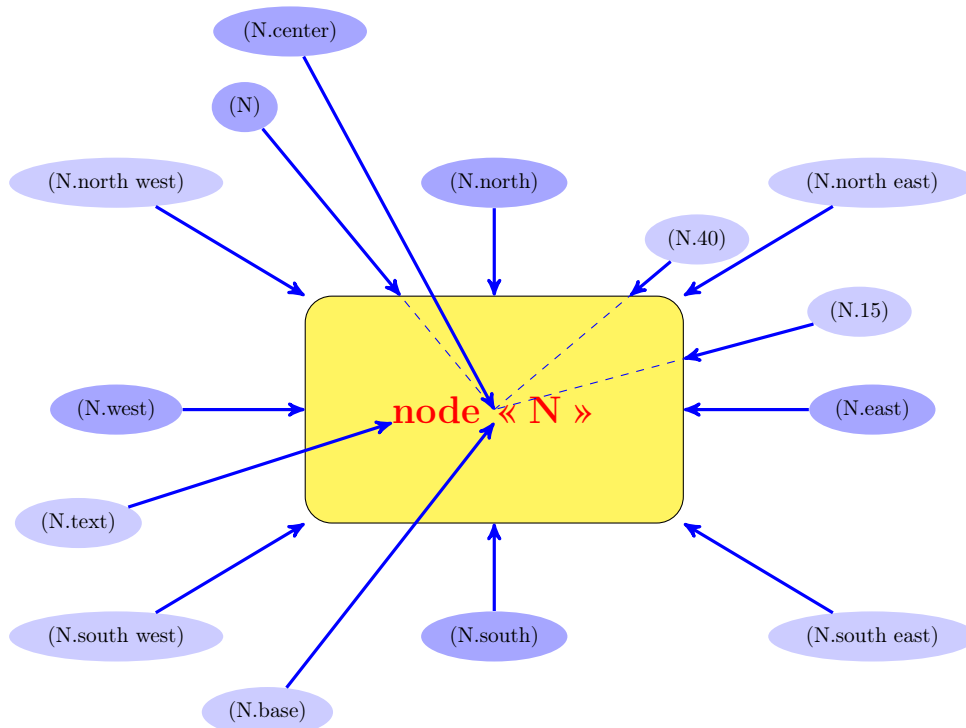
### 6.2.6 Flèches vers les ancrages : N.north, N.center, N.15

Le nom (N) fait référence au centre du nœud comme (N.center), mais n'est cependant pas identique. Une flèche qui pointe vers (N) a son extrémité sur la frontière du nœud, tandis qu'une flèche qui pointe vers (N.center) a son extrémité sur le centre du nœud comme on peut le constater sur la figure suivante qui résume les différents cas :

```

\draw[->] (etiquetteN) -- (N);
\draw[->] (etiquetteNcenter) -- (N.center);
...

```

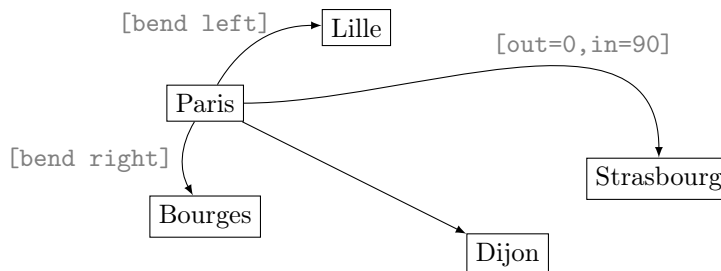


## 6.3 Techniques avancées

### 6.3.1 Tracer un arc sans avancer : edge

En plus des opérations de chemin `--`, `|-`, `-|` et `to`, qui définissent la forme des traits de liaison, on dispose aussi de `edge` (avec les mêmes options que pour l'opération `to`) qui dessine une ligne jusqu'au point suivant, mais laisse le crayon à l'origine pour les liaisons suivantes, en étoile :

```
\draw[->,>=latex] (P) edge[bend left] (L)
      edge[out=0,in=90] (S)
      edge (D)
      edge[bend right] (B);
```



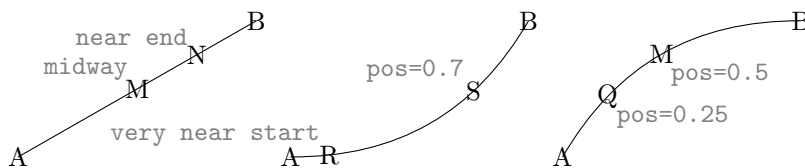
### 6.3.2 Étiquetage des arcs : sloped, midway, pos

Après spécification dans une commande `\draw` d'un chemin `(A)--(B)`, l'opération `node` qui suit place, par défaut, un nœud au point `(B)`. Si on ajoute à l'opération `node` l'option `midway` le nœud sera placé à mi-chemin du point `(A)` et du point `(B)`.

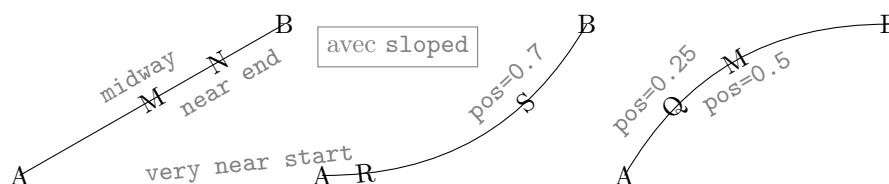
En plus de l'option `midway`, il existe aussi les options `very near start`, `near start`, `near end`, `very near end` pour les placements approximatifs, et l'option `pos=` qui permet une plus grande précision.

Le nombre qui suit `pos=` est le pourcentage du chemin à parcourir pour placer le nœud : 0 en `(A)`, 1 en `(B)`, 0.5 au milieu de `(A)--(B)`.

```
\draw (0,0) node{A} -- (4,2) node{B}
      node[midway]{M}   node[near end]{N};
\draw (5,0) node{A} to[bend right]
      node[very near start]{R}   node[pos=0.7]{S} (9,2) node{B};
\draw (9,0) node{A} to[bend left]
      node[pos=0.5]{M}   node[pos=0.25]{Q} (13,2) node{B};
```



Les étiquettes sont écrites horizontalement par défaut, mais comme on le voit sur les exemples ci-dessous, ajouter l'option `sloped` permet d'incliner les étiquettes selon la pente du chemin : `node[sloped,midway]{M}`



### 6.3.3 Inclinaison des étiquettes : `sloped`, `rotate`

Il est aussi possible de modifier l'inclinaison par défaut du texte de l'étiquette dans l'opération `node` avec l'option `rotate=` suivie d'un angle en degrés :



La rotation du texte se fait à partir de la position normale du texte, c'est-à-dire par rapport à l'horizontale.

La rotation peut aussi être appliquée sur des nœuds placés à une position intermédiaire d'un chemin entre deux points consécutifs :



Si l'option `sloped` est présente sur l'opération `node`, la rotation du texte se fait à partir de la tangente au chemin à la position de l'étiquette.

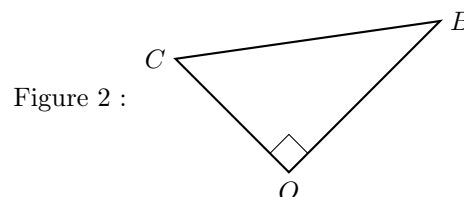
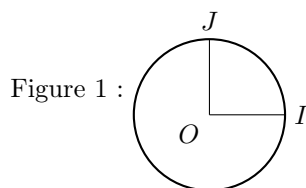
### 6.3.4 Modification de la taille des annotations : `scale`

On peut enfin modifier la taille des étiquettes de l'opération `node` avec l'option `scale=` suivie d'un coefficient multiplicatif :

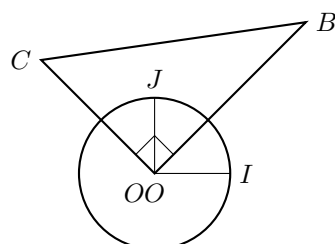


### 6.3.5 Insérer une sous-figure : `scope`, `shift`, `rotate`, `scale`

Supposons qu'on ait deux figures, chacune définie dans un environnement `tikzpicture` :



Dans chaque figure, le point  $O$  a pour coordonnées  $(0,0)$ . Si on copie le code des deux figures à l'intérieur d'un même environnement `tikzpicture`, elles ne seront plus disjointes :



Le problème est alors le suivant : comment créer dans un même environnement `tikzpicture` un dessin comportant ces deux figures, mais disjointes ? Et, de plus, on aimerait pouvoir recopier directement le code de chaque figure séparée, et ne pas avoir à recalculer de nouvelles coordonnées.

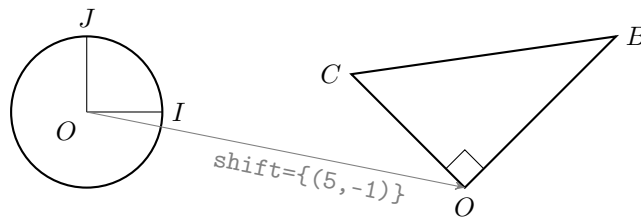
Heureusement, il existe un environnement `scope` qui peut être vu comme un sous-environnement `tikzpicture` auquel on peut affecter des options de décalage comme l'option `xshift` (suivie d'un décalage horizontal), l'option `yshift` (suivi d'un décalage vertical) ou l'option `shift` (suivi des coordonnées d'un vecteur de translation).

Par exemple : `[xshift=2]` `[yshift=25mm]` ou `[shift={(5,2)}`

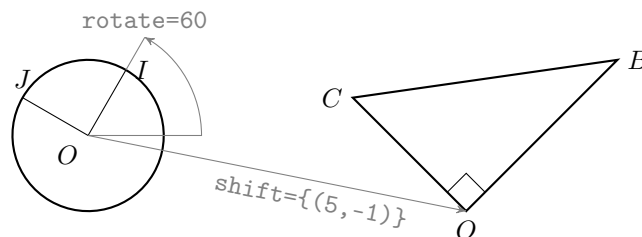
On remarque, dans ce dernier cas, les accolades autour des coordonnées du vecteur de translation. Ces accolades sont nécessaires ici, pour que la virgule ne soit pas considérée comme un séparateur d'options.

Prenons le code de chacune de nos deux figures et insérons les codes respectifs dans un environnement `scope`, que nous plaçons ensuite dans un environnement `tikzpicture`. Pour décaler la seconde figure, on ajoute à son environnement `scope` l'option `[shift={(5,-1)}` :

```
\begin{tikzpicture}
  \begin{scope}
    % ... Figure 1
  \end{scope}
  \begin{scope}[shift={(5,-1)}]
    % ... Figure 2
  \end{scope}
\end{tikzpicture}
```



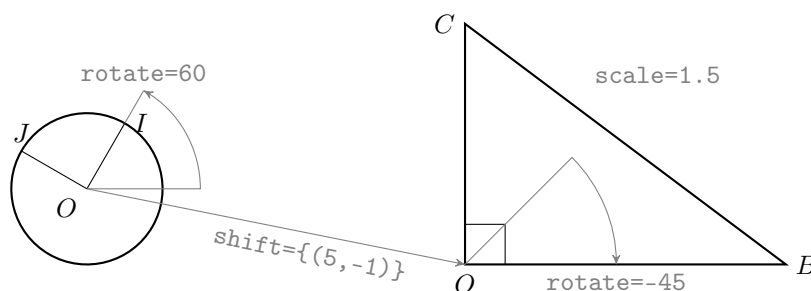
Ajoutons l'option `[rotate=60]` au `scope` qui contient le code de la première figure, une rotation de  $60^\circ$  est donc appliquée à cette figure :



Remarque : la rotation ne concerne que le dessin du chemin. On constate que les textes ont conservé leur alignement original.

C'est un peu surprenant, mais raisonnable : si vous avez réalisé une figure complexe avec de nombreuses annotations textuelles, vous pouvez ainsi la faire tourner facilement, les annotations restent lisibles car elles conservent la même orientation. Il est cependant parfois utile de modifier l'ancrage de certains nœuds.

L'option `[shift={(5,-1)},rotate=-45,scale=1.5]` a maintenant été affectée au `scope` contenant le code de la seconde figure, celle-ci a donc subi une translation, une rotation et sa taille a été augmentée de 50%.



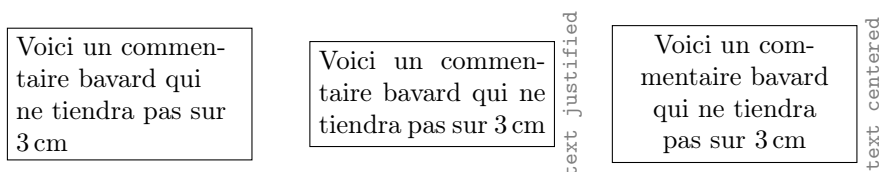
Remarque : Attention, l'effet des options `rotate` ou `scale`, quand elles sont appliquées à un des environnements `scope` ou `tikzpicture`, n'est pas le même que si elles sont appliquées à l'opération `node` :

- `rotate` ou `scale` pour les environnements `scope` ou `tikzpicture` :  
ce sont les lignes du dessin qui constitue la figure qui subissent les transformations. La figure tourne, sa taille est modifiée, mais les annotations textuelles effectuées à l'aide de nœuds gardent leurs tailles et leurs orientations :
- `rotate` ou `scale` pour l'opération `node` :  
ce sont les textes contenus dans le nœud qui subissent les transformations. Une fois ces transformations effectuées, il n'y a plus de changement des textes, ni en taille, ni en orientation lors d'une modifications des options de `scope` ou `tikzpicture`. Cependant, il y a déplacement des textes qui restent liés à leurs points d'ancrage qui ont bougé.

### 6.3.6 Textes longs : `text width`, `justified`, `centered`

Le contenu d'un nœud est en général assez court et placé sur une seule ligne. Toutefois, il est possible de mettre dans un nœud des textes plus longs, sur plusieurs lignes, comme par exemple dans la légende d'une figure.

L'option `text width` permet de limiter la largeur du contenu textuel du nœud et de provoquer ainsi son affichage sur plusieurs lignes :



```
\node[draw,text width=3cm] at(0,0){Voici...cm};
\node[draw,text width=3cm,text justified] at(4,0){Voici...cm};
\node[draw,text width=3cm,text centered] at(8,0){Voici...cm};
```

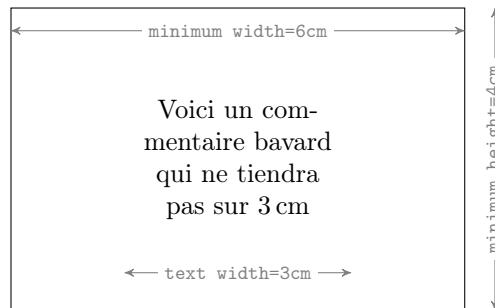
Les options `text justified` ou `text centered` permettent de préciser le type d'alignement du texte.

En général, la largeur d'un nœud est assez limitée. Dans ce cas, les césures provoquées par  $\text{\LaTeX}$ , lors de la mise en page, seront parfois gênantes.

Les options `text badly ragged` ou `text badly centered` permettent d'interdire les césures dans le texte du nœud :



Il est aussi possible d'utiliser simultanément les options `minimum width`, `minimum height` et `text width` dans un même nœud. Ainsi on contrôle la taille du rectangle du nœud :

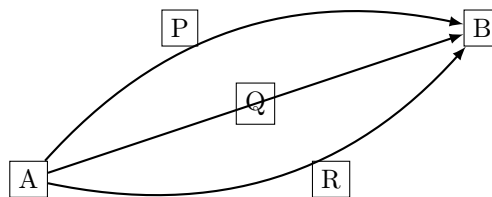


```
\node[draw,text width=3cm,text centered,
      minimum width=6cm,
      minimum height=4cm] at (0,0) {Voici...cm}
```

### 6.3.7 Contournement d'un nœud

Si on construit un graphe complexe, il peut parfois arriver que certains nœuds gênent le passage d'un arc, comme ci-dessous, où les options les plus simples ne permettent pas d'aller de A à B sans rencontrer les autres nœuds.

On veut relier A à B par une flèche qui contourne les obstacles :



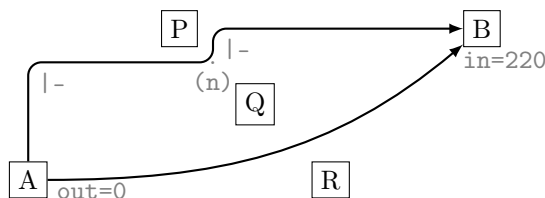
On va d'abord définir un nœud auxiliaire (n), invisible, légèrement à droite et en dessous (2 mm) du coin sud-est du nœud (P) :

```
\coordinate[shift={(2mm,-2mm)}] (n) at (P.south east);
```

Pour cela, on a utilisé l'option `[shift={(2mm,-2mm)}]`, appliquée à l'ancre `(P.south east)` du nœud (P).

On trace ensuite :

```
\draw[->,thick,>=latex,rounded corners=5pt] (A) |- (n) |- (B);
```



Une petite marque discrète « · » a été ajoutée sur la figure pour montrer la position du point auxiliaire (n) « invisible » à côté du point (P).

On remarque qu'il était aussi possible d'éviter les obstacles avec une flèche courbe qui passe entre Q et R, à condition d'ajuster les angles de sortie et d'entrée dans les nœuds, à l'aide des options `out=` et `in=` de l'opération `to`.

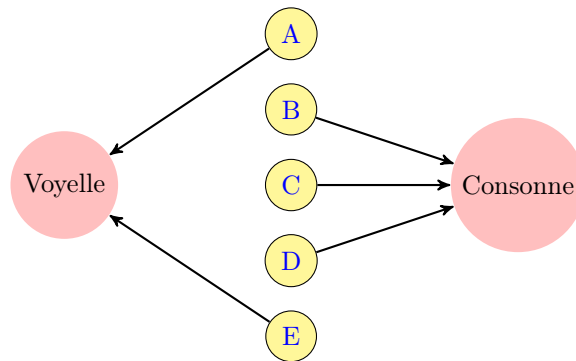


## 6.4 Exercices

Dans chaque exercice, on demande de réaliser la figure spécifiée. Toutes ces figures peuvent être construites uniquement en TikZ avec des nœuds et des arcs, en définissant évidemment des styles à l'aide des différentes options.

La principale difficulté, dans ces exercices, est de réaliser la figure la plus proche possible du modèle en respectant les positions et les angles.

### 6.4.1 Voyelle ou Consonne



#### Aide

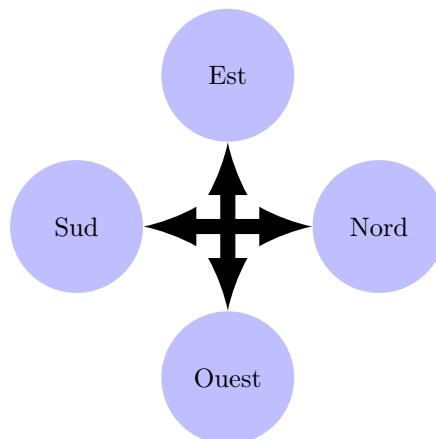
On place les lettres à 1 cm les unes des autres et « Voyelle » et « Consonne » à 3 cm de chaque côté. On définit trois styles :

```

\tikzstyle{lettre}=[circle,draw,fill=yellow!50,text=blue]
\tikzstyle{type}=[circle,fill=red!25]
\tikzstyle{fleche}=[->,>= stealth',thick]
  
```

Il n'y a plus qu'à lier les différents nœuds.

### 6.4.2 Les points cardinaux



#### Aide

On définit le style des points cardinaux :

```

\tikzstyle{point}=[circle,fill=blue!25,minimum width=5em]
  
```

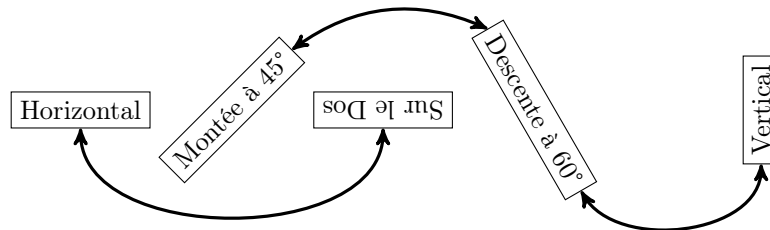
Le mot le plus long est « Ouest », on fixe donc `minimum width=5em`, une largeur d'au moins 5 fois la largeur de la lettre « m », pour que tous les cercles soient de même taille. On définit le style des flèches :

```

\tikzstyle{fleche}=[<->,>=latex,line width=2mm]
On trace :
\node[point] (N) at (2,0){Nord};
\node[point] (S) at (-2,0){Sud};
\draw[fleche] (N)--(S);
...

```

### 6.4.3 Orientations



#### Aide

On définit les nœuds H : (0,0) , M : (2,0), S : (6,0), D : (6,0) et V : (9,0) avec l'option `draw,rotate=` et les angles 45, 180, 60, 90, et le style pour les flèches :

```

\tikzstyle{fleche}=[<->,>=stealth',very thick]

```

Les liens sont assez délicats à établir en raison des rotations des nœuds. La figure a été obtenue avec les options suivantes :

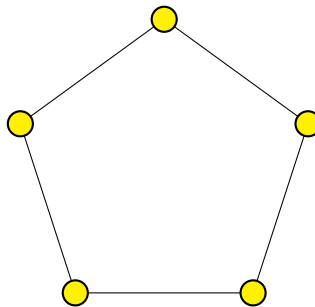
```

\draw[fleche] (H.south) to[bend right=90] (S.north);
\draw[fleche] (M.east) to[bend left] (D.west);
\draw[fleche] (D.east) to[out=-60,in=-90] (V.west);

```

Par exemple, sur la première ligne on relie (H.south) à (S.north) qui est au dessous du nœud (S), car le nœud (S) est sur le dos [`rotate=180`], et son nord est en bas!

### 6.4.4 Pentagone



#### Solution

Pentagone de rayon 2 centré sur l'origine :

```

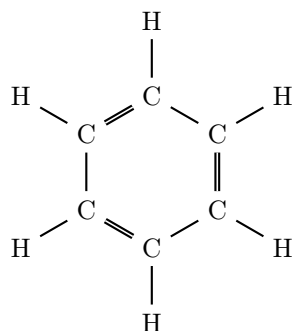
\tikzstyle{sommet}=[circle,draw,thick,fill=yellow]
\draw
(90:2) node[sommet]{}
-- (162:2) node[sommet]{}
-- (234:2) node[sommet]{}
-- (306:2) node[sommet]{}
-- (378:2) node[sommet]{}
-- cycle;

```

Le premier sommet tracé est celui du haut.

### 6.4.5 Benzène

Le benzène est un hydrocarbure aromatique monocyclique, de formule  $C_6H_6$



#### Solution

Les C et les H sont placés et nommés à l'aide d'une commande `\foreach`

L'origine est au centre de la molécule, et les atomes sont placés en coordonnées polaires, l'angle étant défini par la variable `\a`.

Les C sont nommés a, b, c, d et e (noms pris successivement par la variable `\n`) et les H associés sont nommés respectivement aa, bb, cc, dd et ee (noms engendrés successivement par `\n\n`).

```
% les C et les H
\foreach \n/\a in {a/30,b/90,c/150,d/210,e/270,f/330}
  {\node (\n) at (\a:1) {C};
   \node (\n\n) at (\a:2) {H};}

% les liaisons C - H
\foreach \n in {a,b,c,d,e,f} \draw [thick] (\n)--(\n\n);
```

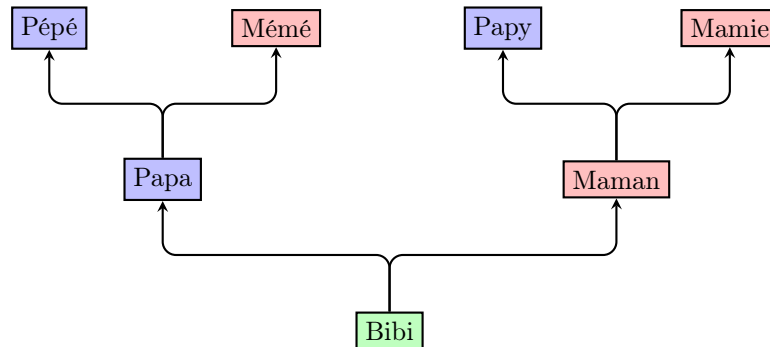
On remarque la possibilité, avec la commande `\foreach`, de construire des noms de nœud par concaténation de variables et de les réutiliser plus tard.

Il suffit ensuite de dessiner les liaisons, simples ou doubles.

```
% les liaisons simples entre C
\draw [thick] (a)--(b);s
\draw [thick] (c)--(d);
\draw [thick] (e)--(f);

% les liaisons doubles entre C
\draw [double,thick] (b)--(c);
\draw [double,thick] (d)--(e);
\draw [double,thick] (f)--(a);
```

### 6.4.6 Arbre généalogique



#### Aide

On définit le style des liens :

```
\tikzstyle{lien}=[->,>=stealth,rounded corners=5pt,thick]
```

On peut ensuite définir, avec la commande `\tikzstyle`, trois styles distincts pour Bibi les hommes et les femmes, mais si l'on dispose de la version 2.00 de TikZ & PGF, on pourra utiliser la commande `\tikzset` pour créer un style paramétrable :

```
\tikzset{individu/.style={draw,thick,fill=#1!25},
individu/.default={green}}
```

Ici, l'option `individu` donne un fond `fill=green!25` par défaut, mais elle peut accepter un nom de couleur qui remplacera le `#1` dans la définition. Ainsi `individu=red` donne un fond `fill=red!25`.

On peut alors placer les nœuds, puis les liens :

```
\node[individu] (B) at (0,0) {Bibi};
\node[individu=blue] (P) at (-3,2) {Papa};
\node[individu=red] (M) at (3,2) {Maman};
...
\draw[lien] (B) |- (-1,1) -| (P);
\draw[lien] (B) |- (1,1) -| (M);
...
```

On remarque l'utilisation de points auxiliaires pour permettre un tracé des flèches des liens en zigzag.

