

ProfLycee

Quelques *petites* commandes pour \LaTeX (au lycée)

Cédric Pierquet

c pierquet -- at -- outlook . fr

Version 2.5.8 – 6 Avril 2023

Résumé : Quelques commandes pour faciliter l'utilisation de \LaTeX pour les mathématiques, au lycée.

- ✦ résoudre, de manière approchée, des équations
- ✦ tracer *facilement* des repères/grilles/courbes
- ✦ tracer des courbes *lisses* avec gestion des extrema et des dérivées
- ✦ présenter du code python ou pseudocode, une console d'exécution Python
- ✦ tracer rapidement un pavé, un tétraèdre
- ✦ simplifier des calculs sous forme fractionnaire, simplifier des racines
- ✦ effectuer des calculs avec des suites récurrentes
- ✦ créer la *toile* pour une suite récurrente
- ✦ afficher et utiliser un cercle trigo
- ✦ afficher un petit schéma pour le signe d'une fonction affine ou d'un trinôme
- ✦ travailler sur les statistiques à deux variables (algébriques et graphiques)
- ✦ convertir entre bin/dec/hex avec détails
- ✦ présenter un calcul de PGCD
- ✦ effectuer des calculs de probas (lois binomiale, exponentielle, de Poisson, normale)
- ✦ créer des arbres de probas « classiques »
- ✦ générer des listes d'entiers aléatoires (avec ou sans répétitions)
- ✦ déterminer la mesure principale d'un angle
- ✦ ...

Merci à Anne pour ses retours et sa relecture!

Merci à Christophe et Denis pour leurs retours et éclairages!

Merci aux membres du groupe  du « Coin \LaTeX » pour leur aide et leurs idées!

\LaTeX

pdf \LaTeX

Lua \LaTeX

TikZ

TeXLive

MiKTeX

Table des matières

I	Introduction	6
1	Le package ProfLycee	6
1.1	« Philosophie » du package	6
1.2	Chargement du package	6
1.3	Librairies, option du package	7
2	Compléments	8
2.1	Le système de « clés/options »	8
2.2	Compilateur(s)	8
2.3	Problèmes éventuels...	8
II	Liste des commandes, par thème	10
III	Outils pour l'analyse	14
3	Résolution approchée d'une équation	14
3.1	Idée	14
3.2	Clés et options	14
4	Présentation d'une solution d'équation par balayage	16
4.1	Idée	16
4.2	Clés et arguments	16
4.3	Interaction avec la commande de résolution approchée	17
5	Suites récurrentes simples	18
5.1	Idées	18
5.2	Clés et arguments	18
5.3	Exemple d'utilisation	19
IV	Outils graphiques, via TikZ	20
6	Repérage et tracé de courbes, en TikZ	20
6.1	Idée	20
6.2	Commandes, clés et options	21
6.3	Commandes annexes	24
6.4	Repère non centré en O	25
7	L'outil « SplineTikz »	26
7.1	Courbe d'interpolation	26
7.2	Code, clés et options	26
7.3	Compléments sur les coefficients de « compensation »	26
7.4	Exemples	27
7.5	Avec une gestion plus fine des « coefficients »	28
7.6	Conclusion	28
8	L'outil « TangenteTikz »	29
8.1	Définitions	29
8.2	Exemple et illustration	29
8.3	Exemple avec les deux outils, et « personnalisation »	30

9	Petits schémas pour le signe d'une fonction affine ou d'un trinôme	31
9.1	Idée	31
9.2	Commandes	31
9.3	Intégration avec tkz-tab	33
10	Suites récurrentes et « toile »	34
10.1	Idée	34
10.2	Commandes	34
10.3	Exemples	34
10.4	Influence des paramètres	36
V	Présentation de codes	37
11	Code Python « simple » via le package listings	37
11.1	Introduction	37
11.2	Commande et options	37
11.3	Insertion via un fichier « externe »	38
11.4	Exemples	38
12	Code Python via le package piton	41
12.1	Introduction	41
12.2	Présentation de code Python	41
12.3	Console en partenariat avec Pyluatex	43
13	Code & Console Python, via les packages Pythontex ou Minted	44
13.1	Librairies	44
13.2	Introduction	44
13.3	Présentation de code Python grâce au package pythontex	44
13.4	Présentation de code Python via le package minted	45
13.5	Console d'exécution Python	46
14	Pseudo-Code	48
14.1	Introduction	48
14.2	Présentation de Pseudo-Code	48
14.3	Compléments	49
15	Terminal Windows/UNIX/OSX	50
15.1	Introduction	50
15.2	Commandes	50
16	Cartouche Capytale	52
16.1	Introduction	52
16.2	Commandes	52
17	Présentation de code \LaTeX	53
17.1	Introduction	53
17.2	Commandes	53
VI	Outils pour la géométrie	54
18	Pavé droit « simple »	54
18.1	Introduction	54
18.2	Commandes	54
18.3	Influence des paramètres	55

19	Tétraèdre « simple »	56
19.1	Introduction	56
19.2	Commandes	56
19.3	Influence des paramètres	57
20	Cercle trigo	58
20.1	Idée	58
20.2	Commandes	58
20.3	Équations trigos	59
21	Style « main levée » en TikZ	61
21.1	Idée	61
21.2	Utilisation basique	61
VII	Outils pour les statistiques	62
22	Paramètres d'une régression linéaire par la méthode des moindres carrés	62
22.1	Idée	62
22.2	Commandes	62
22.3	Intégration dans un environnement TikZ	64
23	Statistiques à deux variables	67
23.1	Idées	67
23.2	Commandes, clés et options	68
23.3	Commandes annexes	71
23.4	Interactions avec CalculsRegLin	72
23.5	Exemple complémentaire, pour illustration	75
24	Boîtes à moustaches	76
24.1	Introduction	76
24.2	Clés et options	76
24.3	Commande pour placer un axe horizontal	77
VIII	Outils pour les probabilités	79
25	Calculs de probabilités	79
25.1	Introduction	79
25.2	Calculs « simples »	79
25.3	Complément avec sortie « formatée »	81
26	Arbres de probabilités « classiques »	84
26.1	Introduction	84
26.2	Options et arguments	84
26.3	Exemples complémentaires	85
27	Petits schémas pour des probabilités continues	87
27.1	Idée	87
27.2	Commandes et options	87
27.3	Remarques et compléments	88
28	Nombres aléatoires	89
28.1	Idée	89
28.2	Clés et options	90

29 Combinatoire	91
29.1 Idée	91
29.2 Utilisation	91
 IX Outils pour l'arithmétique	 92
30 Conversions binaire/hexadécimal/décimal	92
30.1 Idée	92
30.2 Conversion décimal vers binaire	92
30.3 Conversion binaire vers hexadécimal	93
30.4 Conversion binaire ou hexadécimal en décimal	94
31 Conversion « présentée » d'un nombre en base décimale	95
31.1 Idée	95
31.2 Code et clés	95
32 Algorithme d'Euclide pour le PGCD	97
32.1 Idée	97
32.2 Options et clés	97
32.3 Compléments	98
 X Écritures, simplifications	 99
33 Simplification sous forme d'une fractions	99
33.1 Idée	99
33.2 Commande et options	99
34 Ensembles	101
34.1 Idée	101
34.2 Commande et options	101
35 Écriture d'un trinôme, trinôme aléatoire	102
35.1 Idée	102
35.2 Clés et options	102
36 Simplification de racines	104
36.1 Idée	104
36.2 Exemples	104
37 Mesure principale d'un angle	105
37.1 Idée	105
37.2 Exemples	105
 XI Jeux et récréations	 106
38 SudoMaths, en TikZ	106
38.1 Introduction	106
38.2 Clés et options	107
 XII Historique	 109

Première partie

Introduction

1 Le package ProfLycee

1.1 « Philosophie » du package

💡 Idée(s)

Ce `\package`, très largement inspiré (et beaucoup moins abouti) de l'excellent `\ProfCollege` de C. Poulain et des excellents `\tkz-*` d'A. Matthes, va définir quelques outils pour des situations particulières qui ne sont pas encore dans `\ProfCollege`.

On peut le voir comme un (maigre) complément à `\ProfCollege`, et je précise que la syntaxe est très proche (car pertinente de base) et donc pas de raison de changer une *équipe qui gagne*!

Il se charge de manière classique, dans le préambule, par `\usepackage{ProfLycee}`. Il charge des packages utiles, mais j'ai fait le choix de laisser l'utilisateur gérer ses autres packages, comme notamment `\amssymb` qui peut poser souci en fonction de la *position* de son chargement.

L'utilisateur est libre de charger ses autres packages utiles et habituels, ainsi que ses polices et encodages habituels!

🔧 Information(s)

Le package `\ProfLycee` charge et utilise les packages :

- `\xcolor` avec l'option `[table,svgnames]`;
- `\tikz`, `\pgf`, `\pgffor`, `\nicefrac`;
- `\tcolorbox` avec l'option `[most]`;
- `\xparse`, `\xstring`, `\simplekv`, `\xinttools`;
- `\listofitems`, `\xintexpr`, `\xintbinhex`, `\xintgcd`;
- `\tabularray`, `\fontawesome5`, `\randomlist`, `\fancyvrb`.

💡 Idée(s)

J'ai utilisé les packages de C. Tellechea, je vous conseille d'aller jeter un œil sur ce qu'il est possible de faire en \LaTeX avec `\listofitems`, `\randomlist`, `\simplekv` ou encore `\xstring`!

1.2 Chargement du package

🔗 Code \LaTeX

```
%exemple de chargement pour une compilation en (pdf)latex
\documentclass{article}
\usepackage{ProfLycee}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
...
```

🔗 Code \LaTeX

```
%exemple de chargement pour une compilation en (xe/lua)latex
\documentclass{article}
\usepackage{ProfLycee}
\usepackage{fontspec}
...
```

1.3 Bibliothèques, option du package

⚠ Attention

2.5.0 Le package fonctionne désormais avec un système de `librairies`, qui utilisent et chargent des packages spécifiques, avec des compilations particulières, donc l'utilisateur utilisera un système de chargement similaire à celui de `tcolorbox` ou `tikz`, dans le préambule, et une fois le package appelé.

</> Code \LaTeX

```
\usepackage{ProfLycee}
\useproflyclib{...,...}
```

⚙ Information(s)

Les bibliothèques disponibles seront indiquées dans les sections spécifiques. Pour le moment, il existe :

- `python` (page 41);
- `minted` (page 45);
- `pythontex` (page 44).

⚠ Attention

2.5.8 Pour le package `python`, la version minimale requise est la **1.5** pour bénéficier d'un rendu optimal (au niveau des marges) de la présentation du code Python.

⚙ Information(s)

En compilant (notamment avec les bibliothèques `minted` et `pythontex`) on peut spécifier des répertoires particuliers pour les (ou des) fichiers auxiliaires.

Avec l'option **(build)**, l'utilisateur a la possibilité de placer les fichiers temporaires de `minted` et `pythontex` dans un répertoire `build` du répertoire courant.

Dans ce cas il faut créer au préalable le répertoire `build` avant de compiler un fichier, pour éviter toute erreur!

</> Code \LaTeX

```
...
\usepackage[build]{ProfLycee}
\useproflyclib{...}
...
```

⚙ Information(s)

L'option **(build)** charge certains packages (bibliothèques `minted` et `pythontex`) avec les options :

- `\setpythontexoutputdir{./build/pythontex-files-\jobname}`
- `\RequirePackage[outputdir=build]{minted}`

2 Compléments

2.1 Le système de « clés/options »

💡 Idée(s)

L'idée est de conserver – autant que faire se peut – l'idée de **⟨Clés⟩** qui sont :

- modifiables;
- définies (en majorité) par défaut pour chaque commande.

Pour certaines commandes, le système de **⟨Clés⟩** pose quelques soucis, de ce fait le fonctionnement est plus *basique* avec un système d'arguments *optionnels* (souvent entre [...]) ou *obligatoires* (souvent entre {...}).

À noter que les :


- les **⟨Clés⟩** peuvent être mises dans n'importe quel ordre, elles peuvent être omises lorsque la valeur par défaut est conservée;
- les arguments doivent, eux, être positionnés dans le *bon ordre*.

🔧 Information(s)


Les commandes et environnements présentés seront explicités via leur syntaxe avec les options/clés ou arguments.

Autant que faire se peut, des exemples/illustrations/remarques seront proposés à chaque fois.

Les codes seront présentés dans des boîtes  **Code**  **TEX**, si possible avec la sortie dans la même boîte, et sinon la sortie sera visible dans des boîtes  **Sortie**  **TEX**.

Les clés ou options seront présentées dans des boîtes  **Clés**.

🔧 Information(s)




À noter que certaines commandes disponibles sont liées à un environnement  **tikzpicture**, elles peuvent ne pas être autonomes mais permettent de conserver – en parallèle – toute commande liée à TikZ!

2.2 Compilateur(s)

🔧 Information(s)



Le package  **ProfLycee** est compatible avec les compilateurs classiques : latex, pdflatex ou encore lualatex.



En ce qui concerne les codes librairies, il faudra :

-  **pythontex** : compiler en chaîne (xxx)latex + pythontex + (xxx)latex;
-  **minted** : compiler avec shell-escape (ou write18);
-  **piton** : compiler en Lua \TeX et shell-escape (ou write18).

2.3 Problèmes éventuels...

🔧 Information(s)

Certaines commandes sont à intégrer dans un environnement TikZ, afin de pouvoir rajouter des éléments, elles ont été testés dans des environnement  **tikzpicture**, à vérifier que la gestion des axes par l'environnement  **axis** est compatible...

Certains packages ont une fâcheuse tendance à être tatillons sur leurs options (les *fameux* option clash for ...) ou leur *position* dans le chargement, donc attention notamment au chargement de  **xcolor** et de  **amssymb**!
En dehors de cela, ce sont des tests multiples et variés qui permettront de détecter d'éventuels bugs!

↔ Bonne(s) découverte(s) ↔

Deuxième partie

Liste des commandes, par thème

Information(s)

 2.0.0 Cette section contient un *résumé* des différentes commandes et environnements disponibles dans ProfLycee.

Elles sont présentées de manière *succincte*, mais elles sont présentées de manière *détaillée* dans la suite de la documentation.

Code \LaTeX

```
%Résolution approchée d'une équation  $f(x)=k$ 
\ResolutionApprochee[clés]{équation}[macro]

%Présentation d'une solution par balayage (TVI)
\SolutionTVI[options]{fonction}{valeur}

%Calculer le terme d'une suite récurrente simple, toile pour une suite récurrente simple
\CalculTermeReccurrence[options]{fonction associée}
\ToileReccurrence[clés][options du tracé][option supplémentaire des termes]

%Mise en forme de la conclusion d'un seuil
\SolutionSeuil[options]{fonction associée}{seuil}
```

Code \LaTeX

```
%fenêtre de repérage en tikz et courbe
\GrilleTikz[options][options grille ppale][options grille second.]
\AxesTikz[options] \AxexTikz[options]{valeurs} \AxeYtikz[options]{valeurs}
\FenetreSimpleTikz[options](opt axes)<opt axe Ox>{liste valx}<opt axe Oy>{liste valy}
\CourbeTikz[options]{fonction}{valxmin:valxmax}

%courbe d'interpolation, tangente, dans un environnement tikz
\SplineTikz[options]{liste}
\TangenteTikz[options]{liste}

%schémas pour le signe affine/trinôme, dans un environnement tikz
\MiniSchemaSignes(*)[clés]<options tikz>
\MiniSchemaSignesTkzTab[options]{numligne}[échelle][décalage horizontal]
```

Code \LaTeX

```
%présentation de code Python
\begin{CodePythonLst}(*)[largeur]{commandes tcbbox}...\end{CodePythonLst}
\begin{CodePythonLstAlt}(*)[largeur]{commandes tcbbox}...\end{CodePythonLstAlt}
%=:bibliothèque python
\begin{CodePiton}[options piton]{commandes tcbbox}...\end{CodePiton}
\begin{PitonConsole}<Clés>{commandes tcbbox}...\end{PitonConsole}
%=:bibliothèque pythontex
\begin{CodePythontex}[options]{...}\end{CodePythontex}
\begin{CodePythontexAlt}[options]{...}\end{CodePythontexAlt}
\begin{ConsolePythontex}[options]{...}\end{ConsolePythontex}
%=:bibliothèque minted
\begin{CodePythonMinted}(*)[largeur]{commandes tcbbox}...\end{CodePythonMinted}
\begin{CodePythonMintedAlt}(*)[largeur]{commandes tcbbox}...\end{CodePythonMintedAlt}

%présentation de pseudocode
\begin{PseudoCode}(*)[largeur]{commandes tcbbox}...\end{PseudoCode}
\begin{PseudoCodeAlt}(*)[largeur]{commandes tcbbox}...\end{PseudoCodeAlt}
```

</> Code \LaTeX

```
%terminal OS
\begin{TerminalWin}[largeur]{clés}[options]...\end{TerminalWin}
\begin{TerminalUnix}[largeur]{clés}[options]...\end{TerminalUnix}
\begin{TerminalOSX}[largeur]{clés}[options]...\end{TerminalOSX}

%code Cappytale
\CartoucheCapytale(*)[options]{code capytale}
```

</> Code \LaTeX

```
%pavé et tétraèdre, dans un environnement tikz
\PaveTikz[options]
\TetraedreTikz[options]

%cercle trigo, dans un environnement tikz
\CercleTrigo[clés]
```

</> Code \LaTeX

```
%paramètres d'une régression linéaire, nuage de points
\CalculsRegLin[clés]{listeX}{listeY}
\PointsRegLin[clés]{listeX}{listeY}

%stats à 2 variables, dans un environnement tikz
\GrilleTikz[options][options grille ppale][options grille second.]
\AxesTikz[options]
\AxexTikz[options]{valeurs} \AxeYtikz[options]{valeurs}
\FenetreTikz \OrigineTikz
\FenetreSimpleTikz[options](opt axes)<opt axe 0x>{liste valx}<opt axe 0y>{liste valy}
\NuagePointsTikz[options]{listeX}{listeY}
\PointMoyenTikz[options]
\CourbeTikz[options]{formule}{domaine}

%boîte à moustaches, dans un environnement tikz
\BoiteMoustaches[options]
\BoiteMoustachesAxe[options]
```

</> Code \LaTeX

```
%loi binomiale  $B(n,p)$ 
\CalcBinomP{n}{p}{k}
\CalcBinomC{n}{p}{a}{b}
\BinomP(*)[prec]{n}{p}{k}
\BinomC(*)[prec]{n}{p}{a}{b}

%loi de Poisson  $P(l)$ 
\CalcPoissP{l}{k}
\CalcPoissC{l}{a}{b}
\PoissonP(*)[prec]{l}{k}
\PoissonC(*)[prec]{l}{a}{b}
```

</> Code \LaTeX

```
%loi géométrique  $G(p)$ 
\CalcGeomP{p}{k}
\CalcGeomC{l}{a}{b}
\GeomP{p}{k}
\GeomC{l}{a}{b}

%loi hypergéométrique  $H(N,n,m)$ 
\CalcHypergeomP{N}{n}{m}{k}
\CalcHypergeomP{N}{n}{m}{a}{b}
\HypergeomP{N}{n}{m}{k}
\HypergeomC{N}{n}{m}{a}{b}
```

</> Code \LaTeX

```
%loi normale  $N(m,s)$ 
\CalcNormC{m}{s}{a}{b}
\NormaleC(*)[prec]{m}{s}{a}{b}

%loi exponentielle  $E(l)$ 
\CalcExpoC{l}{a}{b}
\ExpoC(*)[prec]{l}{a}{b}

%arbres de probas
\ArbreProbasTikz[options]{donnees}
\begin{EnvArbreProbasTikz}[options]{donnees}...\end{EnvArbreProbasTikz}

%schémas lois continues
\LoiNormaleGraphe[options]<options tikz>{m}{s}{a}{b}
\LoiExpoGraphe[options]<options tikz>{l}{a}{b}
```

</> Code \LaTeX

```
%entier aléatoire entre a et b
\NbAlea{a}{b}{macro}
%nombre décimal (n chiffres après la virgule) aléatoire entre a et b+1 (exclus)
\NbAlea[n]{a}{b}{macro}
%création d'un nombre aléatoire sous forme d'une macro
\VarNbAlea{macro}{calcul}
%liste d'entiers aléatoires
\TirageAleatoireEntiers[options]{macro}
```

</> Code \LaTeX

```
%arrangement  $Anp$ 
\Arrangement(*)[option]{p}{n}

%arrangement  $Cnp$  (p parmi n)
\Combinaison(*)[option]{p}{n}
```

</> Code \LaTeX

```
%conversions
\ConversionDecBin(*)[clés]{nombre}
\ConversionBinHex[clés]{nombre}
\ConversionVersDec[clés]{nombre}
\ConversionBaseDix[clés]{nombre}{base de départ}
\ConversionDepuisBaseDix[options]{nombre en base 10}{base d'arrivée}

%PGCD présenté
\PresentationPGCD[options]{a}{b}
```

Code \LaTeX

```
%conversion en fraction, simplification de racine
\ConversionFraction(*)[option]{argument}
\SimplificationRacine{expression}

%ensemble d'éléments
\EcritureEnsemble[clés]{liste}

%trinôme, trinôme aléatoire
\EcritureTrinome[options]{a}{b}{c}

%mesure principale
\MesurePrincipale[options]{angle}
```

Code \LaTeX

```
%sudomaths
\SudoMaths[options]{liste}
\begin{EnvSudoMaths}[options]{grille}...\end{EnvSudoMaths}
```

Troisième partie

Outils pour l'analyse

3 Résolution approchée d'une équation

3.1 Idée

💡 Idée(s)

2.1.4 L'idée est de proposer une commande pour résoudre, de manière approchée, une équation du type $f(x) = k$ sur un intervalle (fermé) donné.

La méthode utilisée est la **dichotomie**, pour plus de rapidité que la méthode *simple* par balayage.

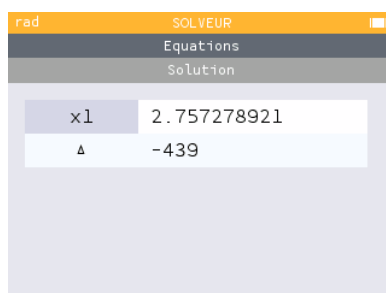
</> Code \LaTeX

```
\ResolutionApprochee[clés]{équation}[macro]
```

</> Code \LaTeX

```
\ResolutionApprochee[Intervalle=0:10]{x**3-2*x**2-x-1=2}%  
$x_0 \approx \num[minimum-decimal-digits=2]{\masolutiond}$ par défaut ;\\  
$x_0 \approx \num[minimum-decimal-digits=2]{\masolutione}$ par excès ;\\  
$x_0 \approx \num[minimum-decimal-digits=2]{\masolutiona}$ arrondi à  $10^{-2}$ .\\  
  
\hfill\includegraphics[scale=0.45]{./graphics/pl-solve_a}\hfill~
```

$x_0 \approx 2,75$ par défaut;
 $x_0 \approx 2,76$ par excès;
 $x_0 \approx 2,76$ arrondi à 10^{-2} .



rad SOLVEUR	
Equations	
Solution	
x1	2.757278921
Δ	-439

3.2 Clés et options

🔗 Clés et options

Quelques explications sur les **clés** et sur les arguments :

- la clé **Précision** pour le nombre de chiffres après la virgule de la solution; défaut **2**
- la clé (obligatoire!) **Intervalle** qui permet de préciser l'intervalle initial de recherche;
- la clé **Variable** qui permet de spécifier la variable de l'équation; défaut **x**
- l'argument *obligatoire* est l'équation, sous la forme $f(\dots) = k$ (ou $f(\dots)$ pour $f(\dots) = 0$);
- l'argument *optionnel* est la base de la *macro* qui sert à stocker les valeurs : défaut **masolution**
- `\<macro>d` pour la valeur approchée par défaut;
- `\<macro>e` pour la valeur approchée par excès;
- `\<macro>a` pour la valeur approchée.

```

\ResolutionApprochee[Precision=4,Intervalle=0:2]{exp(0.5*x)+x**2-4=0}%
Une valeur approchée, à  $10^{-4}$  près, d'une solution de  $\text{\text{e}}^{0,5x}+x^2-4=0$  sur
↪  $\left[0;2\right]$  est  $\beta$  avec :
\begin{itemize}
  \item  $\beta \approx \text{\num[minimum-decimal-digits=4]{\masolutiond}}$  par défaut ;
  \item  $\beta \approx \text{\num[minimum-decimal-digits=4]{\masolutione}}$  par excès ;
  \item  $\beta \approx \text{\num[minimum-decimal-digits=4]{\masolutiona}}$ .
\end{itemize}
\ResolutionApprochee[Variable=t,Intervalle=-1:2]{3*t*exp(-0.5*t+1)=4}[SolA]%
Une valeur approchée, à  $10^{-2}$  près d'une solution de  $3t\text{\text{e}}^{-0,5t+1}=4$  est  $t_1$ 
↪ avec :
\begin{itemize}
  \item  $t_1 \approx \text{\num[minimum-decimal-digits=2]{\SolAd}}$  par défaut ;
  \item  $t_1 \approx \text{\num[minimum-decimal-digits=2]{\SolAe}}$  par excès ;
  \item  $t_1 \approx \text{\num[minimum-decimal-digits=2]{\SolAa}}$ .
\end{itemize}
\ResolutionApprochee[Precision=3,Variable=t,Intervalle=2:10]{3*t*exp(-0.5*t+1)=4}[SolB]%
Une valeur approchée, à  $10^{-2}$  près d'une solution de  $3t\text{\text{e}}^{-0,5t+1}=4$  est  $t_2$ 
↪ avec :
\begin{itemize}
  \item  $t_2 \approx \text{\num[minimum-decimal-digits=2]{\SolBd}}$  par défaut ;
  \item  $t_2 \approx \text{\num[minimum-decimal-digits=2]{\SolBe}}$  par excès ;
  \item  $t_2 \approx \text{\num[minimum-decimal-digits=2]{\SolBa}}$ .
\end{itemize}
\medskip
\hfill\includegraphics[scale=0.45]{./graphics/pl-solve_b}~~
\includegraphics[scale=0.45]{./graphics/pl-solve_c}~~
\includegraphics[scale=0.45]{./graphics/pl-solve_d}\hfill~

```

Une valeur approchée, à 10^{-4} près, d'une solution de $e^{0,5x} + x^2 - 4 = 0$ sur $[0;2]$ est β avec :

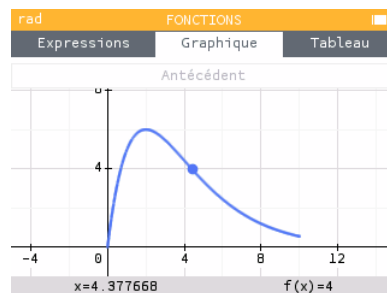
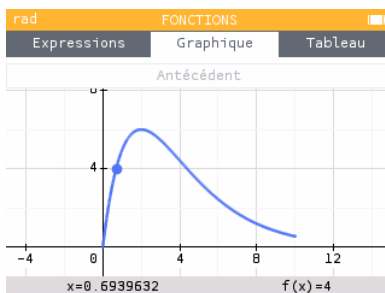
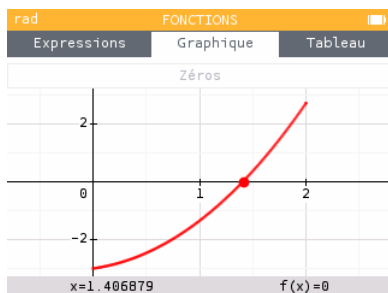
- $\beta \approx 1,4068$ par défaut ;
- $\beta \approx 1,4069$ par excès ;
- $\beta \approx 1,4069$.