## 6.5 Résumé

On construit des graphes en définissant et en nommant des nœuds, qui sont des boîtes de texte, avec la commande `\node`, puis on les relie par des arcs avec la commande `\draw`.

On peut modifier le style, la taille et l'orientation des textes écrits dans les nœuds avec les options `rotate` `scale`.

Les arcs sont tracés à l'aide de la commande `\draw`, et de ses différentes opérations comme `|-`, `-|` ou `to` qui permettent de modifier la forme des arcs. Avec l'opération `edge`, il est aussi possible de tracer des arcs à partir d'un nœud, sans avancer.

Les options comme `thick`, `dashed`, `->`, `*-o` ou `[-(` permettent de préciser les styles de trait ou de flèche.

On peut aussi ajouter des annotations sur les arcs avec l'opération `node` et une des options `midway`, `pos=` ou `sloped`.

Pour la construction de figures complexes, l'environnement `scope` permet de décomposer le diagramme en sous-figures créées d'abord indépendamment, puis assemblées ensuite avec des transformations éventuelles en utilisant des options comme `shift`, `rotate` ou `scale`.

# Chapitre 7

## Graphes : Exemples

On a présenté, dans le chapitre précédent, les outils de base pour dessiner des graphes. On va maintenant réaliser des graphes complexes pour montrer comment mettre en œuvre ces différents outils.


### 7.1 Graphe d'une relation

#### 7.1.1 Relations entre quadrilatères

On connaît différentes relations entre quadrilatères, comme : un carré est un rectangle, un carré est un losange.

On veut tracer un graphe exprimant ces différentes relations entre les types de quadrilatère suivant : parallélogramme, rectangle, carré, losange.

Chaque type de quadrilatère sera représenté par un nœud et chaque relation sera représentée par une flèche.

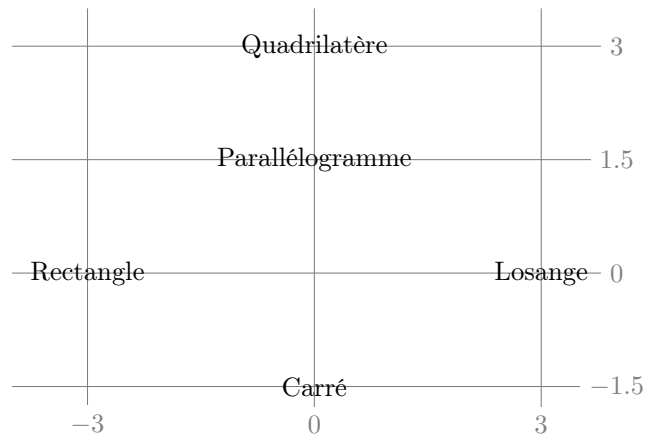
Par exemple « Carré est un Rectangle » :  signifie que l'ensemble des carrés est strictement inclus dans l'ensemble des rectangles.

#### 7.1.2 Des nœuds et des flèches : node et ->

La figure étant assez complexe, on va la construire pas à pas en indiquant, à chaque étape, uniquement ce qui diffère de l'étape précédente.

On place d'abord un nœud (`node`) pour chaque type de quadrilatère et on le nomme à l'aide de son initiale :

```
\begin{tikzpicture}
  \node (Q) at (0,3) {Quadrilatère};
  \node (P) at (0,1.5) {Parallélogramme};
  \node (R) at (-3,0) {Rectangle};
  \node (L) at (3,0) {Losange};
  \node (C) at (0,-1.5) {Carré};
\end{tikzpicture}
```

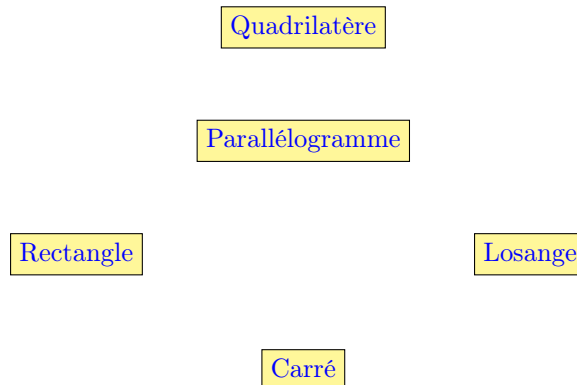


Le placement des nœuds a été fait au crayon sur un papier, puis simulé en centrant chaque nœud sur des points de coordonnées raisonnables.

Pour rendre la figure plus lisible, on va définir un style de nœud en écrivant chaque nom en bleu sur fond jaune clair, dans un rectangle encadré de noir.

On définit un style `quadri` à ajouter comme option sur chaque nœud :

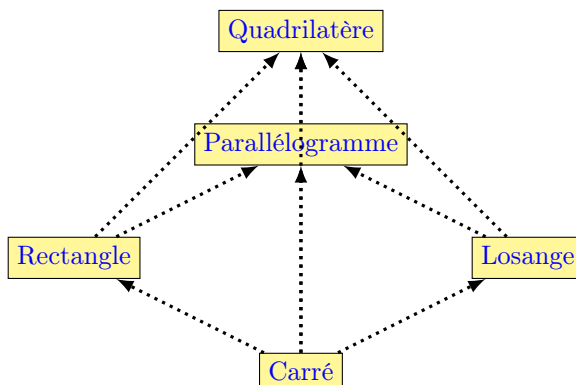
```
\tikzstyle{quadri}=[rectangle,draw,fill=yellow!50,text=blue]
\node[quadri] (Q) at (0,3) {Quadrilatère}; ... etc.
```



On ajoute maintenant les relations « est un » reliant les nœuds par des flèches. On définit `estun`, le style de ces flèches, pour les tracer en pointillés très épais, avec une pointe de type `latex`.

On ajoute ainsi le code suivant à notre figure :

```
\tikzstyle{estun}=[->,dotted,very thick,>=latex]
\draw[estun] (P)--(Q);
\draw[estun] (R)--(Q); \draw[estun] (R)--(P);
\draw[estun] (L)--(Q); \draw[estun] (L)--(P);
\draw[estun] (C)--(Q); \draw[estun] (C)--(P);
\draw[estun] (C)--(L); \draw[estun] (C)--(R);
```

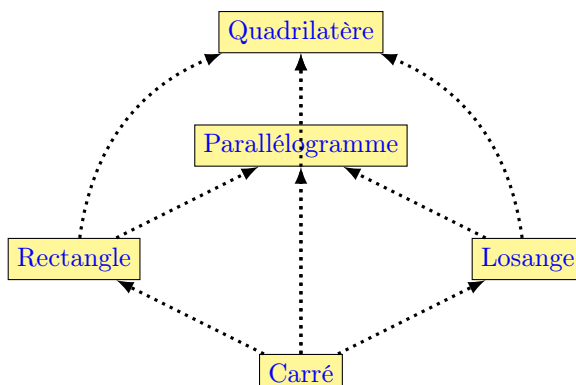


Le résultat manque de lisibilité, car certaines flèches traversent les nœuds et d'autres se superposent.

Évitons d'abord les superpositions en remplaçant les droites par des courbes. Pour cela le lien - est remplacé par un lien `to` avec une option `[bend left]` (courbé à gauche) ou `[bend right]` (à droite).

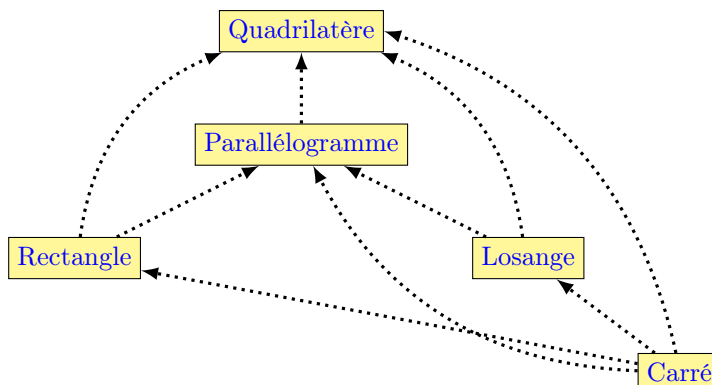
Modifions d'abord « Rectangle-Quadrilatère » et « Losange-Quadrilatère » :

```
\draw[estun] (R) to[bend left] (Q);
\draw[estun] (L) to[bend right] (Q);
```



Le nœud « Carré » est mal placé. Pour séparer les flèches superposées et rendre les relations entre quadrilatères visibles, on déplace le nœud « Carré » à droite du nœud « Losange » et on courbe ensuite la flèches « Carré-Quadrilatère » et la flèche « Carré-Parallélogramme » :

```
\node[quadri] (L) at (4,1.5) {Losange};
\node[quadri] (C) at (7,0) {Carré};
\draw[estun] (C) to[bend right] (Q);
\draw[estun] (C) to[bend left] (P);
```



Il est encore possible d'améliorer la figure.

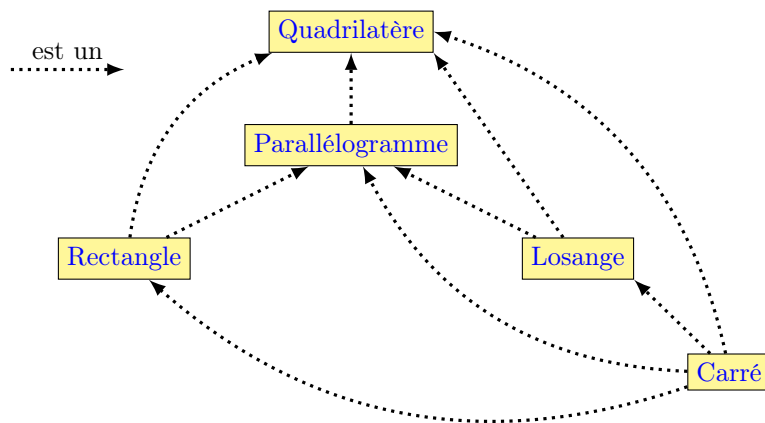
### 7.1.3 Graphe final : courbure bend, ancrage P.east

On va redresser la flèche « Losange-Quadrilatère » et la faire arriver au coin sud-est du nœud « Quadrilatère » : `\draw (L)--(Q.south east);`

On procède ensuite à d'autres petites modifications de position des extrémités des flèches « Carré-Losange » et « Carré-Parallélogramme ».

Pour terminer, on place sur le côté gauche de la figure, une légende qui précise la sémantique de la flèche :

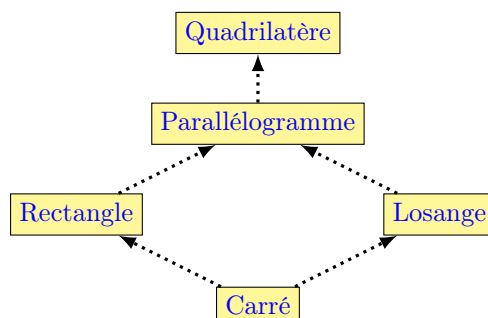
```
\draw[estun] (-4.5,2.5) -- (-3,2.5) node[midway,above]{est un};
```



```
\begin{tikzpicture}
% définition des styles
\tikzstyle{quadri}=[rectangle,draw,fill=yellow!50,text=blue]
\tikzstyle{estun}=[->,>=latex,very thick,dotted]
% les nœuds
\node[quadri] (Q) at (0,3) {Quadrilatère};
\node[quadri] (P) at (0,1.5) {Parallélogramme};
\node[quadri] (R) at (-3,0) {Rectangle};
\node[quadri] (L) at (3,0) {Losange};
\node[quadri] (C) at (5,-1.5) {Carré};
% les flèches
\draw[estun] (P)--(Q);
\draw[estun] (R)to[bend left](Q); \draw[estun] (R)--(P);
\draw[estun] (L)--(Q.south east); \draw[estun] (L)--(P);
\draw[estun] (C)to[bend right](Q.east); \draw[estun] (C)to[bend left](P);
\draw[estun] (C)--(L.south east); \draw[estun] (C)to[bend left](R);
% la légende
\draw[estun] (-4.5,2.5)--(-3,2.5)node[midway,above]{est un};
\end{tikzpicture}
```



L'exemple précédent avait pour but de montrer comment contrôler finement la position des nœuds et des arcs. Il est d'usage, en général, de ne pas représenter les flèches que l'on peut déduire par transitivité comme ci-dessous :



## 7.2 Organigramme informatique

### 7.2.1 Somme des $N$ premiers nombres entiers

Dessiner un organigramme du programme qui calcule la somme des  $N$  premiers nombres entiers avec l'algorithme suivant :

```

DÉBUT
  Lire un entier positif  $N$ 
   $S \leftarrow 0$ 

  TANT QUE  $N > 0$  FAIRE
    |  $S \leftarrow S + N$ 
    |  $N \leftarrow N - 1$ 
  Afficher la somme  $S$ 
FIN
  
```

### 7.2.2 Style des nœuds : draw, ellipse, fill, text

Comme dans l'exemple précédent, on va représenter un graphe composé de nœuds reliés par des flèches. Mais, dans ce cas, il y a plusieurs types de nœuds que l'on désire différencier graphiquement.

On va définir des styles pour chaque type de nœud du diagramme :

- `debutfin` pour les repères de Début et de Fin : `ellipse`;
- `es` pour les Entrées-Sorties : `rounded corners`;
- `instruct` pour les instructions : `rectangle`;
- `test` pour le test : `diamond`.

```

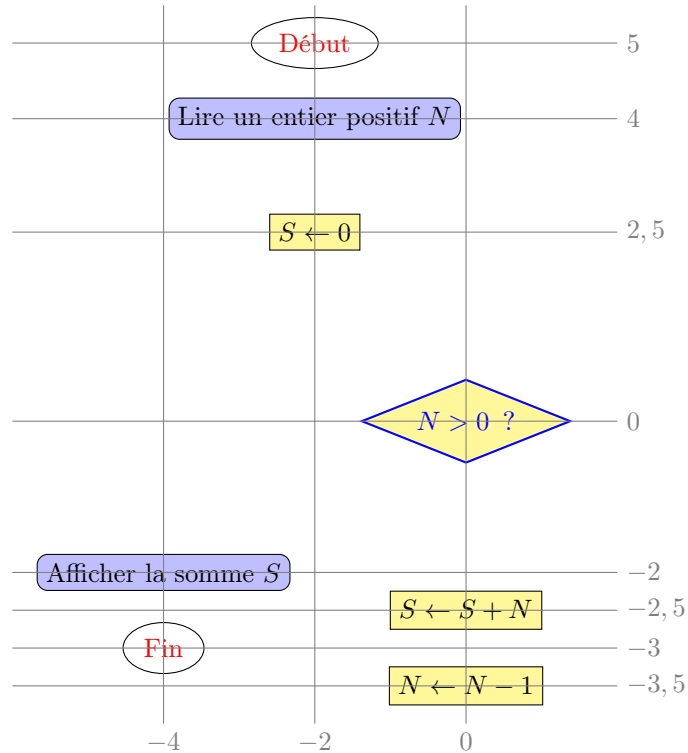
\begin{tikzpicture}
  \tikzstyle{debutfin}=[ellipse,draw,text=red]
  \tikzstyle{instruct}=[rectangle,draw,fill=yellow!50]
  \tikzstyle{test}=[diamond, aspect=2.5,thick,
    draw=blue,fill=yellow!50,text=blue]
  \tikzstyle{es}=[rectangle,draw,rounded corners=4pt,fill=blue!25]
\end{tikzpicture}
  
```

On va ensuite placer les éléments en leur attribuant un nom significatif :

```

\node[debutfin] (debut) at (-2,5) {Début};
\node[es] (lire) at (-2,4) {Lire un entier positif  $N$ };
\node[test] (test) at (0,0) { $N > 0$  \ ?};
\node[instruct] (init) at (-2,2.5) { $S \leftarrow 0$ };
\node[instruct] (plus) at (0,-2.5) { $S \leftarrow S + N$ };
\node[instruct] (moins) at (0,-3.5) { $N \leftarrow N - 1$ };
  
```

```
\node[es] (afficher) at (-4,-2) {Afficher la somme $$};
\node[debutfin] (fin) at (-4,-3) {Fin};
```



### 7.2.3 Forme des flèches : $\Rightarrow$ , rounded corners, $\lrcorner$

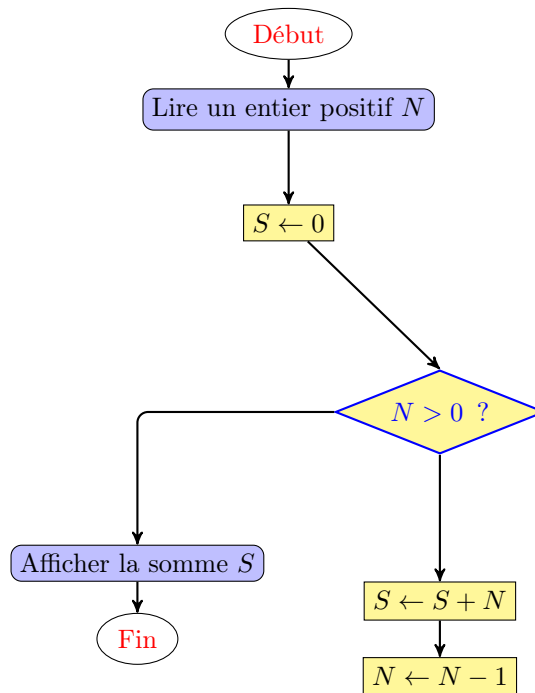
Il suffit de représenter la séquence ordonnée des instructions par des flèches dont le style `suite` est d'abord défini :

```
\tikzstyle{suite}=[->,>=stealth',thick,rounded corners=4pt]
```

```
\draw[suite] (debut) -- (lire);
\draw[suite] (lire) -- (init);
\draw[suite] (init) -- (test.north);
\draw[suite] (test) -- (plus);
\draw[suite] (plus) -- (moins);
\draw[suite] (test) -| (afficher);
\draw[suite] (afficher) -- (fin);
```

On remarquera en particulier : `\draw[suite] (init)--(test.north)`; le branchement au sommet du losange de test.

Et aussi : `\draw[suite] (test)-|(afficher)`; le branchement avec virage à angle droit en sortie du test.

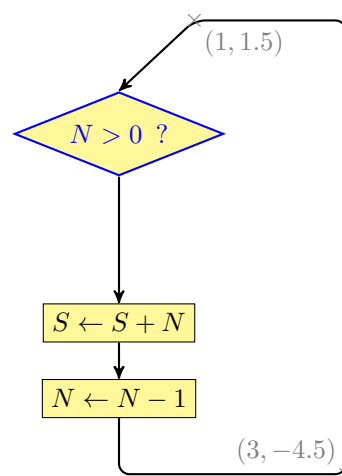


Il manque la flèche qui boucle entre l'instruction  $N \leftarrow N - 1$  et le sommet du losange de test.

Comme elle doit contourner la figure par la droite, on va introduire deux points auxiliaires rendus visibles ci-contre par  $\times$ .

On ajoute donc la commande :

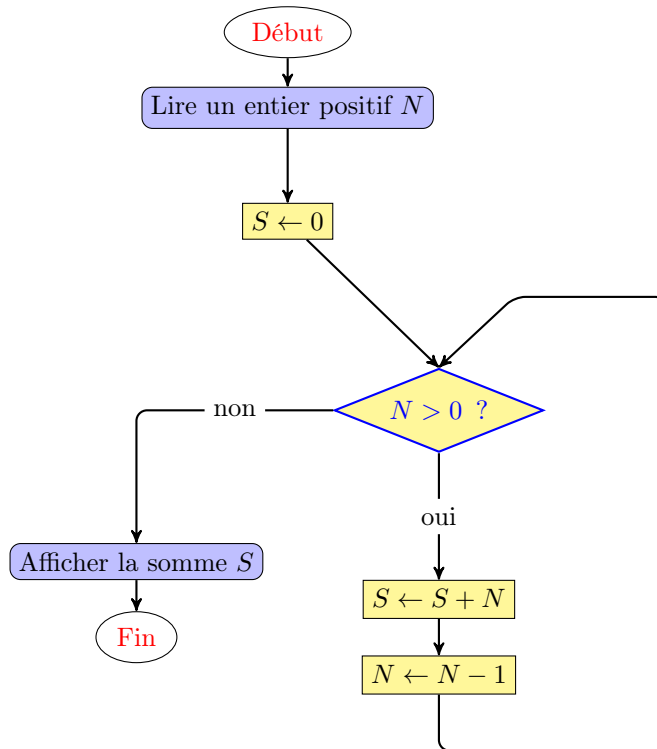
```
\draw[suite] (moins) |- (3,-4.5)
  |- (1,1.5) -- (test.north);
```



Il ne reste plus qu'à étiqueter les sorties du losange de test :

```
\draw[suite] (test) -- (plus)
  node[midway,fill=white] {oui};
\draw[suite] (test) -| (afficher)
  node[near start,fill=white] {non};
```

## 7.2.4 Organigramme final



Code complet de la figure :

```

\begin{tikzpicture}
% style des nœuds
\tikzstyle{debutfin}=[ellipse,draw,text=red]
\tikzstyle{instruct}=[rectangle,draw,fill=yellow!50]
\tikzstyle{test}=[diamond,aspect=2.5,thick,
draw=blue,fill=yellow!50,text=blue]
\tikzstyle{es}=[rectangle,draw,rounded corners=4pt,fill=blue!25]
% style des flèches
\tikzstyle{suite}=[->,>=stealth',thick,rounded corners=4pt]
% placement des nœuds
\node[debutfin] (debut) at (-2,5) {Début};
\node[es] (lire) at (-2,4) {Lire un entier positif $N$};
\node[test] (test) at (0,0) {$N > 0$ \ ?};
\node[instruct] (init) at (-2,2.5) {$S \leftarrow 0$};
\node[instruct] (plus) at (0,-2.5) {$S \leftarrow S + N$};
\node[instruct] (moins) at (0,-3.5) {$N \leftarrow N - 1$};
\node[es] (afficher) at (-4,-2) {Afficher la somme $$$};
\node[debutfin] (fin) at (-4,-3) {Fin};
% Placement des flèches
\draw[suite] (debut) -- (lire);
\draw[suite] (lire) -- (init);
\draw[suite] (init) -- (test.north);
\draw[suite] (test) -- (plus) node[midway,fill=white]{oui};
\draw[suite] (plus) -- (moins);
\draw[suite] (moins)|-(3,-4.5)|-(1,1.5)--(test.north);
\draw[suite] (test)-|(afficher)node[near start,fill=white]{non};
\draw[suite] (afficher) -- (fin);
\end{tikzpicture}

```

## 7.3 Diagrammes syntaxiques

### 7.3.1 Grammaire des expressions mathématiques

Dessiner les diagrammes de définition de la grammaire (simplifiée) des expressions mathématiques :

- Une Expression est une somme de Termes séparés par + ;
- Un Terme est une produit de Facteurs séparés par  $\times$  ;
- Un Facteur est : soit une Variable, soit une Expression entre ( et ) ;
- Une Variable est : soit  $X$ , soit  $Y$ , soit  $Z$ .

Pour le spécialiste, voici cette grammaire, notée en BNF :

```
Expression ::= Terme {"+" Terme}*
Terme ::= Facteur {"\times" Facteur}*
Facteur ::= Variable | "(" , Expression , ")"
Variable ::= "X" | "Y" | "Z"
```

**Remarque :** pour simplifier on se limite à trois variables et on ne prévoit aucune valeur numérique. Voici quelques exemples :

$X$  ;  $X \times Y$  ;  $X + Y$  ;  $X + (Y \times Z)$  ;  $(X + Y) \times (Z + X)$

### 7.3.2 Alignement des nœuds, étiquetage

On connaît bien maintenant les méthodes de construction de graphes que nous allons appliquer à ce problème.

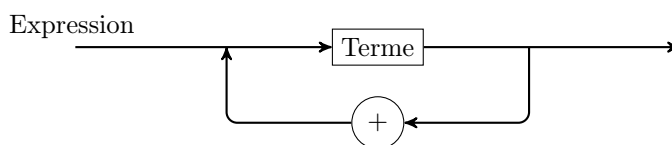
On définit différents styles :

- `element` pour les éléments syntaxiques (Terme Facteur ...);
- `terminal` pour les élément terminaux de la grammaire (+  $\times$  ...);
- `fleche` pour les arcs de liaison.