Une valeur approchée, à 10^{-2} près d'une solution de $3te^{-0,5t+1} = 4$ est t_1 avec :

- $t_1 \approx 0,69$ par défaut ;
- $t_1 \approx 0,70$ par excès ;
- $t_1 \approx 0,69$.

Une valeur approchée, à 10^{-2} près d'une solution de $3te^{-0,5t+1} = 4$ est t_2 avec :

- $t_2 \approx 4,377$ par défaut ;
- $t_2 \approx 4,378$ par excès ;
- $t_2 \approx 4,378$.



4 Présentation d'une solution d'équation par balayage

4.1 Idée

💡 Idée(s)

2.0.4 L'idée est de présenter l'obtention d'une solution approchée d'équation par balayage, dans le cadre du TVI par exemple. Les calculs et tests sont effectués grâce au package `xinttools`, et le formatage par `tabularray` et `sinuitx`.

⚠ Attention

Le code ne trouve pas la solution, il met *juste* en forme mais effectue quand même les calculs d'images et les tests.

🔗 Code \LaTeX

```
\SolutionTVI[options]{fonction}{valeur}
```

4.2 Clés et arguments

🔗 Clés et options

Plusieurs **⟨Clés⟩** sont disponibles pour cette commande, relative donc à une équation du type $f(x) = k$:

- la clé **⟨NomFct⟩** qui permet de spécifier le nom de la fonction; défaut **⟨f⟩**
- la clé **⟨NomSol⟩** qui permet de spécifier le nom de la fonction; défaut **⟨\alpha⟩**
- les clés **⟨va⟩** et **⟨vb⟩** qui sont les bornes inférieure et supérieure de l'encadrement;
- la clé **⟨Precision⟩** qui est la précision des calculs pour les images; défaut **⟨2⟩**
- la clé **⟨Stretch⟩** qui permet d'espacer les lignes; défaut **⟨1.15⟩**
- les booléens **⟨Balayage⟩** ou **⟨Calculatrice⟩** pour afficher un texte en amont; défaut **⟨false⟩**
- le booléen **⟨Majuscule⟩** qui affiche le texte avant, avec une majuscule au début. défaut **⟨true⟩**

Le premier argument *obligatoire* est la fonction, en syntaxe `xint` et avec comme variable x , et le second la valeur de k .

🔗 Code \LaTeX

Pour $f(x)=0$ avec $f(x)=x^2-2$. On obtient

```
\SolutionTVI[va=1.414,vb=1.415,Precision=3]{x**2-2}{0}.
```

Pour $f(x) = 0$ avec $f(x) = x^2 - 2$. On obtient
$$\left\{ \begin{array}{l} f(1,414) \approx -0,001 < 0 \\ f(1,415) \approx 0,002 > 0 \end{array} \right\} \Rightarrow 1,414 < \alpha < 1,415.$$

🔗 Code \LaTeX

Avec $\varphi(t)=3t\,e^{-0,5t+1}=5$,

```
\SolutionTVI[Majuscule=false,Calculatrice,va=1.02,vb=1.03,NomFct=\varphi]{3*x*exp(-0.5*x+1)}{5}
```

Avec $\varphi(t) = 3t e^{-0,5t+1} = 5$, par calculatrice, on obtient
$$\left\{ \begin{array}{l} \varphi(1,02) \approx 4,99 < 5 \\ \varphi(1,03) \approx 5,02 > 5 \end{array} \right\} \Rightarrow 1,02 < \alpha < 1,03$$

Code \LaTeX


On s'intéresse à $g(x) = \text{num}\{1,5\}$ avec $g(x) = \ln(x)$. \backslash
 \backslash `SolutionTVI%`
[Balayage,Stretch=1.5,va=4.48,vb=4.49,NomFct=g,Precision=4,NomSol={x_0}]{log(x)}{1.5}.

On s'intéresse à $g(x) = 1,5$ avec $g(x) = \ln(x)$.

Par balayage, on obtient $\left\{ \begin{array}{l} g(4,48) \approx 1,4996 < 1,5 \\ g(4,49) \approx 1,5019 > 1,5 \end{array} \right. \Rightarrow 4,48 < x_0 < 4,49.$

4.3 Interaction avec la commande de résolution approchée

Idée(s)

 2.1.4 L'idée est de récupérer les valeurs par défaut et par excès pour le TVI grâce à la commande `\ResolutionApprochee`.

Code \LaTeX

On s'intéresse à $g(x) = \text{num}\{1,5\}$ avec $g(x) = \ln(x)$ sur l'intervalle $\left[3;5\right]$.

\backslash `ResolutionApprochee`[Intervalle=3:5]{log(x)=1.5}[SolLn]
 \backslash `SolutionTVI%`
[Balayage,Stretch=1.5,va={\SolLnd},vb={\SolLne},
NomFct=g,Precision=4,NomSol={x_0}]{log(x)}{1.5}.

On s'intéresse à $g(x) = 1,5$ avec $g(x) = \ln(x)$ sur l'intervalle $[3;5]$.

Par balayage, on obtient $\left\{ \begin{array}{l} g(4,48) \approx 1,4996 < 1,5 \\ g(4,49) \approx 1,5019 > 1,5 \end{array} \right. \Rightarrow 4,48 < x_0 < 4,49.$

Information(s)

À terme, peut-être que la commande `\ResolutionApprochee` sera intégrée dans la commande `\SolutionTVI` afin d'automatiser encore plus le procédé.

5 Suites récurrentes simples

5.1 Idées

💡 Idée(s)

2.0.3 L'idée est de proposer des commandes pour effectuer des calculs avec des suites récurrentes du type $u_{n+1} = f(u_n)$:

- calcul de termes avec possibilité d'arrondir;
- présentation de la conclusion de la recherche d'un seuil du type $u_n > S$ ou $u_n < S$.

⚠ Attention

2.1.0 Le code pour le seuil **trouve** également le rang cherché, il met en forme et effectue les calculs d'images.

2.0.5 Le choix a été fait de faire les calculs en mode `float` pour éviter les dépassements de capacité de `xint` liés aux boucles...

🔗 Code \LaTeX

```
%commande pour calculer et formater
\CalculTermeReccurrence[options]{fonction associée}

%mise en forme de la conclusion d'un seuil
\SolutionSeuil[options]{fonction associée}{seuil}
```

5.2 Clés et arguments

🔑 Clés et options

Plusieurs **<Clés>** sont disponibles pour la commande du calcul d'un terme :

- la clé **<No>** qui est le rang initial de la suite;
- la clé **<UNo>** qui est le terme initial de la suite;
- la clé **<Precision>** qui précise l'arrondi éventuel; défaut **<3>**
- la clé **<N>** qui est l'indice du terme à calculer.

L'argument *obligatoire* est la fonction associée à la suite, en syntaxe `xint` et avec comme variable x .

🔗 Code \LaTeX

```
Avec  $\begin{cases} u_0 = 50 \\ u_{n+1} = \frac{1}{u_n + 2} \end{cases}$ .

On obtient  $u_{10} \approx \text{\CalcTermReccurrence[No=0,UNo=50,N=10]{1/(x+2)}}$ .

On obtient  $u_{15} \approx \text{\CalcTermReccurrence[Precision=4,No=0,UNo=50,N=15]{1/(x+2)}}$ .

On obtient  $u_{20} \approx \text{\CalcTermReccurrence[Precision=6,No=0,UNo=50,N=20]{1/(x+2)}}$ .
```

📄 Sortie \LaTeX

Avec $u_0 = 50$ et $u_{n+1} = \frac{1}{u_n + 2}$.

On obtient $u_{10} \approx 0,414$

On obtient $u_{15} \approx 0,4142$

On obtient $u_{20} \approx 0,414214$

sortie par défaut.
avec choix de la précision à 10^{-4} .
avec choix de la précision à 10^{-6} .

Quatrième partie

Outils graphiques, via TikZ

6 Repérage et tracé de courbes, en TikZ

6.1 Idée

💡 Idée(s)

- 2.1.1 L'idée est de proposer des commandes *simplifiées* pour tracer un repère, en TikZ, avec :
- axes et graduations, grille;
 - courbe.

📌 Information(s)

Au niveau du code, il y aura donc plusieurs *aspects* :

- le paramétrage de la fenêtre graphique directement dans la déclaration de l'environnement;
- les commandes de tracés avec options et clés.

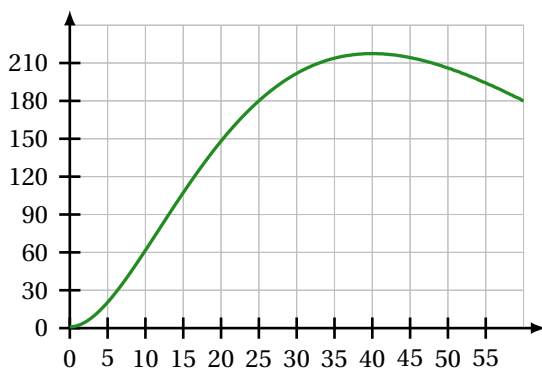
🔗 Code \LaTeX

```
%version basique
\begin{tikzpicture}[paramètres]
  %grille et axes
  \GrilleTikz[options][options grille ppale][options grille second.]
  \AxesTikz[options]
  \AxeXTikz[options]{valeurs}
  \AxeYTikz[options]{valeurs}
  %courbe
  \CourbeTikz[options]{fonction}{valxmin:valxmax}
\end{tikzpicture}
```

🔗 Code \LaTeX

```
%version simplifiée
\begin{tikzpicture}[<paramètres>]
  %grille et axes
  \FenetreSimpleTikz[opt](opt axes)<opt axe Ox>{liste valx}<opt axe Oy>{liste valy}
  %courbe
  \CourbeTikz[options]{fonction}{valxmin:valxmax}
\end{tikzpicture}
```

➡ Sortie \LaTeX



6.2 Commandes, clés et options

Information(s)

Les **paramètres** nécessaires à la bonne utilisation des commandes suivantes sont à déclarer directement dans l'environnement `\tikzpicture`, seules les versions « x » sont présentées ici :

- **<xmin>**, stockée dans `\xmin`; défaut **<-3>**
- **<xmax>**, stockée dans `\xmax`; défaut **<3>**
- **<Ox>**, stockée dans `\axexOx`, origine de l'axe (Ox); défaut **<0>**
- **<xgrille>**, stockée dans `\xgrille`, graduation principale; défaut **<1>**
- **<xgrilles>**, stockée dans `\xgrilles`, graduation secondaire. défaut **<0.5>**

La fenêtre d'affichage (de sortie) sera donc *portée* par le rectangle de coins (xmin; ymin) et (xmax; ymax); ce qui correspond en fait à la fenêtre TikZ *portée* par le rectangle de coins (xmin-Ox; ymin-Oy) et (xmax-Ox; ymax-Oy).

Les commandes ont – pour certaines – pas mal de **<clés>** pour des réglages fins, mais dans la majorité des cas elles ne sont pas forcément *utiles*.

Code L^AT_EX

```
%...code tikz
\GrilleTikz[options][options grille ppale][options grille second.]
```

Clés et options

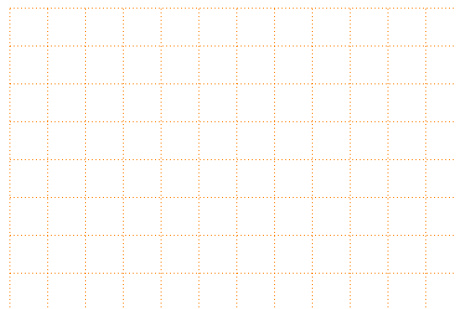
Cette commande permet de tracer une grille principale et/ou une grille secondaire :

- les premières **<clés>** sont les booléens **<Affp>** et **<Affs>** qui affichent ou non les grilles; défaut **<true>**
- les options des grilles sont en TikZ. défaut **<thin,lightgray>** et **<very thin,lightgray>**

Code L^AT_EX

```
\begin{tikzpicture}%
[x=0.1cm,y=0.0167cm, %unités
xmin=0,xmax=60,xgrille=5,xgrilles=5, %axe Ox
ymin=0,ymax=240,ygrille=30,ygrilles=30] %axe Oy
\GrilleTikz
\end{tikzpicture}
~~
\begin{tikzpicture}%
[x=0.1cm,y=0.0167cm, %unités
xmin=0,xmax=60,xgrille=5,xgrilles=5, %axe Ox
ymin=0,ymax=240,ygrille=30,ygrilles=30] %axe Oy
\GrilleTikz[Affp=false] [] [orange,densely dotted]
\end{tikzpicture}
```

Sortie L^AT_EX



</> Code L^AT_EX

```
%...code tikz
\AxesTikz[options]
```

🔑 Clés et options

Cette commande permet de tracer les axes, avec des **<clés>** :

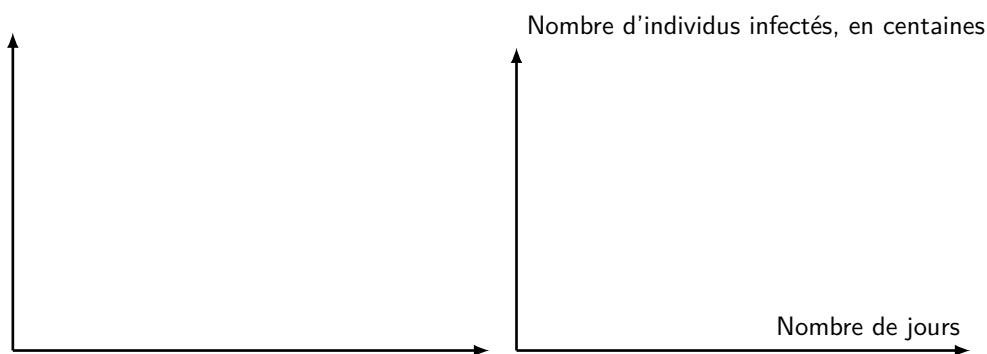
- **<Epaisseur>** qui est l'épaisseur des axes; défaut **<1pt>**
- **<Police>** qui est le style des labels des axes; défaut **<\normalsize\normalfont>**
- **<ElargirOx>** qui est le % l'élargissement **<global>** ou **<G/D>** de l'axe (Ox); défaut **<0/0.05>**
- **<ElargirOy>** qui est le % l'élargissement **<global>** ou **<B/H>** de l'axe (Oy); défaut **<0/0.05>**
- **<Labelx>** qui est le label de l'axe (Ox); défaut **<\$x\$>**
- **<Labely>** qui est le label de l'axe (Oy); défaut **<\$y\$>**
- **<AffLabel>** qui est le code pour préciser quels labels afficher, entre **<x>**, **<y>** ou **<xy>**; défaut **<vide>**
- **<PosLabelx>** pour la position du label de (Ox) en bout d'axe; défaut **<right>**
- **<PosLabely>** pour la position du label de (Oy) en bout d'axe; défaut **<above>**
- **<EchelleFleche>** qui est l'échelle de la flèche des axes; défaut **<1>**
- **<TypeFleche>** qui est le type de la flèche des axes. défaut **<latex>**

</> Code L^AT_EX

```
%code tikz
\AxesTikz

%code tikz
\AxesTikz%
[AffLabel=xy,Labelx={Nombre de jours},Labely={Nombre d'individus infectés, en centaines},%
PosLabelx={above left},PosLabely={above right},%
Police=\small\sffamily,ElargirOx=0,ElargirOy=0]
```

➦ Sortie L^AT_EX



</> Code L^AT_EX

```
%...code tikz
\AxexTikz[options]{valeurs}
\AxeYtikz[options]{valeurs}
```

Clés et options

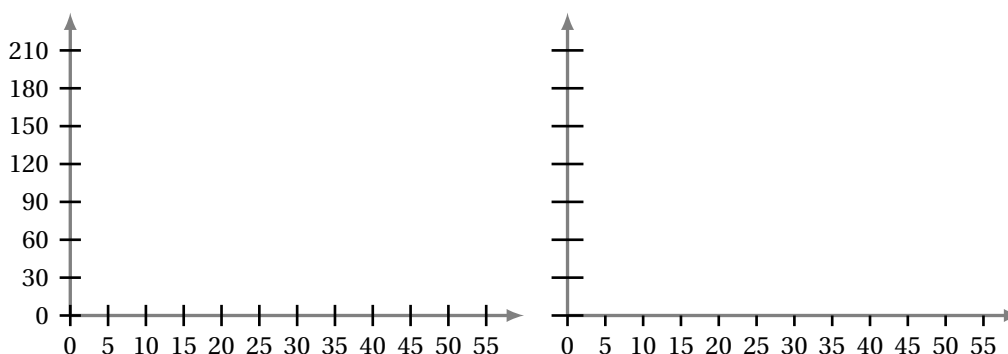
Ces commande permet de tracer les graduations des axes, avec des **clés** identiques pour les deux directions :

- **<Epaisseur>** qui est l'épaisseur des graduations; défaut **<1pt>**
- **<Police>** qui est le style des labels des graduations; défaut **<\normalsize\normalfont>**
- **<PosGrad>** qui est la position des graduations par rapport à l'axe; défaut **<below>** et **<left>**
- **<HautGrad>** qui est la position des graduations (sous la forme **<lgt>** ou **<lgtalgtb>**); défaut **<4pt>**
- le booléen **<AffGrad>** pour afficher les valeurs (formatés avec `\num` donc dépendant de `\sisetup`) des graduations; défaut **<true>**
- le booléen **<AffOrigine>** pour afficher la graduation de l'origine; défaut **<true>**
- le booléen **<Annee>** qui permet de ne pas formater les valeurs des graduations (type année); défaut **<false>**
- `\2.5.6` le booléen **<Trigo>** (uniquement pour l'axe (Ox)) pour des graduations libres en radians; défaut **<false>**
- `\2.5.6` le booléen **<Dfrac>** (uniquement pour l'axe (Ox) en **<Trigo>**) pour forcer les fractions en *grand*. défaut **<false>**

Code \LaTeX

```
%code tikz
\AxexTikz[Police=\small]{0,5,...,55}
\AxyTikz[Police=\small]{0,30,...,210}
%code tikz
\AxexTikz[Police=\small,HautGrad=0pt/4pt]{0,5,...,55}
\AxyTikz[AffGrad=false,HautGrad=6pt]{0,30,...,210}
%des axes fictifs (en gris) sont rajoutés pour la lisibilité du code de sortie
```

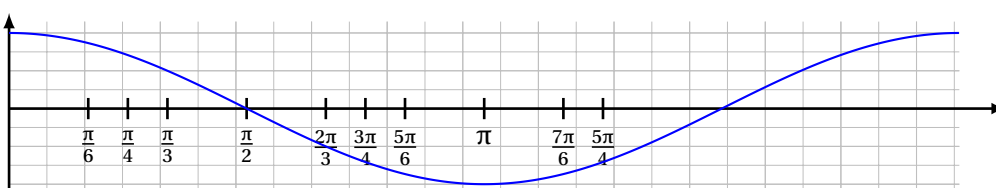
Sortie \LaTeX



Code \LaTeX

```
\begin{tikzpicture}[x=2cm,y=1cm,xmin=0,xmax={2*pi},xgrille=0.5,xgrilles=0.25,
  ymin=-1.15,ymax=1.15,ygrille=0.5,ygrilles=0.25]
  \GrilleTikz \AxesTikz
  \AxexTikz[Trigo]{\pi/6,\pi/4,\pi/3,\pi/2,{2*\pi/3},{3*\pi/4},{5*\pi/6},\pi,{7*\pi/6},{5*\pi/4}}
  \CourbeTikz[thick,blue,samples=250]{cos(deg(\x))}{0:2*pi}
\end{tikzpicture}
```

Sortie \LaTeX



Information(s)

La clé **⟨Trigo⟩** utilise, en interne, une commande qui permet de *transformer* les abscisses, données en langage TikZ, en fraction en \LaTeX .

Code \LaTeX

```
 $\backslash\text{AffAngleRadian}\{0\}\$ \quad \backslash\text{AffAngleRadian}\{\pi\}\$ \quad \backslash\text{AffAngleRadian}\{\pi/4\}\$ \quad \backslash\text{AffAngleRadian}\{2*\pi/3\}\$ \quad \backslash\text{AffAngleRadian}\{-2*\pi/3\}\$ \quad \backslash\text{AffAngleRadian}\{-2*\pi/3\}\$$ 
```

$$0 \quad \pi \quad \frac{\pi}{4} \quad \frac{2\pi}{3} \quad -\frac{2\pi}{3} \quad -\frac{2\pi}{3}$$

6.3 Commandes annexes

Information(s)

Il existe, de manière marginale, quelques commandes complémentaires qui ne seront pas trop détaillées mais qui sont existantes :

- \LaTeX `FenetreTikz` qui restreint les tracés à la fenêtre (utile pour des courbes qui *débordent*);
- \LaTeX `FenetreSimpleTikz` qui permet d'automatiser le tracé des grilles/axes/graduations dans leurs versions par défaut, avec peu de paramétrages;
- \LaTeX `OrigineTikz` pour rajouter le libellé de l'origine si non affiché par les axes.

Code \LaTeX

```
%code tikz
\text{FenetreTikz} \quad \text{\textit{on restreint les tracés}}
\text{FenetreSimpleTikz}\%
[options](opt axes)<opt axe Ox>\{valeurs Ox\}<opt axe Oy>\{valeurs Oy\}
```

Idée(s)

L'idée est de proposer, en *complément*, une commande simplifiée pour tracer une courbe en TikZ.

Code \LaTeX

```
%...code tikz
\text{CourbeTikz}[options]\{formule\}\{domaine\}
```

Clés et options

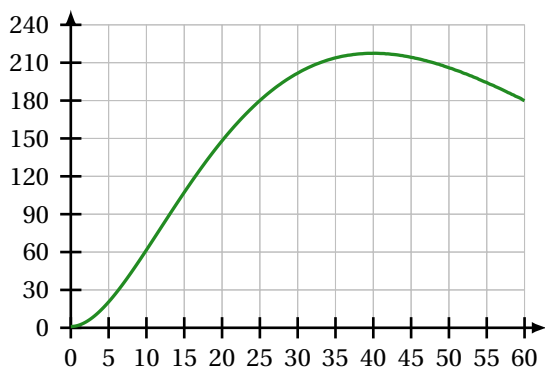
Cette commande permet de rajouter une courbe sur le graphique (sans se soucier de la transformation de son expression) avec les arguments :

- **⟨optionnels⟩** qui sont - en TikZ - les paramètres du tracé;
- le premier *obligatoire*, est - en langage TikZ - l'expression de la fonction à tracer, donc avec \LaTeX `\x` comme variable;
- le second *obligatoire* est le domaine du tracé, sous la forme \LaTeX `valxmin:valxmax`.

Code \LaTeX

```
\begin{tikzpicture}[x=0.1cm,y=0.0167cm, %unités
xmin=0,xmax=60,xgrille=5,xgrilles=5, %axe Ox
ymin=0,ymax=240,ygrille=30,ygrilles=30] %axe Oy
\text{FenetreSimpleTikz}\%
<Police=\small>\{0,5,...,60\}\%
<Police=\small>\{0,30,...,240\} %repère
\text{CourbeTikz}[line width=1.25pt,ForestGreen,samples=250]\%
\{x*\text{x}\exp(-0.05*\text{x})+1\}\{0:60\} %courbe
\end{tikzpicture}
```


Sortie \LaTeX



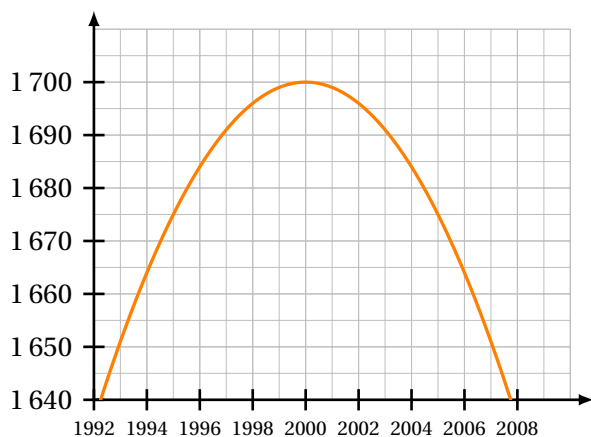
6.4 Repère non centré en O

Idée(s)

Parfois on est amené à travailler dans des repères qui n'ont pas forcément pour origine (0 ; 0). De ce fait - pour éviter des erreurs de `dimension too large` liées à TikZ - il faut *décaler les axes* pour se ramener à une origine en O. L'idée est donc d'utiliser les commandes précédentes, sans se soucier des éventuelles transformations!