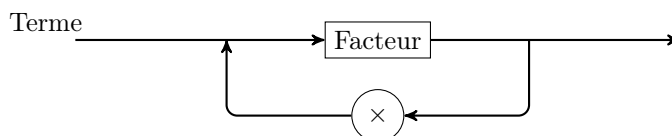
```
\tikzstyle{element}=[rectangle,draw,fill=white]
\tikzstyle{terminal}=[circle,draw]
\tikzstyle{fleche}=[->,>=stealth',thick,rounded corners=4pt]
```

On place ensuite les nœuds en les alignant correctement, puis les arcs orientés pour lier les différents nœuds.

```
\begin{tikzpicture}
  \node[above right] at (-1,0) {Expression};
  \node[element] (T) at (4,0) {Terme};
  \node[terminal] (p) at (4,-1) {$+$};
  \draw[fleche] (0,0) -- (T); \draw[fleche] (T) -- (8,0);
  \draw[fleche] (6,0) |- (p); \draw[fleche] (p) -| (2,0);
\end{tikzpicture}
```



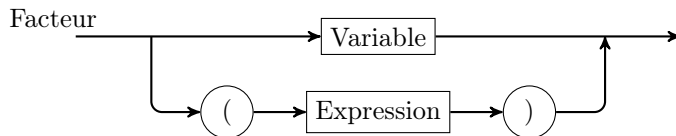
De même, on va construire les diagrammes syntaxiques successifs :



```

\node[above right] at (-1,0) {Terme};
\node[element] (F) at (4,0) {Facteur};
\node[terminal] (m) at (4,-1) {$\times$};
\draw[fleche] (0,0) -- (F); \draw[fleche] (F) -- (8,0);
\draw[fleche] (6,0) |- (m); \draw[fleche] (m) -| (2,0);

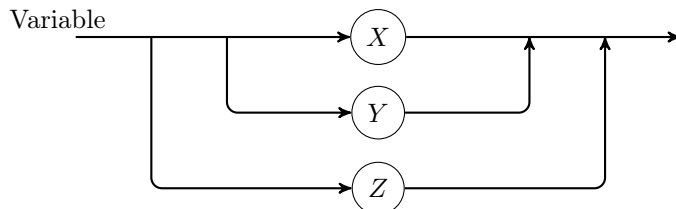
```



```

\node[above right] at (-1,0) {Facteur};
\node[element] (V) at (4,0) {Variable};
\node[terminal] (po) at (2,-1) {$($};
\node[element] (E) at (4,-1) {Expression};
\node[terminal] (pf) at (6,-1) {$)$};
\draw[fleche] (0,0) -- (V); \draw[fleche] (V) -- (8,0);
\draw[fleche] (1,0) |- (po); \draw[fleche] (po)--(E);
\draw[fleche] (E) -- (pf); \draw[fleche] (pf) -| (7,0);

```



```

\node[above right] at (-1,0) {Variable};
\node[terminal] (x) at (4,0) {$X$};
\node[terminal] (y) at (4,-1) {$Y$};
\node[terminal] (z) at (4,-2) {$Z$};
\draw[fleche] (0,0) -- (x); \draw[fleche] (x) -- (8,0);
\draw[fleche] (2,0) |- (y); \draw[fleche] (y) -| (6,0);
\draw[fleche] (1,0) |- (z); \draw[fleche] (z) -| (7,0);

```

### 7.3.3 Regroupement de figures : scope et yshift

Les quatre figures ont été construites dans des environnements `tikzpicture` séparés. On aimerait maintenant les regrouper en une seule figure.

On pourrait recopier simplement leur code dans un seul environnement `tikzpicture` :

```

\begin{tikzpicture}
  \node[above right] at (-1,0) {Expression}; ...
  \node[above right] at (-1,0) {Expression}; ...
  \node[above right] at (-1,0) {Terme}; ...
  \node[above right] at (-1,0) {Variable}; ...
\end{tikzpicture}

```

Malheureusement, dans ce cas, toutes les figures seraient superposées puisque les coordonnées ont été positionnées à l'origine.

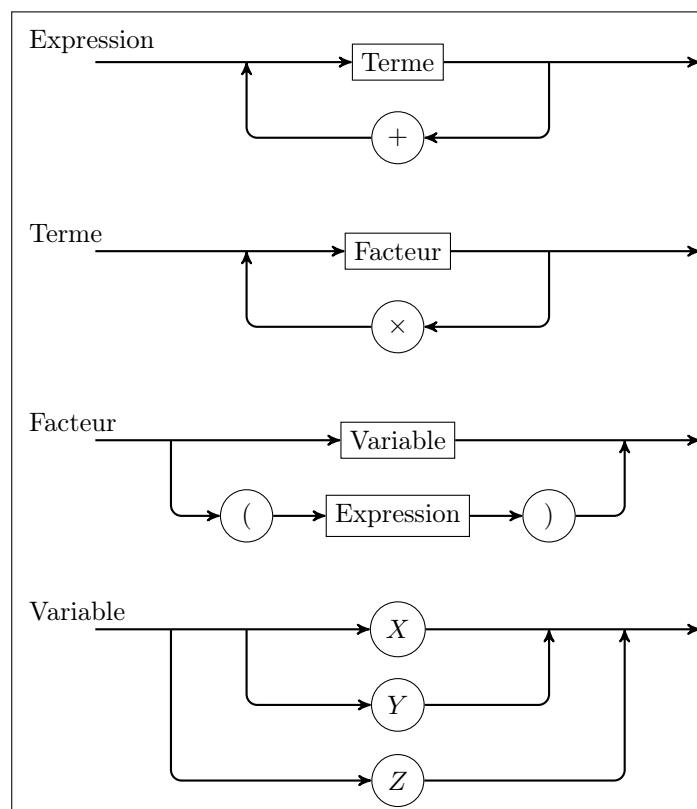
On pourrait alors penser à modifier les coordonnées de tous les nœuds. C'est possible, mais ce serait long.

On va utiliser une instruction qui permet de décaler chaque figure vers le bas de  $-2.5$  cm par rapport à la précédente. En fait, il est plus précis de dire : décaler la deuxième figure de  $-2.5$  cm, la troisième de  $5$  cm, etc.

Le décalage vertical s'obtient avec l'option `yshift`, et pour préciser à quelle partie du code s'applique cette option, on insère le code concerné dans un environnement `scope`. Lorsque `TikZ` applique l'option `[yshift=-2.5cm]` au point  $(-1,0)$ , il trace en fait le point  $(-1,-2.5)$ , etc.

On applique ensuite à chaque `scope` les décalages successifs : `[yshift=-2.5cm]`, `[yshift=-5cm]`, `[yshift=-7.5cm]`.

```
\begin{tikzpicture}
  \node[above right] at (-1,0) {Expression}; ...
  \begin{scope}[yshift=-2.5cm]
    \node[above right] at (-1,0) {Terme}; ...
  \end{scope}
  \begin{scope}[yshift=-5cm]
    \node[above right] at (-1,0) {Terme}; ...
  \end{scope}
  \begin{scope}[yshift=-7.5cm]
    \node[above right] at (-1,0) {Variable}; ...
  \end{scope}
\end{tikzpicture}
```



## 7.4 Graphe de preuve

### 7.4.1 Résolution d'une équation : $2x + 3 = 7$

On veut expliquer, à l'aide d'un schéma, la résolution de l'équation du premier degré :  $2x + 3 = 7$

**Résolution de l'équation**

On soustrait 3 à chaque membre

On simplifie

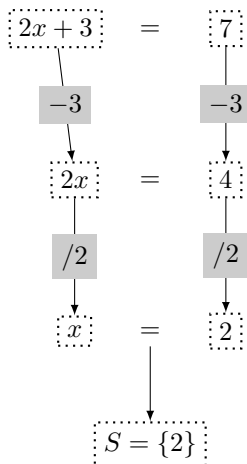
On divise chaque membre par 2

On simplifie

La solution est

$$\begin{aligned}
 2x + 3 &= 7 \\
 2x + 3 - 3 &= 7 - 3 \\
 2x &= 4 \\
 \frac{2x}{2} &= \frac{4}{2} \\
 x &= 2 \\
 S &= \{2\}
 \end{aligned}$$

On veut présenter le schéma de résolution sous cette forme :

**7.4.2 Placement des nœuds : `\node (a) at (x,y), below`**

La figure finale est proposée ci-dessus pour montrer l'objectif à atteindre. En réalité un diagramme de ce type a été fait à la main au brouillon au préalable. On code ensuite successivement les différents éléments en procédant par essais et erreurs pour obtenir le résultat voulu.

On va représenter notre équation à l'aide de 3 nœuds en nommant les membres de gauche (g) et de droite (d). Ainsi les nœuds (g) et (d) pourront être origines des flèches :

```

\begin{tikzpicture}
  \node (g) at (-1.25,4.5) {$2x+3$};
  \node at (0,4.5) {$=$};
  \node (d) at (1,4.5) {$7$};
\end{tikzpicture}

```

$$2x + 3 = 7$$

Un maniaque de la précision remarquera un léger défaut d'alignement du signe égal par rapport à sa position dans l'équation standard :  $2x + 3 = 7$

Pour corriger ça, on peut penser à l'option `[below]` que l'on ajoute au nœud égal :

```

\node[below] at (0,5) {$=$};

```

$$2x + 3 = 7$$

Hélas le résultat est pire ! Il faut ajuster la valeur de la descente en procédant par essais successifs jusqu'à obtenir : `[below=-5pt]`

$$2x + 3 = 7$$

On va aussi encadrer les deux membres de l'équation en définissant le style `membre` qu'on ajoutera ensuite comme option des nœuds (g) et (d).

```

\tikzstyle{membre}= [rectangle,draw,thick,dotted]
\node[membre] (g) at (-1.25,4.5) {$2x+3$};
\node[membre] (d) at (1,4.5) {$7$};

```



$$\boxed{2x + 3} = \boxed{7}$$

On peut maintenant compléter le diagramme :

$$\boxed{2x + 3} = \boxed{7}$$

$$\boxed{2x} = \boxed{4}$$

$$\boxed{x} = \boxed{2}$$

$$\boxed{S = \{2\}}$$

Plusieurs ajustements des coordonnées des différents nœuds ont été nécessaires pour obtenir une disposition satisfaisante et des espacements corrects.

```
\node[membre] (gg) at (-1,2.5) {$2x$};
\node [below=-5pt] at (0,2.5) {$=$};
\node[membre] (dd) at (1,2.5) {$4$};
\node[membre] (ggg) at (-1,0.5) {$x$};
\node [below=-5pt] (egal) at (0,0.5) {$=$};
\node[membre] (ddd) at (1,0.5) {$2$};
\node[membre] (reponse) at (0,-1) {$S=\{2\}$};
```

On remarque que l'on a nommé (egal) le nœud contenant le signe égal de l'avant dernière équation et (reponse) le nœud contenant la dernière équation. En effet, on a l'intention de les relier par une flèche.

### 7.4.3 Placement et étiquetage des flèches : `->`, `midway`

On va donc maintenant placer des flèches qui indiquent, à l'aide d'étiquettes, les opérations à effectuer sur chaque membre de l'équation. Par exemple :

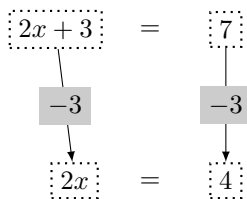
```
\draw[->] (g)--(gg) node[midway]{$-3$};
\draw[->] (d)--(dd) node[midway]{$-3$};
```

$$\begin{array}{ccc} \boxed{2x + 3} & = & \boxed{7} \\ \downarrow -3 & & \downarrow -3 \\ \boxed{2x} & = & \boxed{4} \end{array}$$

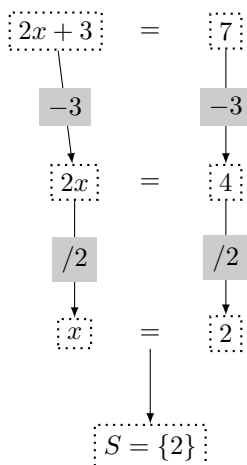
On a utilisé l'option `[->]` de la commande `\draw` pour obtenir une flèche, et l'option `[midway]` de `node` pour placer l'étiquette à mi-chemin de l'origine et de l'extrémité.

La définition pour les flèches et les étiquettes des styles respectifs `operation` et `etiquette` permettent d'améliorer la pointe des flèches et de dessiner des étiquettes sur fond gris clair :

```
\tikzstyle{operation}=[->,>=latex]
\tikzstyle{etiquette}=[midway,fill=black!20]
\draw[operation] (g)--(gg) node[etiquette]{$-3$};
\draw[operation] (d)--(dd) node[etiquette]{$-3$};
```



#### 7.4.4 Flèches courbes : bend, to



```

\begin{tikzpicture}
\tikzstyle{membre}= [rectangle,draw,thick,dotted]
\tikzstyle{operation}=[->,>=latex]
\tikzstyle{etiquette}=[midway,fill=black!20]
\node[membre] (g) at (-1.25,4.5)  {$2x+3$};
\node[below=-5pt] at (0,4.5)  {$=$};
\node[membre] (d) at (1,4.5)  {$7$};
\node[membre] (gg) at (-1,2.5)  {$2x$};
\node [below=-5pt] at (0,2.5)  {$=$};
\node[membre] (dd) at (1,2.5)  {$4$};
\node[membre] (ggg) at (-1,0.5)  {$x$};
\node [below=-5pt] (egal) at (0,0.5)  {$=$};
\node[membre] (ddd) at (1,0.5)  {$2$};
\node[membre] (reponse) at (0,-1)  {$S=\{2\}$};
%flèches
\draw[operation] (g)--(gg) node[etiquette]{$-3$};
\draw[operation] (d)--(dd) node[etiquette] {$-3$};
\draw[operation] (gg)--(ggg) node[etiquette]{$/2$};
\draw[operation] (dd)--(ddd) node[etiquette] {$/2$};
\draw[operation] (egal)--(reponse);
\end{tikzpicture}

```

*Mais on peut faire mieux pour que la figure remplace la solution rédigée.*

On va d'abord éviter d'intercaler les étiquettes «  $-3$  » ou «  $/2$  » entre les équations et les placer, de chaque côté des équations, comme des commentaires « on soustrait 3 » ou « on divise par 2 » :

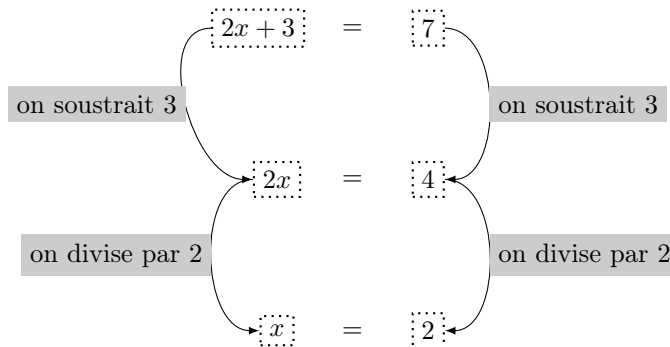
On change la flèche de droite en flèche courbe en remplaçant - par to et en précisant, en option, l'angle de sortie du nœud, pour l'origine de la flèche, et l'angle d'entrée dans le nœud suivant, pour l'extrémité.

On décale ensuite les étiquettes, en ajoutant les options left ou right aux nœuds correspondants.

```

\draw[operation] (g) to[out=180,in=180]
  node[etiquette,left]{on soustrait $3$} (gg);
\draw[operation] (d) to[out=0,in=0]
  node[etiquette,right]{on soustrait $3$} (dd);
\draw[operation] (gg) to[out=180,in=180]
  node[etiquette,left]{on divise par $2$}; (ggg)
\draw[operation] (dd) to[out=0,in=0]
  node[etiquette,right]{on divise par $2$} (ddd);

```



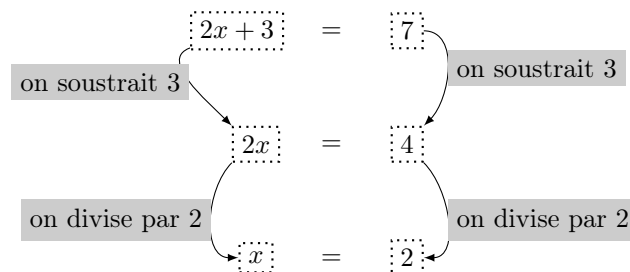
La position des flèches n'est pas encore tout à fait satisfaisante. On va légèrement modifier les angles d'entrée et de sortie et préciser les points d'entrée et de sortie des flèches sur les nœuds.

De plus, on peut remonter légèrement la ligne de la troisième équation.

```

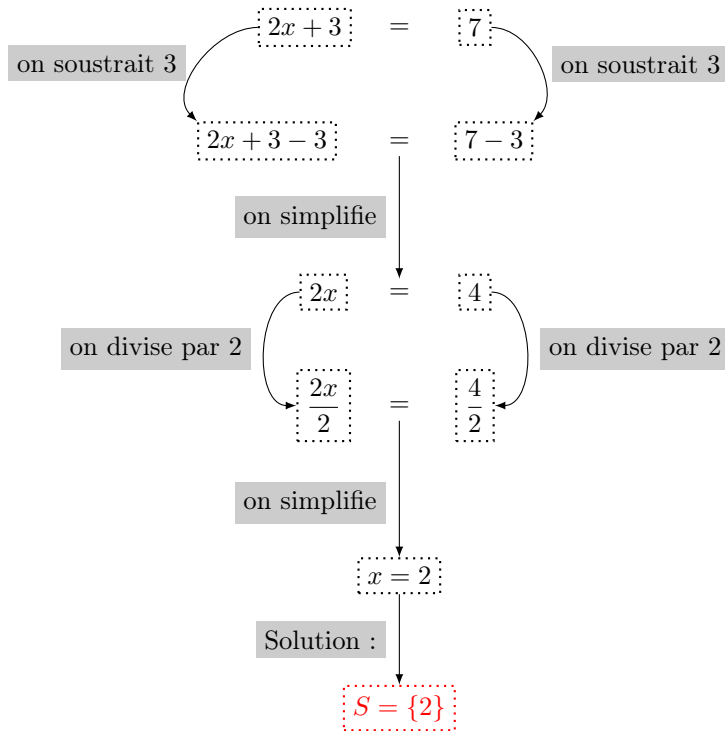
\draw[operation] (g) to[out=200,in=135]
  node[etiquette,left]{on soustrait $3$} (gg.north west);
\draw[operation] (d) to[out=0,in=45]
  node[etiquette,right]{on soustrait $3$} (dd.north east);
\draw[operation] (gg.south west) to[out=-135,in=180]
  node[etiquette,left]{on divise par $2$} (ggg);
\draw[operation] (dd.south east) to[out=-45,in=0]
  node[etiquette,right]{on divise par $2$}; (ddd)

```



#### 7.4.5 Exercice d'amélioration

À titre d'exercice, on laissera le lecteur poursuivre l'amélioration de la figure pour obtenir, par exemple, la présentation suivante :



## 7.5 Résumé

On a vu ici des exemples de graphes complexes et le moyen de les réaliser en utilisant les commandes `\node` et `\draw` et leurs différentes options.

On a montré la nécessité de structurer le dessin en plaçant d'abord les nœuds, puis les arcs à partir d'un modèle fait à la main sur papier au préalable.

Mais on a aussi pris conscience qu'un diagramme se prépare d'abord soigneusement avec un papier et un crayon. Il faut ensuite le modifier par améliorations successives, à l'aide de corrections du plus en plus fines.

## Chapitre 8

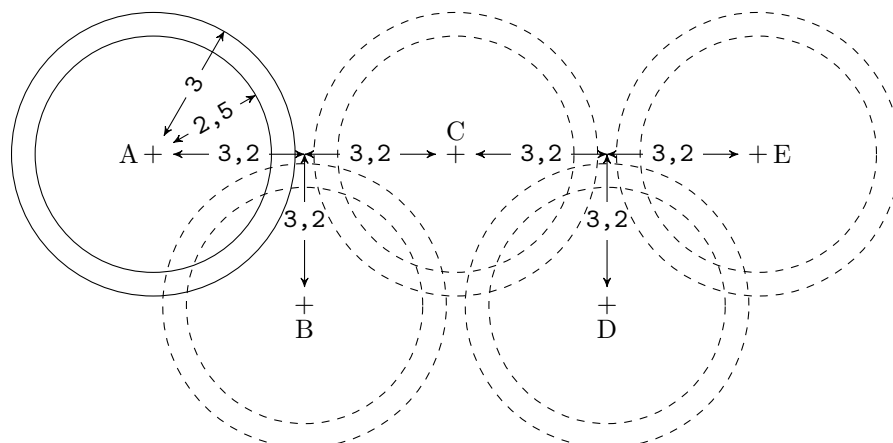
# Des figures aux illustrations

### 8.1 Les anneaux olympiques

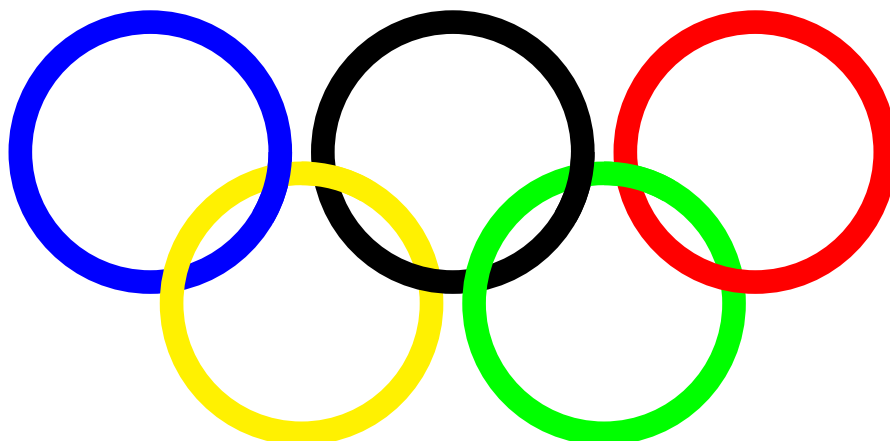
Dans cet exemple, la figure est décorative, mais sa construction exige une grande précision mathématique.

#### Description des anneaux olympiques

Les cinq anneaux olympiques sont centrés aux points A, B, C, D, et E. Leur rayon et leur position sont précisés sur le diagramme suivant :



De plus, ils sont en couleur et entrelacés ainsi :



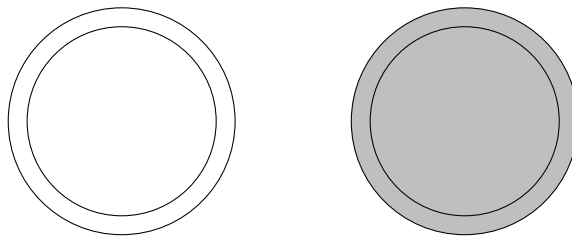
Centre	Pays	Couleur
A	Europe	Bleu
B	Asie	Jaune
C	Afrique	Noir
D	Océanie	Vert
E	Amérique	Rouge

### 8.1.1 Un anneau : `circle`, `fill`, `even odd rule`

On trace d'abord deux cercles concentriques :

```
\draw (0,0) circle(2.5) circle(3);
```

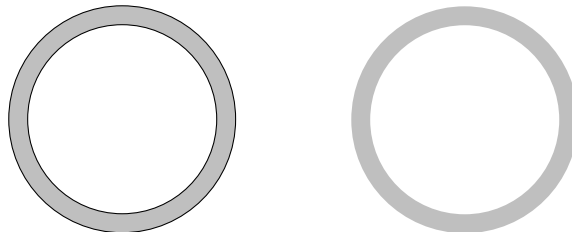
Puis on met une couleur à l'intérieur avec l'option : `fill=lightgray`



Hélas, ce n'est pas ce que l'on désire. On a déjà vu précédemment que la définition de l'intérieur d'une région n'est pas simple (voir le manuel : *interior rules*). Dans le cas présent, l'option à appliquer est : `even odd rule`

On peut éviter de tracer les frontières de l'anneau en remplaçant `\draw` par :

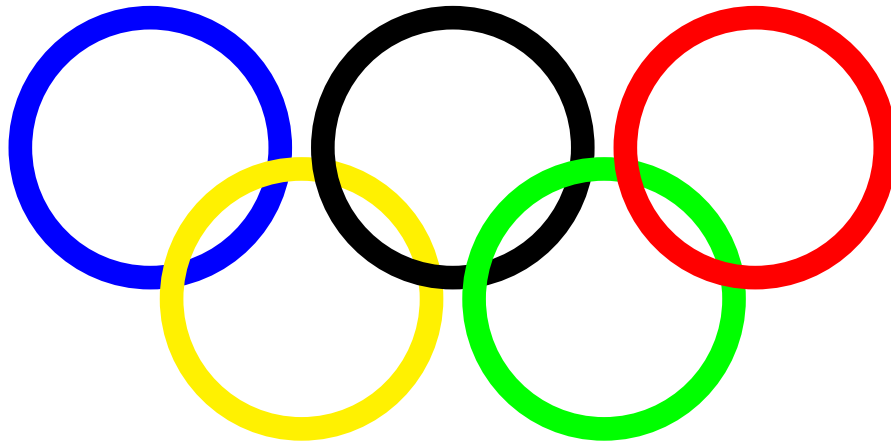
```
\fill[lightgray,even odd rule] (0,0) circle(2.5) circle(3);
```



### 8.1.2 Entrelacer les anneaux : `\coordinate`, `fill` et `arc`

On va maintenant définir et nommer les coordonnées des centres des cercles et tracer les anneaux :

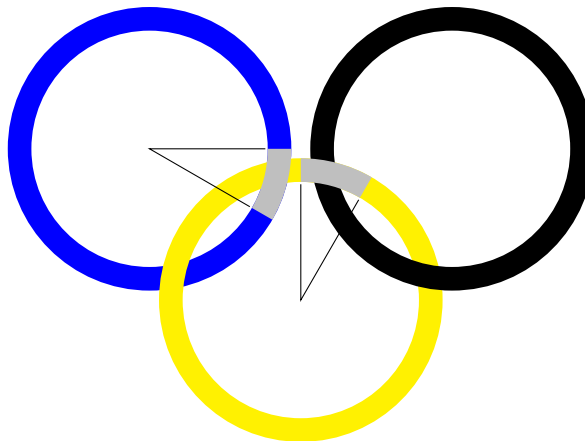
```
\coordinate (A) at (-6.4,0);
\coordinate (B) at (-3.2,-3.2);
\coordinate (C) at (0,0);
\coordinate (D) at (3.2,-3.2);
\coordinate (E) at (6.4,0);
\fill[blue,even odd rule] (A) circle(2.5) circle(3);
\fill[yellow,even odd rule] (B) circle(2.5) circle(3);
\fill[black,even odd rule] (C) circle(2.5) circle(3);
\fill[green,even odd rule] (D) circle(2.5) circle(3);
\fill[red,even odd rule] (E) circle(2.5) circle(3);
```



On reconnaît bien les anneaux olympiques. Hélas, ils ne sont pas entrelacés, mais placés les uns sur les autres selon leur ordre de création.

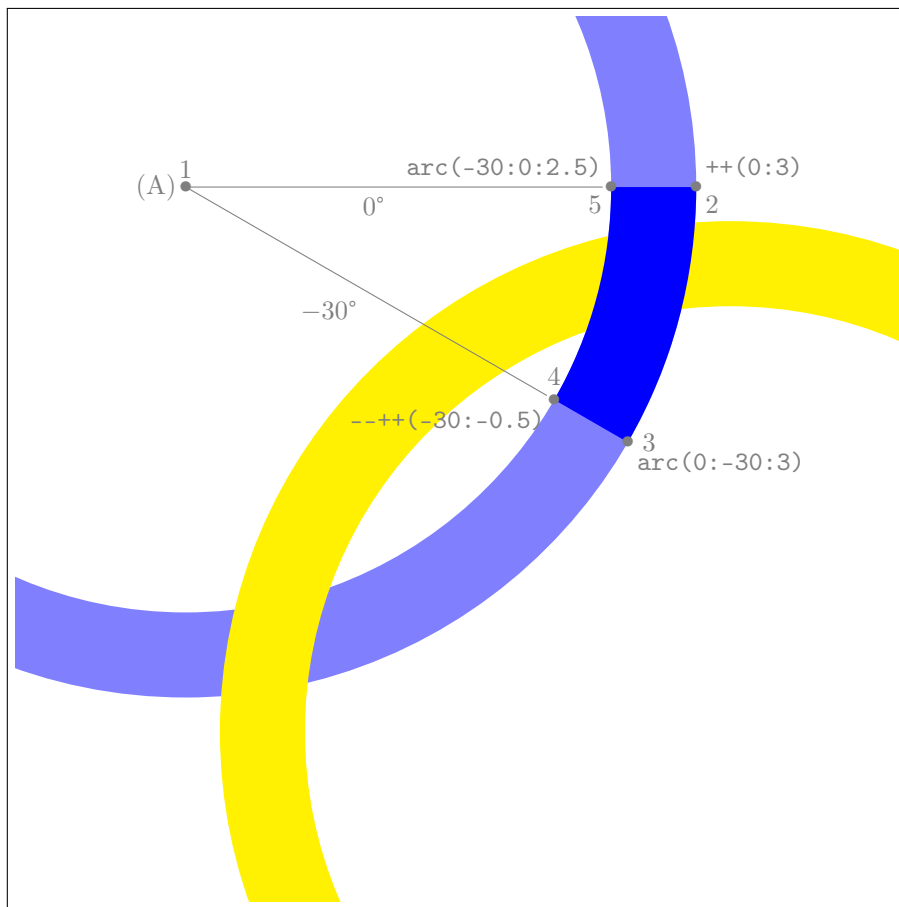
Sur la figure qui suit, on a dessiné des secteurs angulaires de  $30^\circ$  qui repèrent la zone où l'anneau bleu devrait être au dessus de l'anneau jaune et la zone où l'anneau jaune devrait être au dessus de l'anneau noir.

Donc, une fois les anneaux superposés normalement, il suffit de dessiner à nouveau, par dessus, les arcs d'anneaux correspondants aux zones concernées. C'est-à-dire les régions coloriées en gris :



Par exemple, pour entrelacer les deux anneaux bleu et jaune, on va faire passer un arc d'anneau bleu sur le jaune avec la commande :

```
\fill[blue] (A) ++(0:3)
  arc(0:-30:3)      -- ++(-30:-0.5)
  arc(-30:0:2.5)   -- cycle;
```



Suivons pas à pas la frontière de la région coloriée :

1. (A)                    origine
2. ++(0:3)                avance, au départ, *sans tracer* de 3 dans la direction  $0^\circ$
3. arc(0:-30:3)          trace un arc de rayon 3 de  $0^\circ$  à  $-30^\circ$
4. --+(-30:-0.5)        trace vers le centre un trait de 0,5 direction  $-30^\circ$
5. arc(-30:0:2.5)        trace un arc de rayon 2,5 de  $-30^\circ$  à  $0^\circ$
6. --cycle                retour au point de départ 2.

On procède de la même façon pour le recouvrement de la partie jaune en remplaçant l'origine (A) par le point (B) et les angles de  $0^\circ$  à  $-30^\circ$  par l'intervalle des angles de  $90^\circ$  à  $60^\circ = 90^\circ - 30^\circ$ .

En résumé : on trace d'abord tous les anneaux, puis ensuite, on retrace la partie des anneaux entrelacés qui doit être au dessus.

### 8.1.3 La figure complète : `\newcommand`

On remarque que le dessin d'un anneau est déterminé par son centre et sa couleur. La partie de recouvrement est déterminée par son centre, sa couleur et son angle de départ. Il semble donc naturel d'en faire des commandes :

```
\newcommand{\anneau}[2]%
{\fill[#2,even odd rule] (#1) circle (2.5) circle (3);}
```

- #1 est le centre
- #2 est la couleur



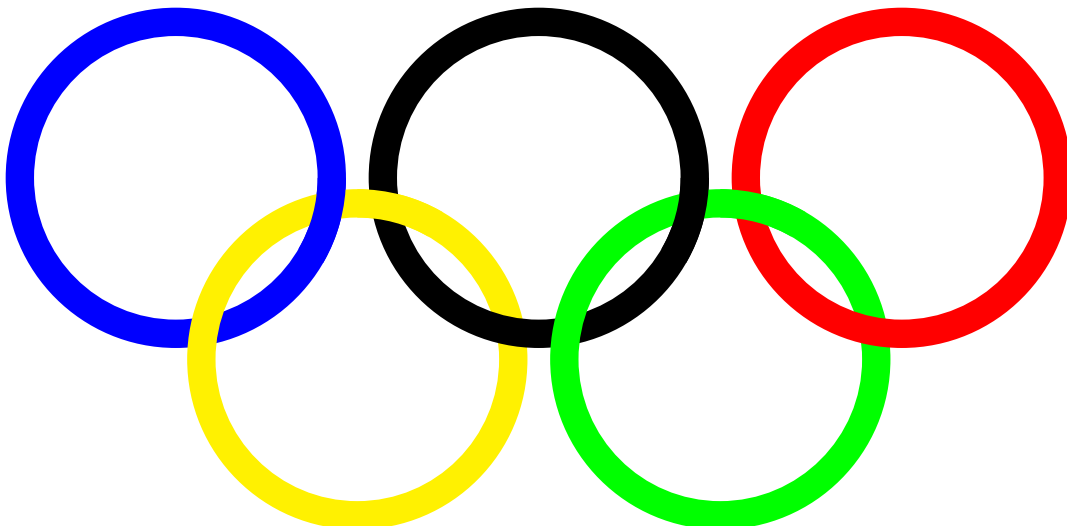
```
\newcommand{\recouvrement}[3]%
{\fill[#3] (#1) ++(#2:3)
  arc (#2:{#2-30}:3) -- ++({#2-30}:-0.5)
  arc ({#2-30}:#2:2.5) -- cycle;}
```

- #1 est le centre
- #2 est la couleur
- #3 est l'angle de départ du recouvrement

On note l'utilisation des calculs dans la définition de cette commande. Pour que cet exemple fonctionne correctement, il ne faut pas oublier d'inclure dans le préambule `\usetikzlibrary{calc}`, après avoir chargé le package `TikZ` (uniquement avec PGF version 2).

On trace alors les anneaux olympiques avec :

```
\anneau{A}{blue}      \anneau{C}{black}    \anneau{D}{green}
  \anneau{B}{yellow}  \anneau{E}{red}
\recouvrement{A}{0}{blue}  \recouvrement{B}{90}{yellow}
\recouvrement{C}{0}{black} \recouvrement{D}{90}{green}
```



## 8.2 Diagrammes de Venn

Dans cet exemple, les diagrammes illustrent un sujet mathématique, mais la construction de ces diagrammes ne demande pas de précision particulière.

Les diagrammes de Venn permettent de faire des schémas d'ensembles. On se limitera à deux ensembles d'un même référentiel.

On choisit de représenter le référentiel  $E$  par un rectangle et les ensembles  $A$  et  $B$  par des cercles concourants dans ce rectangle, mais on pourrait aussi choisir de représenter ces ensembles par d'autres formes quelconques.

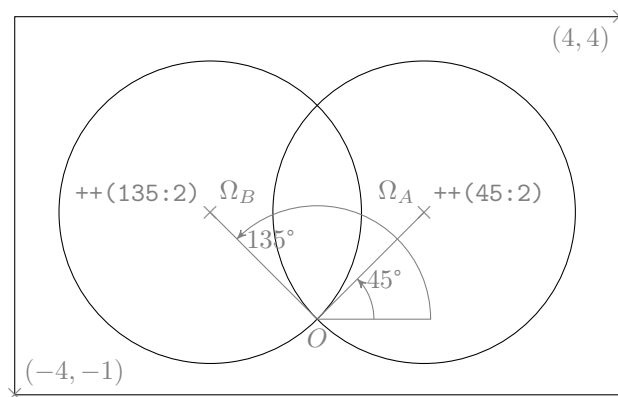
On va construire les diagrammes classiques qui représentent :  $A$ ,  $B$ ,  $A \cup B$ ,  $A \cap B$ ,  $A \setminus B$ ,  $B \setminus A$ ,  $A \Delta B$  et leurs complémentaires dans le référentiel  $E$ .

On proposera deux méthodes différentes pour réaliser ces diagrammes.

### 8.2.1 Ensembles $E$ , $A$ , $B$ : rectangle, cercle, `\newcommand`

À partir de l'origine  $O$  point d'intersection bas des deux cercles, on placera les centres  $\Omega_A$  et  $\Omega_B$  à 2 cm de  $O$ , respectivement à  $135^\circ$  et à  $45^\circ$ , comme on le voit sur la figure :

```
\begin{tikzpicture}
  \draw (-4,-1) rectangle (4,4); % E
  \draw (0,0) ++(135:2) circle (2); % A
  \draw (0,0) ++(45:2) circle (2); % B
\end{tikzpicture}
```



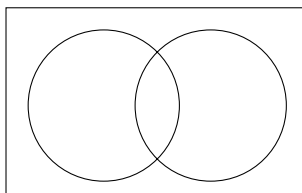
Pour éviter d'avoir à réécrire chaque fois les coordonnées des différents dessins, on propose de les définir comme des commandes  $\LaTeX$  qu'il sera facile d'utiliser ensuite dans les commandes  $TikZ$ . On gagne ainsi du temps à la rédaction, mais surtout, on rend le code plus compréhensible.

On va définir les commandes  $\LaTeX$   $\backslash E$ ,  $\backslash A$  et  $\backslash B$  pour représenter respectivement les ensembles  $E$ ,  $A$  et  $B$  :

```
\newcommand{\E}{(-4,-1) rectangle (4,4)}
\newcommand{\A}{(0,0) ++(135:2) circle (2)}
\newcommand{\B}{(0,0) ++(45:2) circle (2)}
```

Chaque fois que, par exemple, la commande  $\backslash E$  sera rencontrée dans le code  $TikZ$ , elle sera remplacée par le texte `(-4,-1) rectangle (4,4)`

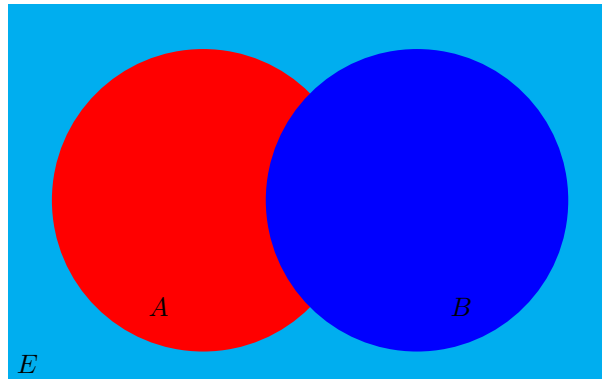
Ainsi, le code `\tikz\draw\E\A\B;` produit :



Pour l'instant, seuls les contours ont été tracés. Il est possible de les colorier en remplaçant la commande `\draw` par la commande `\fill`.

### 8.2.2 Coloriage : `\fill`, `color`, `opacity`

Écrire : `\fill[cyan] \E; \fill[red] \A; \fill[blue] \B;` donne :



Les couleurs sont un peu violentes! Mais surtout, l'ensemble  $B$  qui a été dessiné en dernier, cache une partie de l'ensemble  $A$ .

Pour éclaircir la couleur de  $E$ , on va remplacer `\fill[cyan] \E;` par `\fill[cyan!50] \E;` Hélas, cela provoque une erreur de compilation avec le message :

```
! Missing \encsname inserted. <to be read again> \penalty
```

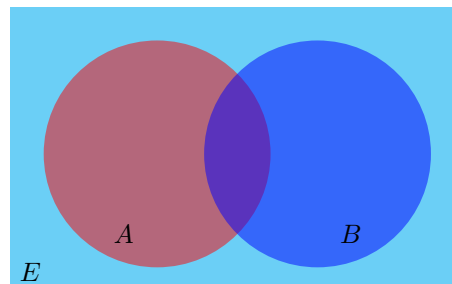
En effet, `\fill[cyan]` est une abréviation de `\fill[color=cyan]` qui est permise par la syntaxe de TikZ, car `cyan` est reconnu sans ambiguïté comme une couleur par la commande `\fill`.

Par contre `cyan!50` provoque une erreur. Dans ce cas, il est obligatoire d'écrire `\fill[color=cyan!50]`.

On peut éviter ce type de problème en écrivant systématiquement `color=` lorsqu'on définit une couleur.

Pour éviter ensuite le recouvrement de l'ensemble  $A$  par l'ensemble  $B$ , on peut définir un coloriage transparent à l'aide de l'option `opacity`.

```
\fill[color=cyan!50] \E;
\fill[opacity=0.5,red] \A;
\fill[opacity=0.5,blue] \B;
```



### 8.2.3 Méthode par superposition de couleurs

Dans cette première méthode, le coloriage des régions sera effectué en plusieurs couches. Certaines zones du dessin seront coloriées plusieurs fois.

Effacer une zone peut être considéré comme la colorier en blanc (la couleur du fond).

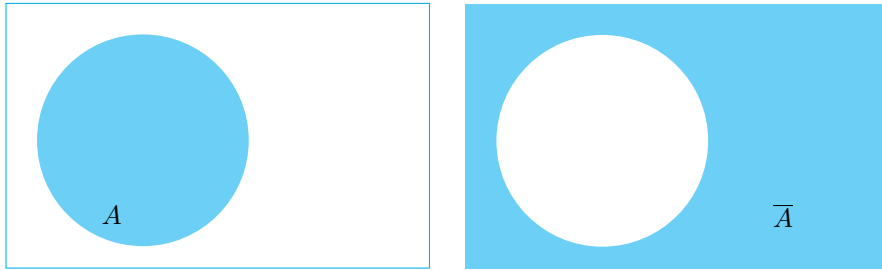
$A$ ,  $B$  et  $A \cup B$  : `\draw` et `\fill`

Le disque de  $A$  est simplement colorié dans le rectangle de  $E$  :

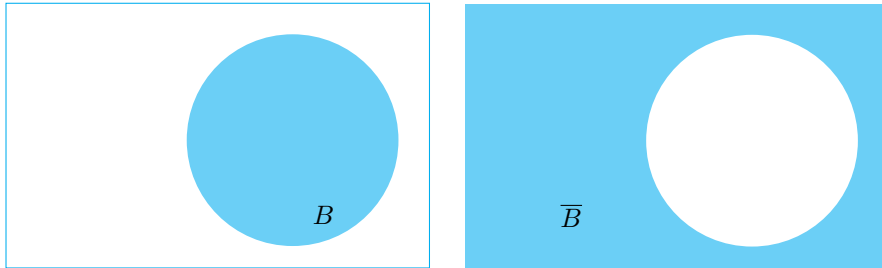
```
\draw[cyan] \E; \fill[cyan] \A;
```

Le complémentaire  $\bar{A}$  est obtenu en coloriant le rectangle de  $E$  puis en effaçant le disque de  $A$ , c'est-à-dire en le coloriant en blanc :

```
\fill[cyan] \E; \fill[white] \A;
```

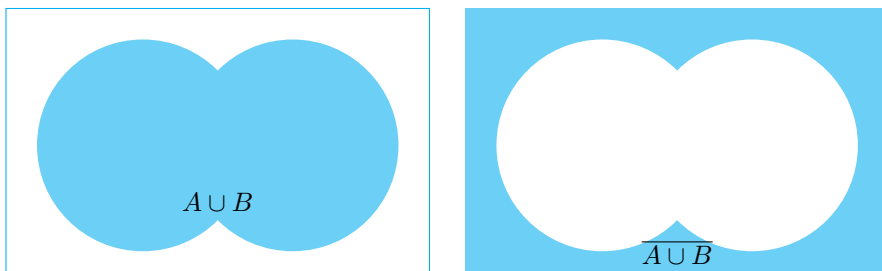


De même pour les ensembles  $B$  et  $\bar{B}$  :



Pour l'ensemble  $A \cup B$ , on va superposer les disques représentant  $A$  et  $B$ , et pour l'ensemble  $\overline{A \cup B}$ , on fera de même en coloriant en blanc :

```
\fill[cyan] \E; \fill[white] \A; \fill[white] \B;
```



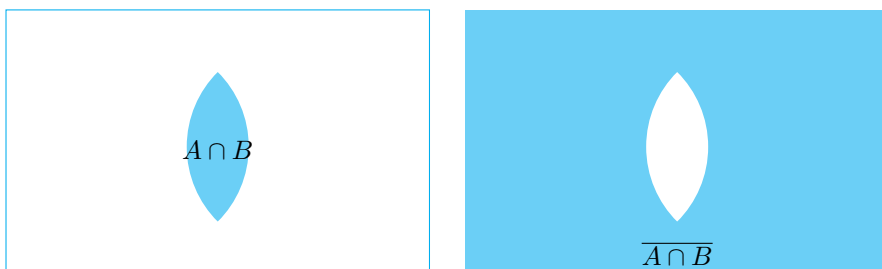
$A \cap B$ ,  $A \setminus B$ ,  $B \setminus A$  et  $A \Delta B$  : `\clip` et `scope`

Pour l'ensemble  $A \cap B$ , il faut colorier uniquement la partie du cercle de  $A$  qui est aussi dans le cercle de  $B$ .

Pour cela, on va utiliser la commande `\clip` qui définit une région de découpage dans laquelle les commandes suivantes seront effectuées. On placera le code dans un environnement `scope` pour limiter la portée du découpage :

```
\draw[cyan] \E;
\begin{scope}
  \clip \B;
  \fill[cyan] \A;
\end{scope}
```

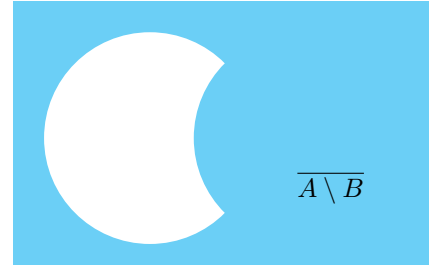
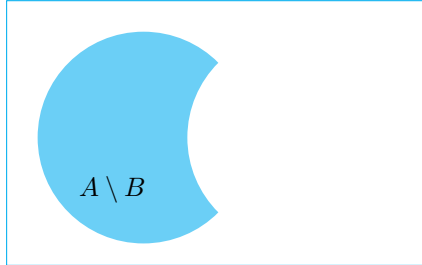
```
\fill[cyan] \E;
\begin{scope}
  \clip \B;
  \fill[white] \A;
\end{scope}
```



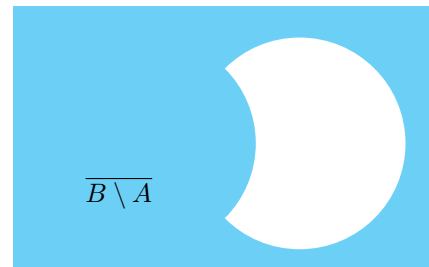
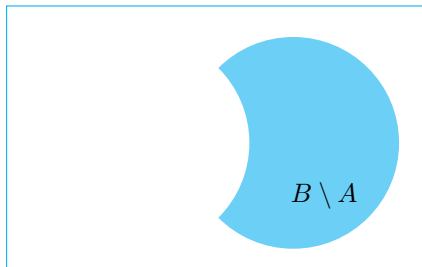
L'ensemble  $A \setminus B$  et son complémentaire  $\overline{A \setminus B}$  peuvent se dessiner par simple coloriage-effacement :

```
\draw[cyan] \E;
\fill[cyan] \A;
\fill[white] \B;
```

```
\fill[cyan] \E;
\fill[white] \A;
\fill[cyan] \B;
```



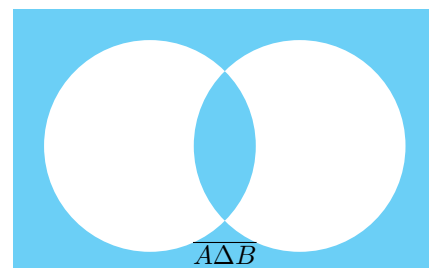
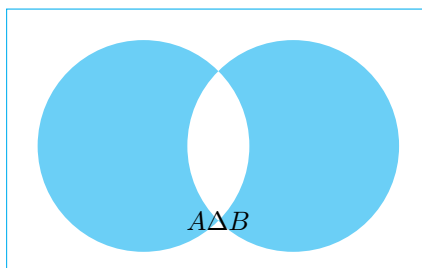
De même pour les ensembles  $B \setminus A$  et  $\overline{B \setminus A}$  :



Enfin :  $A \Delta B = (A \cup B) \setminus (A \cap B)$  et on a déjà vu précédemment comment représenter la réunion et l'intersection :

```
\draw[cyan] \E;
\fill[cyan] \A;
\fill[cyan] \B;
\begin{scope}
  \clip \B;
  \fill[white] \A;
\end{scope}
```

```
\fill[cyan] \E;
\fill[white] \A;
\fill[white] \B;
\begin{scope}
  \clip \B;
  \fill[cyan] \A;
\end{scope}
```



### 8.2.4 Méthode par coloriage entre les frontières

Dans cette seconde méthode, on va d'abord définir les frontières qui entourent les différents ensembles, puis colorier l'intérieur des régions définies par ces frontières.

Comme on l'a déjà dit précédemment, le mode par défaut de coloriage des régions est l'option `nonzero rule` qu'il est inutile de spécifier, l'autre mode qu'il est possible de définir est `even odd rule`.

La définition de ces modes est compliquée (voir le manuel : *interior rules*). En particulier, le mode par défaut tient compte du sens de parcours de la frontière pour déterminer la région à colorier. Or, quand on trace un rectangle ou un cercle, le sens du tracé est inconnu. Dans ce cas, le coloriage donne très souvent des résultats inattendus et difficiles à interpréter.

**Conseil :** Dès qu'une région est compliquée (en plusieurs morceaux ou avec des trous), il est très fortement conseillé d'utiliser `even odd rule` comme option de coloriage. Il faut donc, dans ce cas, le spécifier explicitement dans chaque commande `\fill`.

Dans le mode `even odd rule`, le coloriage se fait de façon assez naturelle, l'extérieur est blanc, l'intérieur est colorié. Il suffit donc de savoir ce qu'on appelle intérieur et extérieur :

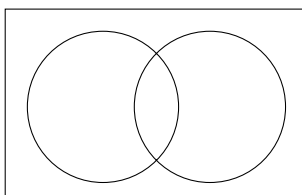
- l'espace situé autour de la figure totale est à l'extérieur
- chaque fois qu'on franchit une frontière :
  - si on est à l'extérieur on passe à l'intérieur
  - si on est à l'intérieur on passe à l'extérieur

**Attention :** Dans toute cette section `even odd rule` est explicitement spécifié dans chaque commande `\fill`.

### Définition des frontières : rectangle, circle et arc

Les frontières de  $E$ ,  $A$  et  $B$  ont été définies précédemment avec les commandes `\E`, `\A` et `\B`.

Le code `\tikz\draw\E\A\B;` produit :

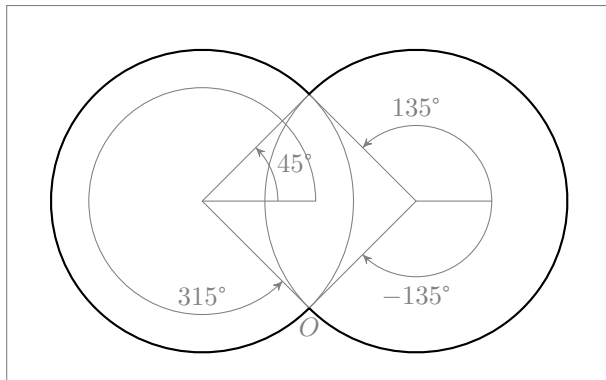


Nous allons définir, de même pour  $A \cup B$ ,  $A \cap B$ ,  $A \setminus B$  et  $B \setminus A$ , les commandes respectives `\AuB`, `\AnB`, `\AmB` et `\BmA`.