Code \LaTeX

```
\begin{tikzpicture}[x=0.35cm,y=0.07cm,Ox=1992,xmin=1992,xmax=2010,%
  xgrille=2,xgrilles=1,Oy=1640,ymin=1640,ymax=1710,ygrille=10,ygrilles=5]
  \FenetreSimpleTikz<Annee,Police=\scriptsize>{1992,1994,...,2008}{1640,1650,...,1700}
  \FenetreTikz
  \CourbeTikz[line width=1.25pt,orange,samples=500]{-(\x-2000)*(\x-2000)+1700}{\xmin:\xmax}
\end{tikzpicture}
```



7 L'outil « SplineTikz »

7.1 Courbe d'interpolation

Information(s)

On va utiliser les notions suivantes pour paramétrer le tracé « automatique » grâce à `\splinecontrols` :

- il faut rentrer les **points de contrôle**;
- il faut préciser les **pentés des tangentes** (pour le moment on travaille avec les mêmes à gauche et à droite...);
- on peut « affiner » les portions de courbe en paramétrant des **coefficients** (voir un peu plus loin...).

Pour déclarer les paramètres :

- liste des points de contrôle (minimum 2!!) par : $x_1/y_1/d_1$ $x_2/y_2/d_2$. . . avec les points $(x_i; y_i)$ et $f'(x_i)=d_i$;
- coefficients de contrôle par `coeffs=...` :
 - `coeffs=x` pour mettre tous les coefficients à x ;
 - `coeffs=C1$C2$...` pour spécifier les coefficients par portion (donc il faut avoir autant de $\$$ que pour les points!);
 - `coeffs=C1G/C1D$...` pour spécifier les coefficients par portion et par partie gauche/droite;
 - on peut mixer avec `coeffs=C1$C2G/C2D$...`.

7.2 Code, clés et options

Code \LaTeX

```
\begin{tikzpicture}
...
\SplineTikz[options]{liste}
...
\end{tikzpicture}
```

Clés et options

Certains paramètres et **clés** peuvent être gérés directement dans la commande `\splinetikz` :

- la couleur de la courbe par la clé **⟨Couleur⟩**; défaut **⟨red⟩**
- l'épaisseur de la courbe par la clé **⟨Epaisseur⟩**; défaut **⟨1.25pt⟩**
- du style supplémentaire pour la courbe peut être rajouté, grâce à la clé **⟨Style⟩**; défaut **⟨vide⟩**
- les coefficients de *compensation* gérés par la clé **⟨Coeffs⟩**; défaut **⟨3⟩**
- les points de contrôle, affichés ou non par la clé booléenne **⟨AffPoints⟩**; défaut **⟨false⟩**
- la taille des points de contrôle est gérée par la clé **⟨TaillePoints⟩**. défaut **⟨2pt⟩**

7.3 Compléments sur les coefficients de « compensation »

Idée(s)

Le choix a été fait ici, pour *simplifier* le code, le travailler sur des courbes de Bézier.

Pour *simplifier* la gestion des nombres dérivés, les points de contrôle sont gérés par leurs coordonnées *polaires*, les coefficients de compensation servent donc – grosso modo – à gérer la position radiale.

Le coefficient **⟨3⟩** signifie que, pour une courbe de Bézier entre $x = a$ et $x = b$, les points de contrôle seront situés à une distance radiale de $\frac{b-a}{3}$.

Pour *écarter* les points de contrôle, on peut du coup *réduire* le coefficient de compensation!

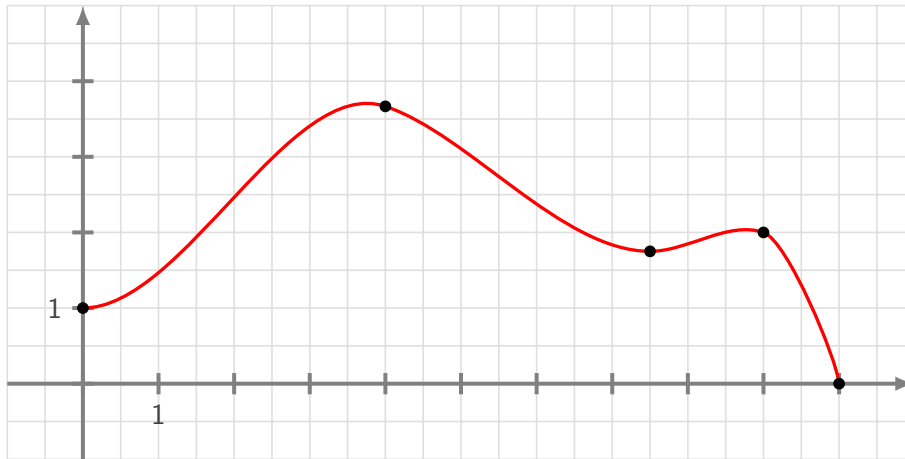
Pour des intervalles *étroits*, la *pente* peut paraître abrupte, et donc le(s) coefficient(s) peuvent être modifiés, de manière fine.

Si jamais il existe (un ou) des points *anguleux*, le plus simple est de créer les splines en plusieurs fois.

7.4 Exemples

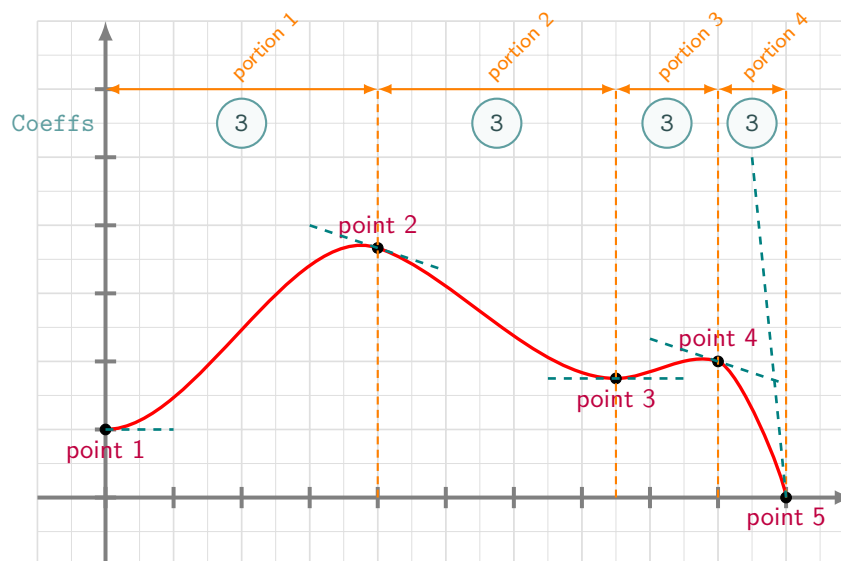
Code \LaTeX

```
%code tikz
\def\x{0.9cm}\def\y{0.9cm}
\def\xmin{-1}\def\xmax{11}\def\xgrille{1}\def\xgrilles{0.5}
\def\ymin{-1}\def\ymax{5}\def\ygrille{1}\def\ygrilles{0.5}
%axes et grilles
\draw[xstep=\xgrilles,ystep=\ygrilles,line width=0.6pt,lightgray!50] (\xmin,\ymin) grid
  - (\xmax,\ymax);
\draw[line width=1.5pt,->,gray,>=latex] (\xmin,0)--(\xmax,0) ;
\draw[line width=1.5pt,->,gray,>=latex] (0,\ymin)--(0,\ymax) ;
\foreach \x in {0,1,...,10} {\draw[gray,line width=1.5pt] (\x,4pt) -- (\x,-4pt) ;}
\foreach \y in {0,1,...,4} {\draw[gray,line width=1.5pt] (4pt,\y) -- (-4pt,\y) ;}
\draw[darkgray] (1,-4pt) node[below,font=\sffamily] {1} ;
\draw[darkgray] (-4pt,1) node[left,font=\sffamily] {1} ;
%splines
\def\LISTE{0/1/0$4/3.667/-0.333$7.5/1.75/0$9/2/-0.333$10/0/-10}
\SplineTikz[AffPoints,Coeffs=3,Couleur=red]{\LISTE}
```



Information(s)

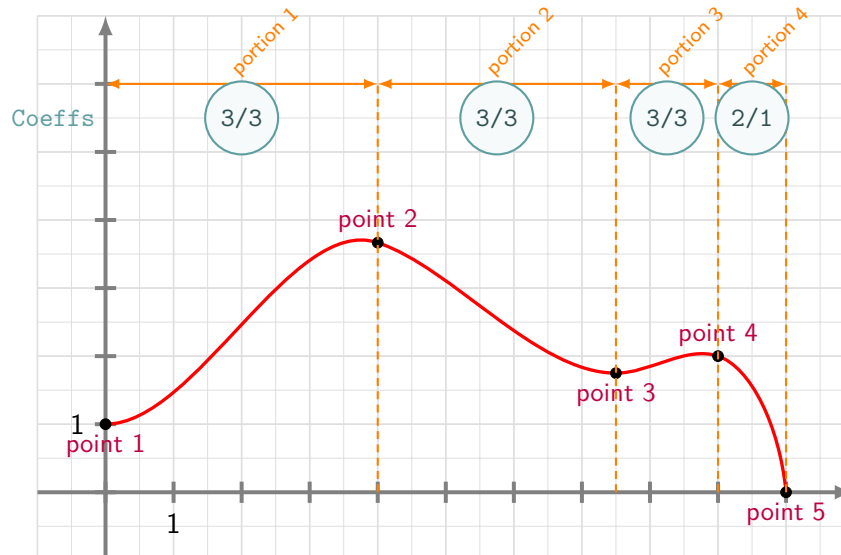
Avec des explications utiles à la compréhension :



7.5 Avec une gestion plus fine des « coefficients »

Information(s)

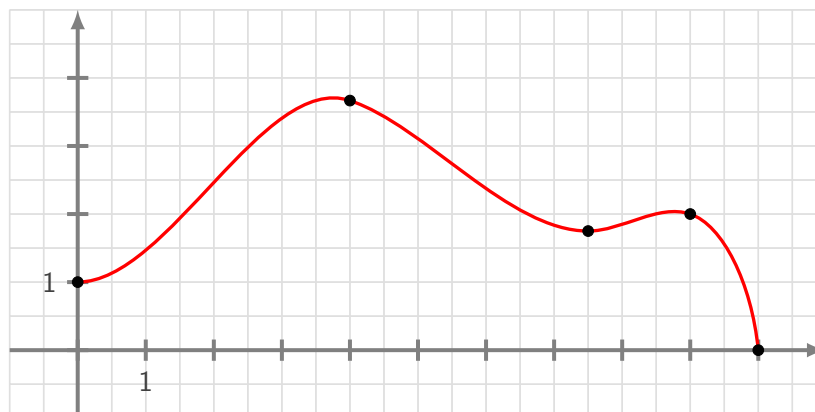
Dans la majorité des cas, le *coefficient* ③ permet d'obtenir une courbe (ou une portion) très satisfaisante! Dans certains cas, il se peut que la portion paraisse un peu trop « abrupte ». On peut dans ce cas *jouer* sur les coefficients de cette portion pour *arrondir* un peu tout cela (ie diminuer le coeff...)!



Code \LaTeX

```
...
%splines
\def\LISTE{0/1/0$4/3.667/-0.333$7.5/1.75/0$9/2/-0.333$10/0/-10}
\SplineTikz[AffPoints,Coeffs=3$3$3$2/1]{\LISTE}
...
```

Sortie \LaTeX



7.6 Conclusion

Information(s)

Le plus « simple » est donc :

- de déclarer la liste des points de contrôle, grâce à `\def\LISTE{x1/y1/d1$x2/y2/d2$...}`;
- de saisir la commande `\SplineTikz[...]{\LISTE}`;
- d'ajuster les options et coefficients en fonction du rendu!

8 L'outil « TangenteTikz »

8.1 Définitions

💡 Idée(s)

En parallèle de l'outil `\SplineTikz`, il existe l'outil `\TangenteTikz` qui va permettre de tracer des tangentes à l'aide de la liste de points précédemment définie pour l'outil `\SplineTikz`.

NB : il peut fonctionner indépendamment de l'outil `\SplineTikz` puisque la liste des points de travail est gérée de manière autonome !

🔗 Code \LaTeX

```
\begin{tikzpicture}
...
\TangenteTikz[options]{liste}
...
\end{tikzpicture}
```

🔑 Clés et options

Cela permet de tracer la tangente :

- au point numéro **⟨Point⟩** de la liste **⟨liste⟩**, de coordonnées x_i/y_i avec la pente d_i ;
- avec une épaisseur de **⟨Épaisseur⟩**, une couleur **⟨Couleur⟩** et un style additionnel **⟨Style⟩**;
- en la traçant à partir de **⟨xl⟩** avant x_i et jusqu'à **⟨xr⟩** après x_i .

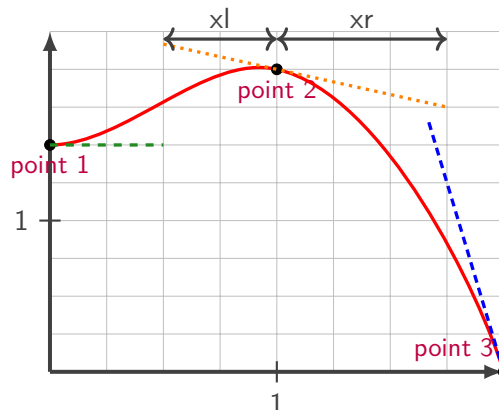
8.2 Exemple et illustration

🔗 Code \LaTeX

```
\begin{tikzpicture}
...
\def\LISTE{0/1.5/0$1/2/-0.333$2/0/-5}
% spline
\SplineTikz[AffPoints,Coeffs=3$2,Couleur=red]{\LISTE}
% tangente
\TangenteTikz[xl=0,xr=0.5,Couleur=ForestGreen,Style=dashed]{\LISTE}
\TangenteTikz[xl=0.5,xr=0.75,Couleur=orange,Style=dotted,Point=2]{\LISTE}
\TangenteTikz[xl=0.33,xr=0,Couleur=blue,Style=densely dashed,Point=3]{\LISTE}
...
\end{tikzpicture}
```

📄 Sortie \LaTeX

On obtient le résultat suivant (avec les éléments rajoutés utiles à la compréhension) :



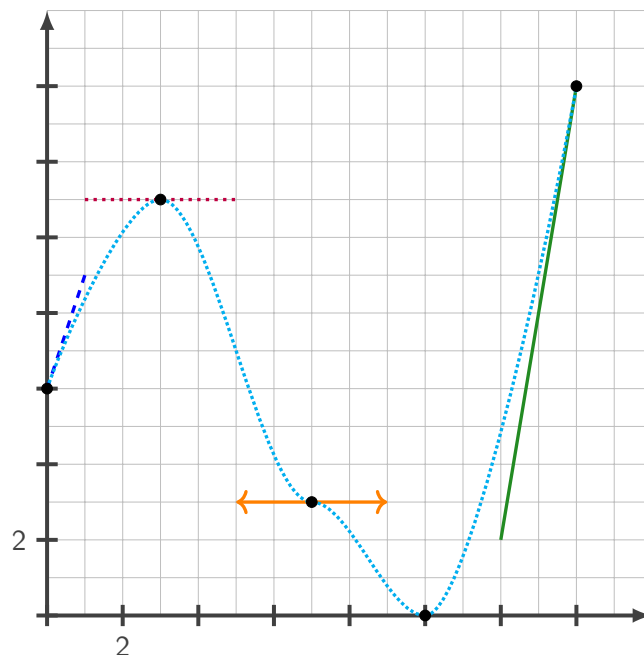
8.3 Exemple avec les deux outils, et « personnalisation »

Code \LaTeX

```
\tikzset{%
  xmin/.store in=\xmin,xmin/.default=-5,xmin=-5,
  xmax/.store in=\xmax,xmax/.default=5,xmax=5,
  ymin/.store in=\ymin,ymin/.default=-5,ymin=-5,
  ymax/.store in=\ymax,ymax/.default=5,ymax=5,
  xgrille/.store in=\xgrille,xgrille/.default=1,xgrille=1,
  xgrilles/.store in=\xgrilles,xgrilles/.default=0.5,xgrilles=0.5,
  ygrille/.store in=\ygrille,ygrille/.default=1,ygrille=1,
  ygrilles/.store in=\ygrilles,ygrilles/.default=0.5,ygrilles=0.5,
  xunit/.store in=\xunit,unit/.default=1,xunit=1,
  yunit/.store in=\yunit,unit/.default=1,yunit=1
}

\begin{tikzpicture}[x=0.5cm,y=0.5cm,xmin=0,xmax=16,xgrilles=1,ymin=0,ymax=16,ygrilles=1]
  \draw[xstep=\xgrilles,ystep=\ygrilles,line width=0.3pt,lightgray] (\xmin,\ymin) grid
    - (\xmax,\ymax) ;
  \draw[line width=1.5pt,->,darkgray,>=latex] (\xmin,0)--(\xmax,0) ;
  \draw[line width=1.5pt,->,darkgray,>=latex] (0,\ymin)--(0,\ymax) ;
  \foreach \x in {0,2,...,14} {\draw[darkgray,line width=1.5pt] (\x,4pt) -- (\x,-4pt) ;}
  \foreach \y in {0,2,...,14} {\draw[darkgray,line width=1.5pt] (4pt,\y) -- (-4pt,\y) ;}
  %la liste pour la courbe d'interpolation
  \def\liste{0/6/3$3/11/0$7/3/0$10/0/0$14/14/6}
  %les tangentes "stylis  es"
  \TangenteTikz[xl=0,xr=1,Couleur=blue,Style=dashed]{\liste}
  \TangenteTikz[xl=2,xr=2,Couleur=purple,Style=dotted,Point=2]{\liste}
  \TangenteTikz[xl=2,xr=2,Couleur=orange,Style=<->,Point=3]{\liste}
  \TangenteTikz[xl=2,xr=0,Couleur=ForestGreen,Point=5]{\liste}
  %la courbe en elle-m  me
  \SplineTikz[AffPoints,Coeffs=3,Couleur=cyan,Style=densely dotted]{\liste}
\end{tikzpicture}
```

Sortie \LaTeX



9 Petits schémas pour le signe d'une fonction affine ou d'un trinôme

9.1 Idée

💡 Idée(s)

L'idée est d'obtenir une commande pour tracer (en TikZ) un petit schéma pour *visualiser* le signe d'une fonction affine ou d'un trinôme.

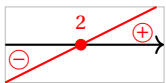
Le code est largement inspiré de celui du package `\tnsana` même si la philosophie est un peu différente.

Comme pour les autres commandes TikZ, l'idée est de laisser la possibilité à l'utilisateur de définir et créer son environnement TikZ, et d'insérer la commande `\MiniSchemaSignes` pour afficher le schéma.

2.1.9 Il est à noter que la version *étoilée* rend la commande autonome, sans besoin de créer l'environnement TikZ.

🔗 Code \LaTeX

```
\MiniSchemaSignes*
```



9.2 Commandes

🔗 Code \LaTeX

```
\begin{tikzpicture}[<options>]
\MiniSchemaSignes[clés]
\end{tikzpicture}
```

🔗 Code \LaTeX

```
{\tikz[options] \MiniSchemaSignes[clés]}
%ou
\MiniSchemaSignes*[clés]<options tikzpicture>
```

🔗 Clés et options

2.1.9 La version *étoilée* de la commande permet de basculer en mode *autonome*, c'est-à-dire sans avoir besoin de créer son environnement TikZ.

Le premier argument, *optionnel* et entre [...], contient les **(Clés)** sont disponibles pour cette commande :

- la clé **(Code)** qui permet de définir le type d'expression (voir en-dessous); défaut **(da+)**
- la clé **(Couleur)** qui donne la couleur de la représentation; défaut **(red)**
- la clé **(Racines)** qui définit la ou les racines; défaut **(2)**
- la clé **(Largeur)** qui est la largeur du schéma; défaut **(2)**
- la clé **(Hauteur)** qui est la hauteur du schéma; défaut **(1)**
- un booléen **(Cadre)** qui affiche un cadre autour du schéma. défaut **(true)**

Le second argument, *optionnel* et entre <...>, permet de spécifier (pour la commande *étoilée*), des options à passer à l'environnement `\tikzpicture`.

🔗 Clés et options

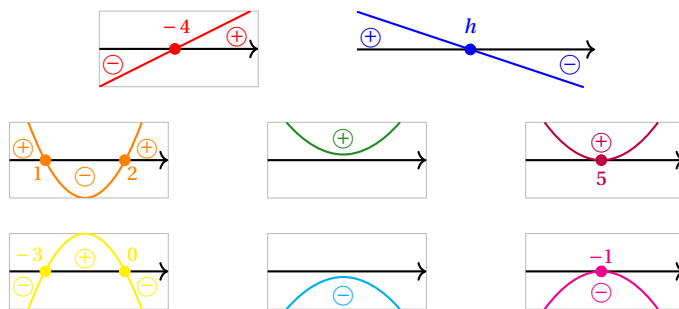
Pour la clé **(code)**, il est construit par le type (a pour affine ou p comme parabole) puis les éléments caractéristiques (a+ pour $a > 0$, d0 pour $\Delta = 0$, etc) :

- **(Code=da+)** := une droite croissante;
- **(Code=da-)** := une droite décroissante;
- **(Code=pa+d+)** := une parabole *souriante* avec deux racines;
- etc

Code \LaTeX

```
\begin{center}
\MiniSchemaSignes*[Code=da+,Racines=-4]
~~~~~
\MiniSchemaSignes*[Code=da-,Racines={h},Couleur=blue,Largeur=3,Cadre=false]
\end{center}
%
\begin{center}
\MiniSchemaSignes*[Code=pa+d+,Racines={1/2},Couleur=orange]
~~~~~
\MiniSchemaSignes*[Code=pa+d-,Couleur=ForestGreen]
~~~~~
\MiniSchemaSignes*[Code=pa+d0,Racines={5},Couleur=purple]
\end{center}
%
\begin{center}
\MiniSchemaSignes*[Code=pa-d+,Racines={-3/0},Couleur=yellow]
~~~~~
\MiniSchemaSignes*[Code=pa-d-,Couleur=cyan]
~~~~~
\MiniSchemaSignes*[Code=pa-d0,Racines={-1},Couleur=magenta]
\end{center}
```

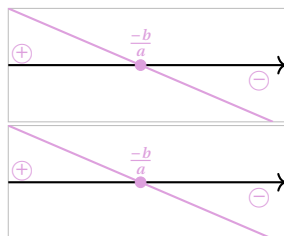
Sortie \LaTeX



Code \LaTeX

```
\begin{tikzpicture}
\MiniSchemaSignes[Largeur=3.5,Hauteur=1.5,Code=da-,Racines=\tfrac{-b}{a},Couleur=Plum]
\end{tikzpicture}

\MiniSchemaSignes*[Code=da-,Racines=\tfrac{-b}{a},Couleur=Plum]<x=1.75cm,y=1.5cm>
```



9.3 Intégration avec tkz-tab

💡 Idée(s)

Ces schémas peuvent être de plus utilisés, via la commande `\MiniSchemaSignesTkzTab` pour illustrer les signes obtenus dans un tableau de signes présentés grâce au package `tkz-tab`.
Pour des raisons internes, le fonctionnement de la commande `\MiniSchemaSignesTkzTab` est légèrement différent et, pour des raisons que j'ignore, le code est légèrement différent en *interne* (avec une *déconnexion* des caractères : et \) pour que la librairie TikZ `calc` puisse fonctionner (mystère pour le moment...)

🔗 Code \LaTeX

```
\begin{tikzpicture}
  %commandes tkztab
  \MiniSchemaSignesTkzTab[options]{numligne}[echelle][décalage horizontal]
\end{tikzpicture}
```

🔑 Clés et options

Les **⟨Clés⟩** pour le premier argument *optionnel* sont les mêmes que pour la version *initiale* de la commande précédente.

En ce qui concerne les autres arguments :

- le deuxième argument, *obligatoire*, est le numéro de la ligne à côté de laquelle placer le schéma ;
- le troisième argument, *optionnel* et valant **⟨0.85⟩** par défaut, est l'échelle à appliquer sur l'ensemble du schéma (à ajuster en fonction de la hauteur de la ligne) ;
- le quatrième argument, *optionnel* et valant **⟨1.5⟩** par défaut, est lié à l'écart horizontal entre le bord de la ligne du tableau et le schéma.

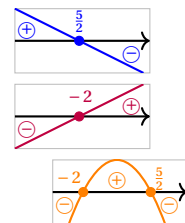
À noter que si l'un des arguments optionnels (le n°3 et/ou le n°4) sont utilisés, il vaut mieux préciser les 2 !

🔗 Code \LaTeX

```
\begin{center}
\begin{tikzpicture}
  \tkzTabInit[]{$x$/1,$-2x+5$/1,$2x+4$/1,$p(x)$/1}{$-\infty$,$-2$,$2,5$,$+\infty$}
  \tkzTabLine{+,t,+,z,-,}
  \tkzTabLine{-,z,+,t,+,}
  \tkzTabLine{-,z,+,z,-,}
  \MiniSchemaSignesTkzTab[Code=da-,Racines={\tfrac{5}{2}},Couleur=blue]{1}
  \MiniSchemaSignesTkzTab[Code=da+,Racines={-2},Couleur=purple]{2}
  \MiniSchemaSignesTkzTab[Code=pa-d+,Racines={-2/{\tfrac{5}{2}}},Couleur=orange]{3}[0.85][2]
\end{tikzpicture}
\end{center}
```

➡ Sortie \LaTeX

x	$-\infty$	-2	$2,5$	$+\infty$
$-2x + 5$	+	+	0	-
$2x + 4$	-	0	+	+
$p(x)$	-	0	+	-



10 Suites récurrentes et « toile »

10.1 Idée

💡 Idée(s)

L'idée est d'obtenir une commande pour tracer (en TikZ) la « toile » permettant d'obtenir – graphiquement – les termes d'une suite récurrente définie par une relation $u_{n+1} = f(u_n)$.

Comme pour les autres commandes TikZ, l'idée est de laisser l'utilisateur définir et créer son environnement TikZ, et d'insérer la commande `\ToileRecurrence` pour afficher la « toile ».

10.2 Commandes

🔗 Code \LaTeX