Pour :  $A \cup B$  : `\draw (0,0) arc(-135:135:2) arc(45:315:2);`

Et donc la commande :

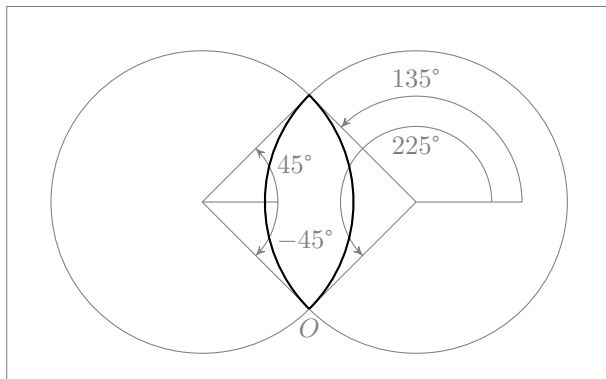
`\newcommand{\AuB}{(0,0) arc(-135:135:2) arc(45:315:2)}`



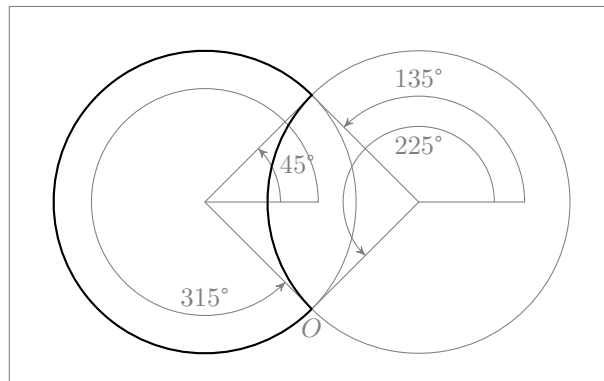
Pour :  $A \cap B$  : `\draw (0,0) arc (-45:45:2) arc (135:225:2);`

Et donc la commande :

`\newcommand{\AnB}{(0,0) arc (-45:45:2) arc (135:225:2)}`



Pour :  $A \setminus B$  : `\draw (0,0) arc (225:135:2) arc (45:315:2);`  
 Et donc la commande :  
`\newcommand{\AmB}{(0,0) arc (225:135:2) arc (45:315:2)}`

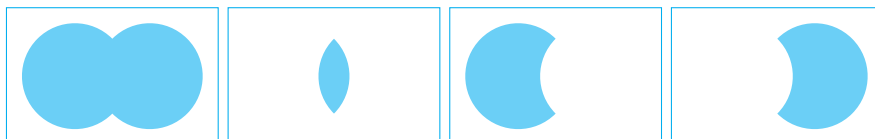


Et de même pour  $B \setminus A$  la commande :  
`\newcommand{\BmA}{(0,0) arc (-135:135:2) arc (45:-45:2)}`

**Coloriage des régions :** `\fill, even odd rule`

Maintenant que les frontières sont définies, le coloriage ne pose plus de problème et on va obtenir les mêmes résultats que précédemment avec :

`\draw[cyan] \E;`  
`\fill[cyan] \AuB; % ou respectivement \AnB \AmB \BmA`



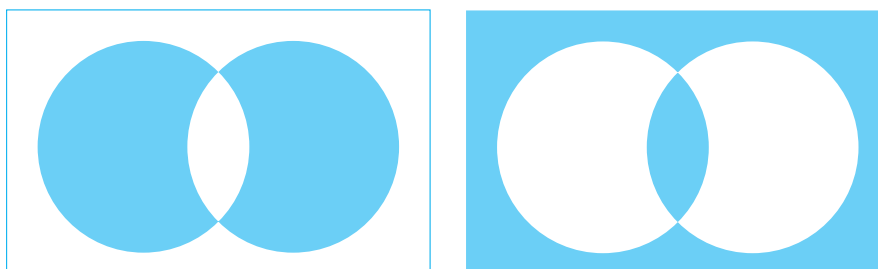
Et les complémentaires avec :

`\draw[cyan,even odd rule] \E \AuB; % resp. \AnB \AmB \BmA`



Pour l'ensemble  $A \Delta B$ , on n'a pas défini de frontière particulière. C'est en effet inutile, car l'option `even odd rule` permet un coloriage correct en utilisant pour frontière de  $A \Delta B$  la réunion de celle de  $A$  et de  $B$  ainsi :

`\draw[cyan] \E;`  
`\fill[cyan,even odd rule] \A \B;`                      `\fill[cyan,even odd rule] \E \A \B;`

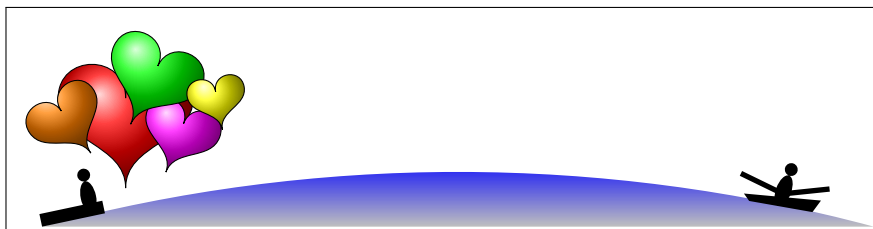


### 8.3 Personnages et décors

Dans cet exemple, on a choisi d'illustrer un problème de mathématiques récréatives. Les choix esthétiques sont ceux de l'auteur, mais on laisse le lecteur libre de refaire la figure selon sa propre inspiration.

#### Cœur brisé

Vous venez de plaquer l'ex-amour de votre vie!

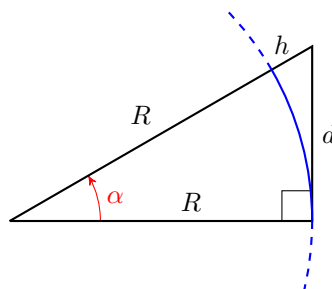


Vous l'abandonnez sur la jetée, altitude de ses yeux humides : 4 m et vous ramez irrésistiblement vers le large, altitude de vos yeux impitoyables : 1 m.

*À quelle distance du rivage échapperez-vous à son regard déchirant, en disparaissant à l'horizon ?*

#### Solution

- $R$  est le rayon de la terre ;
- $h$  la hauteur des yeux au dessus du sol ;
- $d$  la distance entre l'œil et l'horizon ;
- $c$  la distance au sol (arc de cercle).



On a  $\cos(\alpha) = \frac{R}{R+h}$  donc  $c = R\alpha$  et  $d = R \tan(\alpha)$

On peut prendre  $R \approx 6365$  km,  $h_1 = 0,001$  km,  $h_2 = 0,004$  km.

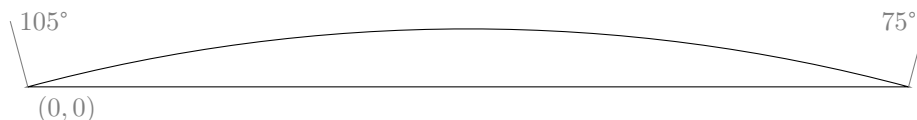
Comme  $\alpha$  est très petit, on a  $c_1 \approx d_1 \approx 3,568$  km et  $c_2 \approx d_2 \approx 7,136$  km

**Donc le cœur est définitivement brisé à environ 15 km**

#### 8.3.1 L'océan : \shade, arc, top color, bottom color

On va dessiner un arc avec un grand rayon :

```
\draw (0,0) arc (105:75:25) -- cycle;
```




On peut alors remplacer la commande `\draw`, la commande `\shade` pour colorier l'océan et définir les couleurs des dégradés, par exemple les options : `top color=blue` et `bottom color=lightgray`





### 8.3.2 Le quai : `\fill`, `rectangle`, `rotate`


Le quai est un rectangle : `\fill (0,0)rectangle(1,0.2);` 

Il suffit de le tourner un peu en ajoutant l'option `[rotate=12]` pour le mettre en place à gauche de l'image :



### 8.3.3 Les personnages : `\fill`, `ellipse`, `circle`

Les personnages sont de petites silhouettes constituées d'une ellipse et d'un cercle dessinés avec la commande `\fill`, et dont les dimensions ont été rendues satisfaisantes avec quelques essais :

`\fill (0,0)ellipse(0.12 and 0.25) (0,0.35)circle(0.1);` 

Quelques essais sont encore nécessaires pour placer le personnage sur le quai à l'aide des options `[rotate=12,shift={(0.8,0.3)}]` :



Le second personnage rame dans une barque que nous allons d'abord dessiner avant de l'y placer.

```
\fill (0,0)--(1,0)--(1.1,0.2)--(-0.1,0.1)--cycle;% barque
\draw[line width=2pt] (-0.2,0.4)--(0.5,0.2)--(1.2,0.4);% rames
```



On ajoute le personnage en l'inclinant légèrement pour produire un effet plus dynamique :

```
\fill[shift={(0.5,0.25)},rotate=-10] % personnage
(0,0)ellipse(0.12 and 0.25) (0,0.35)circle(0.1);
```



Il n'y a plus qu'à le placer la barque au large en insérant les instructions précédentes dans un environnement `scope` avec les options appropriées :

```
\begin{scope} [shift={(11,0.4)},rotate=-10]
% ...
\end{scope}
```



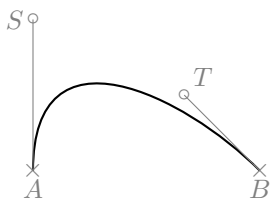
Il faut bien insister sur le fait que pour arriver à placer correctement les différents éléments du dessin, de nombreux essais ont été nécessaires.

### 8.3.4 Le cœur : `\draw`, `.. controls and ..`

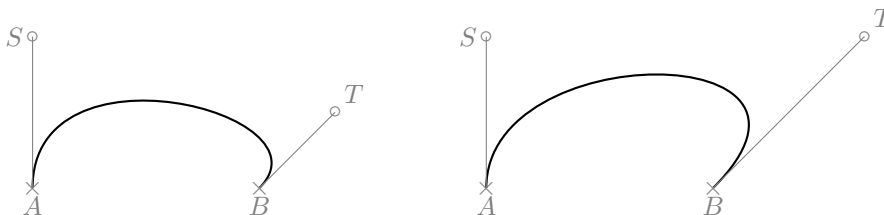
Pour dessiner une figure quelconque, on peut utiliser des courbes de Bézier. Une courbe de Bézier est définie à l'aide de quatre points de contrôle :

- $A$ , le point de départ de la courbe ;
- $S$ , le point tel que  $AS$  est la tangente à la courbe en  $A$  ;
- $T$ , le point tel que  $BT$  est la tangente à la courbe en  $B$  ;
- $B$ , le point d'arrivée de la courbe.

```
\draw (A) .. controls (S) and (T) .. (B);
```



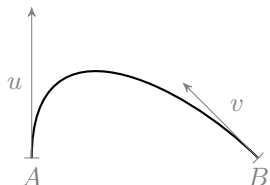
Une fois les points  $A$  et  $B$  placés, on peut modifier la courbure en jouant de façon assez intuitive avec les points  $S$  ou  $T$ . On montre ici deux déplacements du point  $T$  et l'effet produit sur la courbe :



On peut considérer que plus  $AS$  ou  $BT$  sont longs, plus la courbe colle à la tangente. On constate donc que les déplacements des points de contrôle  $S$  et  $T$  ont un effet global sur la forme de la courbe, mais sans en déplacer les extrémités  $A$  et  $B$ .

Une autre syntaxe permet de définir les vecteurs tangents  $\vec{u} = \overrightarrow{AS}$  et  $\vec{v} = \overrightarrow{BT}$  en coordonnées relatives par rapport à  $A$  et  $B$  respectivement :

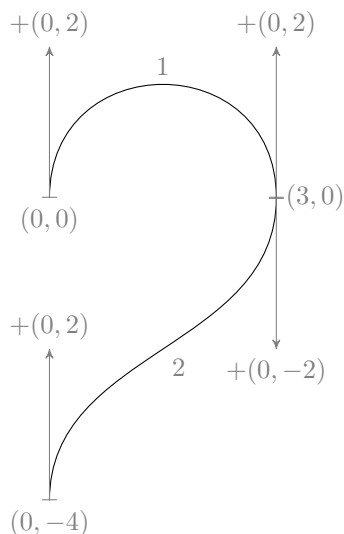
```
\draw (A) .. controls +(u) and +(v) .. (B);
```



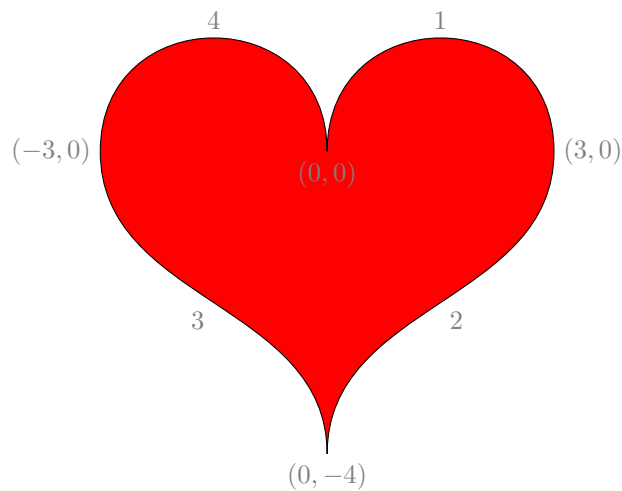
On peut maintenant construire des formes complexes constituées d'arcs de Bézier raccordés bout à bout. Si, de plus, on aligne les vecteurs tangents aux extrémités communes, on obtient facilement des courbes lisses.

Par exemple, on peut commencer à dessiner un cœur ainsi :

```
\draw (0,0) ..controls +(0,2) and +(0,2).. (3,0); % arc 1
\draw (3,0) ..controls +(0,-2) and +(0,2).. (0,-4); % arc 2
```



En ajoutant deux arcs supplémentaires, on complète le cœur avec une symétrie par rapport à l'axe vertical : `\draw[fill=red] (0,0) ... cycle;`



### 8.3.5 Cœurs multicolores : `\shift`, `rotate`, `ball color`

Les options de la commande `\draw` peuvent être modifiées pour changer l'aspect du cœur, comme par exemple :



```
\draw [scale=0.25,rotate=45,ball color=blue]
(0,0) .. controls +(0,2) and +(0,3) .. (3,0)
      .. controls +(0,-2) and +(0,2) .. (0,-4)
      .. controls +(0,2) and +(0,-2) .. (-3,0)
      .. controls +(0,2) and +(0,2) .. (0,0);
```

On va donc créer une commande ayant pour paramètres les options de la commande `\draw` et faire appel à cette commande plusieurs fois, pour dessiner les cœurs à des endroits différents avec des orientations différentes :

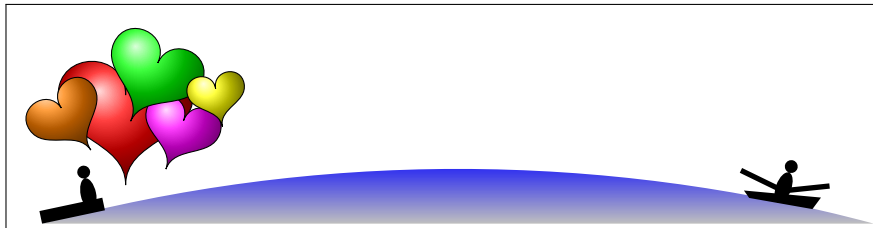
```
% Commande \coeur
\newcommand{\coeur}[1][fill=red] % rouge par défaut
{\draw [#1] (0,0)
  .. controls +(0,2) and +(0,2) .. (3,0)
  .. controls +(0,-2) and +(0,2) .. (0,-4)
  .. controls +(0,2) and +(0,-2) .. (-3,0)
  .. controls +(0,2) and +(0,2) .. (0,0);}
% Dessin
\coeur[shift={(1.3,2)},ball color=red,scale=0.35]
\coeur[shift={(0.3,1.8)},ball color=orange,scale=0.2,rotate=35]
\coeur[shift={(2.2,1.6)},ball color=magenta,scale=0.2,rotate=-20]
\coeur[shift={(1.8,2.5)},ball color=green,scale=0.25,rotate=-25]
\coeur[shift={(2.7,2.1)},ball color=yellow,scale=0.15,rotate=10]
```



(0,0) •

### 8.3.6 La figure complète : scope, shift, rotate

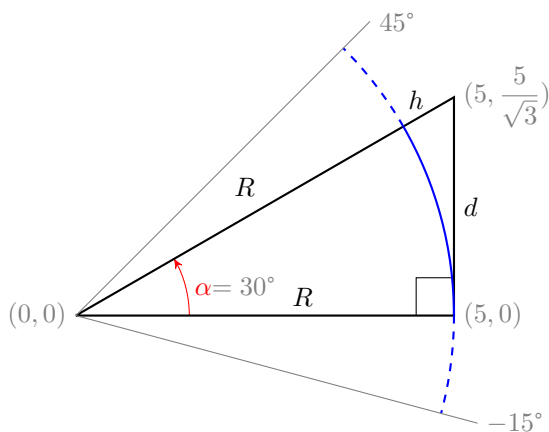
On regroupe enfin tous les éléments pour obtenir l'illustration de l'énoncé du problème :



### 8.3.7 La solution : scope, shift, rotate

La figure proposée dans la solution est une figure géométrique classique qu'il sera facile de réaliser à titre d'exercice.

Pour simplifier les calculs d'angles et de coordonnées, dessinons un demi-triangle équilatéral :



```
\begin{tikzpicture}
% angle alpha
\draw[->,red,>=stealth'] (1.5,0) arc (0:30:1.5);
\draw[red] (15:1.5) node[right]{\alpha};
% angle droit
\draw (4.5,0)|-(5,0.5);
% triangle
\draw[thick] (0,0)-- node[pos=0.6,above]{R} (5,0)
-- node[midway,right]{d} (5,{5/sqrt(3)})
-- node[pos=0.1,above]{h}
node[pos=0.5,above left]{R}(0,0);
% arcs
\draw[blue,thick,dashed] (0:5) arc (0:-15: 5);
\draw[blue,thick] (0:5) arc (0:30: 5);
\draw[blue,thick,dashed] (30:5) arc (30:45: 5);
\end{tikzpicture}
```

## 8.4 Résumé

On a vu dans ce chapitre comment dessiner des figures complexes en composant des lignes élémentaires comme `rectangle`, `circle`, `arc` ou en utilisant des courbes de Bézier. Les courbes dessinées peuvent être les frontières de régions qu'il est possible de colorier avec `fill`.

Ainsi `TikZ` permet non seulement de construire des figures purement géométriques de grande précision, mais il peut aussi permettre de réaliser certains travaux d'illustration.

# Chapitre 9

## Compléments techniques

### 9.1 Transformations avec `scope`

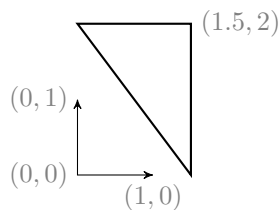
On va étudier ici l'effet des options de transformation `xshift`, `yshift`, `shift`, `scale` et `rotate`, sur une même figure, que l'on désire dessiner plusieurs fois sur un diagramme unique, à des positions différentes.

Pour cela, on placera cette figure dans sa position d'origine, en écrivant son code dans un environnement `tikzpicture`, et en insérant une copie de ce code dans un environnement `scope`, contrôlé par des options de transformation.

Les exemples suivants seront donc tous rédigés sur ce modèle :

```
\begin{tikzpicture}
  % code de la Figure
  \begin{scope} [<options de transformation>]
    % code de la Figure
  \end{scope}
\end{tikzpicture}
```

Voici la petite figure qui va nous servir à visualiser les transformations. Elle est inscrite dans un rectangle de 1,5 cm sur 2 cm de côtés :

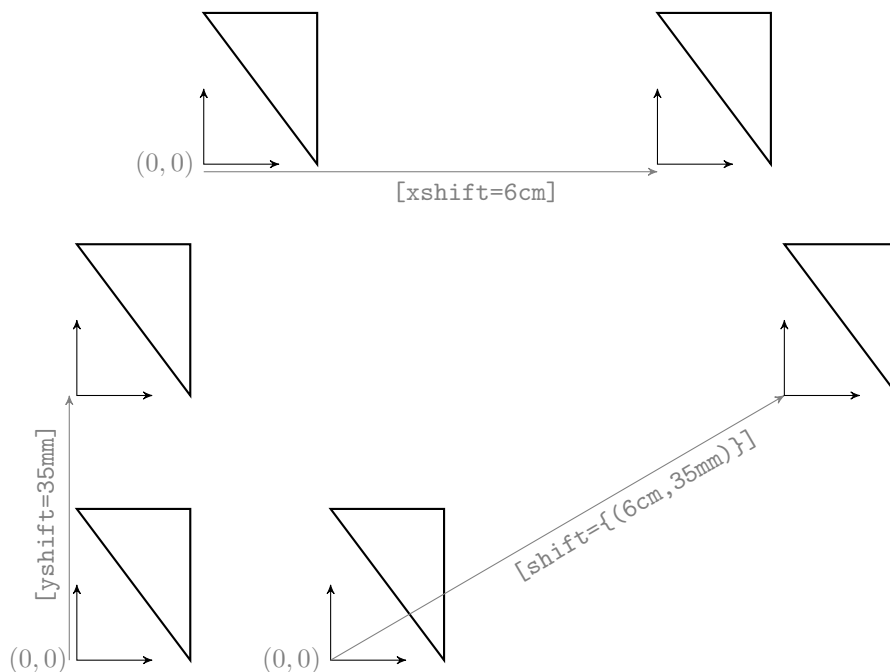


```
\begin{tikzpicture}
  % code de la Figure
  \draw [<->,>=stealth'] (1,0) -- (0,0) -- (0,1);
  \draw[thick] (1.5,0) -- (0,2) -- (1.5,2) -- cycle;
\end{tikzpicture}
```

Les transformations dont il est question ici sont appliquées au repère dans lequel sont exécutées les commandes TikZ qui construisent la figure. La figure semble donc subir les mêmes transformations, mais il s'agit en réalité de changement du repère dans lequel est dessinée la figure. Il faut donc faire attention à l'ordre dans lequel sont effectuées ces transformations.

#### 9.1.1 Translations : `xshift`, `yshift` ou `shift`

Pour éviter de superposer les deux dessins, on va d'abord effectuer une translation en utilisant une des options `xshift`, `yshift` ou `shift` :

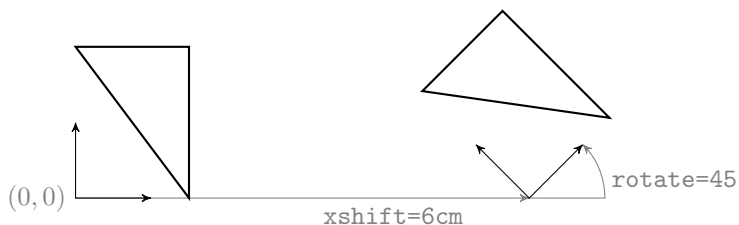


On remarquera que dans le cas de l'option `shift=`, les coordonnées du vecteur de translation doivent être insérées entre des accolades. Ceci est nécessaire, car la virgule qui sépare les coordonnées ne doit pas être interprétée comme un séparateur d'options.

On va maintenant montrer comment faire tourner ou dilater la figure. Dans tous les cas suivants, on utilisera au préalable l'option `xshift=6cm` pour éviter la superposition des deux figures.

### 9.1.2 Combinaison de translation et rotation : `[xshift=6cm,rotate=45]`

L'option `rotate=` attend pour paramètre l'angle de la rotation en degrés :

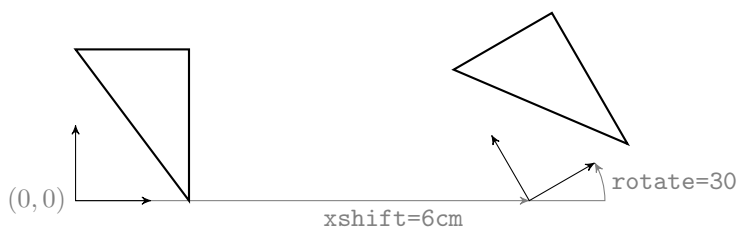


**Attention :** On conseille vivement de toujours écrire en premier l'option de translation (`xshift`, `yshift` ou `shift`), mais ce n'est pas obligatoire.

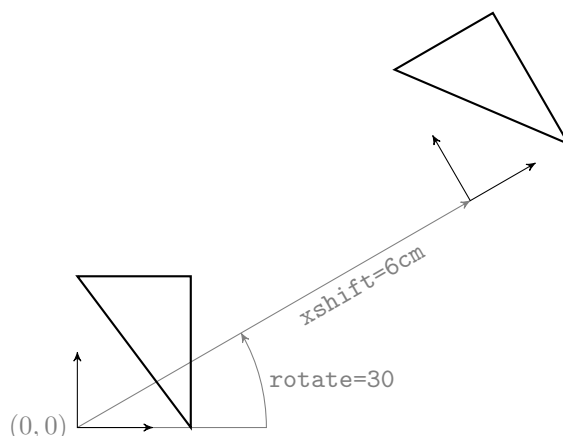
En effet le centre de la rotation est l'origine *actuelle*, c'est-à-dire l'origine de la figure après qu'elle ait subi le `xshift=6cm` dans l'exemple précédent.

Mais si on effectue la rotation en premier, le centre est l'origine (0,0) de départ. Dans ce cas la translation suivante est alors effectuée dans un repère ayant déjà subi la rotation.

Voici ce que donne : `\begin{scope}[xshift=6cm,rotate=30]`



Si on le compare à : `\begin{scope}[rotate=30,xshift=6cm]`



on constate que l'ordre des options est important et le résultat est plus difficile à interpréter dans le second cas.

Il est donc plus facile de toujours faire la translation en premier. Le résultat final est alors plus facile à comprendre.

### 9.1.3 Translation et changement d'échelle : `[xshift=6cm,scale=0.5]`

Le changement d'échelle est une homothétie, appliquée au repère dans lequel sont exécutées les commandes de construction de la figure, ce qui provoque effectivement une modification de la taille de la figure.

L'option `scale=` attend pour paramètre le rapport d'homothétie :

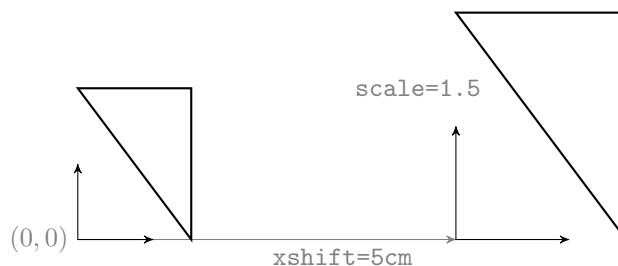


**Attention :** On conseille, comme dans le cas précédent, de toujours écrire en premier une option de translation (`xshift`, `yshift` ou `shift`) bien que ce ne soit pas obligatoire.

En effet le centre de l'homothétie est l'origine *actuelle*, c'est-à-dire l'origine du repère dans lequel est tracée la figure après qu'elle ait subi le `xshift=6cm` dans l'exemple précédent.

Si on effectue l'homothétie en premier, le centre est l'origine (0,0) de départ. Dans ce cas la translation suivante est alors effectuée dans un repère ayant déjà subi l'homothétie et dont les unités ont été multipliées par le rapport d'homothétie.