```
...
\begin{tikzpicture}[options]
...
\ToileRecurrence[clés][options du tracé][options supplémentaires des termes]
...
\end{tikzpicture}
```

🔗 Clés et options

Plusieurs **arguments** (optionnels) sont disponibles :

- le premier argument optionnel définit les **Clés** de la commande :
 - la clé **Fct** qui définit la fonction f ; défaut **vide**
 - la clé **Nom** qui est le *nom* de la suite; défaut **u**
 - la clé **No** qui est l'indice initial; défaut **0**
 - la clé **Uno** qui est la valeur du terme initial; défaut **vide**
 - la clé **Nb** qui est le nombre de termes à construire; défaut **5**
 - la clé **PosLabel** qui est le placement des labels par rapport à l'axe (Ox); défaut **below**
 - la clé **DecalLabel** qui correspond au décalage des labels par rapport aux abscisses; défaut **6pt**
 - la clé **TailleLabel** qui correspond à la taille des labels; défaut **small**
 - un booléen **AffTermes** qui permet d'afficher les termes de la suite sur l'axe (Ox). défaut **true**
- le deuxième argument optionnel concerne les **options** du tracé de l'escalier en *langage TikZ*; défaut **thick,color=magenta**;
- le troisième argument optionnel concerne les **options** du tracé des termes en *langage TikZ*. défaut **dotted**.

🔗 Information(s)

Il est à noter que le code n'est pas autonome, et doit être intégré dans un environnement `\tikzpicture`.

L'utilisateur est donc libre de définir ses styles pour l'affichage des éléments de son graphique, et il est libre également de rajouter des éléments en plus du tracé de la *toile*!

La macro ne permet – pour le moment – ni de tracer la bissectrice, ni de tracer la courbe...

En effet, il y aurait trop d'options pour ces deux éléments, et l'idée est quand même de conserver une commande *simple*! Donc l'utilisateur se chargera de tracer et de personnaliser sa courbe et sa bissectrice!

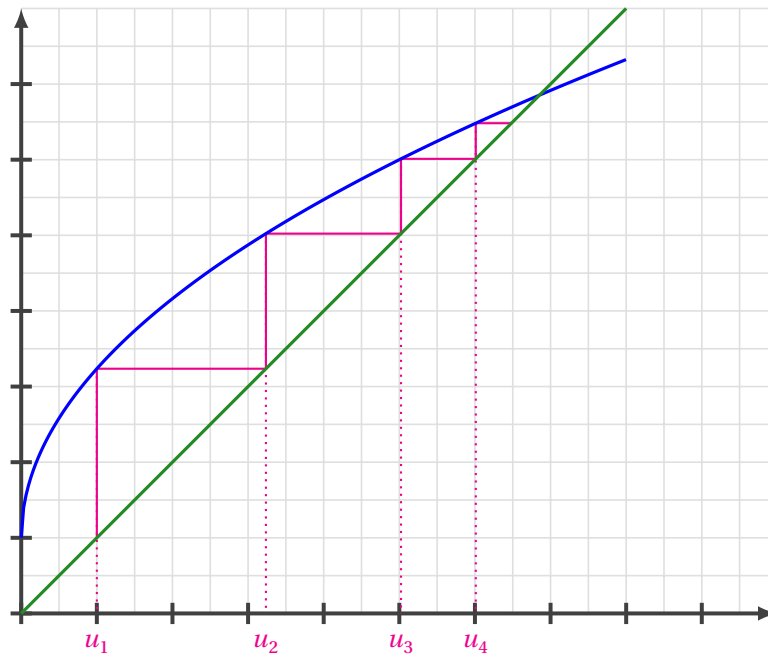
10.3 Exemples

🔗 Information(s)

On va tracer la *toile* des 4 premiers termes de la suite récurrente
$$\begin{cases} u_1 = 1 \\ u_{n+1} = \sqrt{5u_n} + 1 \text{ pour tout entier } n \geq 1 \end{cases}$$

</> Code \LaTeX

```
%code tikz
\def\x{1.5cm}\def\y{1.5cm}
\def\xmin{0}\def\xmax{10}\def\xgrille{1}\def\xgrilles{0.5}
\def\ymin{0}\def\ymax{8}\def\ygrille{1}\def\ygrilles{0.5}
%axes et grilles
\draw[xstep=\xgrilles,ystep=\ygrilles,line width=0.6pt,lightgray!50] (\xmin,\ymin) grid
-- (\xmax,\ymax);
\draw[line width=1.5pt,->,darkgray,>=latex] (\xmin,0)--(\xmax,0) ;
\draw[line width=1.5pt,->,darkgray,>=latex] (0,\ymin)--(0,\ymax) ;
\foreach \x in {0,1,...,9} {\draw[darkgray,line width=1.5pt] (\x,4pt) -- (\x,-4pt) ;}
\foreach \y in {0,1,...,7} {\draw[darkgray,line width=1.5pt] (4pt,\y) -- (-4pt,\y) ;}
%fonction définie et réutilisable
\def\f{sqrt(5*\x)+1}
%toile
\ToileRecurrence[Fct={\f},No=1,Uno=1,Nb=4,DecalLabel=4pt]
%éléments supplémentaires
\draw[very thick,blue,domain=0:8,samples=250] plot (\x,{\f}) ;
\draw[very thick,ForestGreen,domain=0:8,samples=2] plot (\x,\x) ;
```



Information(s)

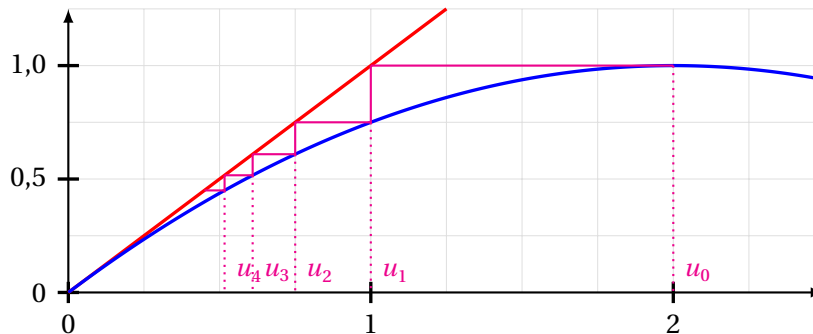
Peut-être que – ultérieurement – des options *booléennes* seront disponibles pour un tracé *générique* de la courbe et de la bissectrice, mais pour le moment la macro ne fait *que* l'escalier.

10.4 Influence des paramètres

Code \LaTeX

```
\begin{center}
\begin{tikzpicture}[x=4cm,y=3cm]
  %axes + grilles + graduations
  ...
  %fonction
  \def\f{-0.25*\x*\x+\x}
  %tracés
  \begin{scope}
    \clip (0,0) rectangle (2.5,1.25) ;
    \draw[line width=1.25pt,blue,domain=0:2.5,samples=200] plot (\x,{\f}) ;
  \end{scope}
  \ToileRecurrence[Fct={\f},No=0,Uno=2,Nb=5,PosLabel=above right,DecalLabel=0pt]
\end{tikzpicture}
\end{center}
```

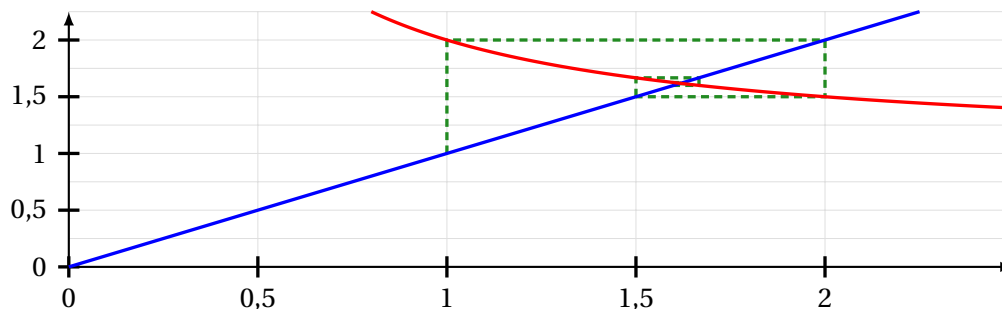
Sortie \LaTeX



Code \LaTeX

```
\begin{center}
\begin{tikzpicture}[x=5cm,y=1.5cm]
  ...
  \def\f{1+1/\x}
  \ToileRecurrence%
    [Fct={\f},No=0,Uno=1,Nb=7,PosLabel=above right,DecalLabel=0pt,AffTermes=false]%
    [line width=1.25pt,ForestGreen,densely dashed] []
  \draw[line width=1.25pt,blue,domain=0:2.25,samples=2] plot(\x,{\x});
  \draw[line width=1.25pt,red,domain=0.8:2.5,samples=250] plot(\x,{\f});
\end{tikzpicture}
\end{center}
```

Sortie \LaTeX



Cinquième partie

Présentation de codes

11 Code Python « simple » via le package listings

11.1 Introduction

💡 Idée(s)

Le package `listings` permet d'insérer et de formater du code, notamment du code Python. En *partenariat* avec `tcolorbox`, on peut donc présenter *joliment* du code Python!

🔧 Information(s)

Le package `listings` ne nécessite pas de compilation particulière, au contraire d'autres (comme `pythontex` ou `minted` ou `piton`) qui seront présentés ultérieurement.

🔧 Information(s)

Les styles utilisés pour formater le code Python ne sont pas modifiables. Ils donnent un rendu proche de celui des packages comme `pythontex` ou `minted` ou `piton`.

Donc, si plusieurs *méthodes* sont utilisées pour insérer du code Python (via les *méthodes* suivantes), le rendu pourra être légèrement différent.

11.2 Commande et options

💡 Idée(s)

L'environnement `CodePythonLst` permet de présenter du code Python, dans une `tcolorbox` avec deux styles particuliers (`2.5.8`).

🔗 Code \LaTeX

```
\begin{CodePythonLst}(*)[largeur]{commandes tcolorbox}
...
\end{CodePythonLst}
```

🔗 Code \LaTeX

```
\begin{CodePythonLstAlt}(*)[largeur]{commandes tcolorbox}
...
\end{CodePythonLstAlt}
```

🔗 Clés et options

Plusieurs **arguments** sont disponibles :

- la version *étoilée* qui permet de ne pas afficher les numéros de lignes;
- le premier argument (*optionnel*), concerne la **largeur** de la `tcolorbox`; défaut `\linewidth`;
- le second argument (*obligatoire*), concerne des **options** de la `tcolorbox` en *langage tcolorbox*, comme l'alignement.

⚠ Attention

Les environnements `DeclareTCBListing` créés par `tcolorbox` et `listings` ne sont pas compatibles avec les options **gobble** (pour supprimer les tabulations d'environnement), donc il faut bien penser à « aligner » le code à gauche, pour éviter des tabulations non esthétiques!

11.3 Insertion via un fichier « externe »

💡 Idée(s)

Pour des raisons pratiques, il est parfois intéressant d'avoir le code Python dans un fichier externe au fichier `tex`, ou bien créé directement par le fichier `tex` (via `scontents`, notamment, mais non chargé par `ProfLycee`).

Dans ce cas, il n'est pas nécessaire d'aligner le code « à gauche », en utilisant une commande alternative.

Si cette méthode est utilisée, il ne faut oublier de charger le package `scontents`.

🔗 Code \LaTeX

```
\usepackage{scontents} %si script déclaré dans le fichier tex
...
\CodePythonLstFichier(*)[largeur]{commandes tcbbox}{script}
```

11.4 Exemples

🔗 Code \LaTeX

```
\begin{CodePythonLst}{} %les {}, même vides, sont nécessaires (bug avec # sinon !)
#environnement par défaut
nb = int(input("Saisir un entier positif"))
if (nb %7 == 0) :
    print(f"{nb} est bien divisible par 7")
#endif

def f(x) :
    return x**2
\end{CodePythonLst}
```

➡ Sortie \LaTeX

```
1 #environnement par défaut
2 nb = int(input("Saisir un entier positif"))
3 if (nb %7 == 0) :
4     print(f"{nb} est bien divisible par 7")
5 #endif
6
7 def f(x) :
8     return x**2
```

Code Python

🔗 Code \LaTeX

```
\begin{CodePythonLstAlt}*[0.75\linewidth]{flush right}
#largeur de 50%, sans numéro, et aligné à droite
nb = int(input("Saisir un entier Python positif"))
if (nb %7 == 0) :
    print(f"{nb} est bien divisible par 7")
#endif

def f(x) :
    return x**2
\end{CodePythonLstAlt}
```

➔ Sortie L^AT_EX

⌘ Code Python

```
#largeur de 50%, sans numéro, et aligné à droite
nb = int(input("Saisir un entier Python positif"))
if (nb %7 == 0) :
    print(f"{nb} est bien divisible par 7")
#endif

def f(x) :
    return x**2
```

⌘ Code L^AT_EX

```
\begin{scontents}[overwrite,write-out=testscript.py]
# Calcul de la factorielle en langage Python
def factorielle(x):
    if x < 2:
        return 1
    else:
        return x * factorielle(x-1)

# rapidité de tracé
import matplotlib.pyplot as plt
import time
def trace_parabole_tableaux():
    depart=time.clock()
    X = [] # Initialisation des listes
    Y = []
    a = -2
    h = 0.001
    while a<2:
        X.append(a) # Ajout des valeurs
        Y.append(a*a) # au "bout" de X et Y
        a = a+h
    # Tracé de l'ensemble du tableau de valeurs
    plt.plot(X,Y,".b")
    fin=time.clock()
    return "Temps : " + str(fin-depart) + " s."
\end{scontents}

%environnement centré, avec numéros, largeur 9cm
\CodePythonLstFichier[9cm]{center}{testscript.py}
```

```

1  # Calcul de la factorielle en langage Python
2  def factorielle(x):
3      if x < 2:
4          return 1
5      else:
6          return x * factorielle(x-1)
7
8  # rapidité de tracé
9  import matplotlib.pyplot as plt
10 import time
11 def trace_parabole_tableaux():
12     depart=time.clock()
13     X = [] # Initialisation des listes
14     Y = []
15     a = -2
16     h = 0.001
17     while a<2:
18         X.append(a) # Ajout des valeurs
19         Y.append(a*a) # au "bout" de X et Y
20         a = a+h
21     # Tracé de l'ensemble du tableau de
        valeurs
22     plt.plot(X,Y,".b")
23     fin=time.clock()
24     return "Temps : " + str(fin-depart) + "
        s."

```


12 Code Python via le package piton

12.1 Introduction

Information(s)

2.5.0 Cette section nécessite de charger la librairie `piton` dans le préambule.
2.5.7 Une console Python est disponible, elle nécessite le package `pyluatex`, qui n'est pas chargé par `ProfLycee`, du fait de l'obligation de spécifier le *chemin* pour l'exécutable Python!

Code \LaTeX

```
\usepackage[executable=...]{pyluatex} %si utilisation de la console REPL
\useproflyclib{piton}
```

Information(s)

La librairie `piton` (qui charge `piton`, est compatible uniquement avec $\text{Lua}\LaTeX$!) permet d'insérer du code Python avec une coloration syntaxique en utilisant la bibliothèque Lua LPEG.

En *partenariat* avec `tcolorbox`, on peut avoir une présentation de code Python!

Depuis la version **0.95** de `piton`, `<left-margin=auto>` est disponible et activée dans `ProfLycee`.

Depuis la version **0.99** de `piton`, `<break-lines>` est disponible et activée dans `ProfLycee`.

Depuis la version **1.0** de `piton`, `<tabs-auto-gobble>` est disponible et activée dans `ProfLycee`.

Attention

Le package `piton` nécessite donc obligatoirement l'emploi de $\text{Lua}\LaTeX$!

Ce package n'est chargé que si la compilation détectée est en $\text{Lua}\LaTeX$!

2.5.7 L'utilisation de la console **REPL** nécessite une compilation en `--shell-escape` ou `-write18`!

2.5.7 Les packages `pyluatex` et `pythontex` utilisent des commandes de même nom, donc la présente documentation n'utilisera pas le package `pyluatex`. Une documentation annexe spécifique est disponible.


12.2 Présentation de code Python

Code \LaTeX

```
\begin{CodePiton}[options]{options tcolorbox}
...
\end{CodePiton}
```

Clés et options

Plusieurs **<clés>** sont disponibles :

- la clé booléenne **<Lignes>** pour afficher ou non les numéros de lignes; défaut **<true>**
- la clé booléenne **<Gobble>** pour activer les options liées au gobble; défaut **<true>**
- la clé **<Largeur>** qui correspond à la largeur de la `tcolorbox`; défaut **<\linewidth>**
- la clé **<TaillePolice>** pour la taille des caractères; défaut **<\footnotesize>**
- la clé **<Alignement>** qui paramètre l'alignement de la `tcolorbox`; défaut **<center>**
- **2.5.7** la clé **<Style>** (parmi **<Moderne / Classique>**) pour changer le style; défaut **<Moderne>**
- **2.5.7** le booléen **<Filigrane>** pour afficher, le logo  en filigrane; défaut **<false>**
- **2.5.7** le booléen **<BarreTitre>** (si **<Style=Moderne>**) pour afficher le titre; défaut **<true>**
- **2.5.7** le booléen **<Cadre>** (si **<Style=Moderne>**) pour afficher le cadre. défaut **<true>**

Information(s)

Du fait du paramétrage des boîtes `tcolorbox`, il se peut que le rendu soit non conforme si elle doit être insérée dans une autre `tcolorbox`...

Information(s)

Pour éviter des problèmes avec le code interprété par piton, les `{}` de l'argument obligatoire sont nécessaires au bon fonctionnement du code.

Code L^AT_EX

```
\begin{CodePiton}{}  
#environnement piton avec numéros de ligne, pleine largeur, style moderne  
def arctan(x,n=10):  
    if x < 0:  
        return -arctan(-x) #> (appel récursif)  
    elif x > 1:  
        return pi/2 - arctan(1/x) #> (autre appel récursif)  
    else:  
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )  
\end{CodePiton}
```

Code Python

```
1 #environnement piton avec numéros de ligne, pleine largeur, style moderne  
2 def arctan(x,n=10):  
3     if x < 0:  
4         return -arctan(-x) (appel récursif)  
5     elif x > 1:  
6         return pi/2 - arctan(1/x) (autre appel récursif)  
7     else:  
8         return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

Code L^AT_EX

```
\begin{CodePiton}[Style=Classique,Filigrane]{}  
#environnement piton avec numéros, style classique, filigrane  
def arctan(x,n=10):  
    if x < 0:  
        return -arctan(-x) #> (appel récursif)  
    elif x > 1:  
        return pi/2 - arctan(1/x) #> (autre appel récursif)  
    else:  
        return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )  
\end{CodePiton}
```

Code Python

```
1 #environnement piton avec numéros, style classique, filigrane  
2 def arctan(x,n=10):  
3     if x < 0:  
4         return -arctan(-x) (appel récursif)  
5     elif x > 1:  
6         return pi/2 - arctan(1/x) (autre appel récursif)  
7     else:  
8         return sum( (-1)**k/(2*k+1)*x**(2*k+1) for k in range(n) )
```

</> Code L^AT_EX

```
\begin{CodePiton}[Alignement=flush right,Largeur=13cm]{}
def f(x) :
    return x**2
\end{CodePiton}

\begin{CodePiton}[Alignement=flush left,Largeur=11cm]{}
def f(x) :
    return x**2
\end{CodePiton}

\begin{itemize} %Avec des indentations d'environnement :
  \item On essaye avec un \texttt{itemize} :
  %
  \begin{CodePiton}[Largeur=12cm,Style=Classique,Cadre=false]{}
    def f(x) :
        return x**2
  \end{CodePiton}
  \item Et avec un autre \texttt{itemize} :
  %
  \begin{CodePiton}[Largeur=12cm,Style=Classique,Cadre=false,BarreTitre=false]{}
    #avec numéros, de largeur 12cm, centré, classique, sans cadre/titre
    def f(x) :
        return x**2
  \end{CodePiton}
\end{itemize}
```

Code Python

```
1 #avec numéros, de largeur 13cm, aligné à droite
2 def f(x) :
3     return x**2
```

Code Python

```
1 #avec numéros, de largeur 11cm, aligné à gauche
2 def f(x) :
3     return x**2
```

— On essaye avec un itemize :

</> Code Python

```
1 #avec numéros, de largeur 12cm, centré, classique, sans cadre
2 def f(x) :
3     return x**2
```

— Et avec un autre itemize :

```
1 #avec numéros, de largeur 12cm, centré, classique, sans \
+ ↪ cadre/titre
2 def f(x) :
3     return x**2
```

12.3 Console en partenariat avec Pyluatex

Information(s)

2.5.7 Une console d'exécution (type REPL) est disponible, et la documentation associée est en marge de la présente documentation.

13 Code & Console Python, via les packages Pythontex ou Minted

13.1 Librairies

Information(s)

2.5.0 Cette section nécessite de charger les librairies `\minted` et/ou `\pythontex` dans le préambule.

Code \LaTeX

```
\useproflyclib{minted}
\useproflyclib{pythontex}
%ou
\useproflyclib{minted,pythontex}
```

13.2 Introduction

Idée(s)

2.5.0 La librairie `\pythontex` permet d'insérer et d'exécuter du code Python. On peut :

- **2.5.8** présenter du code Python (deux styles disponibles);
- exécuter du code Python dans un environnement type « console »;
- charger du code Python, et éventuellement l'utiliser dans la console.

Attention

Attention : il faut dans ce cas une compilation en plusieurs étapes, comme par exemple `pdflatex` puis `pythontex` puis `pdflatex`!

Voir par exemple <http://lesmathsduyeti.fr/fr/informatique/latex/pythontex/>!

Information(s)

Compte tenu de la *relative complexité* pour gérer les options (par paramètres/clés...) des *tcbbox* et des *fancyvrb*, les style sont « fixés » tels quels, et seules la taille et la position de la *tcbbox* sont modifiables. Si toutefois vous souhaitez personnaliser davantage, il faudra prendre le code correspondant et appliquer vos modifications!

Cela peut donner – en tout cas – des idées de personnalisation en ayant une base *préexistante*!

13.3 Présentation de code Python grâce au package pythontex

Idée(s)

L'environnement `\CodePythontex` est donc lié à `\pythontex` (chargé par `\ProfLycee`, avec l'option *autogobble*) permet de présenter du code Python, dans une `\tcolorbox` avec deux styles particuliers (**2.5.8**).

Code \LaTeX

```
\begin{CodePythontex}[options]{ } %les {} vides sont nécessaires
...
\end{CodePythontex}
```

Code \LaTeX

```
\begin{CodePythontexAlt}[options]{ } %les {} vides sont nécessaires
...
\end{CodePythontexAlt}
```

🔑 Clés et options

Comme précédemment, des **<Clés>** qui permettent de *légèrement* modifier le style :

- **<Largeur>** : largeur de la *tcbbox*; défaut `\linewidth`
- **<Centre>** : booléen pour centrer ou non la *tcbbox*; défaut `<false>`
- **<TaillePolice>** : taille des caractères; défaut `\footnotesize`
- **<EspacementVertical>** : option (*stretch*) pour l'espacement entre les lignes; défaut `<1>`
- **<Lignes>** : booléen pour afficher ou non les numéros de ligne. défaut `<true>`

🔗 Code \LaTeX

```
\begin{CodePythontex}{} %bien mettre les {} !!
#environnement Python(tex) par défaut
def f(x) :
    return x**2
\end{CodePythontex}
```

🔗 Sortie \LaTeX

```
1 #environnement Python(tex) par défaut
2 def f(x) :
3     return x**2
```

🔗 Code Python

🔗 Code \LaTeX

```
\begin{CodePythontexAlt}[Largeur=12cm,Centre,Lignes=false]{}
#environnement Python(tex) classique, centré, sans lignes
def f(x) :
    return x**2
\end{CodePythontexAlt}
```

🔗 Sortie \LaTeX

🔗 Code Python

```
#environnement Python(tex) classique, centré, sans lignes
def f(x) :
    return x**2
```

13.4 Présentation de code Python via le package minted

🔗 Information(s)

Pour celles et ceux qui ne sont pas à l'aise avec le package `\text{\minted}` et notamment sa spécificité pour compiler, il existe le package `\text{\minted}` qui permet de présenter du code, et notamment Python.

2.5.8 Deux styles sont désormais disponibles.

2.5.0 C'est donc la librairie `\text{\minted}` qu'il faudra charger.

🔗 Attention

Le package `\text{\minted}` nécessite quand même une compilation avec l'option `--shell-escape` ou `-write18` !

🔗 Code \LaTeX

```
\begin{CodePythonMinted}(*) [largeur]{options}
...
\end{CodePythonMinted}
```

</> Code \LaTeX

```
\begin{CodePythonMintedAlt}(*)[largeur]{options}
...
\end{CodePythonMintedAlt}
```

🔑 Clés et options

Plusieurs **<arguments>** sont disponibles :

- la version *étoilée* qui permet de ne pas afficher les numéros de lignes;
- le 1^{er} argument *optionnel* concerne la **<largeur>** de la \LaTeX `tcbox`; défaut **<12cm>**
- le 2nd argument *obligatoire* concerne les **<options>** de la \LaTeX `tcbox` en langage *tcbox*. défaut **<vide>**

</> Code \LaTeX

```
\begin{CodePythonMinted}[13cm]{center}
#environnement Python(minted) centré avec numéros, de largeur 13cm
def f(x) :
    return x**2
\end{CodePythonMinted}
```

⊕ Sortie \LaTeX

```
1 #environnement Python(minted) centré avec numéros
2 def f(x) :
3     return x**2
```

Code Python

</> Code \LaTeX

```
\begin{CodePythonMintedAlt}*[0.8\linewidth]{}
#environnement Python(minted), style alt, sans numéro, de largeur 0.8\linewidth
def f(x) :
    return x**2
\end{CodePythonMintedAlt}
```

⊕ Sortie \LaTeX

</> Code Python

```
#environnement Python(minted), style alt, sans numéro, de largeur 0.8\linewidth
def f(x) :
    return x**2
```

13.5 Console d'exécution Python

💡 Idée(s)

\LaTeX `pythontex` permet également de *simuler* (en exécutant également!) du code Python dans une *console*, avec la librairie \LaTeX `pythontex` du coup!

C'est l'environnement \LaTeX `ConsolePythontex` qui permet de le faire.

</> Code \LaTeX

```
\begin{ConsolePythontex}[options]{} %les {} vides sont nécessaires
...
\end{ConsolePythontex}
```

🔑 Clés et options

Les **<Clés>** disponibles sont :

- **<Largeur>** : largeur de la *console*; défaut **<\linewidth>**
- **<Centre>** : booléen pour centrer ou non la *console*; défaut **<false>**
- **<TaillePolice>** : taille des caractères; défaut **<\footnotesize>**
- **<EspacementVertical>** : option (*stretch*) pour l'espacement entre les lignes; défaut **<1>**
- **<Label>** : booléen pour afficher ou non le titre. défaut **<true>**

🔗 Code \LaTeX

```
\begin{ConsolePythontex}{}
#console Python(tex) par défaut
from math import sqrt
1+1
sqrt(12)
\end{ConsolePythontex}
```

➡ Sortie \LaTeX

_____ Début de la console python _____

```
>>> #console Python(tex) par défaut
>>> from math import sqrt
>>> 1+1
2
>>> sqrt(12)
3.4641016151377544
```

_____ Fin de la console python _____

🔗 Code \LaTeX

```
\begin{ConsolePythontex}[Largeur=14cm,Label=false,Centre]{}
#console Python(tex) centrée sans label, 14cm
table = [[1,2],[3,4]]
table[0][0]

from random import randint
tableau = [[randint(1,20) for j in range(0,6)] for i in range(0,3)]
tableau
len(tableau), len(tableau[0]), tableau[1][4]
\end{ConsolePythontex}
```

➡ Sortie \LaTeX

```
>>> #console Python(tex) centrée sans label, 14cm
>>> table = [[1,2],[3,4]]
>>> table[0][0]
1

>>> from random import randint
>>> tableau = [[randint(1,20) for j in range(0,6)] for i in range(0,3)]
>>> tableau
[[16, 13, 12, 5, 8, 8], [12, 9, 20, 17, 15, 13], [1, 5, 2, 9, 5, 13]]
>>> len(tableau), len(tableau[0]), tableau[1][4]
(3, 6, 15)
```

📄 Information(s)

Le package `\usepackage{pythonx}` peut donc servir à présenter du code Python, comme `\usepackage{minted}` ou `\usepackage{piston}`, sa particularité est toutefois de pouvoir *exécuter* du code Python pour une présentation de type *console*.

14 Pseudo-Code

14.1 Introduction

Information(s)

Le package `listings` permet d'insérer et de présenter du code, et avec `tcolorbox` on peut obtenir une présentation similaire à celle du code Python. Pour le moment la *philosophie* de la commande est un peu différente de celle du code Python, avec son système de **Clés**.

14.2 Présentation de Pseudo-Code

Idée(s)

Les environnements `PseudoCode` ou `PseudoCodeAlt` permet de présenter du (pseudo-code) dans une `tcolorbox`, avec deux styles à disposition (`2.5.8`).

Attention

De plus, le package `listings` avec `tcolorbox` ne permet pas de gérer le paramètre *autogobble*, donc il faudra être vigilant quant à la position du code (pas de tabulation en fait...)

Code \LaTeX

```
\begin{PseudoCode}(*)[largeur]{options tcbbox}
%attention à l'indentation, gobble ne fonctionne pas...
...
\end{PseudoCode}
```

Code \LaTeX

```
\begin{PseudoCodeAlt}(*)[largeur]{options tcbbox}
%attention à l'indentation, gobble ne fonctionne pas...
...
\end{PseudoCodeAlt}
```

Clés et options

Plusieurs **arguments** (optionnels) sont disponibles :

- la version *étoilée* qui permet de ne pas afficher les numéros de lignes;
- le premier argument optionnel concerne la **largeur** de la `tcbbox`; défaut **12cm**
- `2.5.8` l'argument obligatoire entre `{...}` concerne les **options** de la `tcbbox`.

Code \LaTeX

```
%en pas oublier les {}, même vides !
\begin{PseudoCode}{} %non centré, de largeur par défaut (12cm) avec lignes
List = [...]          # à déclarer au préalable
n = longueur(List)
Pour i allant de 0 à n-1 Faire
  Afficher(List[i])
FinPour
\end{PseudoCode}
```


⊕ Sortie L^AT_EX

Pseudo-Code

```
1 List ← [...]          # à déclarer au préalable
2 n ← longueur(List)
3 Pour i allant de 0 à n-1 Faire
4     Afficher(List[i])
5 FinPour
```

⌞ Code L^AT_EX

```
\begin{PseudoCodeAlt}[15cm]{center} %centré, de largeur 15cm
List = [...]          # à déclarer au préalable
n = longueur(List)
Pour i allant de 0 à n-1 Faire
    Afficher(List[i])
FinPour
\end{PseudoCodeAlt}
```

⊕ Sortie L^AT_EX

⌞ PseudoCode

```
1 List ← [...]          # à déclarer au préalable
2 n ← longueur(List)
3 Pour i allant de 0 à n-1 Faire
4     Afficher(List[i])
5 FinPour
```

14.3 Compléments

📢 Attention

À l'instar de packages existants, la *philosophie* ici est de laisser l'utilisateur gérer *son* langage pseudo-code. J'ai fait le choix de ne pas définir des mots clés à mettre en valeur car cela reviendrait à *imposer* des choix! Donc ici, pas de coloration syntaxique ou de mise en évidence de mots clés, uniquement un formatage basique!

🔧 Information(s)

Le style `listings` utilisé par la commande a l'option **(mathescape)** activée, et accessible grâce aux délimiteurs **((...*))**.

Cela permet d'insérer du code L^AT_EX dans l'environnement `PseudoCode` (attention au fontes par contre!).

⌞ Code L^AT_EX

```
\begin{PseudoCode}*[12cm]{ }
#Utilisation du mode mathescape
Afficher (*\og*) .....(*\fg*)
m = (*$\tfrac{\texttt{1}}{\texttt{2}}$*)
\end{PseudoCode}
```

⊕ Sortie L^AT_EX

Pseudo-Code

```
#Utilisation du mode mathescape
Afficher « ..... »
m ←  $\frac{1}{2}$ 
```

15 Terminal Windows/UNIX/OSX

15.1 Introduction

💡 Idée(s)

L'idée des commandes suivantes est de permettre de simuler des fenêtres de Terminal, que ce soit pour Windows, Ubuntu ou OSX.

L'idée de base vient du package `termsim`, mais ici la gestion du code et des fenêtres est légèrement différente. Le contenu est géré par le package `listings`, sans langage particulier, et donc sans coloration syntaxique particulière.

🚨 Attention

Comme pour le pseudo-code, pas d'autogobble, donc commandes à aligner à gauche!

15.2 Commandes

🔗 Code \LaTeX

```
\begin{TerminalWin}[largeur]{titre=...}[options tcbox]
...
\end{TerminalWin}

\begin{TerminalUnix}[largeur]{titre=...}[options tcbox]
...
\end{TerminalUnix}

\begin{TerminalOSX}[largeur]{titre=...}[options tcbox]
...
\end{TerminalOSX}
```

🔗 Clés et options

Peu d'options pour ces commandes :

- le premier, *optionnel*, est la \langle largeur \rangle de la `tcbox`; défaut \langle \linewidth \rangle
- le deuxième, *obligatoire*, permet de spécifier le titre par la clé \langle titre \rangle . défaut \langle Terminal Windows/UNIX/OSX \rangle
- le troisième, *optionnel*, concerne les \langle options \rangle de la `tcbox` en langage *tcolorbox*. défaut \langle vide \rangle

🔗 Information(s)

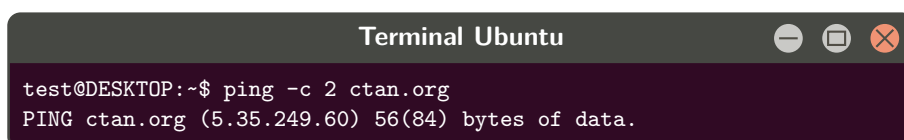
Le code n'est pas formaté, ni mis en coloration syntaxique.

De ce fait tous les caractères sont autorisés : même si l'éditeur pourra détecter le % comme le début d'un commentaire, tout sera intégré dans le code mis en forme!

🔗 Code \LaTeX

```
\begin{TerminalUnix}[12cm]{Titre=Terminal Ubuntu}[center] %12cm, avec titre modifié et centré
test@DESKTOP:~$ ping -c 2 ctan.org
PING ctan.org (5.35.249.60) 56(84) bytes of data.
\end{TerminalUnix}
```

🔗 Sortie \LaTeX



</> Code L^AT_EX

```
\begin{TerminalWin}[15cm]{} %largeur 15cm avec titre par défaut
Microsoft Windows [version 10.0.22000.493]
(c) Microsoft Corporation. Tous droits réservés.
C:\Users\test>ping ctan.org

Envoi d'une requête 'ping' sur ctan.org [5.35.249.60] avec 32 octets de données :
Réponse de 5.35.249.60 : octets=32 temps=35 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=37 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=35 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=39 ms TTL=51

Statistiques Ping pour 5.35.249.60:
Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
Minimum = 35ms, Maximum = 39ms, Moyenne = 36ms
\end{TerminalWin}

\begin{TerminalOSX}[0.5\linewidth]{Titre=Terminal MacOSX}[flush right] %1/2-largeur et titre
~ modifié et droite
[test@server]$ ping -c 2 ctan.org
PING ctan.org (5.35.249.60) 56(84) bytes of data.
\end{TerminalOSX}
```

➤ Sortie L^AT_EX

>_ Terminal Windows

```
Microsoft Windows [version 10.0.22000.493]
(c) Microsoft Corporation. Tous droits réservés.
C:\Users\test>ping ctan.org

Envoi d'une requête 'ping' sur ctan.org [5.35.249.60] avec 32 octets de données :
Réponse de 5.35.249.60 : octets=32 temps=35 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=37 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=35 ms TTL=51
Réponse de 5.35.249.60 : octets=32 temps=39 ms TTL=51

Statistiques Ping pour 5.35.249.60:
Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en millisecondes :
Minimum = 35ms, Maximum = 39ms, Moyenne = 36ms
```

Terminal Ubuntu

```
test@DESKTOP:~$ ping -c 2 ctan.org
PING ctan.org (5.35.249.60) 56(84) bytes of data.
```

Terminal MacOSX

```
[test@server]$ ping -c 2 ctan.org
PING ctan.org (5.35.249.60) 56(84) bytes of data.
```

16 Cartouche Capytale

16.1 Introduction

💡 Idée(s)

L'idée est d'obtenir des cartouches tels que Capytale les présente, pour partager un code afin d'accéder à une activité Python.

16.2 Commandes

🔗 Code \LaTeX

```
\CartoucheCapytale(*)[options]{code capytale}
```

🔑 Clés et options

Peu d'options pour ces commandes :

- la version *étoilée* qui permet de passer de la police $\langle\texttt{sffamily}\rangle$ à la police $\langle\texttt{ttfamily}\rangle$, et donc dépendante des fontes du document;
- le deuxième, *optionnel*, permet de rajouter des caractères après le code (comme un espace);
défaut $\langle\texttt{vide}\rangle$
- le troisième, *obligatoire*, est le code capytale à afficher.

🔗 Code \LaTeX

```
\CartoucheCapytale{abcd-12345}           %lien simple, en sf
\CartoucheCapytale[~]{abcd-12345}        %lien avec ~ à la fin, en sf
\CartoucheCapytale*{abcd-12345}          %lien simple, en tt
\CartoucheCapytale*[~]{abcd-12345}       %lien avec ~ à la fin, en tt
```

➦ Sortie \LaTeX

abcd-12345 🔗

abcd-12345 🔗

abcd-12345 🔗

abcd-12345 🔗

🔧 Information(s)

Le cartouche peut être « cliquable » grâce à `\href`.

🔗 Code \LaTeX

```
\usepackage{hyperref}
\urlstyle{same}
...
\href{https://capytale2.ac-paris.fr/web/c/abcd-12345}{\CartoucheCapytale{abcd-12345}}
```

➦ Sortie \LaTeX

abcd-12345 🔗

17 Présentation de code \LaTeX

17.1 Introduction

💡 Idée(s)

2.0.6 L'idée est de proposer un environnement pour présenter du code \LaTeX . Ce n'est pas forcément lié à l'enseignement en Lycée mais pourquoi pas!

Il s'agit d'un environnement créé en \LaTeX `tcolorbox`, et utilisant la présentation *basique* de code via \LaTeX `listings`.

17.2 Commandes

🔗 Code \LaTeX

```
\begin{PresentationCode}[Couleur]{options tcbbox}
...
\end{PresentationCode}
```

🔗 Clés et options

Peu de personnalisations pour ces commandes :

- le premier argument, *optionnel*, permet de préciser la *couleur* de la présentation; défaut **ForestGreen**
- le second, *obligatoire*, correspond aux éventuelles options liées à la \LaTeX `tcolorbox`.

🔗 Information(s)

Il est à noter que, même dans le cas d'option vide pour la \LaTeX `tcolorbox`, les \LaTeX `{}` sont nécessaires.
On peut par exemple utiliser l'option **<listing only>** pour ne présenter *que* le code source.

🔗 Code \LaTeX

```
\begin{PresentationCode}{}
\xdef\ValAleaA{\fpeval{randint(1,100)}}
\xdef\ValAleaB{\fpeval{randint(1,100)}}
\end{PresentationCode}
```

Avec $\$A=\text{\ValAleaA\$}$ et $\$B=\text{\ValAleaB\$}$, on a $\$A\times B=\text{\inteval{\ValAleaA * \ValAleaB}\$}$.

```
\end{PresentationCode}
```

```
\begin{PresentationCode}[DarkBlue]{}
On peut faire beaucoup de choses avec \LaTeX{} !
\end{PresentationCode}
```

```
\xdef\ValAleaA{\fpeval{randint(1,100)}}
\xdef\ValAleaB{\fpeval{randint(1,100)}}

```

Code \LaTeX

Avec $\$A=\text{\ValAleaA\$}$ et $\$B=\text{\ValAleaB\$}$, on a $\$A\times B=\text{\inteval{\ValAleaA * \ValAleaB}\$}$.

Avec $A = 29$ et $B = 11$, on a $A \times B = 319$.

On peut faire beaucoup de choses avec $\text{\LaTeX}\{\}$!

Code \LaTeX

On peut faire beaucoup de choses avec \LaTeX !

Sixième partie

Outils pour la géométrie

18 Pavé droit « simple »

18.1 Introduction

💡 Idée(s)

L'idée est d'obtenir un pavé droit, dans un environnement TikZ, avec les nœuds créés et nommés directement pour utilisation ultérieure.

18.2 Commandes

🔗 Code L^AT_EX

```
\begin{tikzpicture}[options tikz]
  \PaveTikz[options]
  ...
\end{tikzpicture}
```

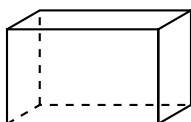
🔗 Clés et options

Quelques <clés> sont disponibles pour cette commande :

- | | |
|---|---------------------------------|
| — <Largeur> : largeur du pavé; | défaut <2> |
| — <Profondeur> : profondeur du pavé; | défaut <1> |
| — <Hauteur> : hauteur du pavé; | défaut <1.25> |
| — <Angle> : angle de fuite de la perspective; | défaut <30> |
| — <Fuite> : coefficient de fuite de la perspective; | défaut <0.5> |
| — <Sommets> : liste des sommets (avec délimiteur \$!); | défaut <A\$B\$C\$D\$E\$F\$G\$H> |
| — <Math> : booléen pour forcer le mode math des sommets; | défaut <false> |
| — <Epaisseur> : épaisseur des arêtes (en <i>langage simplifié</i> TikZ); | défaut <thick> |
| — <Aff> : booléen pour afficher les noms des sommets; | défaut <false> |
| — <Plein> : booléen pour ne pas afficher les arêtes <i>invisibles</i> ; | défaut <false> |
| — <Cube> : booléen pour préciser qu'il s'agit d'un cube (seule la valeur <Largeur> est util(isé)e). | défaut <false> |

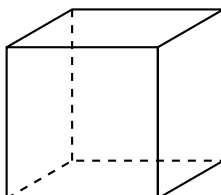
🔗 Code L^AT_EX

```
%code tikz
\PaveTikz
```



🔗 Code L^AT_EX

```
%code tikz
\PaveTikz[Cube,Largeur=2]
```



Information(s)

La ligne est de ce fait à insérer dans un environnement TikZ, avec les options au choix pour cet environnement.

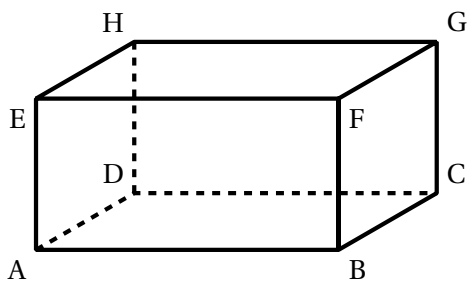
Le code crée les nœuds relatifs aux sommets, et les nomme comme les sommets, ce qui permet de les réutiliser pour éventuellement compléter la figure!

18.3 Influence des paramètres

Code \LaTeX

```
\begin{tikzpicture}[line join=bevel]
  \PaveTikz[Aff,Largeur=4,Profondeur=3,Hauteur=2,Epaisseur={ultra thick}]
\end{tikzpicture}
```

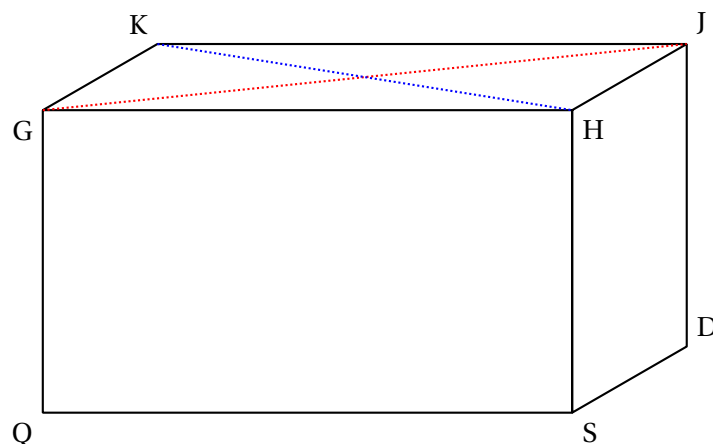
Sortie \LaTeX



Code \LaTeX

```
\begin{center}
  \begin{tikzpicture}[line join=bevel]
    \PaveTikz[Plein,Aff,Largeur=7,Profondeur=3.5,Hauteur=4,Sommets=QSSD$F$G$H$J$K]
    \draw[thick,red,densely dotted] (G)--(J) ;
    \draw[thick,blue,densely dotted] (K)--(H) ;
  \end{tikzpicture}
\end{center}
```

Sortie \LaTeX



19 Tétraèdre « simple »

19.1 Introduction

💡 Idée(s)

L'idée est d'obtenir un tétraèdre, dans un environnement TikZ, avec les nœuds créés et nommés directement pour utilisation ultérieure.

19.2 Commandes

🔗 Code \LaTeX

```
\begin{tikzpicture}[options tikz]
  \TetraedreTikz[options]
  ...
\end{tikzpicture}
```

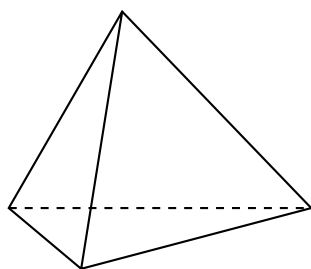
🔑 Clés et options

Quelques <clés> sont disponibles pour cette commande :

- | | |
|--|---------------------|
| — <Largeur> : <i>largeur</i> du tétraèdre; | défaut <4> |
| — <Profondeur> : <i>profondeur</i> du tétraèdre; | défaut <1.25> |
| — <Hauteur> : <i>hauteur</i> du tétraèdre; | défaut <3> |
| — <Alpha> : angle <i>du sommet de devant</i> ; | défaut <40> |
| — <Beta> : angle <i>du sommet du haut</i> ; | défaut <60> |
| — <Sommets> : liste des sommets (avec délimiteur \$!); | défaut <A\$B\$C\$D> |
| — <Math> : booléen pour forcer le mode math des sommets; | défaut <false> |
| — <Epaisseur> : épaisseur des arêtes (en <i>langage simplifié</i> TikZ); | défaut <thick> |
| — <Aff> : booléen pour afficher les noms des sommets; | défaut <false> |
| — <Plein> : booléen pour ne pas afficher l'arête <i>invisible</i> . | défaut <false> |

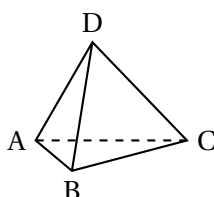
🔗 Code \LaTeX

```
%code tikz
\TetraedreTikz
```



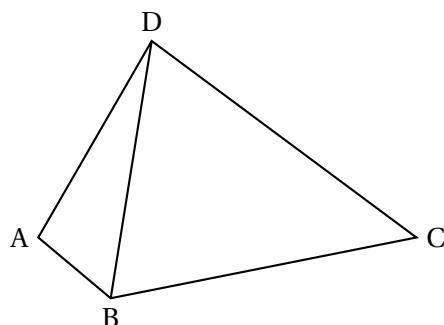
🔗 Code \LaTeX

```
%code tikz
\TetraedreTikz[Aff,Largeur=2,Profondeur=0.625,Hauteur=1.5]
```



Code \LaTeX

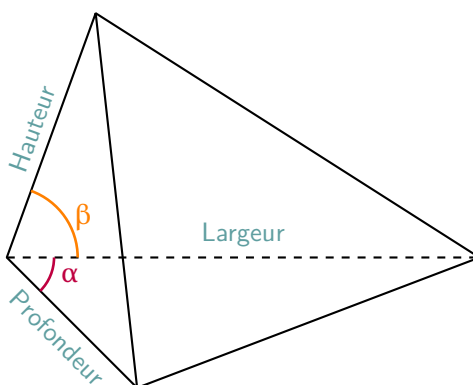
```
%code tikz
\TetraedreTikz[Plein,Aff,Largeur=5,Beta=60]
```



19.3 Influence des paramètres

Information(s)

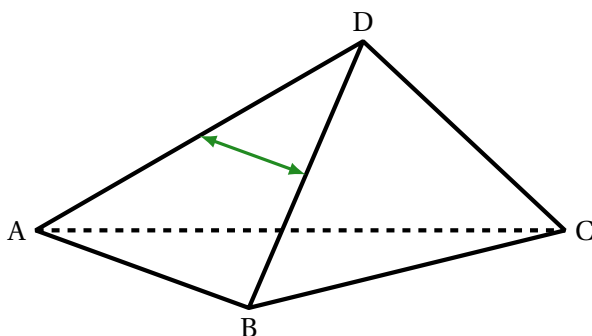
Pour *illustrer* un peu les **clés**, un petit schéma, avec les différents paramètres utiles.



Code \LaTeX

```
\begin{center}
\begin{tikzpicture}[line join=bevel]
\TetraedreTikz[Aff,Largeur=7,Profondeur=3,Hauteur=5,Epaisseur={ultra
  ↳ thick},Alpha=20,Beta=30]
\draw[very thick,ForestGreen,<->,>=latex] ($ (A)!0.5!(D)$ )--($ (B)!0.5!(D)$ ) ;
\end{tikzpicture}
\end{center}
```

Sortie \LaTeX



20 Cercle trigo

20.1 Idée

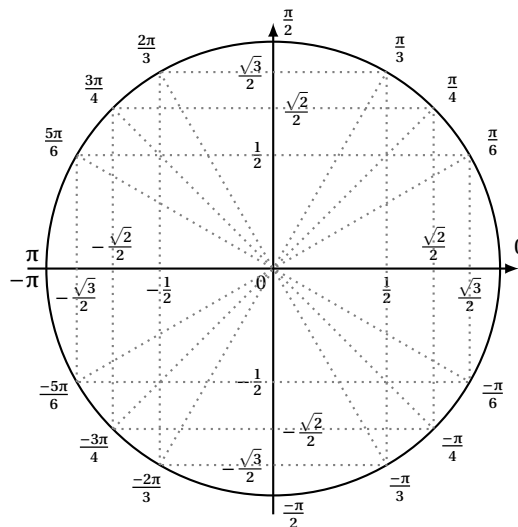
💡 Idée(s)

L'idée est d'obtenir une commande pour tracer (en TikZ) un cercle trigonométrique, avec personnalisation des affichages.

Comme pour les autres commandes TikZ, l'idée est de laisser l'utilisateur définir et créer son environnement TikZ, et d'insérer la commande `\CercleTrigo` pour afficher le cercle.