Voici ce que donne : `\begin{scope}[xshift=5cm,scale=1.5]`



Si on le compare à : `\begin{scope}[scale=1.5,xshift=5cm]`

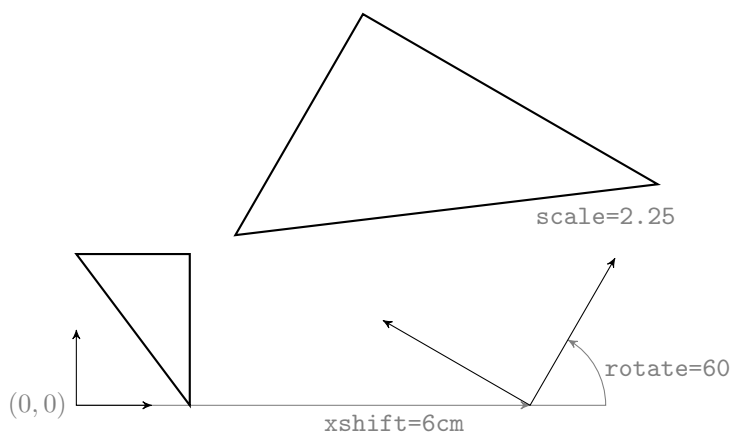


on constate, comme précédemment avec la rotation, que l'ordre des options est important et le résultat est toujours surprenant dans le second cas.

On a vu, sur les exemples, que le coefficient de l'option `scale` (rapport d'homothétie), peut être inférieur à 1, pour rétrécir, ou supérieur à 1, pour dilater la figure. Il peut aussi être négatif.

On peut évidemment composer les trois transformations, translation en premier, homothétie et rotation ensuite dans un ordre quelconque.

Par exemple : `\begin{scope}[xshift=6cm,rotate=60,scale=2.25]`



Les options `[xshift=6cm,scale=2.25,rotate=60]`, écrites dans cet ordre, donnent la même figure.

**Attention :** Toutes ces options de transformation s'appliquent au repère dans lequel sera tracée la figure définie dans l'environnement `scope`, c'est-à-dire que les transformations ont un effet sur l'ensemble des chemins (`path`) dessinés (`draw`). Par contre, ces transformations n'ont *aucun effet* sur l'épaisseur des traits, sur la taille ou l'inclinaison des textes contenus dans les nœuds (`node`) qui annotent éventuellement la figure.

#### 9.1.4 Épaisseur des traits : `\draw` et `line width`

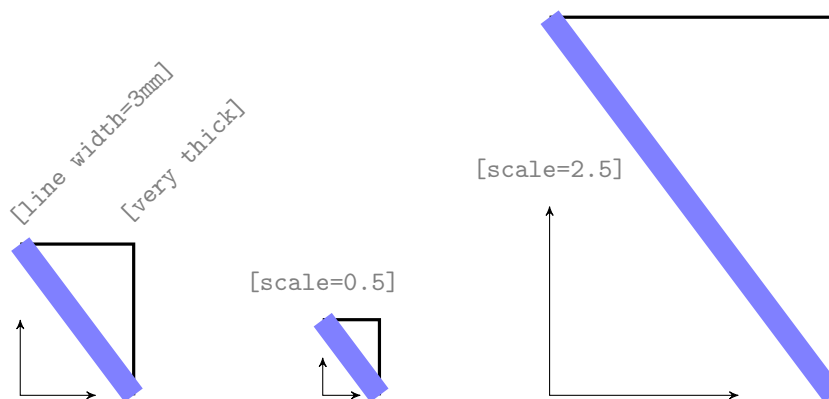
**Attention :** En première lecture, vous pouvez ignorer ce paragraphe assez technique. Par contre si vous avez l'intention de faire des transformations précises sur des figures dans lesquelles les traits ont des épaisseurs différentes, il sera nécessaire de le lire attentivement.

On va considérer une figure de référence qui ressemble à la précédente, mais avec des traits d'épaisseurs différentes :

```
\draw [<->,>=stealth'] (1,0) -- (0,0) -- (0,1);
\draw[very thick] (0,2) -- (1.5,2) -- (1.5,0);
\draw[line width=3mm,color=blue!50] (0,2) -- (1.5,0);
```

Elle est située à gauche et a été reproduite deux fois avec des options `scale` différentes pour chaque environnement `scope` :





On constate que la forme de la figure a été transformée à l'exception de l'épaisseur des traits qui est restée constante. L'homothétie porte sur les chemins (`path`) dessinés par `\draw` mais pas sur leur épaisseur définie par des options de la commande `\draw`.

L'épaisseur du trait peut être définie avec l'option `line width=` suivie d'une dimension, ou avec une des options suivantes : `ultra thin` (0.1pt), `very thin` (0.2pt), `thin` (qui est `line width=0.4pt` épaisseur par défaut si aucune option n'est spécifiée), `semithick` (0.6pt), `thick` (0.8pt), `very thick` (1.2pt) et `ultra thick` (1.6pt)

Si on veut obtenir des figures semblables, il faut modifier le code situé à l'intérieur de l'environnement `scope` pour définir l'épaisseur de chaque trait en multipliant l'épaisseur originale par le coefficient de l'option `scale`.

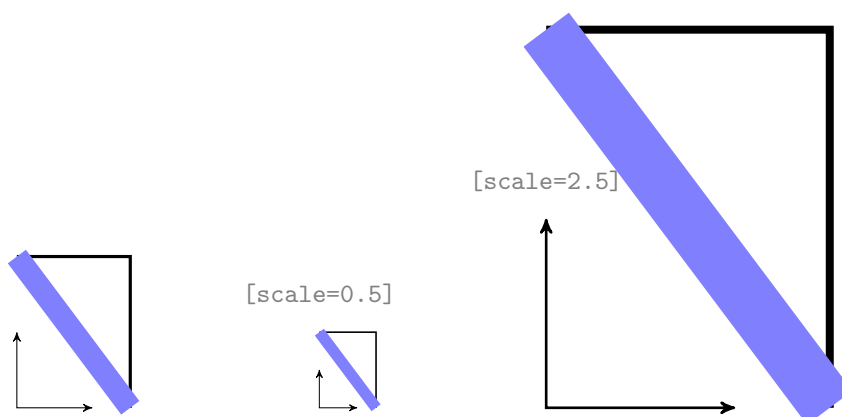
Dans le cas où l'environnement `scope` est contrôlé par `[scale=0.5]`, toute commande de la forme : `\draw[line width=<d>] ...`; où `<d>` est une dimension, sera remplacée par : `\draw[line width={0.5*<d>}] ...`;

Par exemple :

```
\draw ...; par \draw[line width={0.5*0.4pt}] ...;
\draw[very thick] ...; par \draw[line width={0.5*1.6pt}] ...;
\draw[line width=3mm] ...; par \draw[line width={0.5*3mm}] ...;
```

Ne pas oublier d'insérer les calculs entre accolades.

Maintenant, le changement d'échelle est appliqué à la figure complète (épaisseur des traits comprise). Voici ce que donnera, après correction, notre figure :



**Remarque :** Pour que cet exemple fonctionne correctement, il ne faut pas oublier d'inclure dans le préambule `\usetikzlibrary{calc}` après avoir chargé le package `TikZ` (uniquement avec PGF version 2).

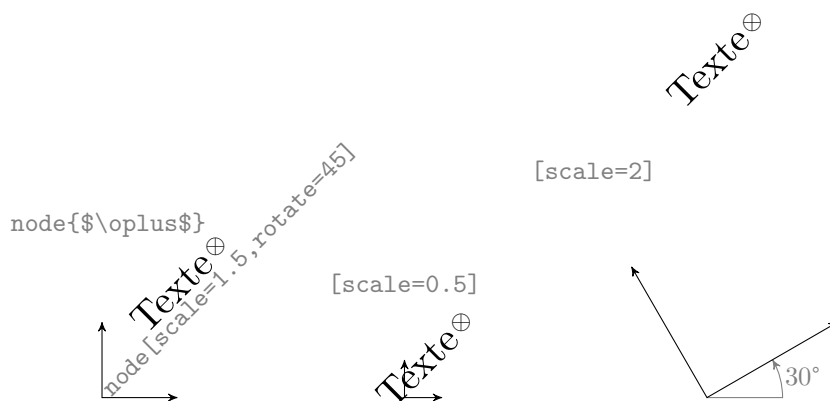
### 9.1.5 Taille et inclinaison de textes : transform shape

**Attention :** En première lecture, vous pouvez ignorer ce paragraphe qui est assez technique. Par contre si vous avez l'intention de faire des transformations précises sur des figures contenant de nombreuses annotations textuelles, il sera nécessaire de le lire attentivement.

On va considérer une figure de référence qui ressemble à la précédente, mais avec des annotations textuelles variées en taille et en orientation :

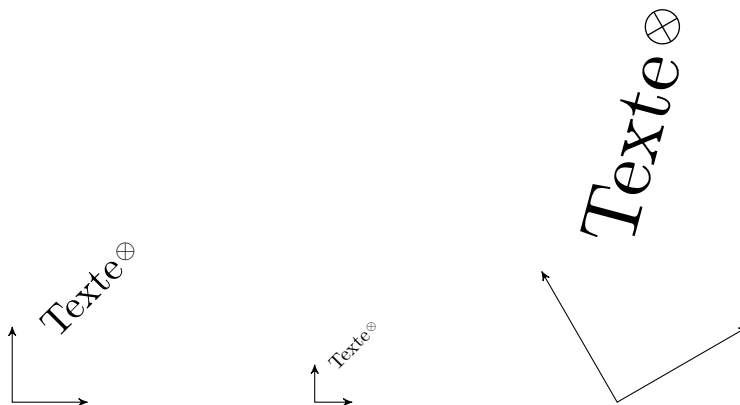
```
\draw [<->,>=stealth'] (1,0) -- (0,0) -- (0,1);
\path (1.5,2) node{${\oplus}$}
      node[left,scale=1.5,rotate=45]{Texte};
```

et lui faire subir deux transformations :



On constate que la forme de la figure a été transformée et que les annotations textuelles sont correctement placées, *mais elles ont conservé la même taille et la même inclinaison*. La plupart du temps, on veut simplement modifier la taille de la figure et c'est donc tout à fait satisfaisant, les annotations gardent une taille assortie à celle du texte qui accompagne la figure.

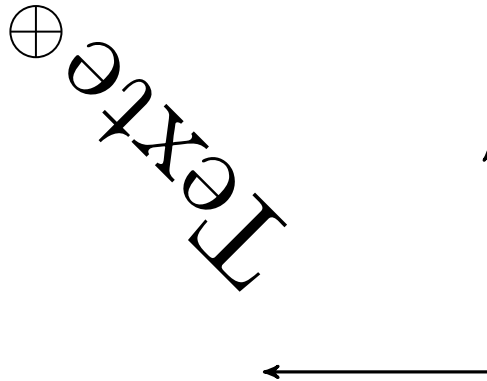
Par contre, si on a l'intention de montrer une image réduite ou dilatée du diagramme avec des texte proportionnels correctement inclinés, il va falloir ajouter à la figure l'option `[transform shape]` qui va faire subir aux nœuds les transformations définies dans les environnements `{scope}` :



On remarque qu'on a bien obtenu des figures de formes semblables, textes compris. Seule l'épaisseur des traits est restée constante.

### 9.1.6 Exercice

À titre d'exercice, faites subir un quart de tour et triplez la figure.



#### Solution

```
\begin{tikzpicture} [scale=3,rotate=90,transform shape]
  \draw[<->,>=stealth',very thick] (1,0)--(0,0)--(0,1);
  \path (1.5,2) node{\oplus}
          node[left,scale=1.5,rotate=45]{Texte};
\end{tikzpicture}
```

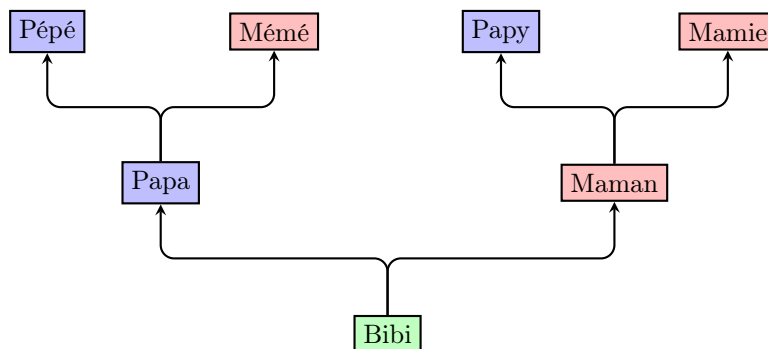
## 9.2 Au sujet des arbres

En mathématiques, la structure d'arbre est un cas important de la théorie des graphes. TikZ propose de nombreuses opérations spécifiques pour la construction des arbres.

Nous allons montrer sur un exemple vu précédemment sous forme d'exercice, comment dessiner d'un arbre généalogique :

```
% définition des styles
\tikzstyle{lien}=[->,>=stealth,rounded corners=5pt,thick]
\tikzset{individu/.style={draw,thick,fill=#1!25},
         individu/.default={green}}

% définition de l'arbre
\begin{tikzpicture}
  \node[individu] (B) at (0,0) {Bibi};
  \node[individu=blue] (P) at (-3,2) {Papa};
  \node[individu=red] (M) at (3,2) {Maman};
  \node[individu=blue] (GPP) at (-4.5,4) {Pépé};
  \node[individu=red] (GMP) at (-1.5,4) {Mémé};
  \node[individu=blue] (GPM) at (1.5,4) {Papy};
  \node[individu=red] (GMM) at (4.5,4) {Mamie};
  \draw[lien] (B) |- (-1,1) -| (P);
  \draw[lien] (B) |- (1,1) -| (M);
  \draw[lien] (P) |- (-4,3) -| (GPP);
  \draw[lien] (P) |- (-2,3) -| (GMP);
  \draw[lien] (M) |- (2,3) -| (GPM);
  \draw[lien] (M) |- (4,3) -| (GMM);
\end{tikzpicture}
```



On va d'abord rappeler le vocabulaire usuel concernant les arbres.

Ci-dessus, le nœud « Bibi » est la racine de l'arbre. Ce nœud « Bibi » a deux nœuds fils : le nœud « Papa » et le nœud « Maman ». Les nœuds terminaux, ou feuilles de l'arbre, sont donc les nœuds « Pépé », « Mémé », « Papy » et « Mamie ».

**Remarque :** Cet arbre représente les ancêtres de Bibi, donc, du point de vue mathématique, Papa et Maman, les parents de Bibi, sont bien les *nœuds fils* du nœud « Bibi », *au sens de la structure d'arbre*, de même que « Papy » et « Mamie » sont mathématiquement deux *nœuds frères* du *nœud parent* « Maman ».

### 9.2.1 Définition : `\node` `node` et `child`

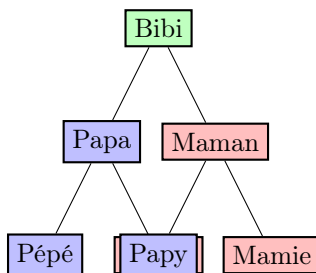
Si on définit un nœud avec la commande `\node`, l'opération `child` permet de lui associer des fils sous la forme suivante :

```
\node {Racine} child{ node{Fils1}...} child{ node{Fils2}...};
```

et ceci de façon récursive.

On peut donc définir l'exemple précédent ainsi :

```
\begin{tikzpicture}
\node [individu] {Bibi}
  child { node [individu=blue]{Papa}
    child { node [individu=blue]{Pépé} }
    child { node [individu=red]{Mémé} }
  }
  child { node [individu=red]{Maman}
    child { node [individu=blue]{Papy} }
    child { node [individu=red]{Mamie} }
  };
\end{tikzpicture}
```

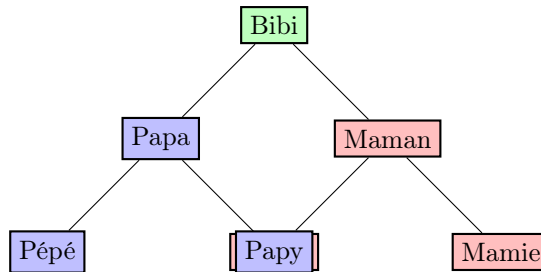


On remarque que les nœuds ont été définis avec leur place dans l'arbre, les arcs ont été automatiquement tracés. Par contre la distance entre les frères est par défaut toujours la même et au second niveau, les nœuds se superposent.

### 9.2.2 Espacement des frères : sibling distance

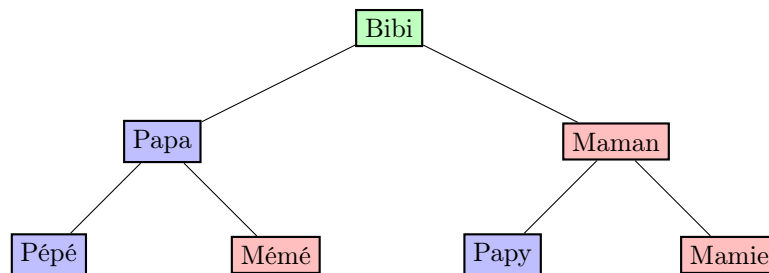
Une option de l'opération `child` permet de modifier cette distance entre frères : `sibling distance`. Par exemple à la racine :

```
\node [individu]{Bibi} [sibling distance=3cm] ...
```



On constate hélas que si la distance entre frères a augmenté, il y a toujours superposition au second niveau car l'option `sibling distance` propage son effet sur tous les sous-arbres. Il faut donc redéfinir cette option à chaque étage en faisant varier la distance selon le niveau, par exemple :

```
\node [individu] {Bibi} [sibling distance=6cm]
  node [individu=blue]{Papa} [sibling distance=3cm]
  node [individu=red]{Maman} [sibling distance=3cm]
```



Construire un arbre est toujours une tâche assez délicate. En pratique, il est toujours nécessaire de bien réfléchir pour fixer correctement la `sibling distance`, et cela, à tous les niveaux.

Heureusement, il existe des noms de style créés automatiquement lors de la construction d'un arbre et que l'on peut définir globalement, ce qui évite de le faire sur chaque fils à chaque niveau.

Le style `level` s'applique à l'arbre entier. Les styles `level 1`, `level 2`, etc. s'appliquent respectivement aux fils de niveau 1, 2, etc.

On obtient exactement le même arbre en supprimant, sur tous les nœuds où on les avait mises, les options `sibling distance`, à condition de définir globalement pour la figure les styles `level 1` et `level 2` :

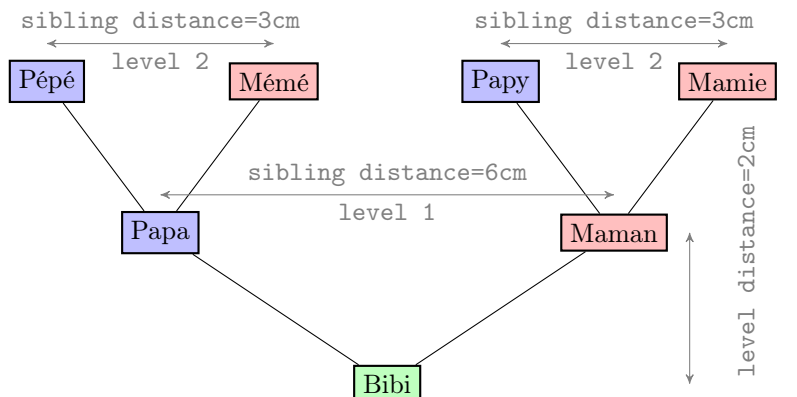
```
\begin{tikzpicture}
  [level 1/.style={sibling distance=6cm},
  level 2/.style={sibling distance=3cm}]
\node [individu] {Bibi}
  child{ node [individu=blue]{Papa}
    % etc.
  }
  child{ node [individu=red]{Maman}
    % etc.
  };
\end{tikzpicture}
```

### 9.2.3 Forme globale : level distance et grow

On peut aussi préciser la distance entre les niveaux avec l'option `level distance` et modifier la direction de croissance de l'arbre avec une des options `grow` ou `grow'`

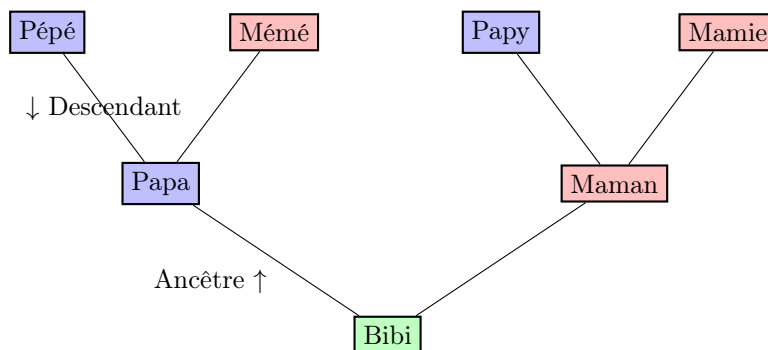
- `grow=down` : les fils sont au-dessous du père, de gauche à droite (`grow'` : de droite à gauche) (valeur par défaut)
- `grow=up` : les fils sont au-dessus du père, de droite à gauche (`grow'` : de gauche à droite)
- `grow=right` : les fils sont à droite du père, de bas en haut (`grow'` : de haut en bas)
- `grow = left` : les fils sont à gauche du père, de haut en bas (`grow'` : de bas en haut)

Par exemple : `\node{Bibi} [grow'=up,level distance=2cm,sibling distance=6cm]`



### 9.2.4 Étiquetage des arcs : edge from parent

Parfois, il peut être utile de placer des textes sur les arcs. L'opération `edge from parent`, placée à la fin de la définition d'un fils (`child`), si elle est suivie de nœuds (`node`), place les textes de ces nœuds sur l'arc reliant le fils à son parent, comme le montre la figure suivante :



Pour bien comprendre la figure obtenue ci-dessus, lire attentivement le code suivant qui a permis de placer les textes sur les arcs :

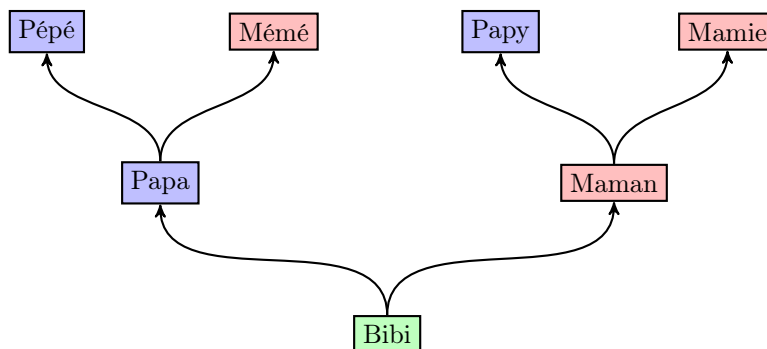
```
\node [individu] {Bibi}
child{ node [individu=blue]{Papa}
  child{ node [individu=blue]{Pépé}
    edge from parent node{ $\downarrow$  Descendant}
  }
  child{node [individu=red]{Mémé}}
  edge from parent node[below left]{Ancêtre  $\uparrow$ }
}
child{ node [individu=red]{Maman}
  child{node [individu=blue]{Papy}}
  child{node [individu=red]{Mamie}}
};
```

### 9.2.5 Style des arcs : edge from parent path

Pour finir on va montrer comment on peut modifier le tracé et le style des arcs reliant les nœuds.

L'option `edge from parent path` permet de définir l'arc comme un chemin quelconque. Pour faire référence aux nœuds origine et extrémité du chemin, on peut utiliser les commandes respectives `\tikzparentnode` et `\tikzchildnode`, par exemple on peut définir comme option de l'environnement `tikzpicture` :

```
[edge from parent path={[->,>=stealth',thick]
(\tikzparentnode) to [out=90,in=-90] (\tikzchildnode)}]
```



Les nombreuses autres opérations sont disponibles dans TikZ pour améliorer la représentation des arbres, mais elle sortent du cadre de cet ouvrage.

On conseille donc au lecteur qui voudrait dessiner des arbres complexes de se reporter au manuel de référence.

## 9.3 Liaisons entre figures : overlay

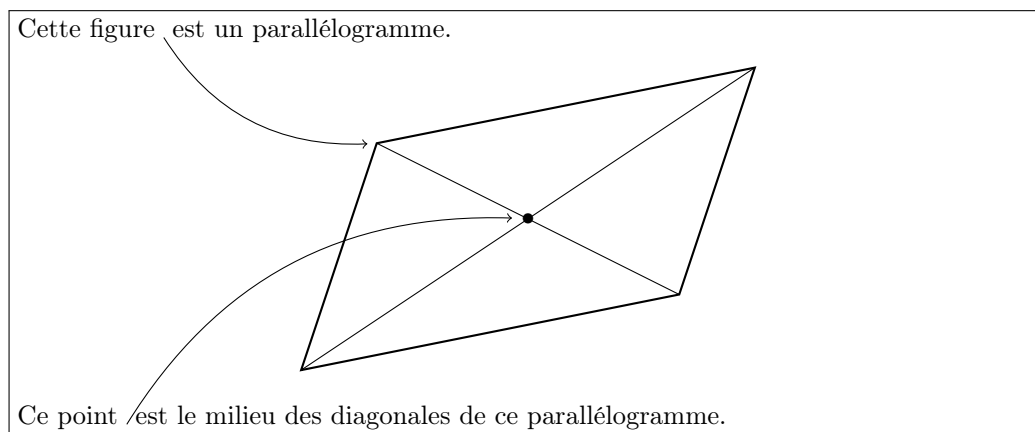
Parfois on peut avoir envie de *montrer du doigt* certains points remarquables d'une figure. Pour cela, on veut en général écrire, *en dehors de la figure*, dans le texte d'accompagnement, des commentaires relatifs à des points particuliers désignés sur la figure.

Par « en dehors de la figure » on veut dire, dans la page L<sup>A</sup>T<sub>E</sub>X, mais en dehors de l'environnement `tikzpicture` qui définit la figure. On peut aussi dessiner sur la page des liens entre deux environnements `tikzpicture` distincts définissant deux figures différentes.

Voici, dans le cadre ci-dessous, un exemple contenant du texte L<sup>A</sup>T<sub>E</sub>X et quatre figures TikZ dont deux visibles :

- la première figure est le parallélogramme ;
- la seconde est le couple de flèches.

Les deux autres figures TikZ sont invisibles, ce sont des nœuds qui repèrent simplement les origines des flèches dans le texte L<sup>A</sup>T<sub>E</sub>X.