🔗 Code \LaTeX

```
%code tikz  
\CercleTrigo
```



20.2 Commandes

🔗 Code \LaTeX

```
...  
\begin{tikzpicture}[options tikz]  
...  
  \CercleTrigo[clés]  
...  
\end{tikzpicture}
```

🔑 Clés et options

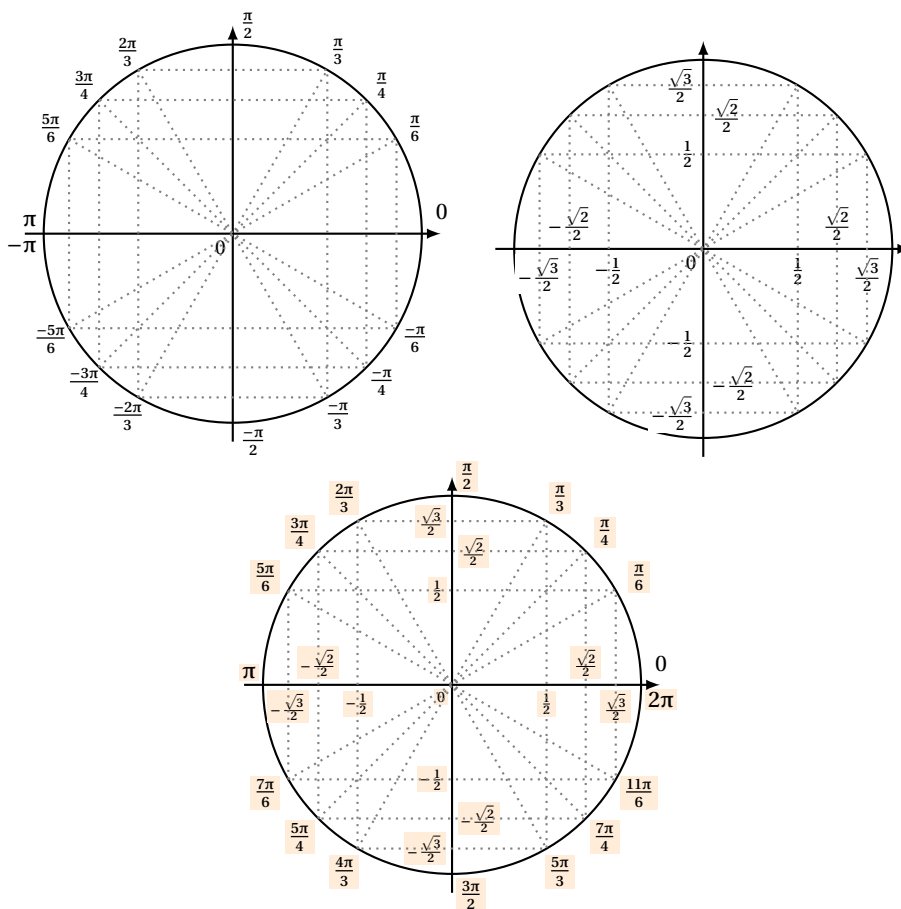
Plusieurs **Clés** sont disponibles pour cette commande :

- | | |
|---|------------------------------------|
| — la clé <Rayon> qui définit le rayon du cercle; | défait <3> |
| — la clé <Epaisseur> qui donne l'épaisseur des traits de base; | défait <thick> |
| — la clé <Marge> qui est l'écartement de axes; | défait <0.25> |
| — la clé <TailleValeurs> qui est la taille des valeurs remarquables; | défait <scriptsize> |
| — la clé <TailleAngles> qui est la taille des angles; | défait <footnotesize> |
| — la clé <CouleurFond> qui correspond à la couleur de fond des labels; | défait <white> |
| — la clé <Decal> qui correspond au décalage des labels par rapport au cercle; | défait <10pt> |
| — un booléen <MoinsPi> qui bascule les angles « -pi » à « zero deux pi »; | défait <true> |
| — un booléen <AffAngles> qui permet d'afficher les angles; | défait <true> |
| — un booléen <AffTraits> qui permet d'afficher les <i>traits de construction</i> ; | défait <true> |
| — un booléen <AffValeurs> qui permet d'afficher les valeurs remarquables. | défait <true> |

</> Code \LaTeX

```
\begin{center}
\begin{tikzpicture}[line join=bevel]
\ CercleTrigo[Rayon=2.5,AffValeurs=false,Decal=8pt]
\end{tikzpicture}
~~~~~
\begin{tikzpicture}[line join=bevel]
\ CercleTrigo[Rayon=2.5,AffAngles=false]
\end{tikzpicture}
~~~~~
\begin{tikzpicture}[line join=bevel]
\ CercleTrigo[Rayon=2.5,MoinsPi=false,CouleurFond=orange!15]
\end{tikzpicture}
\end{center}
```

⊕ Sortie \LaTeX



20.3 Équations trigos

📌 Information(s)

En plus des **(Clés)** précédentes, il existe un complément pour *visualiser* des solutions d'équations simples du type $\cos(x) = \dots$ ou $\sin(x) = \dots$.

🔑 Clés et options

Les **<Clés>** pour cette possibilité sont :

- un booléen **<Equationcos>** pour *activer* « cos = » ; défaut **<false>**
- un booléen **<Equationsin>** pour *activer* « sin = » ; défaut **<false>**
- la clé **<sin>** qui est la valeur de l'angle (en degrés) du sin ; défaut **<30>**
- la clé **<cos>** qui est la valeur de l'angle (en degrés) cos ; défaut **<45>**
- la clé **<CouleurSol>** qui est la couleur des *solutions*. défaut **<blue>**

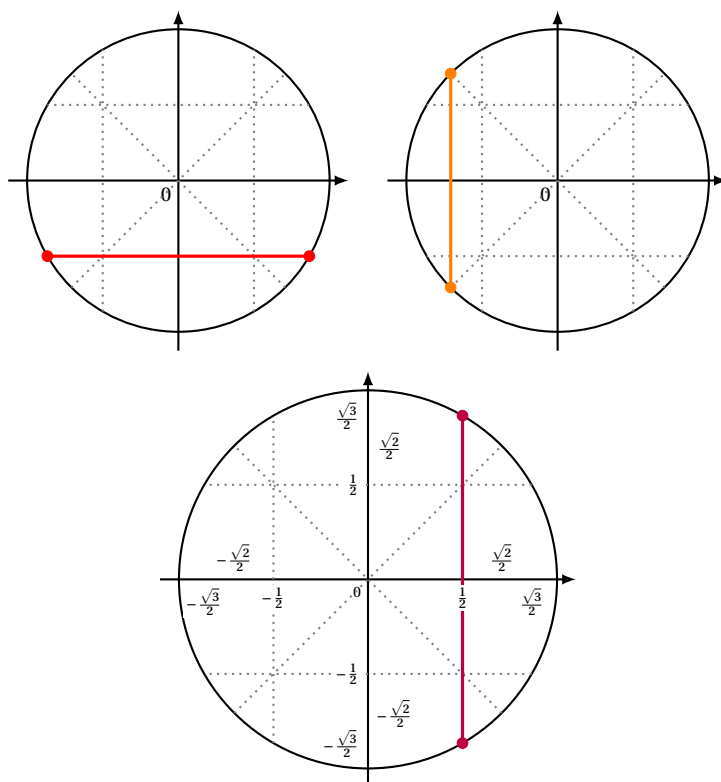
🔗 Code \LaTeX

```
\begin{center}
\begin{tikzpicture}
\CercleTrigo[%
AffAngles=false,AffValeurs=false,AffTraits=false,Rayon=2,Equationsin,sin=-30,CouleurSol=red]
\end{tikzpicture}
~~~~~
\begin{tikzpicture}
\CercleTrigo[%
AffAngles=false,AffValeurs=false,AffTraits=false,Rayon=2,Equationcos,cos=135,
↪ CouleurSol=orange]
\end{tikzpicture}

\medskip

\begin{tikzpicture}
\CercleTrigo[%
AffTraits=false,AffAngles=false,Rayon=2.5,Equationcos,cos=60,CouleurSol=purple,
↪ TailleValeurs=\tiny]
\end{tikzpicture}
\end{center}
```

➡ Sortie \LaTeX



21 Style « main levée » en TikZ

21.1 Idée

💡 Idée(s)

L'idée est de *proposer* un style *tout prêt* pour simuler un tracé, en TikZ, à « main levée ».
Il s'agit d'un style *basique* utilisant la librairie `decorations` avec random steps.

🔗 Code L^AT_EX

```
\tikzset{%
  mainlevee/.style args={#1et#2}{decorate,decoration={random steps, segment
    ↳ length=#1,amplitude=#2}},
  mainlevee/.default={5mm et 0.6pt}
}
```

21.2 Utilisation basique

📌 Information(s)

Il s'agit ni plus ni moins d'un style TikZ à intégrer dans les tracés et constructions TikZ!

🔑 Clés et options

Concernant le style en lui-même, deux paramètres peuvent être précisés via **mainlevee=#1 et #2** :

- **#1** correspond à l'option segment length (longueur des segments *types*); défaut **5mm**
- **#2** correspond à l'option amplitude (amplitude maximale de la *déformation*). défaut **0.6pt**

Les valeurs **mainlevee=5mm et 0.6pt** donnent des résultats – à mon sens – satisfaisants, mais l'utilisateur pourra modifier à loisir ces paramètres!

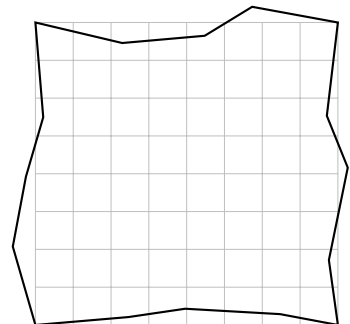
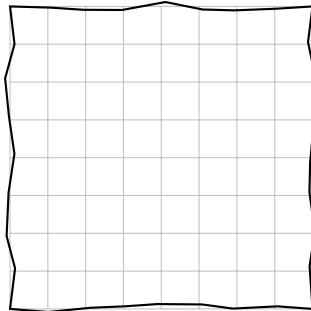
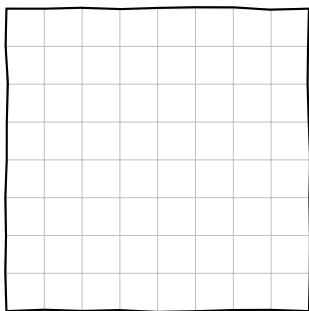
🔗 Code L^AT_EX

```
%la grille a été rajoutée pour la sortie
\begin{tikzpicture}
  \draw[thick,mainlevee] (0,0) --- (4,0) --- (0,4) --- (-4,0) --cycle ;
\end{tikzpicture}

\begin{tikzpicture}
  \draw[thick,mainlevee=5mm et 2pt] (0,0) --- (4,0) --- (0,4) --- (-4,0) --cycle ;
\end{tikzpicture}

\begin{tikzpicture}
  \draw[thick,mainlevee=10mm et 3mm] (0,0) --- (4,0) --- (0,4) --- (-4,0) --cycle ;
\end{tikzpicture}
```

➡ Sortie L^AT_EX



Outils pour les statistiques

22 Paramètres d'une régression linéaire par la méthode des moindres carrés

22.1 Idée

💡 Idée(s)

L'idée est d'utiliser une commande qui va permettre de calculer les paramètres principaux d'une régression linéaire par la méthode des moindres carrés.

Le package `pgfpots` permet de le faire nativement, mais le moteur de calculs de pgf peut poser souci avec de grandes valeurs, donc ici cela passe par `xfp` qui permet de *gagner* en précision!

L'idée est que cette macro calcule et stocke les paramètres dans des variables (le nom peut être personnalisé!) pour exploitation ultérieure :

- en calculs *purs*;
- dans un environnement TikZ via pgfplots ou bien en *natif*;
- dans un environnement PSTricks;
- dans un environnement METAPOST (à vérifier quand même);
- ...

🔗 Code \LaTeX

```
...
\CalculsRegLin[clés]{listeX}{listeY} %listes avec éléments séparés par des ,
...
```

📄 Information(s)

La commande `\CalculsRegLin` va définir également des macros pour chaque coefficient, qui de ce fait seront réutilisables après!

22.2 Commandes

🔑 Clés et options

Quelques **⟨Clés⟩** sont disponibles pour cette commande, essentiellement pour *renommer* les paramètres :

- la clé **⟨NomCoeffa⟩** qui permet de définir la variable qui contiendra a ; défaut **⟨COEFFa⟩**
- la clé **⟨NomCoeffb⟩** qui permet de définir la variable qui contiendra b ; défaut **⟨COEFFb⟩**
- la clé **⟨NomCoeffr⟩** qui permet de définir la variable qui contiendra r ; défaut **⟨COEFFr⟩**
- la clé **⟨NomCoeffrd⟩** qui permet de définir la variable qui contiendra r^2 ; défaut **⟨COEFFrd⟩**
- la clé **⟨NomXmin⟩** qui permet de définir la variable qui contiendra x_{\min} ; défaut **⟨LXmin⟩**
- la clé **⟨NomXmax⟩** qui permet de définir la variable qui contiendra x_{\max} ; défaut **⟨LXmax⟩**

🔗 Code \LaTeX

```
%les espaces verticaux n'ont pas été écrits ici
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
\CalculsRegLin{\LLX}{\LLY}
```

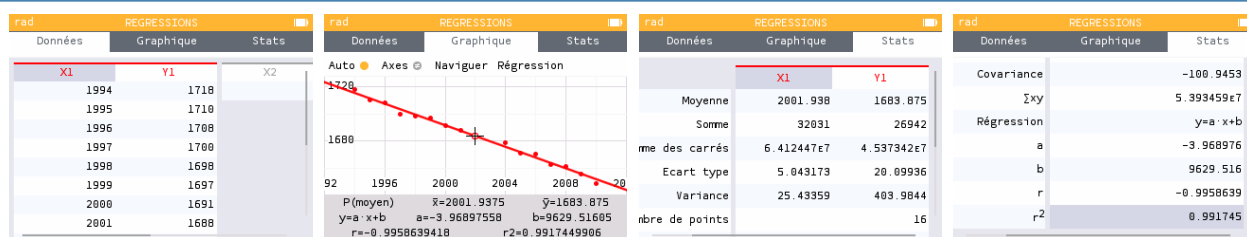
Code \LaTeX

```
%vérif des calculs (noms non modifiables...)
Liste des X := \showitems\X.
Liste des Y := \showitems\Y.
Somme des X := \LXSomme{} et somme des Y := \LYSomme.
Moyenne des X := \LXmoy{} et moyenne des Y := \LYmoy.
Variance des X := \LXvar{} et variance des Y := \LYvar{}
Covariance des X/Y := \LXYvar.
%les coefficients, avec des noms modifiables !
Min des X := \LXmin{} et Max des X := \LXmax.
Coefficient $a=\text{\COEFFa}$.
Coefficient $b=\text{\COEFFb}$.
Coefficient $r=\text{\COEFFr}$.
Coefficient $r^2=\text{\COEFFrd}$.
```

Sortie \LaTeX

Liste des X := 1994 1995 1996 1997 1998 1999 2000 2001 2002 2004 2005 2006 2007 2008 2009 2010 .
 Liste des Y := 1718 1710 1708 1700 1698 1697 1691 1688 1683 1679 1671 1670 1663 1661 1656 1649 .
 Somme des X := 32031 et somme des Y := 26942.
 Moyenne des X := 2001.9375 et moyenne des Y := 1683.875.
 Variance des X := 25.43359375 et variance des Y := 403.984375
 Covariance des X/Y := -100.9453125.
 Min des X := 1994 et Max des X := 2010.
 Coefficient $a = -3.968975579788051$. Coefficient $b = 9629.516049761941$.
 Coefficient $r = -0.9958639418357528$. Coefficient $r^2 = 0.9917449906486436$.

Information(s)



Information(s)

Les macros qui contiennent les paramètres de la régression sont donc réutilisables, en tant que nombres réels, donc exploitables par \LaTeX `siunitx` et \LaTeX `xfp` pour affichage *fin*! Ci-dessous un exemple permettant de visualiser tout cela.

</> Code \LaTeX

```
%les espaces verticaux n'ont pas été écrits ici
\def\LstX{0,1,3,4,5,6}
\def\LstY{-35,-37.4,-37.7,-39.9,-39,-39.6}
%on lance les calculs et on change le nom des "macros-résultats"
\CalculsRegLin[NomCoeffa=TESTa,NomCoeffb=TESTb,NomCoeffr=TESTr,NomCoeffrd=TESTrd,%
               NomXmin=TESTmin,NomXmax=TESTmax]{\LstX}{\LstY}
%commandes complémentaires
\DeclareDocumentCommand\arrond{ s 0{3} m }{% * signe / précision / nb
  \IfBooleanTF{#1}{\num[print-implicit-plus]{\fpeval{round(#3,#2)}}}
  → {\num{\fpeval{round(#3,#2)}}}
}
%paramètres
Les valeurs extr. de X sont \TESTmin{} et \TESTmax. Une éq. est $y=\arrond[3]{\TESTa}x
→ \arrond*[3]{\TESTb}$.
Le coeff. de corrélation est $r=\arrond[4]{\TESTr}$, et son carré est
→ $r^2=\arrond[4]{\TESTrd}$.
```

⊕ Sortie \LaTeX

Les valeurs extrêmes de x sont 0 et 6. Une équation de la droite de régression de y en x est

$$y = -0,701x - 35,881.$$

Le coefficient de corrélation linéaire est $r = -0,8918$, et son carré est $r^2 = 0,7954$.

⚙ Information(s)

rad	REGRESSIONS	
Données	Graphique	Stats
X1	Y1	X2
0	-35	
1	-37.4	
3	-37.7	
4	-39.9	
5	-39	
6	-39.6	

rad	REGRESSIONS	
Données	Graphique	Stats
Covariance		-3.133333
$\sum xy$		-742.7
Régression		$y=a \cdot x+b$
a		-0.7006211
b		-35.88137
r		-0.891847
r^2		0.7953911

22.3 Intégration dans un environnement TikZ

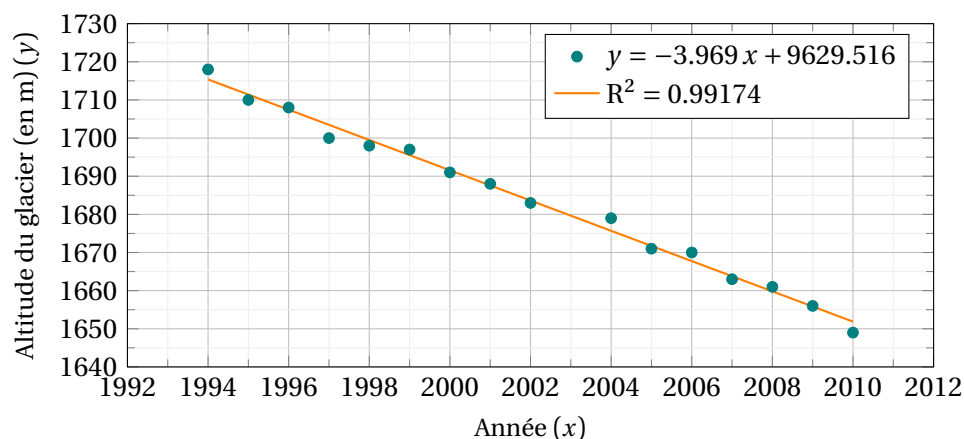
⚙ Information(s)

La commande étant « autonome », elle va pouvoir être intégrée dans des environnements graphiques pour permettre un tracé *facile* de la droite de régression.

</> Code \LaTeX

```
\begin{tikzpicture}
\begin{axis}[options des axes, non présentées ici...]
\addplot[teal, only marks] table{
  X Y
  1994 1718 1995 1710 1996 1708 1997 1700 1998 1698 1999 1697 2000 1691 2001 1688
  2002 1683 2004 1679 2005 1671 2006 1670 2007 1663 2008 1661 2009 1656 2010 1649
};
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
\CalculsRegLin{\LLX}{\LLY}
\addplot [thick,orange,domain=\LXmin:\LXmax,samples=2]{\COEFFa*x+\COEFFb};
\addlegendentry{$y = \fpeval{round(\COEFFa,3)}\,x + \fpeval{round(\COEFFb,3)}$};
\addlegendentry{$R^2=\fpeval{round(\COEFFrd,5)}$};
\end{axis}
\end{tikzpicture}
```


Sortie \LaTeX



Information(s)

Il existe également une commande auxiliaire, `\PointsRegLin` pour afficher le nuage de points avec quelques options, dans un environnement *TikZ* classique (sans *pgfplot*)...

Code \LaTeX

```
...
\begin{tikzpicture}[<options>]
  ...
  \PointsRegLin[clés]{listeX}{listeY}
  ...
\end{tikzpicture}
```

Clés et options

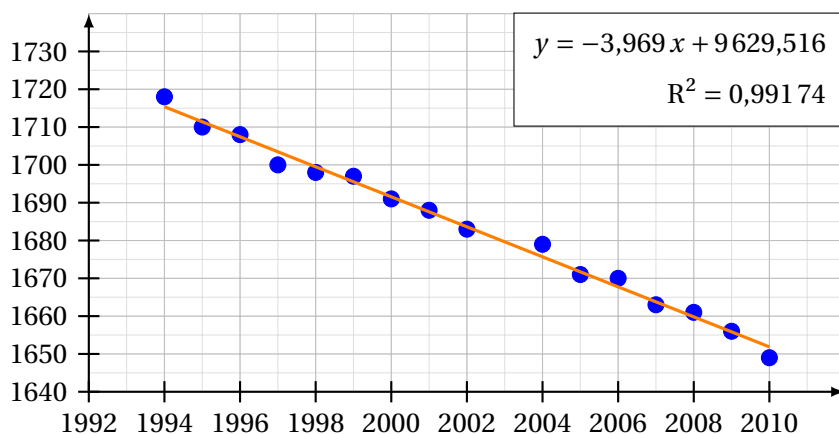
Quelques **<Clés>** sont disponibles pour cette commande, essentiellement pour la mise en forme du nuage :

- la clé **<Couleur>** pour la couleur des points du nuage; défaut **<teal>**
- la clé **<Taille>** pour la taille des points (type *cercle*); défaut **<2pt>**
- la clé **<Ox>** pour spécifier la valeur initiale O_x (si changement d'origine); défaut **<0>**
- la clé **<Oy>** pour spécifier la valeur initiale O_y (si changement d'origine). défaut **<0>**

</> Code L^AT_EX

```
\begin{tikzpicture}[x=0.5cm,y=0.05cm]
  \draw[xstep=1,ystep=5,lightgray!50,very thin] (0,0) grid (20,100);
  \draw[xstep=2,ystep=10,lightgray,thin] (0,0) grid (20,100);
  \draw[thick,->,>=latex] (0,0)--(20,0) ;
  \draw[thick,->,>=latex] (0,0)--(0,100) ;
  \foreach \x in {1992,1994,...,2010} \draw[thick] ({\x-1992},4pt)--({\x-1992},-4pt)
  \node[below] {$\x$} ;
  \foreach \y in {1640,1650,...,1730} \draw[thick] (4pt,{\y-1640})--(-4pt,{\y-1640})
  \node[left] {$\y$} ;
  \def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
  \def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
  \def\Ox{1992}\def\Oy{1640}
  \CalculsRegLin{\LLX}{\LLY}
  \PointsRegLin[Ox=1992,Oy=1640,Couleur=blue,Taille=3pt]{\LLX}{\LLY}
  \draw[orange,very thick,samples=2,domain=\LXmin:\LXmax] plot
  \node {$y=\num{\fpeval{round(\COEFFa,3)}}\x+\num{\fpeval{round(\COEFFb,3)}}$} ; \\\
  \node {$R^2=\num{\fpeval{round(\COEFFrd,5)}}$} ; \\\
};
\end{tikzpicture}
```

⊕ Sortie L^AT_EX



23 Statistiques à deux variables

23.1 Idées

💡 Idée(s)

L'idée est de *prolonger* le paragraphe précédent pour proposer un environnement TikZ adapté à des situations venant de statistiques à deux variables.

Un des soucis pour ces situations est le fait que le repère dans lequel on travaille n'a pas forcément pour origine (0;0).

De ce fait – pour éviter des erreurs de `dimension too large` liées à TikZ – il faut *décaler les axes* pour se ramener à une origine en O.

Le code, intimement lié à un environnement `tikzpicture`, va donc :

- préciser les informations utiles comme `xmin`, `xmax`, `Ox`, `xgrille`, etc
- proposer des commandes (sans se soucier des *translations*!) pour :
 - tracer une grille (principale et/ou secondaire);
 - tracer les axes (avec légendes éventuelles) et éventuellement les graduer;

En utilisant les commandes de régression linéaire du paragraphe précédent, il sera de plus possible (sans calculs!) de :

- représenter le nuage de points;
- placer le point moyen;
- tracer la droite d'ajustement (obtenue par `ProfLycee`) ou une autre courbe.

🔧 Information(s)

Le package `pgfplots` peut être utilisé pour traiter ce genre de situation, mais ne l'utilisant pas, j'ai préféré préparer des macros permettant de s'affranchir de ce package (est-ce pertinent, ça c'est une autre question...).

🔗 Code \LaTeX

```
%Listes et calculs
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
\CalculsRegLin{\LLX}{\LLY}
```

🔗 Code \LaTeX

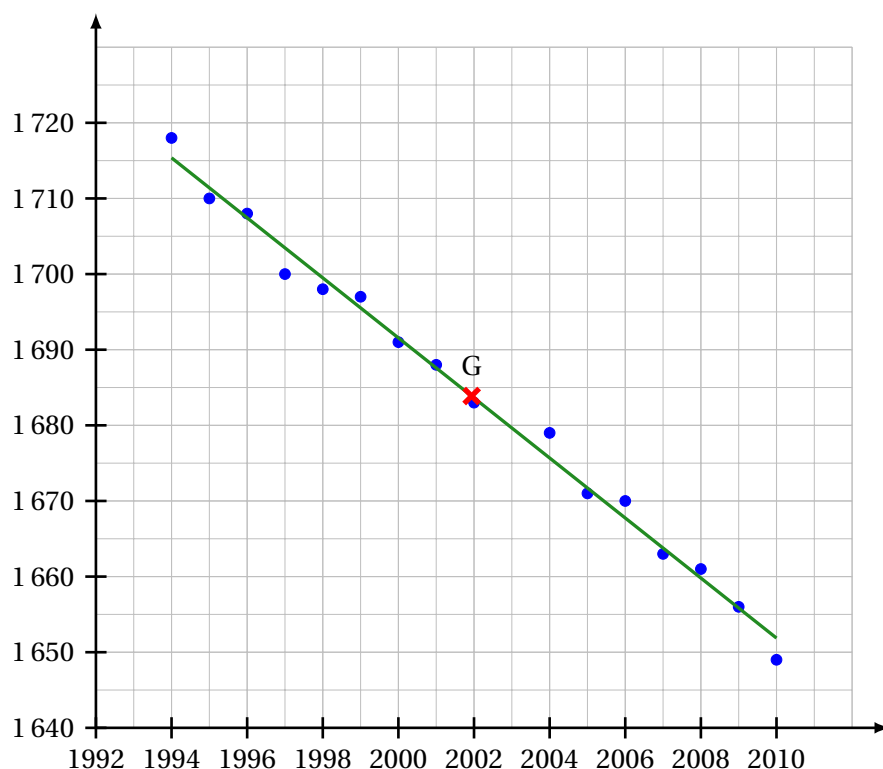
```
%tracé (simple), les options seront présentées juste après
\begin{tikzpicture}%
  [x=0.5cm,y=0.1cm,                                     %unités
  Ox=1992,xmin=1992,xmax=2012,xgrille=2,xgrilles=1,      %axe Ox
  Oy=1640,ymin=1640,ymax=1730,ygrille=10,ygrilles=5]    %axe Oy
  \GrilleTikz \AxesTikz                                  %grilles et axes
  \AxexTikz [Annee] {1992,1994,...,2010}                 %axeOx
  \AxeYtikz {1640,1650,...,1720}                          %axeOy
  \NuagePointsTikz {\LLX}{\LLY}                          %nuage
  \CourbeTikz [line width=1.25pt,ForestGreen,samples=2] %
    {\COEFFa*\x+\COEFFb}{\LXmin:\LXmax}                  %droite de régression
  \PointMoyenTikz                                         %point moyen
\end{tikzpicture}
```

Code \LaTeX

```
%tracé avec options fenêtre par défaut
\begin{tikzpicture}%
[....]
\FenetreSimpleTikz<Annee>{1992,1994,...,2010}{1640,1650,...,1720}
\NuagePointsTikz{\LLX}{\LLY}
\CourbeTikz[line width=1.25pt,ForestGreen,samples=2]%
{\COEFFa*\x+\COEFFb}{\LXmin:\LXmax}
\PLnuageptmoy
\end{tikzpicture}
```

%paramètres
%fenêtre "simple"
%nuage
%droite de régression
%point moyen

Sortie \LaTeX



23.2 Commandes, clés et options

Information(s)

Les **paramètres** nécessaires à la bonne utilisation des commandes suivantes sont à déclarer directement dans l'environnement `\tikzpicture`, seules les versions « x » sont présentées ici :

- **<xmin>**, stockée dans `\xmin`; défaut **<-3>**
- **<xmax>**, stockée dans `\xmax`; défaut **<3>**
- **<Ox>**, stockée dans `\axexOx`, origine de l'axe (Ox); défaut **<0>**
- **<xgrille>**, stockée dans `\xgrille`, graduation principale; défaut **<1>**
- **<xgrilles>**, stockée dans `\xgrilles`, graduation secondaire. défaut **<0.5>**

La fenêtre d'affichage (de sortie) sera donc *portée* par le rectangle de coins (xmin; ymin) et (xmax; ymax); ce qui correspond en fait à la fenêtre TikZ *portée* par le rectangle de coins (xmin - Ox; ymin - Oy) et (xmax-Ox; ymax-Oy).

Les commandes ont – pour certaines – pas mal de **clés** pour des réglages fins, mais dans la majorité des cas elles ne sont pas forcément *utiles*.

Information(s)

Pour illustrer les commandes et options de ce paragraphe, la base sera le graphique présenté précédemment.

Code \LaTeX

```
%...code tikz
\GrilleTikz[options][options grille ppale][options grille second.]
```

Clés et options

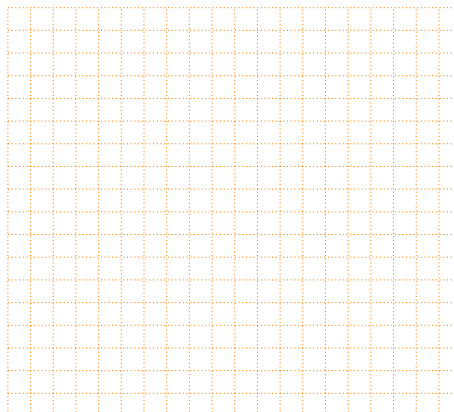
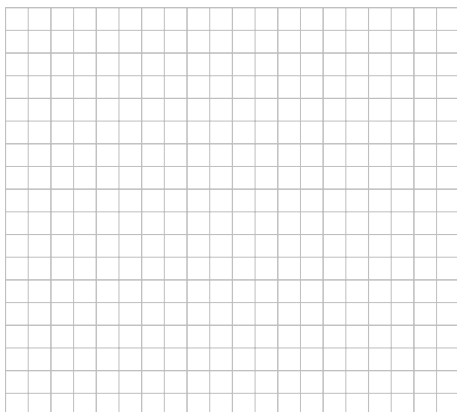
Cette commande permet de tracer une grille principale et/ou une grille secondaire :

- les premières \langle clés \rangle sont les booléens \langle Affp \rangle et \langle Affs \rangle qui affichent ou non les grilles; défaut \langle true \rangle
- les options des grilles sont en TikZ. défaut \langle thin,lightgray \rangle et \langle very thin,lightgray \rangle

Code \LaTeX

```
\begin{tikzpicture}%
[x=0.3cm,y=0.06cm,%
Ox=1992,xmin=1992,xmax=2012,xgrille=2,xgrilles=1,%
Oy=1640,ymin=1640,ymax=1730,ygrille=10,ygrilles=5]
\GrilleTikz
\end{tikzpicture}
~~
\begin{tikzpicture}%
[x=0.3cm,y=0.06cm,%
Ox=1992,xmin=1992,xmax=2012,xgrille=2,xgrilles=1,%
Oy=1640,ymin=1640,ymax=1730,ygrille=10,ygrilles=5]
\GrilleTikz[Affp=false] [] [orange,densely dotted]
\end{tikzpicture}
```

Sortie \LaTeX



Code \LaTeX

```
%...code tikz
\AxesTikz[options]
```

🔑 Clés et options

Cette commande permet de tracer les axes, avec des **clés** :

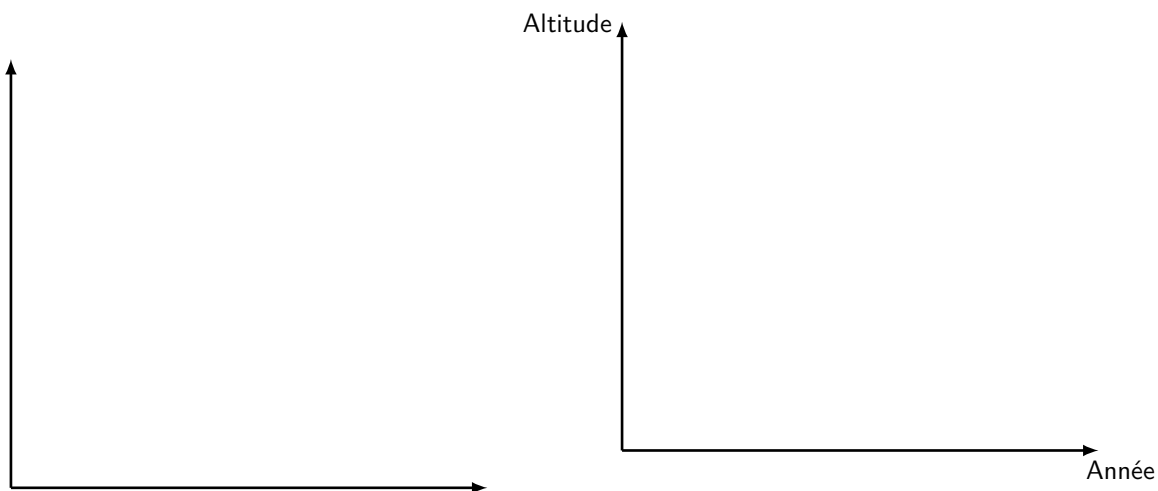
- **Épaisseur** qui est l'épaisseur des traits; défaut **1.25pt**
- **Police** qui est le style des labels des axes; défaut **\normalsize\normalfont**
- **2.1.2 ElargirOx** qui est le % l'élargissement **global** ou **G/D** de l'axe (Ox); défaut **0/0.05**
- **2.1.2 ElargirOy** qui est le % l'élargissement **global** ou **B/H** de l'axe (Oy); défaut **0/0.05**
- **Labelx** qui est le label de l'axe (Ox); défaut **\$x\$**
- **Labely** qui est le label de l'axe (Oy); défaut **\$y\$**
- **AffLabel** qui est le code pour préciser quels labels afficher, entre **x**, **y** ou **xy**; défaut **vide**
- **PosLabelx** pour la position du label de (Ox) en bout d'axe; défaut **right**
- **PosLabely** pour la position du label de (Oy) en bout d'axe; défaut **above**
- **EchelleFleche** qui est l'échelle de la flèche des axes; défaut **1**
- **TypeFleche** qui est le type de la flèche des axes. défaut **latex**

🔗 Code L^AT_EX

```
%code tikz
\AxesTikz

%code tikz
\AxesTikz%
[AffLabel=xy,Labelx={Année},Labely={Altitude},%
PosLabelx={below right},PosLabely={above left},%
Police=\small\sffamily]
```

➕ Sortie L^AT_EX



🔗 Code L^AT_EX

```
%...code tikz
\AxexTikz[options]{valeurs}
\AxyTikz[options]{valeurs}
```

🔑 Clés et options

Ces commande permet de tracer les graduations des axes, avec des **<clés>** identiques pour les deux directions :

- **<Epaisseur>** qui est l'épaisseur des graduations; défaut **<1pt>**
- **<Police>** qui est le style des labels des graduations; défaut **<\normalsize\normalfont>**
- **<PosGrad>** qui est la position des graduations par rapport à l'axe; défaut **<below>** et **<left>**
- **<HautGrad>** qui est la position des graduations (sous la forme **<lgt>** ou **<lgtalgtb>**); défaut **<4pt>**
- le booléen **<AffGrad>** pour afficher les valeurs (formatés avec `\num` donc dépendant de `\sisetup`) des graduations; défaut **<true>**
- le booléen **<AffOrigine>** pour afficher la graduation de l'origine; défaut **<true>**
- le booléen **<Annee>** qui permet de ne pas formater les valeurs des graduations (type année). défaut **<false>**

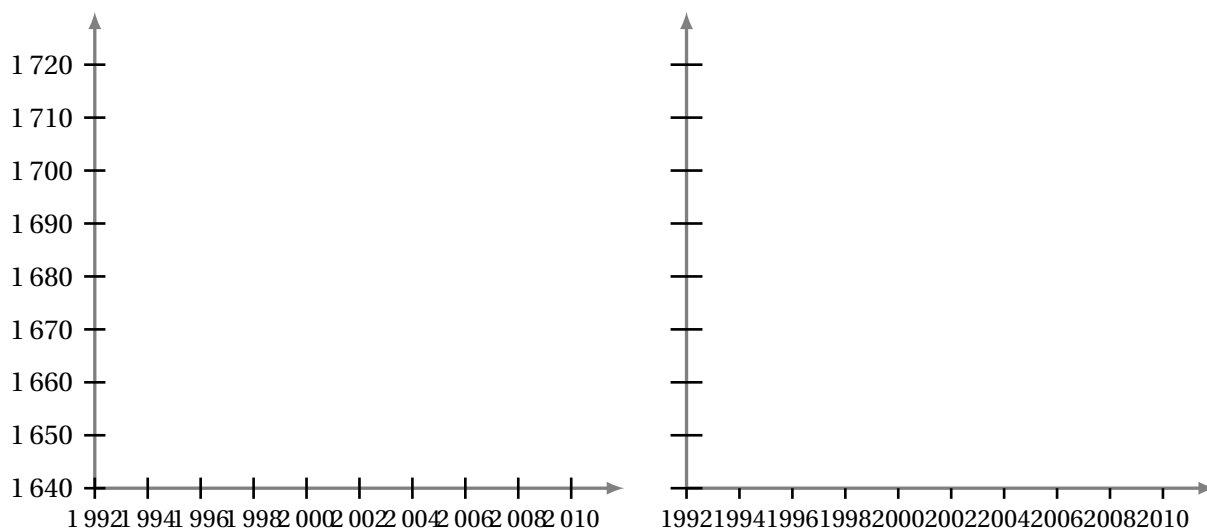
🔗 Code L^AT_EX

```
%code tikz
\AxexTikz[Police=\small]{1992,1994,...,2010}
\AxexTikz{1640,1650,...,1720}

%code tikz
\AxeYtikz[Police=\small,Annee,HautGrad=0pt/4pt]{1992,1994,...,2010}
\AxeYtikz[AffGrad=false,HautGrad=6pt]{1640,1650,...,1720}

%des axes fictifs (en gris) sont rajoutés pour la lisibilité du code de sortie
```

🔗 Sortie L^AT_EX



23.3 Commandes annexes

🔑 Information(s)

Il existe, de manière marginale, quelques commandes complémentaires qui ne seront pas trop détaillées mais qui sont présentes dans l'introduction :

- `\FenetreTikz` qui restreint les tracés à la fenêtre (utile pour des courbes qui *débordent*);
- `\FenetreSimpleTikz` qui permet d'automatiser le tracé des grilles/axes/graduations dans leurs versions par défaut, avec peu de paramétrages;
- `\OrigineTikz` pour rajouter le libellé de l'origine si non affiché par les axes.

</> Code \LaTeX

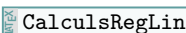
```
%code tikz
\FenetreTikz %on restreint les tracés
\FenetreSimpleTikz[opt](opt axes)<opt axe Ox>\{liste valx\}<opt axe Oy>\{liste valy\}
```

23.4 Interactions avec CalculsRegLin

</> Code \LaTeX

```
%...code tikz
\NuagePointsTikz[options]{listeX}{listeY}
```

🔑 Clés et options

Cette commande, liée à la commande  permet de représenter le nuage de points associé aux deux listes, avec les (clés) suivantes :

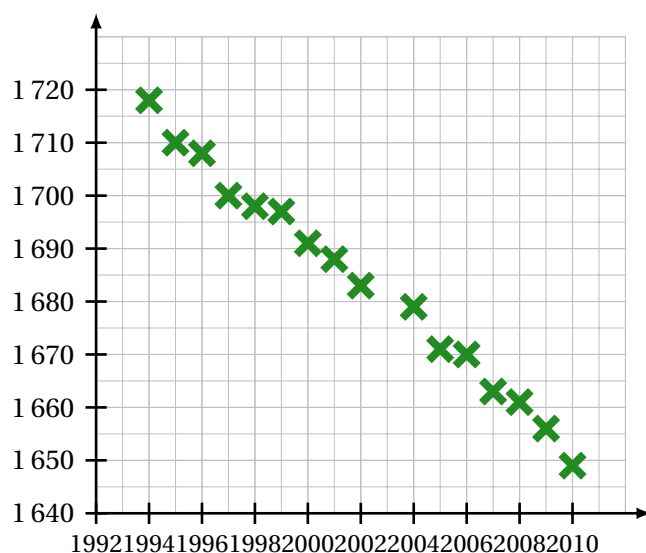
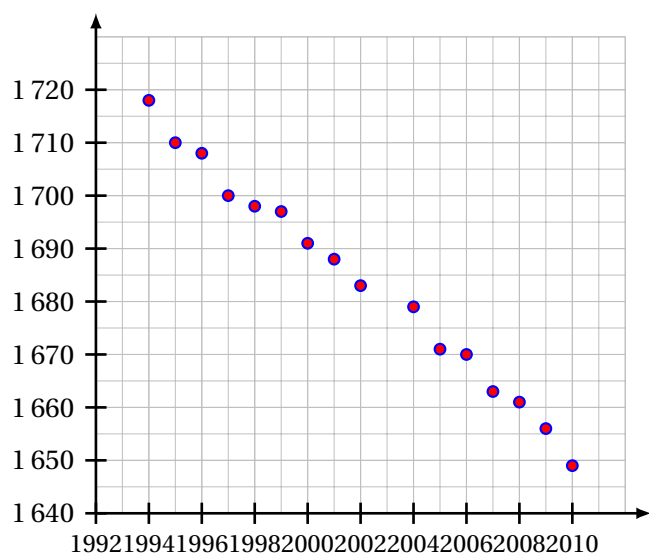
- **<Taille>** qui est la taille des points du nuage; défaut **<2pt>**
- **<Style>** parmi **<o>** (rond) ou **<x>** (croix) ou **<+>** (plus); défaut **<o>**
- **<Couleur>** qui est la couleur (éventuellement **<couleurA/couleurB>** pour les ronds). défaut **<blue>**

</> Code \LaTeX

```
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}

\begin{tikzpicture}[...]
  \NuagePointsTikz[Couleur=blue/red]{\LLX}{\LLY}
\end{tikzpicture}
~~
\begin{tikzpicture}[...]
  \NuagePointsTikz[Couleur=ForestGreen,Style=x,Taille=6pt]{\LLX}{\LLY}
\end{tikzpicture}
```

⊕ Sortie \LaTeX



</> Code \LaTeX

```
%...code tikz
\PointMoyenTikz[options]
```


Clés et options

Cette commande permet de rajouter le point moyen du nuage, calculé par la commande `\CalculsRegLin`, avec les **clés** :

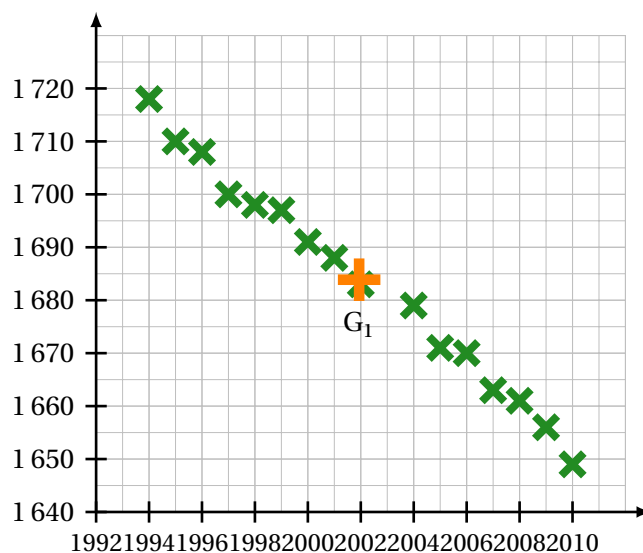
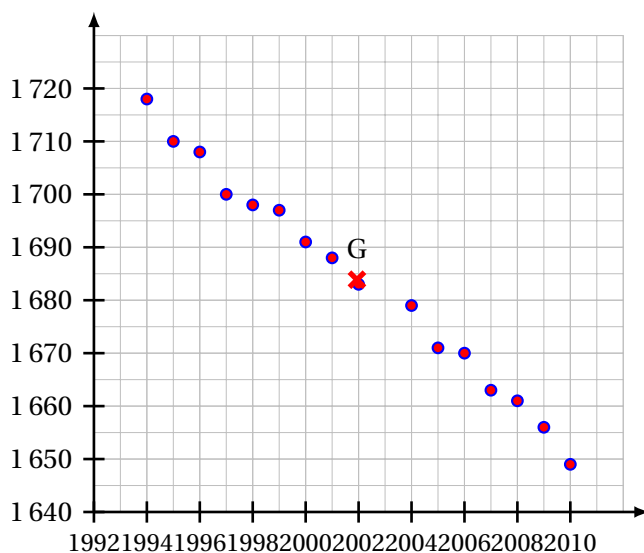
- **<Police>**, comme précédemment; défaut `<\normalsize\normalfont>`;
- **<Taille>**, taille du point moyen; défaut `<4pt>`;
- **<Couleur>**, couleur du point moyen; défaut `<red>`;
- **<Style>** parmi `<o>` (rond) ou `<x>` (croix) ou `<+>` (plus); défaut `<o>`;
- **<yg>**, abscisse du point moyen, récupérable via `\CalculsRegLin`; défaut `<\LXmoy>`;
- **<yg>**, ordonnée du point moyen, récupérable via `\CalculsRegLin`; défaut `<\LYmoy>`;
- **<Nom>**, label du point moyen; défaut `<G>`;
- **<Pos>** qui est la position du label par rapport au point; défaut `<above>`;
- **<Decal>** qui est l'éloignement de la position du label par rapport au point; défaut `<0pt>`;
- la booléen **<AffNom>** qui affiche ou non le libellé. défaut `<true>`;

Code \LaTeX

```
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
\CalculsRegLin{\LLX}{\LLY}

\begin{tikzpicture}[...]
  \NuagePointsTikz[Couleur=blue/red]{\LLX}{\LLY}
  \PointMoyenTikz
\end{tikzpicture}
~~
\begin{tikzpicture}[...]
  \NuagePointsTikz[Couleur=ForestGreen,Style=x,Taille=6pt]{\LLX}{\LLY}
  \PointMoyenTikz[Couleur=orange,Taille=8pt,Style=+,Nom={G_1},Pos=below]
\end{tikzpicture}
```

Sortie \LaTeX



Code \LaTeX

```
%...code tikz
\CourbeTikz[options]{formule}{domaine}
```

🔑 Clés et options

Cette commande permet de rajouter une courbe sur le graphique (sans se soucier de la transformation de son expression) avec les arguments :

- **<optionnels>** qui sont – en TikZ – les paramètres du tracé ;
- le premier *obligatoire*, est – en langage TikZ – l’expression de la fonction à tracer, donc avec $\text{\texttt{\textbackslash x}}$ comme variable ;
- le second *obligatoire* est le domaine du tracé , sous la forme $\text{\texttt{valxmin:valxmax}}$.

🧩 Information(s)

L’idée principale est de récupérer les variables de la régression linéaire pour tracer la droite d’ajustement à *moindres frais*!

🧩 Information(s)

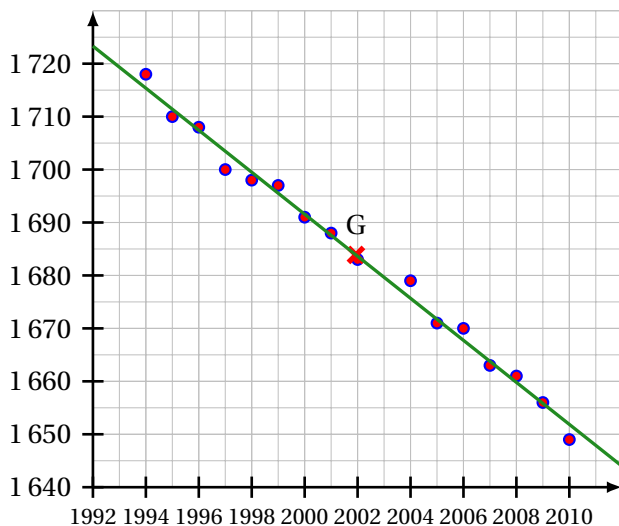
Toute courbe peut être tracée sur ce principe, par contre il faudra saisir la fonction *à la main*.

🔗 Code $\text{\texttt{\LaTeX}}$

```
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}
\CalculsRegLin{\LLX}{\LLY}

\begin{tikzpicture}[...]
  \NuagePointsTikz[Couleur=blue/red]{\LLX}{\LLY} \PointMoyenTikz
  \CourbeTikz[line width=1.25pt,ForestGreen,samples=2]{\COEFFa*\x+\COEFFb}{\xmin:\xmax}
\end{tikzpicture}
```

📤 Sortie $\text{\texttt{\LaTeX}}$

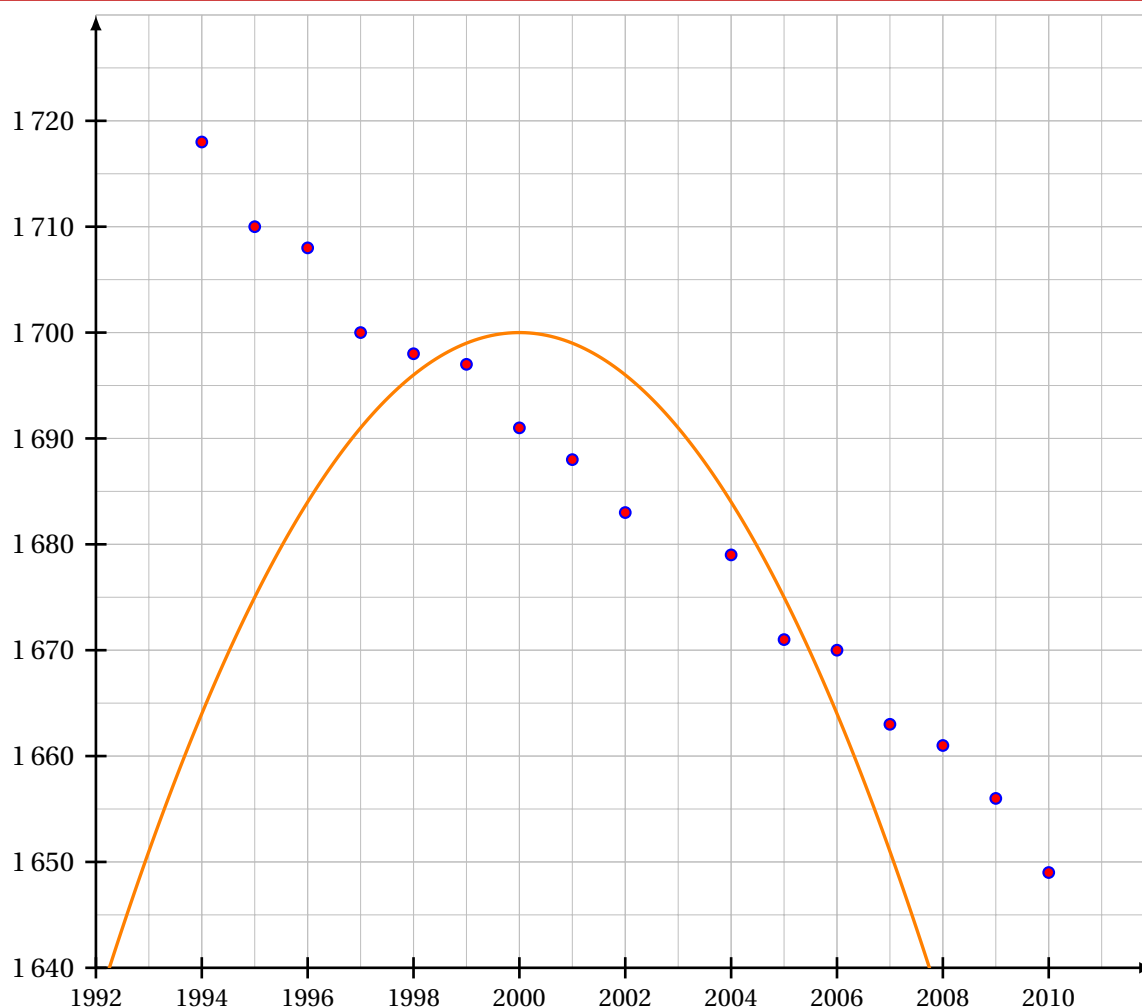


23.5 Exemple complémentaire, pour illustration

Code \LaTeX