Les extrémités des flèches sont des nœuds situés dans quatre figures TikZ différentes et nommés :

(`figure`) , le point situé entre les mots « figure » et « est » de la première ligne de texte ;  
 (`parallogramme`) , le sommet nord-ouest du parallélogramme ;  
 (`centre`) , le centre du parallélogramme ;  
 (`point`) , le point situé entre les mots « point » et « est » de la ligne de texte qui suit la figure.  
 On veut pouvoir dessiner les flèches avec les instructions suivantes :

```
\draw [->] (figure) to[bend right,thick] (parallogramme);
\draw [->] (point) to[bend left,thick,dashed] (centre);
```

Deux questions se posent :

- comment peut-on définir les nœuds (`figure`) et (`point`) ?
- comment partager des noms de nœuds entre différentes figures ?  
 (en effet, on sait que la portée d'un nom est limitée à la figure dans laquelle il est défini)

### 9.3.1 Définitions globales des noms : remember picture

Pour définir le nœud (`figure`), il faut insérer une figure TikZ entre le mot « figure » et le mot « est » de la première ligne de texte, et pour rendre la portée de ce nom globale, il faut ajouter l'option [`remember picture`].

Cette figure `\tikz[remember picture]\coordinate(figure);` est un

On procède de même pour le nœud (`point`) :

Ce point `\tikz[remember picture]\coordinate(point);` est le milieu

Ces deux figures TikZ ne dessinent rien sur les lignes de texte. C'est pour cela qu'on les a appelées *figures invisibles*.

La figure elle-même ne pose pas de problème. On a simplement défini deux points, le centre (`centre`) et le point (`parallogramme`) le sommet en haut à gauche du parallélogramme, et pour rendre la portée de ces noms globale il faut aussi ajouter l'option [`remember picture`]

```
\begin{tikzpicture} [remember picture]
  \draw (-3,-2)--(3,2) (-2,1)--(2,-1);
  \draw[thick] (-3,-2)--(2,-1)--(3,2)--(-2,1)--cycle;
  \node (centre) at (0,0){$\bullet$};
  \node (parallogramme) at (-2,1){};
\end{tikzpicture}
```

### 9.3.2 Dessiner d'une figure à l'autre : overlay

Les flèches sont ensuite dessinées par dessus la figure et le texte dans une nouvelle figure définie avec les deux options [`remember picture,overlay`]. La première, `remember picture`, indique qu'il faut utiliser les noms globaux, la seconde, `overlay`, indique que la figure sera dessinée en surimpression sur la page. Le code suivant peut être placé n'importe où, à condition que ce soit après la définition des noms utilisés :

```
\begin{tikzpicture}[remember picture,overlay]
  \draw [->] (figure) to[bend right,thick] (parallogramme);
  \draw [->] (point) to[bend left,thick,dashed] (centre);
\end{tikzpicture}
```

**Attention :** La mise en place d'une figure avec les options `overlay` et `remember picture` exige au moins deux compilations du texte source.

En général, après la première compilation, les parties définies avec l'option `overlay` sont mal placées sur la page. Il ne faut pas s'en inquiéter, le résultat satisfaisant sera obtenu après une autre compilation.



### 9.3.3 La page courante est un nœud : `current page`

TikZ fournit un nom de nœud spécial `current page` qui donne accès à la page courante sous la forme d'un rectangle qui représente la page complète. C'est un rectangle dont chaque sommet a pour coordonnées un des coins de la page. On peut donc, à partir de ce nœud, définir n'importe quelle position absolue sur la page courante.

On peut par exemple écrire TikZ au centre de la page avec le code suivant :

```
\begin{tikzpicture}[remember picture,overlay]
  \node [rotate=60,scale=10,text opacity=0.3]
    at (current page.center) {Ti\textit{\color{orange}k}Z};
\end{tikzpicture}
```

**Attention :** Il est parfois difficile de comprendre quelle est la page courante, car la mise en page est faite automatiquement par L<sup>A</sup>T<sub>E</sub>X.

Si le dessin ne se situe pas dans la page prévue, on peut essayer de déplacer le code de la figure dans le texte source, ou forcer un saut de page à l'aide de la commande `\newpage`, et compiler à nouveau deux fois.

## 9.4 Résumé

On a ajouté ici des précisions sur l'utilisation de l'environnement `scope` qui permet d'insérer une sous-figure dans une figure. L'effet des options `xshift`, `yshift`, `shift`, `scale` et `rotate` sur l'épaisseur des traits et sur les annotations textuelles a été étudié en détail.

On a présenté, sur un exemple simple, quelques uns des outils de base que TikZ fournit spécifiquement pour la construction de graphes de type arbre.

On a aussi montré comment il est possible d'écrire n'importe où sur une page et créer des liens entre différentes figures définies et le texte du document, à l'aide des options `remember picture` et `overlay`.



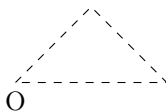
# Annexe A

## La syntaxe de TikZ

La syntaxe de TikZ admet de nombreuses constructions spécialisées, avec parfois des raccourcis et des exceptions, et il peut être difficile de s'y retrouver. Pour cela il peut être bon de classer les formes syntaxiques en grandes catégories : environnements, commandes, options, opérations de chemin, coordonnées.

Toutes ces catégories se retrouvent dans l'exemple suivant, qui trace un triangle en pointillés :

```
\begin{tikzpicture}
\draw [dashed] (0,0) node[below]{O} -- (1,1) -- (2,0) -- cycle;
\end{tikzpicture}
```



`{tikzpicture}` est un environnement, `\draw` est une commande, `[dashed]` est une option, `(0,0)` est un couple de coordonnées, `node` est une opération de chemin avec une option `[below]` et un argument `{O}`, `--` est une opération de chemin avec un argument `(1,1)`, `-- cycle` est aussi une opération de chemin.

### A.1 Les environnements : `{tikzpicture}`, `{scope}`

TikZ ne définit que deux environnements au sens de L<sup>A</sup>T<sub>E</sub>X : `{tikzpicture}` et `{scope}`. Le premier est utilisé chaque fois qu'on veut insérer une figure TikZ dans un document L<sup>A</sup>T<sub>E</sub>X, le deuxième peut être utilisé à l'intérieur du précédent pour définir un contexte ayant ses propres options graphiques.

### A.2 Les commandes

Ce sont des commandes au sens de L<sup>A</sup>T<sub>E</sub>X, qui se reconnaissent à ce qu'elles commencent par le caractère `\`.

#### Commandes utilisables dans tout le document

Certaines commandes peuvent être utilisées dans le document L<sup>A</sup>T<sub>E</sub>X en dehors de l'environnement `{tikzpicture}`.

Ce sont :

```
\usetikzlibrary
```

Pour charger, au début, dans le préambule du document, certaines bibliothèques TikZ utiles comme : `arrows`, `patterns` ou `calc`.

Par exemple : `\usetikzlibrary{patterns, arrows}`

**\tikzset**

Pour définir des options globalement dans tout le document.

Par exemple : `\tikzset{rougegris/.style={draw=gray,fill=red}}`

Appliquée à un nœud, cette option encadrera le nœud en gris et coloriera l'intérieur en rouge. Par exemple : `node[rougegris]{texte}`.

La commande `\tikzset` peut aussi être à l'intérieur d'un environnement `{tikzpicture}`. Son effet sera alors limité à cet environnement.

**\tikzstyle**

C'est l'ancienne forme de la commande `\tikzset`. Elle reste toujours utilisable, sous la forme `\tikzstyle{rougegris}={draw=gray,fill=red}`

**\tikz**

C'est l'équivalent de l'environnement `{tikzpicture}` pour les figures dont le code est court (un paragraphe au plus).

Par exemple : `\tikz \draw (0,0) circle (1);` pour tracer un cercle centré à l'origine et de rayon 1.

**\foreach**

C'est une instruction de programmation rendue disponible dans tout le document dès le chargement de TikZ. Elle permet d'exécuter des instructions de manière répétée (au sens informatique, c'est une *boucle*).

Par exemple : `\foreach \x in {1,2,3} {$x=\x$, }` donne :

$x = 1, x = 2, x = 3,$

La commande peut être utilisée aussi à l'intérieur d'un environnement `{tikzpicture}`.

Par exemple, pour tracer cinq cercles concentriques :

`\tikz \foreach \r in {1,2,...,5} \draw (0,0) circle (\r mm);`



## Commandes dans l'environnement `{tikzpicture}`

Les autres commandes ne peuvent être utilisées que dans un environnement `{tikzpicture}`. Les principales sont :

`\clip, \coordinate, \draw, \fill, \filldraw, \node, \path`

## A.3 Les coordonnées

### A.3.1 Forme générale : (...)

Syntaxiquement, la position d'un point est indiquée par une expression de coordonnées, comme  $(x,y)$  (coordonnées cartésiennes) ou  $(a:r)$  (coordonnées polaires) ou  $(nom)$  (point nommé), ou  $(nom.ancree)$  (ancree d'un noeud, comme  $(n.north)$ ), ou  $++(x,y)$  (coordonnées relatives à la position du crayon).

Il y a d'autres formes, mais d'une manière générale, une expression de coordonnées s'écrit entre parenthèses.

### A.3.2 Calculs sur les nombres : package pgfmath

Depuis la version 2 de TikZ, il est possible d'écrire des calculs portant sur les nombres grâce au package `pgfmath`, qui est chargé automatiquement avec TikZ. Les formules de calcul disponibles sont celles qu'on peut utiliser pour le tracé de courbes avec l'opération `plot`.

Dans une expression de coordonnées, on peut utiliser ces calculs :

$(\{\sqrt{3}/2\}, 1/2)$  désigne le point de coordonnées  $\left(\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$ .

Comme on le voit, si le calcul utilise des parenthèses, il faut l'encadrer par des accolades.

### A.3.3 Calculs sur les coordonnées : bibliothèque calc

Une autre sorte de calcul est rendue possible avec le chargement de la bibliothèque `calc`. Contrairement au package `\pgfmath`, dont le chargement est automatique et implicite, le chargement de `calc` doit être explicite, avec `\usetikzlibrary{calc}`. Nous n'avons pas parlé dans cet ouvrage des possibilités offertes par `calc`. Se référer au manuel officiel : 12.4 *Coordinate Calculations* page 113.

Avec cette bibliothèque, il est possible de faire des calculs portant directement sur les *couples* de coordonnées (par exemple faire la somme de deux couples, faire le produit d'un nombre par un couple). Mais cela introduit une syntaxe spéciale : les calculs de ce type doivent être écrits entre les symboles (`$` et `$`), comme `$(1,2) + (3,4)$`.

En plus de ces calculs avec des opérations mathématiques, cette syntaxe permet d'effectuer de manière interne les calculs liés à des constructions géométriques :

- Point de tangence de la tangente menée d'un point à un cercle ;
- Barycentre de plusieurs points, combinaisons linéaires ;
- Point d'intersection de deux segments ;
- Image d'un point par une similitude directe dont on donne le centre, le rapport et l'angle ;
- Projection orthogonale d'un point sur un segment ;
- Point à une distance donnée d'un autre sur un segment.

Ces possibilités peuvent être intéressantes pour éviter d'avoir à calculer par soi-même des coordonnées. En ceci, elles rapprochent TikZ d'un logiciel de construction géométrique comme GeoGebra.

Cependant, la syntaxe de ces instructions est souvent un peu ésotérique et manque d'unité. Voici un exemple tiré du manuel : `$(1,1)!.5!60:(2,2)$` désigne l'image du point  $(2,2)$  par la similitude directe de centre  $(1,1)$ , de rapport 0.5 et d'angle  $60^\circ$ .

Si le seul but est d'éviter les calculs, on peut aussi le faire en utilisant GeoGebra ou un logiciel de calcul formel comme logiciel auxiliaire. Mais cela fige le résultat des calculs et il faut tout recommencer dès qu'on veut changer une coordonnée quelque part.

## A.4 Les opérations de chemin

Ce sont les opérations qu'on peut écrire dans les commandes qui définissent des chemins ou qui agissent sur des chemins, comme dans

```
\draw, \fill, \filldraw, \path, \clip.
```

### Syntaxe *opération argument*

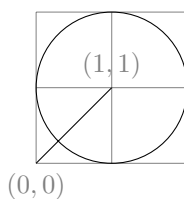
La plupart des opérations ont une syntaxe de la forme *opération argument*, où *opération* est le nom de l'opération et *argument* est l'argument. La syntaxe d'une opération dépend, pour l'argument, de cette opération. La plupart des arguments sont simples (un couple de coordonnées ou un nom de nœud), excepté pour `plot` et `child` (cette dernière étant récursive).

La seule opération qui n'a pas de nom est l'opération de positionnement explicite, qu'on utilise au début d'un chemin pour fixer la position du crayon : elle a seulement un argument, qui est la position en question.

Dans l'exemple suivant, les expressions désignant des opérations ont été encadrées :

```
\draw (0,0) -- (1,1) circle (1);
```

La première est l'opération de positionnement explicite du crayon en  $(0,0)$ , la deuxième celle de tracé d'un segment, la troisième celle de tracé d'un cercle. Après chaque opération, la position du crayon est réactualisée.



Cette opération de positionnement peut être utilisée plusieurs fois dans un même chemin. Dans ce cas-là, le chemin peut être en différents morceaux non connectés. Par exemple :

```
\draw (0,0) -- (1,1) (2,2) -- (3,3);
```

Il y a deux positionnements explicites (0,0) et (2,2). Le chemin est formé de deux segments non connectés. Cette possibilité est à utiliser avec prudence, elle rend le code plus difficile à interpréter. Il vaudrait mieux écrire :

```
\draw (0,0) -- (1,1);
\draw (2,2) -- (3,3);
```

C'est un peu plus long à écrire, mais plus clair et plus modulaire. Cependant, un avantage de regrouper les opérations en un seul chemin est de pouvoir factoriser les options graphiques en un seul endroit, dans la commande `\draw`.

## Principales opérations

Les principaux symboles ou noms d'opérations sont :

```
--, -|, |- , --cycle
rectangle, circle, arc, grid, plot, --plot, to
```

Les opérations se rapportant aux nœuds sont spéciales :

```
node, edge, child
```

On écrit ces dernières à l'intérieur d'un chemin, mais ce qu'elles définissent n'est pas considéré comme un élément du chemin. Cette distinction est importante pour l'ordre du tracé (les nœuds sont tracés après le reste), les remplissages avec `fill` et la fermeture des régions avec `--cycle` (les nœuds ne font pas partie de la région à remplir).

Il existe quelques autres opérations dont nous n'avons pas ou peu parlé :

```
..controls and .., ellipse, parabola, sin, cos, let
```

## A.5 Les options

### Les options explicites : [...]

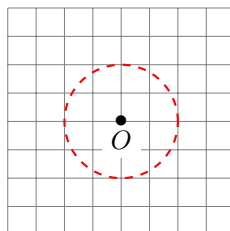
Syntaxiquement, on écrit les options entre crochets, séparées par des virgules. La spécification d'une option peut prendre plusieurs formes, dont les principales sont `[clé]` et `[clé=valeur]`.

Elles peuvent s'appliquer à tout un environnement `{tikzpicture}`, à un environnement `{scope}`, à un chemin utilisé par `\draw`, `\fill`, `\clip`.

`\filldraw`, ou `\path`, et à la plupart des opérations de chemin, par exemple `node`, `to` ou `grid`.

Exemple :

```
\begin{tikzpicture} [scale=0.75]
\draw [gray, very thin] (-2,-2) grid [step=0.5cm] (2,2);
\draw (0,0) node [below, fill=white] {$0$} node{$\bullet$};
\draw [dashed,red,thick] (0,0) circle(1);
\end{tikzpicture}
```



## Les options implicites : raccourcis

Le mécanisme qui interprète les options est défini par le package `pgfkeys`, chargé automatiquement avec `TikZ`, et qui peut être utilisé en dehors de `TikZ`. Ce mécanisme comporte des possibilités d'abstraction, d'abréviations et de valeurs par défaut qui sont souvent pratiques, mais qui peuvent sembler parfois mystérieuses et entraîner des confusions, d'autant plus que certains raccourcis peuvent aussi être obtenus à l'aide de commandes qui cachent le mécanisme d'options.

### Exemple : `coordinate`

Un bon exemple est donné par le mot-clé `coordinate` permettant d'attribuer un nom symbolique à un point ou un nœud.

On le rencontre sous différentes formes : comme commande, comme opération de chemin, comme clé d'option, comme valeur d'option.

Voici différents exemples ayant pour effet de nommer (o) l'origine du repère :

- commande : `\coordinate (o) at (0,0);`
- opération de chemin : `\path (0,0) coordinate (o);`
- clé d'option : `\path (0,0) node [coordinate] (o);`
- valeur d'option : `\path (0,0) node [shape=coordinate] (o);`

En fait, le « vrai » concept de `coordinate` est le dernier : c'est une valeur possible de l'option `shape` pour un nœud. Dire que `shape` a pour valeur `coordinate` signifie que le nœud n'a pas d'étendue, et donc qu'il peut être vu comme un simple point, et donc qu'il peut servir à repérer les coordonnées de ce point.

Mais penser à ce concept en ces termes est à un niveau de détail technique un peu compliqué pour l'usage habituel, qui consiste seulement à donner un nom à un point. C'est pour cela que le langage offre les autres possibilités, qui se ramènent toutes de manière interne à dire qu'il y a un nœud (parfois implicite) dont l'option `shape` a pour valeur `coordinate`.

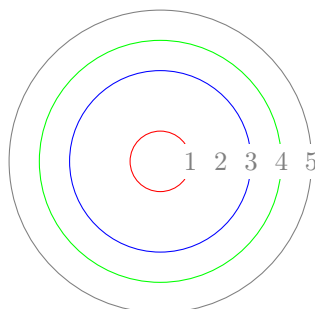
### Exemple : `draw`

Pour tracer un chemin, on utilise habituellement la commande `\draw`. Mais ce n'est qu'un raccourci pour dire que l'option `draw` du chemin a pour valeur une certaine couleur (la couleur actuellement en vigueur, c'est-à-dire la valeur de l'option `color`).

Autrement dit, le mot-clé `draw` indique une option de *couleur* plutôt que réellement une instruction demandant de dessiner. Mais, dans le mécanisme interne, le fait de spécifier cette couleur a pour conséquence de rendre le tracé effectif.

Exemple :

```
\begin{tikzpicture} [scale=0.4]
\draw [red] (0,0) circle (1) ;           % 1 trace en rouge
\path [color=blue] (0,0) circle (2);    % 2 ne trace pas
\path [color=blue,draw] (0,0) circle (3); % 3 trace en bleu
\path [color=blue,draw=green] (0,0) circle (4); % 4 trace en vert
\path [draw=gray] (0,0) circle (5);    % 5 trace en gris
\end{tikzpicture}
```



On constate à cette occasion que la commande de base pour un chemin est `\path`, qui se contente de définir un chemin, mais ne le trace pas forcément. Les autres commandes de chemin, comme `\clip`, `\draw`, `\fill`, ne sont en fait que des raccourcis qui se ramènent de manière interne à la commande `\path`, en affectant certaines options.

## A.6 Utiliser des commandes $\text{\LaTeX}$ dans TikZ

TikZ est une extension de  $\text{\LaTeX}$ . Bien qu'il définisse ses propres règles syntaxiques, il peut quand même bénéficier de certains mécanismes de  $\text{\LaTeX}$ , en particulier l'utilisation de commandes vues comme des macros de remplacement de texte.

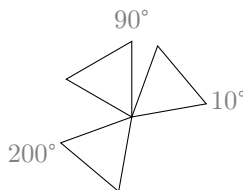
Le manuel ne précise pas exactement ce qui est possible ou pas, mais l'expérience et les exemples montrent qu'il est possible de définir ses propres commandes comme abréviations pour engendrer des parties du texte source.

Par exemple, on peut définir une commande qui trace un triangle équilatéral de côté 1 dont un sommet est  $(0,0)$  et dont un côté est dirigé suivant l'angle polaire donné en paramètre #1. On utilise l'option `rotate` pour tourner le triangle dans la direction voulue.

```
\newcommand{\triangleE}[1]
{\draw[rotate=#1] (0,0) -- (1,0) -- (1/2, {sqrt(3)/2}) -- cycle;}
```

Si on a plusieurs triangles de ce type à tracer, cela évite d'avoir à écrire à chaque fois le code détaillé :

```
\begin{tikzpicture}
\triangleE{10}
\triangleE{90}
\triangleE{200}
\end{tikzpicture}
```



On voit donc qu'à l'intérieur d'un environnement `{tikzpicture}`, il est possible d'utiliser une commande définie ailleurs. Cette commande sera développée sous forme de code source TikZ et sera ensuite interprétée par TikZ.



## Annexe B

# Erreur ! Que faire ?

Soyons optimistes, on suppose ici que le lecteur sait déjà comment corriger les erreurs qui peuvent survenir lors de la rédaction d'un document L<sup>A</sup>T<sub>E</sub>X.

On signale simplement ici les principales sources d'erreurs typiques de TikZ.

Lors d'une erreur de compilation, il faut d'abord consulter les dernières lignes affichées sur la *console*. On appelle *console* la fenêtre dans laquelle sont affichés les messages de compilation et les messages d'erreur.

On va signaler ici les messages les plus fréquents provoqués par les erreurs dans TikZ et qui doivent attirer l'attention.

On va expliquer leur origine et comment corriger les erreurs.

### Oubli du « ; »

Si on voit le message suivant :

```
! Package tikz Error; Giving up on this path.  
Did you forget a semicolon?.
```

C'est l'erreur la plus fréquente quand on commence à utiliser TikZ. Toute commande TikZ se termine par un point-virgule que l'on oublie souvent.

La correction est facile.

### Les nombres trop grands

Si on voit le message suivant :

```
! Dimension too large.  
<recently read> \pgf@xx
```

Ce message est suivi du numéro d'une ligne contenant en général des coordonnées trop grandes. En effet, dans TikZ, il vaut mieux limiter les coordonnées des points à des valeurs comprises entre -500 et +500.

Il suffit donc de concevoir le dessin de façon à maintenir les coordonnées de tous les points de la figure dans un intervalle plus limité.

### Le « ! » dans la définition des couleurs

Si on voit le message suivant :

```
! Missing \endcsname inserted.  
<to be read again>  
\penalty
```

Ce message assez mystérieux est suivi du numéro d'une ligne qui contient en général une option de couleur comme par exemple : `\draw[red!50]...`

Le responsable de l'erreur est ici le « ! » et le raccourci dans la définition de la couleur du tracé.

On corrige l'erreur en évitant le raccourci : `\draw[color=red!50]...`

## Le problème de babel français et de « : »

**Remarque :** Dans la dernière version de TikZ & PGF (la version 2.10) qui est actuellement fournie avec la distribution T<sub>E</sub>Xlive 2011, ce problème de `babel` et du « : » est corrigé.

Si on dispose d'une version plus ancienne et si on voit le message suivant :

```
! Paragraph ended before \tikz@plot@samples@recalc was complete.
<to be read again>
      \par
```

ou le message suivant :

```
! File ended while scanning use of \tikz@plot@samples@recalc.
< inserted text >
      \par
```

Ces messages assez mystérieux sont suivis du numéro d'une ligne contenant en général une commande `\plot` avec une option `[domain=a:b]`. C'est le deux-points qui est responsable de l'erreur.

Le plus simple, pour corriger cette erreur, est d'ajouter la commande `\shorthandoff{:}` dès le début de l'environnement `{tikzpicture}`

Mais on peut aussi voir le message suivant :

```
! Argument of \tikz@plot@samples@recalc has an extra }.
<inserted text>
      \par
```

Ceci peut se produire si l'environnement `{tikzpicture}` est inséré dans une commande `\fbox`. Dans ce cas la commande `\shorthandoff{:}` doit être placée à l'extérieur de la commande `\fbox`, ainsi :

```
{\shorthandoff{:}
\fbox{\begin{tikzpicture}
  \draw plot[domain=-2:5] (\x,\x/5);
\end{tikzpicture}}}
```

Ce problème d'incompatibilité des packages `tikz` et `babel` français est étudié en détail dans le chapitre « Courbes ».

Un package récent, `microtype`, permet de résoudre le problème globalement de façon simple.

Il suffit d'ajouter :

```
\usepackage[babel=true,kerning=true]{microtype} après la déclaration d'utilisation du pa-
ckage babel.
```

Ainsi le caractère « : » ne semble plus poser de problème.

**Remarque :** Dans la dernière version de TikZ & PGF (la version 2.10) qui est actuellement fournie avec la distribution T<sub>E</sub>Xlive 2011, ce problème de `babel` et du « : » est corrigé.

## Annexe C

# Où trouver de l'aide ?

Un site Internet, créé par les auteurs, accompagne ce livre. On y trouvera le code de tous les exemples présentés ici, plus quelques compléments.

À partir du site, il est aussi possible de contacter les auteurs : Toutes les questions, toutes les critiques et toutes les suggestions sont les bienvenues.

<http://math.et.info.free.fr/TikZ/index.html>

### Livres à lire

On trouve sur Internet la documentation de l'auteur Till Tantau :

[Version la plus récente de « TikZ & PGF Manual »](#)

Ce document au format PDF de 560 pages est une référence très complète, mais reste d'un abord un peu difficile...

Si vous êtes plutôt débutant en  $\text{\LaTeX}$ , nous vous conseillons en particulier chez l'éditeur [H&K](#) : «  [\$\text{\LaTeX}\$  pour l' impatient »](#). La lecture préalable de cet ouvrage remarquable, simple, mais cependant très complet vous sera très utile :

[http://www.h-k.fr/liens/tp/latex\\_pour\\_impatient.htm](http://www.h-k.fr/liens/tp/latex_pour_impatient.htm)

### Autres liens vers Internet

#### En anglais

L'auteur de TikZ est [Till Tantau](#) qui a aussi créé pour  $\text{\LaTeX}$  le package Beamer qui permet de réaliser des présentations sous forme de diaporama.

<http://www.tcs.uni-luebeck.de/mitarbeiter/tantau/>

On trouvera sur le site [TeXample.net](#) des exemples proposés par l'auteur et concernant de très nombreux domaines :

<http://www.texample.net/tikz/examples/author/till-tantau/>

#### En français

Allez sur le site [Altermundus](#) : <http://www.altermundus.fr/>



# Annexe D

## Glossaire

Ce glossaire contient la liste alphabétique des commandes, opérations et options du langage *TikZ* utilisées dans ce livre et brièvement commentées.

Pour chaque item il y a une référence à la documentation de Till Tantau dans laquelle cet item est défini.

Cette référence est donnée sous la forme : Titre de la section, numéro de la section et page.

[Version la plus récente de « TikZ & PGF Manual »](#)

### Conventions d'écriture

Dans le langage de *TikZ*, le même mot peut souvent représenter des éléments syntaxiques distincts. Pour aider à comprendre les nombreux raccourcis et les variations syntaxiques on notera dans ce glossaire les éléments différents ainsi :

- les commandes préfixées par une contre-oblique, ainsi : `\draw` ;
- les coordonnées ou les noms entre parenthèses, ainsi : `(x,y)` `(a:r)` (P) ;
- les options entre crochets, ainsi : `[above]` ;
- les environnements entre accolades, ainsi : `{scope}` ;
- les opérations de chemin et autres mots clés, ainsi : `arc west` ;
- les concepts présentés dans ce livre, ainsi : *chemin*.

### Liste alphabétique

-- (The Line-To Operation 13.2 page 118)

-- est le symbole de l'opération qui trace un segment sous la forme :

`\draw (a,b) -- (c,d)` ; du point (a,b) au point (c,d)

On peut aussi obtenir un tracé en lignes horizontales et verticales avec `|-` ou `-|` ou des lignes courbes avec `to` ou `.. controls and ..`

[->] (Graphic Parameters : Arrow Tips 14.3.4 page 134)

C'est une option graphique applicable à un trait ou un chemin, pour dessiner une pointe de flèche placée à l'extrémité du trait.

Exemple : `\draw [->] (a,b) -- (c,d)` ; Différents styles de pointe de flèche sont possibles, comme par exemple `|->`, `*-o`, `|<->`, `[-(` ou `->>`. On peut changer aussi le dessin des pointes avec l'option `>=` suivie de `latex` ou `stealth`.

(x,y), (x,y,z), (a:r), (P) (Specifying coordinates 12 page 103)

*TikZ* propose plusieurs façons d'exprimer les coordonnées des points :

- (x,y) coordonnées cartésiennes à 2 dimensions ;
- (x,y,y) coordonnées cartésiennes à 3 dimensions ;
- (a:r) coordonnées polaires angle et distance au centre ;

- (P) coordonnées du nœud dont le nom (P) a été défini par exemple avec :  
`\draw (x,y) node(P){Texte};` ou `\coordinate (P) at (x,y);`
- `++(x,y)` (Relative and Incremental Coordinates 12.3 page 112)  
`++` est le symbole qui précède des coordonnées pour indiquer que celles-ci doivent être considérées comme relatives au point précédent.  
`\draw (4,3) -- ++(1,3);` équivalent à `\draw (4,3) -- (5,6);`
- `[above]`, `[above left]`, `[above right]` (Positioning Nodes 15.5 page 154)  
Options de placement des nœuds. (Voir : `node`)
- arbre* (Making Trees Grow 17 page 183)  
(Voir : `child`)
- `arc` (The Arc Operation 13.8 page 121)  
`arc` est l'opération de chemin qui trace un arc sous la forme :  
`\draw (x,y) arc (d:f:r);` à partir du point (x,y) un arc de rayon r dont le rayon origine forme un angle de d° avec l'horizontale et le rayon extrémité à un angle de f° avec l'horizontale.
- `[below]`, `[below left]`, `[below right]` (Positioning Nodes 15.5 page 154)  
Options de placement des nœuds. (Voir : `node`)
- `[bend left]`, `[bend right]` (Curves 40.2 page 367)  
Options pour fixer la courbure d'un ligne. (Voir : `to`)
- `[bottom color=]`, `[top color=]` (Choosing a Shading Color 14.5.2 page 140)  
Options pour fixer les couleurs de dégradés. (Voir : `\shade`)
- `[text centered]` (Options for the Text in Nodes 15.4 page 152)  
Option de mise en forme de texte. (Voir : `node`)
- chemin* (Actions on Paths 14 page 130)  
(Voir : `\path` ou `\draw`)
- `child` (Making Trees Grow 17 page 183)  
`child` est une opération placée à la suite d'une opération `node` qui permet de définir un fils du nœud dans une structure d'arbre.  
De nombreuses options sont disponibles pour contrôler les dimensions de l'arbre comme `sibling distance` ou `level distance` ou la direction dans laquelle il se développe comme `grow`.  
On peut aussi étiqueter les nœuds à l'aide de `edge from parent` et dessiner les arcs entre les nœuds avec `edge from parent path`.
- `circle` (The Circle and Ellipse Operations 13.7 page 121)  
`circle` est l'opération de chemin qui trace un cercle sous la forme :  
`\draw (x,y) circle (r);` de centre (x,y) et de rayon (r).  
On peut aussi tracer une ellipse en donnant les rayons sur les deux axes sous la forme  
`\draw (x,y) circle (r and s);`
- `[circle]` (Nodes and Their Shapes 15.2 page 147)  
Abréviation de `[shape=circle]`, fixe la forme d'un nœud. (Voir : `node`)
- `\clip` (Clipping and Fading : Soft Clipping 14.7 page 142)  
`\clip...` est la commande qui restreint la région d'affichage. Tous les dessins qui suivent cette commande seront limités à cette région.

- `[color=]` (Specifying a Color 14.2 page 131)  
Option pour définir la couleur. (Voir : `\draw`, `\fill`, `node` ou `{scope}`)
- `.. controls and ..` (The Curve-To Operation 13.3 page 119)  
(A) `..controls (U) and (V) .. (B)` trace une courbe de Bézier entre le point (A) et le point (B), les tangentes en (A) et (B) étant respectivement déterminées par les points de contrôle (U) et (V).
- `[coordinate]` (Nodes and Their Shapes 15.2 page 146)  
Abréviation de `[shape=coordinate]` qui fixe la forme d'un nœud.  
C'est une forme de nœud spéciale, de dimension nulle, qui permet de considérer le nœud comme un simple point et de le nommer pour un usage ultérieur. (Voir : `node`)
- `\coordinate` (Nodes and Their Shapes 15.2 page 146)  
`\coordinate (A) at (x,y)`; est la forme abrégée de la commande qui permet de nommer et de placer un nœud de dimension nulle :  
`\path (x,y) coordinate (A);` ou  
`\draw (x,y) node[coordinate] (A){}`; ou  
`\node (A) at (x,y) node[shape=coordinate]{};`
- `--cycle` (The Cycle Operation 13.4 page 119)  
`--cycle;` est l'opération de fin de chemin qui consiste à relier le dernier point au premier.
- (`current page`) (Referencing the Current Page Node 15.13.2 page 170)  
Nœud spécial qui représente la page entière. (Voir : `[overlay]`)
- `[dashed]` (Graphic Parameters : Dash Pattern 14.3.2 page 134)  
Option de style pointillé en tirets. (Voir : `\draw`)
- `[diamond]` (Nodes and Their Shapes 15.2 page 147)  
Abréviation de `[shape=diamond]`, fixe la forme d'un nœud. (Voir : `node`)
- `[domain=]` (Plotting a Function 18.5 page 194)  
Option pour le domaine de définition d'une fonction. (Voir : `plot`)
- `[dotted]` (Graphic Parameters : Dash Pattern 14.3.2 page 134)  
Option de style pointillé. (Voir : `\draw`)
- `[double]` (Graphic Parameters : Double Lines 14.3.5 page 136)  
Option de style de trait double. (Voir : `\draw`)
- `\draw` (Drawing a Path 14.3 page 131)  
`\draw...` est la commande qui permet de tracer un chemin.  
De nombreuses options permettent de préciser le mode de tracé. On peut fixer l'épaisseur des traits avec `thin`, `thick`, `line width=`, le type de trait avec `dashed`, `dotted`, `double` etc.
- `east` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Option d'ancrage `[anchor=east]` d'un nœud. (Voir : `node`)
- `.east` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Pour la référence (P.`east`) point d'ancrage à un nœud. (Voir : `node`)
- `edge from parent` (Edges From the Parent Node 17.6 page 191)  
Opération d'accès au parent d'un nœud dans un arbre. (Voir : `child`)

- `edge from parent path`, (Edges From the Parent Node 17.6 page 191)  
Style pour définir les arcs dans un arbre. (Voir : `child`)
- `[ellipse]` (Nodes and Their Shapes 15.2 page 147)  
Abréviation de `[shape=ellipse]`, fixe la forme d'un nœud. (Voir : `node`)
- `[even odd rule]` (Interior Rules 14.4.2 page 138)  
Règle de définition de région. (Voir : `\clip` ou `\fill`)
- figure** (Creating a Picture 11.2 page 96)  
(Voir : `{tikzpicture}` `\tikz`)
- `\fill` (Filling a Path 14.4 page 136)  
`\fill...` est la commande qui colorie la région limitée par un chemin.  
La couleur du coloriage peut être précisée avec par exemple :  
`\draw[fill=red]...` ou aussi `\fill[color=red]...` On peut aussi obtenir des effets de transparence avec l'option `opacity`.  
Il existe deux règles pour définir le mode de coloriage des régions limitées par un chemin : `nonzero rule`, par défaut ou `even odd rule`
- `\filldraw` (Drawing a Path 14.3 page 131) (Filling a Path 14.4 page 136)  
`\filldraw..` est la forme abrégée de la commande `\path[fill,draw]..` qui permet de tracer le chemin et colorier la région. (Voir : `\fill` et `\draw`)
- `\foreach` (The Foreach Statement 44 page 389)  
La commande `\foreach` s'utilise sous la forme :  
`\foreach <variables> in <liste> <commandes>`  
— `<variables>` est une ou plusieurs commandes L<sup>A</sup>T<sub>E</sub>X séparées par des barres. Exemples : `\x` ou `\x/\y/\z`  
— `<liste>` est une liste de valeurs séparées par des virgules :  
Exemples : `{2, texte, 0.5}` ou avec 3 variables `{1/un/I, 5/cinq/V}`  
— `<commandes>` sont des instructions qui contiennent la ou les variables et qui seront exécutées de façon répétitive en donnant aux variables successivement les valeurs fournies dans la liste.
- grid** (The Grid Operation 13.9 page 121)  
`grid` est l'opération de chemin consistant à tracer un quadrillage (grille).  
La forme d'utilisation la plus simple est `\draw (A) grid (B)`; qui trace une grille dont (A) et (B) sont deux sommets d'une diagonale.  
Le pas de la grille peut être fixé avec les options `step`, `xstep` ou `ystep`
- `[grow=]` (Default Growth Function 17.5.2 page 187)  
Option pour fixer la direction de croissance d'un arbre. (Voir : `child`)
- `[in=]`, `[out=]` (Curves 40.2 page 367)  
Options pour fixer la courbure. (Voir : `to`)
- `[text justified]` (Options for the Text in Nodes 15.4 page 152)  
Option de mise en forme de texte. (Voir : `node`)
- `latex` (Arrow Tip Library 22 page 224)  
Style de pointe de flèche, `[>=latex]`. (Voir : `->`)
- `[left]`, `[right]` (Positioning Nodes 15.5 page 154)  
Options de placement des nœuds. (Voir : `node`)



- [level distance=] (Default Growth Function 17.5.2 page 187)  
Option pour fixer l'écart entre les niveaux dans un arbre. (Voir : `child`)
- [line width=] (Graphic Parameters : Line Width 14.3.1 page 132)  
Option pour fixer l'épaisseur du trait. (Voir : `\draw`)
- [mark=] (Placing Marks on the Plot 18.7 page 198)  
Option pour marquer les points d'une courbe avec `*`, `+`, `x`. (Voir : `plot`)
- [midway] (Placing Nodes on a Line or Curve Explicitly 15.8 page 161)  
Option pour placer des nœuds sur une ligne. (Voir : `--` ou `to`)
- [minimum size] (Creating Nodes 59.2 page 474)  
Ou aussi [minimum width] et [minimum height]  
Option pour fixer les dimensions d'un nœud. (Voir : `node`)
- [near end] (Placing Nodes on a Line or Curve Explicitly 15.8 page 161)  
Option pour placer des nœuds sur une ligne. (Voir : `--` ou `to`)
- [near start] (Placing Nodes on a Line or Curve Explicitly 15.8 page 161)  
Option pour placer des nœuds sur une ligne. (Voir : `--` ou `to`)
- nœud* (Nodes and Edges 15 page 146)  
(Voir : `node` ou `\node`)
- `node` (Nodes and Edges 15 page 146)  
`node` est une opération de chemin qui écrit un texte  $\LaTeX$ , placé par défaut dans une boîte rectangulaire centrée à la position du crayon.  
Son utilisation de base est `\draw (a,b) node {texte};`  
De nombreuses options sont possibles, pour contrôler différents aspects :  
— la position : `above`, `below`, `left`, `right`, `above left`, etc.  
— La couleur du texte : `node [red] {texte}`  
— La couleur du fond : `node [fill=red] {texte}`  
— la forme : `circle`, `rectangle`, `coordinate`
- `\node` (Nodes and Their Shapes 15.2 page 146)  
`\node (A) at (x,y){Texte};` est la forme abrégée de la commande :  
`\path (x,y) node (A){Texte};`
- [nonzero rule] (Interior Rules 14.4.2 page 138)  
Règle de définition de région. (Voir : `\clip` ou `\fill`)
- `north` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Option d'ancrage [`anchor=north`] d'un nœud. (Voir : `node`)  
Autres valeurs, [`anchor=north west`], [`anchor=north east`]
- `.north` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Pour la référence (`P.north`) point d'ancrage à un nœud. (Voir : `node`)  
Autres valeurs, (`P.north west`), (`P.north east`)
- `\newcommand` définition d'une commande. (Voir :  $\LaTeX$ )  
On peut aussi utiliser `\def` en  $\TeX$ .
- [opacity=] (Transparency 19 page 202)  
Option pour définir le niveau de transparence. (Voir : `\fill`)

- [out=], [in=] (Curves 40.2 page 367)  
Options pour fixer la courbure d'une ligne. (Voir : to)
- [overlay] (Referencing Nodes Outside the Current Pictures 15.13 page 169)  
L'option `overlay` associée à l'option `remember picture` permet de tracer, sur une même page, des chemins entre des figures définies dans des environnements `{tikzpicture}` distincts.  
Il est aussi possible de dessiner n'importe où sur la page en utilisant le nœud spécial (`current page`) qui représente le rectangle de la feuille.
- `\path` (Actions on Paths 14 page 130)  
`\path...` est la commande de base de TikZ qui définit un chemin.  
Toutes les autres commandes en sont des cas particuliers comme :  
`\clip...`, `\draw...`, `\fill...`, `\filldraw...`, `\shade...`, etc.
- `plot` (Plots of Functions 18 page 193)  
`plot` est une opération de chemin qui trace la courbe représentative d'une fonction donnée par une formule.  
On peut préciser le domaine de définition de la fonction avec l'option `[domain=]` et le nombre de points à calculer avec l'option `[samples=]`.  
Par exemple : `plot [domain=-2:2, samples=50] (\x, 2*\x+3)`
- `plot coordinates` (Plotting Points Given Inline 18.3 page 194)  
`plot coordinates` est une opération de chemin qui trace une représentation graphique d'une liste de points.  
Par défaut, une courbe reliant les points est dessinée. On peut aussi tracer des barres, horizontales avec l'option `[xcomb]`, verticales avec `[ycomb]` ou à partir de l'origine avec `[polar comb]`.  
On peut aussi marquer les points de la courbe avec l'option `[mark=]`.  
La liste des coordonnées de tous les points à afficher est définie en ligne, entre accolades.  
Par exemple : `plot [ycomb] coordinates {(1,2) (3,5) (5,1)}`
- `plot file` (Plotting Points Read From an External File 18.4 page 194)  
`plot file` est une opération de chemin qui trace une représentation graphique d'une liste de points. (Voir : `plot coordinates`)  
La liste des coordonnées de tous les points à afficher est définie dans un fichier texte dont le nom est donné entre accolades.  
Par exemple : `plot [mark=*] file {data.txt}`
- `plot function` (Plotting a Function Using Gnuplot 18.6 page 196)  
`plot function` est une opération de chemin qui trace la courbe représentative d'une fonction donnée par une formule Gnuplot.  
La formule est transmise à Gnuplot qui va calculer les coordonnées de tous les points à afficher. (Voir : `plot`)  
On peut préciser le domaine de définition de la fonction avec l'option `[domain=]` et le nombre de points à calculer avec l'option `[samples=]`.  
Par exemple :  
`plot [domain=-pi:pi, samples=50] function {x+sin(x)}`
- `polar comb` (Smooth Plots, Sharp Plots, and Comb Plots 18.8 page 199)  
Option pour tracer des barres à partir de l'origine. (Voir : `plot`)
- [pos=] (Placing Nodes on a Line or Curve Explicitly 15.8 page 161)  
Option pour placer des nœuds sur une ligne. (Voir : `--` ou `to`)

- rectangle** (The Rectangle Operation 13.5 page 120)  
**rectangle** est l'opération de chemin consistant à tracer un rectangle.  
 La forme d'utilisation la plus simple est `\draw (A) rectangle (B)`; qui trace un rectangle dont (A) et (B) sont deux sommets d'une diagonale.
- [**rectangle**] (Nodes and Their Shapes 15.2 page 147)  
 Abréviation de [**shape=rectangle**] fixe la forme d'un nœud. (Voir : **node**)
- région** (Interior Rules 14.4.2 page 138)  
 Une région est limitée par un chemin. (Voir : `\clip` ou `\fill`)
- [**remember picture**] (Referencing a Node in a Different Picture 15.13.1 page 169)  
 Option qui mémorise les noms des nœuds définis dans des environnements `{tikzpicture}` distincts. (Voir : [**overlay**])
- [**right**], [**left**] (Positioning Nodes 15.5 page 154)  
 Option de placement des nœuds. (Voir : **node**)
- [**rotate=**] (Coordinate Transformations 21.3 page 218)  
 Option pour effectuer une rotation. (Voir : **node** ou `{scope}`)
- [**rounded corners=**] (Rounding Corners 13.6 page 120)  
 Option qui arrondit les angles des lignes. (Voir : `\draw` ou **node**)
- [**scale=**] (Coordinate Transformations 21.3 page 218)  
 Option pour effectuer un changement d'échelle. (Voir : **node** ou `{scope}`)
- `{scope}` (Using Scopes to Structure a Picture 11.3 page 99)  
 L'environnement `{scope}` permet de dessiner une sous-figure et de lui appliquer globalement différentes transformations ou styles, comme : **shift**, **scale**, **rotate**, etc.
- [**samples=**] (Plotting a Function 18.5 page 194)  
 Option qui fixe le nombre des points à calculer pour une fonction. (Voir : **plot**)
- `\shade` (Shading a Path 14.5 page 139)  
`\shade...` est la forme abrégée de la commande `\path[shade]...` qui permet de colorier la région définie par un chemin avec des effets de dégradés de couleurs.  
 Différentes options permettent de préciser le dégradé comme **shading** ou les couleurs comme **top color** ou **bottom color**.
- [**shift=**], [**xshift=**], [**yshift=**] (Coordinate Transformations 21.3 page 218)  
 Option pour effectuer une translation. (Voir : **node** ou `{scope}`)
- [**sibling distance=**] (Default Growth Function 17.5.2 page 187)  
 Option pour fixer l'écart entre fils dans un arbre. (Voir : **child**)
- [**sloped**] (Placing Nodes on a Line or Curve Explicitly 15.8 page 161)  
 Option pour aligner des nœuds sur une ligne. (Voir : `--` ou **to**)
- [**smooth**] (Smooth Plots, Sharp Plots, and Comb Plots 18.8 page 199)  
 Option qui provoque le lissage d'une courbe. (Voir : **plot**)
- south** (Positioning Nodes Using Anchors 15.5.1 page 154)  
 Option d'ancrage [**anchor=south**] d'un nœud. (Voir : **node**)  
 Autres valeurs, [**anchor=south west**], [**anchor=south east**]

- `.south` (Positioning Nodes Using Anchors 15.5.1 page 154)  
 Pour la référence (`P.south`) point d'ancrage à un nœud. (Voir : `node`)  
 Autres valeurs, (`P.south west`), (`P.south east`)
- `stealth` (Arrow Tip Library 22 page 224)  
 Style de pointe de flèche, [`>=stealth`]. (Voir : `->`)
- [`step=`], [`xstep=`], [`ystep=`] (The Grid Operation 13.9 page 121)  
 Option pour fixer le pas de la grille. (Voir : `\grid`)
- [`tension=`] (Smooth Plots, Sharp Plots, and Comb Plots 18.8 page 199)  
 Option qui contrôle le niveau de lissage d'une courbe. (Voir : `plot`)
- [`text width=`] (Options for the Text in Nodes 15.4 page 152)  
 Option pour fixer mise en forme du texte d'un nœud. (Voir : `node`)
- [`thick`], [`very thick`] (Graphic Parameters : Line Width 14.3.1 page 132)  
 Option pour fixer l'épaisseur du trait. (Voir : `\draw`)
- [`thin`], [`very thin`] (Graphic Parameters : Line Width 14.3.1 page 132)  
 Options pour fixer l'épaisseur du trait. (Voir : `\draw`)
- TikZ** (Loading the Package and the libraries 11.1 page 96)  
 Pour utiliser TikZ, on doit charger le package `tikz` :  
`\usepackage{tikz}`  
 Si on veut de plus utiliser des bibliothèques additionnelles, il faut les charger explicitement, par exemple :  
`\usetikzlibrary{arrows,calc}`  
 Ne pas confondre le package `tikz` avec `\tikz`, qui est une commande équivalente à un environnement `{tikzpicture}` (contenant un code bref).
- `\tikz` (Creating a Picture Using a Command 11.2.2 page 98)  
 Commande brève pour la création de figure TikZ. (Voir : `{tikzpicture}`)
- `{tikzpicture}` (Creating a Picture Using an Environment 11.2.1 page 96)  
 On écrit les commandes de dessin dans l'environnement `{tikzpicture}`  
`\begin{tikzpicture} ... \end{tikzpicture}`  
 Pour les dessins dont le code tient en un paragraphe, on peut utiliser la commande brève :  
`\tikz \draw (0,0) -- (1,1);`
- `\tikzset` (How Graphic Options Are Processed 11.4.1 page 100)  
`\tikzset` est une commande TikZ qui permet de définir des options, des styles ou des clés.  
 Par exemple :  
`\tikzset{fondRouge/.style={fill=red}}`
- `\tikzstyle` (How Graphic Options Are Processed 11.4.1 page 100)  
 C'est une ancienne commande de la version 1.00 de TikZ & PGF utilisée pour définir une option de style. Par exemple :  
`\tikzstyle{fondRouge}=[fill=red]`  
 On peut toujours utiliser cette commande dans la version 2.00, mais maintenant, on utilisera plutôt la commande `\tikset`
- `to` (The To Path Operation 13.13 page 125)  
 (`A`) `to` (`B`) est une opération de chemin comme (`A`)--(`B`) mais on peut lui associer des options pour obtenir des lignes courbes, comme : [`bend left`], [`bend right`], [`in=`] ou [`out=`]

- [`top color=`], [`bottom color=`] (Choosing a Shading Color 14.5.2 page 140)  
Options pour fixer les couleurs de dégradés. (Voir : `\shade`)
- `west` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Option d'ancrage [`anchor=west`] d'un nœud. (Voir : `node`)
- `.west` (Positioning Nodes Using Anchors 15.5.1 page 154)  
Pour la référence (`P.west`) point d'ancrage à un nœud. (Voir : `node`)
- [`xcomb`] (Smooth Plots, Sharp Plots, and Comb Plots 18.8 page 199)  
Option pour tracer des barres horizontales. (Voir : `plot`)
- [`x=`] (The XY- and XYZ-Coordinate Systems 21.2 page 217)  
Option qui permet de changer le vecteur unité des X, qui vaut par défaut : [`x={ (1cm,0pt) }`] par rapport à un repère absolu.
- [`y=`] (The XY- and XYZ-Coordinate Systems 21.2 page 217)  
Option qui permet de changer le vecteur unité des Y, qui vaut par défaut : [`y={ (0pt,1cm) }`] par rapport à un repère absolu.
- [`ycomb`] (Smooth Plots, Sharp Plots, and Comb Plots 18.8 page 199)  
Option pour tracer des barres verticales. (Voir : `plot`)
- [`z=`] (The XY- and XYZ-Coordinate Systems 21.2 page 217)  
Option qui permet de changer le vecteur unité des Z, qui vaut par défaut : [`z={ (-0.3535cm,-0.3535cm) }`] par rapport à un repère absolu.  
En effet, les points de l'espace sont effectivement affichés dans un repère à deux dimensions qui vérifie par défaut :  $\vec{k} = -\frac{1}{2\sqrt{2}}\vec{i} - \frac{1}{2\sqrt{2}}\vec{j}$ .  
Pour obtenir le repère mathématique français usuel on peut définir :  
[`x={ (-0.353cm,-0.353cm) }`], [`z={ (0cm,1cm) }`], [`y={ (1cm,0cm) }`]



```

% avec \usetikzlibrary{fadings} en préambule pour le dégradé

\newcommand{\Fin}{node[xshift=-1.5ex,rotate=10]{F}
node[rotate=170]{i}
node[xshift=1.5ex,rotate=45]{n}}

\begin{tikzpicture}[scale=10,transform shape]
\draw (0,0) \Fin;
\draw (-1em,-1ex) -- (1em,-1ex);
\path[scope fading=south] (-1em,-0.25em) rectangle (1em,-3.75ex);
\draw[yscale=-1] (0,2ex) \Fin;
\end{tikzpicture}

```

**F I n**  


---

*E i U*