```
\def\LLX{1994,1995,1996,1997,1998,1999,2000,2001,2002,2004,2005,2006,2007,2008,2009,2010}  
\def\LLY{1718,1710,1708,1700,1698,1697,1691,1688,1683,1679,1671,1670,1663,1661,1656,1649}  
  
%la courbe n'a pas de lien avec le nuage  
%elle illustre l'interaction des commandes "nuage" avec les autres commandes  
  
\begin{tikzpicture}[...]  
  \NuagePointsTikz[Couleur=blue/red]{\LLX}{\LLY} \FenetreTikz %on fixe la fenêtre  
  \CourbeTikz[line width=1.25pt,orange,samples=250]{-(\x-2000)*(\x-2000)+1700}{\xmin:\xmax}  
\end{tikzpicture}
```

Sortie \LaTeX



24 Boîtes à moustaches

24.1 Introduction

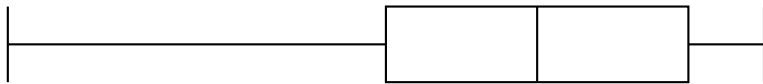
💡 Idée(s)

L'idée est de proposer une commande, à intégrer dans un environnement TikZ, pour tracer une boîte à moustaches grâce aux paramètres, saisis par l'utilisateur.

Le code ne calcule pas les paramètres, il ne fait *que* tracer la boîte à moustaches!

🔗 Code \LaTeX

```
\begin{tikzpicture}
  \BoiteMoustaches{10/15/17/19/20}
\end{tikzpicture}
```



📌 Information(s)

Étant donnée que la commande est intégrée dans un environnement TikZ, les unités peuvent/doivent donc être précisées, *comme d'habitude*, si besoin.

24.2 Clés et options

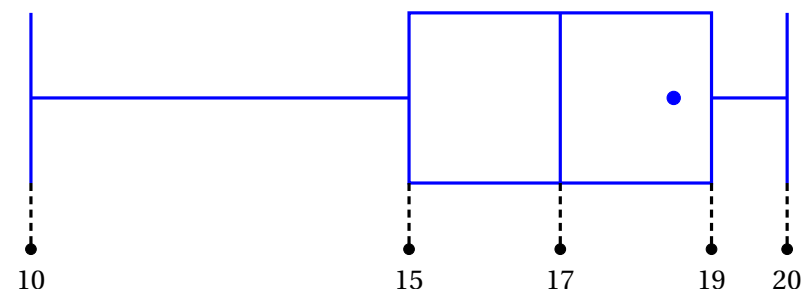
🔗 Clés et options

Quelques **clés** sont disponibles pour cette commande :

- la clé **Couleur** qui est la couleur de la boîte; défaut **black**
- la clé **Elevation** qui est la position verticale (ordonnée des moustaches) de la boîte; défaut **1.5**
- la clé **Hauteur** qui est la hauteur de la boîte; défaut **1**
- la clé **Moyenne** qui est la moyenne (optionnelle) de la série;
- la clé **Epaisseur** qui est l'épaisseur des traits de la boîte; défaut **thick**
- la clé **Remplir** qui est la couleur de remplissage de la boîte; défaut **white**
- le booléen **AffMoyenne** qui permet d'afficher ou non la moyenne (sous forme d'un point); défaut **false**
- le booléen **Pointilles** qui permet d'afficher des pointillés au niveau des paramètres; défaut **false**
- le booléen **Valeurs** qui permet d'afficher les valeurs des paramètres au niveau des abscisses. défaut **false**

🔗 Code \LaTeX

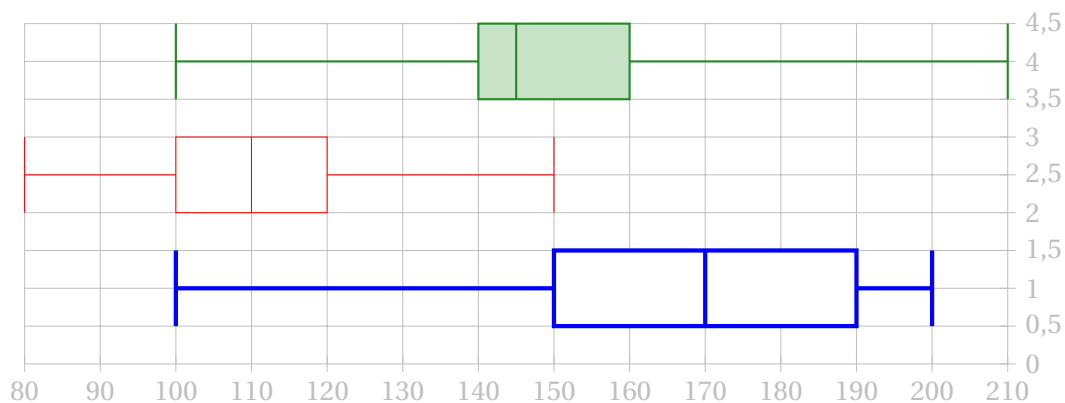
```
\begin{tikzpicture}
  \BoiteMoustaches[Epaisseur=very thick,Moyenne=18.5,Couleur=blue,AffMoyenne,%
  Pointilles,Valeurs,Hauteur=2.25,Elevation=2]{10/15/17/19/20}
\end{tikzpicture}
```



Code \LaTeX

```
%une grille a été rajoutée pour visualiser la "position verticale"
\begin{center}
\begin{tikzpicture}[x=0.1cm]
\BoiteMoustaches[Epaisseur=ultra thick,Couleur=blue]{100/150/170/190/200}
\BoiteMoustaches[Epaisseur=thin,Elevation=2.5,Couleur=red]{80/100/110/120/150}
\BoiteMoustaches%
[Elevation=4,Couleur=ForestGreen,Remplir=ForestGreen!25]{100/140/145/160/210}
\end{tikzpicture}
\end{center}
```

Sortie \LaTeX



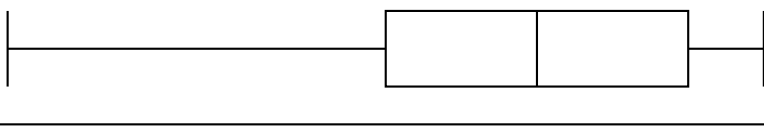
24.3 Commande pour placer un axe horizontal

Idée(s)

L'idée est de proposer, en parallèle de la commande précédente, une commande pour tracer un axe horizontal « sous » les éventuelles boîtes à moustaches.

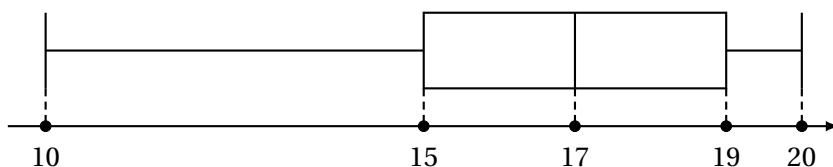
Code \LaTeX

```
\begin{tikzpicture}
\BoiteMoustachesAxe[Min=10,Max=20]
\BoiteMoustaches{10/15/17/19/20}
\end{tikzpicture}
```



Code \LaTeX

```
\begin{tikzpicture}
\BoiteMoustachesAxe[Min=10,Max=20]
\BoiteMoustaches[Valeurs,Pointilles]{10/15/17/19/20}
\end{tikzpicture}
```



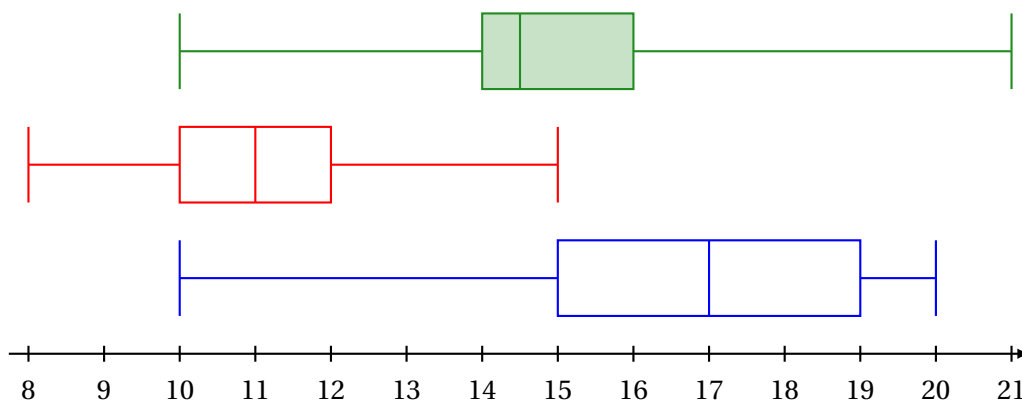
🔑 Clés et options

Quelques **clés** sont disponibles pour cette commande :

- la clé **<Min>** qui est la valeur minimale de l'axe horizontal;
- la clé **<Max>** qui est la valeur maximale de l'axe horizontal;
- la clé **<Elargir>** qui est le pourcentage d'élargissement de l'axe; défaut **<0.1>**
- la clé **<Epaisseur>** qui est l'épaisseur des traits de la boîte; défaut **<thick>**
- la clé **<Valeurs>** qui est la liste (compréhensible en TikZ) des valeurs à afficher.

🔗 Code \LaTeX

```
\begin{tikzpicture}
  \BoiteMoustachesAxe[Min=8,Max=21,AffValeurs,Valeurs={8,9,...,21},Elargir=0.02]
  \BoiteMoustaches[Moyenne=18.5,Couleur=blue]{10/15/17/19/20}
  \BoiteMoustaches[Elevation=2.5,Couleur=red]{8/10/11/12/15}
  \BoiteMoustaches[Elevation=4,Couleur=ForestGreen,Remplir=ForestGreen!25]{10/14/14.5/16/21}
\end{tikzpicture}
```



📌 Information(s)

Le placement des différentes boîtes n'est pas automatique, donc il faut penser à cela avant de se lancer dans le code.

Sachant que la hauteur par défaut est de 1, il est – a priori – intéressant de placer les boîtes à des **<élévations>** de 1 puis 2,5 puis 4 etc

Huitième partie

Outils pour les probabilités

25 Calculs de probabilités

25.1 Introduction

💡 Idée(s)

L'idée est de proposer des commandes permettant de calculer des probabilités avec des lois classiques :

- binomiale;
- normale;
- exponentielle;
- de Poisson;
- géométrique;
- hypergéométrique.

🔧 Information(s)

Les commandes sont de deux natures :

- des commandes pour calculer, grâce au package `xintexpr`;
- des commandes pour formater le résultat de `xintexpr`, grâce à `siunitx`.

De ce fait, les options de `siunitx` de l'utilisateur affecteront les formatages du résultat, la commande va « forcer » les arrondis et l'écriture scientifique.

25.2 Calculs « simples »

🔗 Code \LaTeX

```
%loi binomiale B(n,p)
\CalcBinomP{n}{p}{k}           %P(X=k)
\CalcBinomC{n}{p}{a}{b}        %P(a<=X<=b)

%loi de Poisson P(l)
\CalcPoissP{l}{k}              %P(X=k)
\CalcPoissC{l}{a}{b}           %P(a<=X<=b)

%loi géométrique G(p)
\CalcGeomP{p}{k}               %P(X=k)
\CalcGeomC{l}{a}{b}            %P(a<=X<=b)

%loi hypergéométrique H(N,n,m)
\CalcHypergeomP{N}{n}{m}{k}    %P(X=k)
\CalcHypergeomP{N}{n}{m}{a}{b} %P(a<=X<=b)

%loi normale N(m,s)
\CalcNormC{m}{s}{a}{b}         %P(a<=X<=b)

%loi exponentielle E(l)
\CalcExpoC{l}{a}{b}            %P(a<=X<=b)
```

🔑 Clés et options

Les probabilités calculables sont donc – comme pour beaucoup de modèles de calculatrices – les probabilités **P**onctuelles ($P(X = k)$) et **C**umulées ($P(a \leq X \leq b)$).

Pour les probabilités cumulées, on peut utiliser le caractère `*` comme borne (a ou b), pour les probabilités du type $P(X \leq b)$ et $P(X \geq a)$.

</> Code \LaTeX

```
% X -> B(5,0.4)
$P(X=3) \approx \text{\CalcBinomP}{5}{0.4}{3}$.
$P(X\leqslant 1) \approx \text{\CalcBinomC}{5}{0.4}{*}{1}$.

% X -> B(100,0.02)
$P(X=10) \approx \text{\CalcBinomP}{100}{0.02}{10}$.
$P(15\leqslant X\leqslant 25) \approx \text{\CalcBinomC}{100}{0.02}{15}{25}$.

% Y -> P(5)
$P(Y=3) \approx \text{\CalcPoissP}{5}{3}$.
$P(Y\geqslant 2) \approx \text{\CalcPoissC}{5}{2}{*}$.

% T -> G(0.5)
$P(T=100) \approx \text{\CalcPoissP}{0.5}{3}$.
$P(T\leqslant 5) \approx \text{\CalcPoissC}{0.5}{*}{5}$.

% W -> H(50,10,5)
$P(W=4) \approx \text{\CalcHypergeomP}{50}{10}{5}{4}$.
$P(1\leqslant W\leqslant 3) \approx \text{\CalcHypergeomC}{50}{10}{5}{1}{3}$.
```

↪ Sortie \LaTeX

- $X \hookrightarrow \mathcal{B}(5;0,4)$:
 $P(X = 3) \approx 0.2304$.
 $P(X \leqslant 1) \approx 0.33696$.
- $X \hookrightarrow \mathcal{B}(100;0,02)$:
 $P(X = 10) \approx 0.00002877077765846743$.
 $P(15 \leqslant X \leqslant 25) \approx 0.000000001670210428685021$.
- $Y \hookrightarrow \mathcal{P}_5$:
 $P(Y = 3) \approx 0.1403738958142806$.
 $P(Y \geqslant 2) \approx 0.9595723180054873$.
- $T \hookrightarrow \mathcal{G}_{0,5}$:
 $P(T = 3) \approx 0.125$.
 $P(T \leqslant 5) \approx 0.96875$.
- $W \hookrightarrow \mathcal{H}(50;10;5)$:
 $P(W = 4) \approx 0.003964583058015065$.
 $P(1 \leqslant W \leqslant 3) \approx 0.6853536974456758$.

</> Code \LaTeX

```
% X -> N(0,1)
$P(X\leqslant 1) \approx \text{\CalcNormC}{0}{1}{*}{1}$.
$P(-1,96\leqslant Z\leqslant 1,96) \approx \text{\CalcNormC}{0}{1}{-1.96}{1.96}$.

% X -> N(550,30)
$P(Y\geqslant 600) \approx \text{\CalcNormC}{550}{30}{600}{*}$.
$P(500\leqslant Y\leqslant 600) \approx \text{\CalcNormC}{550}{30}{500}{600}$.

% Z -> E(0.001)
$P(Z\geqslant 400) \approx \text{\CalcExpoC}{0.001}{400}{*}$.
$P(300\leqslant Z\leqslant 750) \approx \text{\CalcExpoC}{0.001}{300}{750}$.
```


⊕ Sortie \LaTeX

- $X \hookrightarrow \mathcal{N}(0;1)$:
 $P(X \leq 1) \approx 0.841344680841397$.
 $P(-1,96 \leq Z \leq 1,96) \approx 0.9500039553976748$.
- $Y \hookrightarrow \mathcal{N}(550;30)$:
 $P(Y \geq 600) \approx 0.0477903462453939$.
 $P(500 \leq Y \leq 600) \approx 0.9044193075092122$.
- $Z \hookrightarrow \mathcal{E}_{0,001}$:
 $P(Z \geq 400) \approx 0.6703200460356393$.
 $P(300 \leq Z \leq 750) \approx 0.2684516679407032$.

25.3 Complément avec sortie « formatée »

💡 Idée(s)

L'idée est ensuite de formater le résultat obtenu par `\xintexpr`, pour un affichage homogène. L'utilisateur peut donc utiliser « sa » méthode pour formater les résultats obtenus par `\xintexpr`!

⌕ Code \LaTeX

```
%avec un formatage manuel  
\num[exponent-mode=scientific]{\CalcBinomP{100}{0.02}{10}}
```

⊕ Sortie \LaTeX

- $X \hookrightarrow \mathcal{B}(100;0,02)$:
 $P(X = 10) \approx 2,877\,077\,765\,846\,743 \times 10^{-5}$.

💡 Idée(s)

Le package `ProfLycee` propose – en complément – des commandes pour formater, grâce à `siunitx`, le résultat.

Les commandes ne sont donc, dans ce cas, pas préfixées par `\calc` :

- formatage sous forme décimale *pure* : $0,00\dots$;
- formatage sous forme scientifique : $n,\dots \times 10^{\dots}$.

Code \LaTeX

```
%loi binomiale  $B(n,p)$ 
\BinomP(*) [prec]{n}{p}{k}           % $P(X=k)$ 
\BinomC(*) [prec]{n}{p}{a}{b}        % $P(a \leq X \leq b)$ 

%loi de Poisson  $P(l)$ 
\PoissonP(*) [prec]{l}{k}           % $P(X=k)$ 
\PoissonC(*) [prec]{l}{a}{b}        % $P(a \leq X \leq b)$ 

%loi géométrique  $G(p)$ 
\GeomP{p}{k}                         % $P(X=k)$ 
\GeomC{l}{a}{b}                     % $P(a \leq X \leq b)$ 

%loi hypergéométrique  $H(N,n,m)$ 
\HypergeomP{N}{n}{m}{k}            % $P(X=k)$ 
\HypergeomC{N}{n}{m}{a}{b}         % $P(a \leq X \leq b)$ 

%loi normale  $N(m,s)$ 
\NormaleC(*) [prec]{m}{s}{a}{b}     % $P(a \leq X \leq b)$ 

%loi exponentielle  $E(l)$ 
\ExpoC(*) [prec]{l}{a}{b}          % $P(a \leq X \leq b)$ 
```

Clés et options

Quelques précisions sur les commandes précédentes :

- la version étoilée $\langle * \rangle$ des commandes formate le résultat en mode scientifique;
- l'argument optionnel (par défaut $\langle 3 \rangle$) correspond à quant à lui à l'arrondi.

Code \LaTeX

```
%  $X \rightarrow N(550,30)$ 
$P(Y \geqslant 600) \approx \text{\NormaleC}[4]{550}{30}{600}{*}$.
$P(500 \leqslant Y \leqslant 600) \approx \text{\NormaleC}[4]{550}{30}{500}{600}$.

%  $X \rightarrow B(100,0.02)$ 
$P(X=10) \approx \text{\BinomP}[7]{100}{0.02}{10} \approx \text{\BinomP*}[7]{100}{0.02}{10}$.
$P(15 \leqslant X \leqslant 25) \approx \text{\BinomC}[10]{100}{0.02}{15}{25} \approx \text{\BinomC*}[10]{100}{0.02}{15}{25}$.

%  $H \rightarrow H(50,10,5)$ 
$P(W=4) \approx \text{\HypergeomP}[5]{50}{10}{5}{4}$.
$P(1 \leqslant W \leqslant 3) \approx \text{\HypergeomC}[4]{50}{10}{5}{1}{3}$.

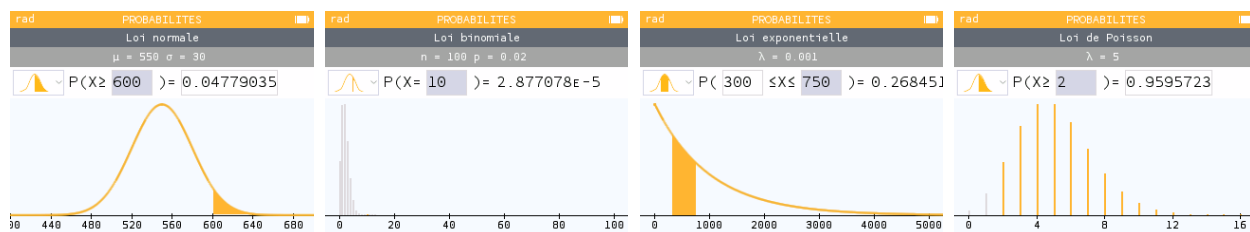
%  $Z \rightarrow E(0,001)$  :
$P(Z \geqslant 400) \approx \text{\ExpoC}[0.001]{400}{*}$.
$P(300 \leqslant Z \leqslant 750) \approx \text{\ExpoC}[0.001]{300}{750}$.

%  $T \rightarrow P(5)$ 
$P(T=3) \approx \text{\PoissonP}[5]{3}$.
$P(T \geqslant 2) \approx \text{\PoissonC}[4]{5}{2}{*}
```

➔ Sortie \LaTeX

- $Y \hookrightarrow \mathcal{N}(550; 30)$:
 $P(Y \geq 600) \approx 0,0478$.
 $P(500 \leq Y \leq 600) \approx 0,9044$.
- $X \hookrightarrow \mathcal{B}(100; 0,02)$:
 $P(X = 10) \approx 0,0000288 \approx 2,88 \times 10^{-5}$.
 $P(15 \leq X \leq 25) \approx 0,000000017 \approx 1,7 \times 10^{-9}$.
- $W \hookrightarrow \mathcal{H}(50; 10; 5)$:
 $P(W = 4) \approx 0,00396$.
 $P(1 \leq W \leq 3) \approx 0,6854$.
- $Z \hookrightarrow \mathcal{E}_{0,001}$:
 $P(Z \geq 400) \approx 0,670$.
 $P(300 \leq Z \leq 750) \approx 0,268$.
- $T \hookrightarrow \mathcal{P}_5$:
 $P(T = 3) \approx 0,140$.
 $P(T \geq 2) \approx 0,9596$.

🧩 Information(s)



26 Arbres de probabilités « classiques »

26.1 Introduction

💡 Idée(s)

L'idée est de proposer des commandes pour créer des arbres de probabilités classiques (et homogènes), en TikZ, de format :

- 2×2 ou 2×3 ;
- 3×2 ou 3×3 .

Les (deux) commandes sont donc liées à un environnement `tikzpicture`, et elles créent les nœuds de l'arbre, pour exploitation ultérieure éventuelle.

🔗 Code \LaTeX

```
%commande simple pour tracé de l'arbre
\ArbreProbasTikz[options]{donnees}

%environnement pour tracé et exploitation éventuelle
\begin{EnvArbreProbasTikz}[options]{donnees}
  code tikz supplémentaire
\end{EnvArbreProbasTikz}
```

26.2 Options et arguments

📌 Information(s)

Les **⟨donnees⟩** seront à préciser sous forme

`<sommet1>/<proba1>/<position1>, <sommet2>/<proba2>/<position2>, ...`

avec comme « sens de lecture » de la gauche vers la droite puis du haut vers le bas (on balaye les *sous-arbres*), avec comme possibilités :

- **2.5.3** une donnée **⟨proba⟩** peut être laissée vide ou spécifiée avec des macros;
- une donnée **⟨position⟩** peut valoir **⟨above⟩** (au-dessus), **⟨below⟩** (en-dessous) ou être laissée **⟨vide⟩** (sur).

🔑 Clés et options

Quelques **⟨Clés⟩** (communes) pour les deux commandes :

- la clé **⟨Unite⟩** pour préciser l'unité de l'environnement TikZ; défaut **⟨1cm⟩**
- la clé **⟨EspaceNiveau⟩** pour l'espace (H) entre les étages; défaut **⟨3.25⟩**
- la clé **⟨EspaceFeuille⟩** pour l'espace (V) entre les feuilles; défaut **⟨1⟩**
- la clé **⟨Type⟩** pour le format, parmi **⟨2x2⟩** ou **⟨2x3⟩** ou **⟨3x2⟩** ou **⟨3x3⟩**; défaut **⟨2x2⟩**
- la clé **⟨Police⟩** pour la police des nœuds; défaut **⟨\normalfont\normalsize⟩**
- la clé **⟨PoliceProbas⟩** pour la police des probas; défaut **⟨\normalfont\small⟩**
- le booléen **⟨InclineProbas⟩** pour incliner les probas; défaut **⟨true⟩**
- le booléen **⟨Fleche⟩** pour afficher une flèche sur les branches; défaut **⟨false⟩**
- la clé **⟨StyleTrait⟩** pour les branches, en langage TikZ; défaut **⟨vide⟩**
- la clé **⟨EpaisseurTrait⟩** pour l'épaisseur des branches, en langage TikZ; défaut **⟨semithick⟩**

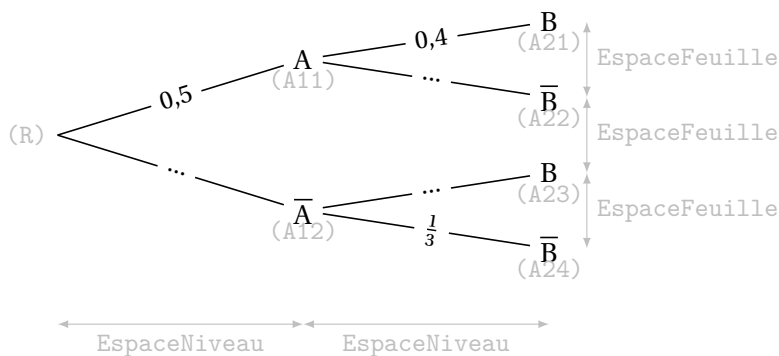
Code \LaTeX

```
\def\ArbreDeuxDeux{
  $A$/\num{0.5}/,
  $B$/\num{0.4}/,
  $\overline{B}$/\dots/,
  $\overline{A}$/\dots/,
  $B$/\dots/,
  $\overline{B}$/$\frac{1}{3}$/$/
}

\ArbreProbasTikz{\ArbreDeuxDeux}
```

%des éléments, en gris, ont été rajoutés pour illustrer certaines options

Sortie \LaTeX



Information(s)

Les nœuds créés par les commandes sont :

- \LaTeX R pour la racine;
- \LaTeX A1x pour les nœuds du 1^{er} niveau (de haut en bas);
- \LaTeX A2x pour les nœuds du 2^d niveau (de haut en bas).

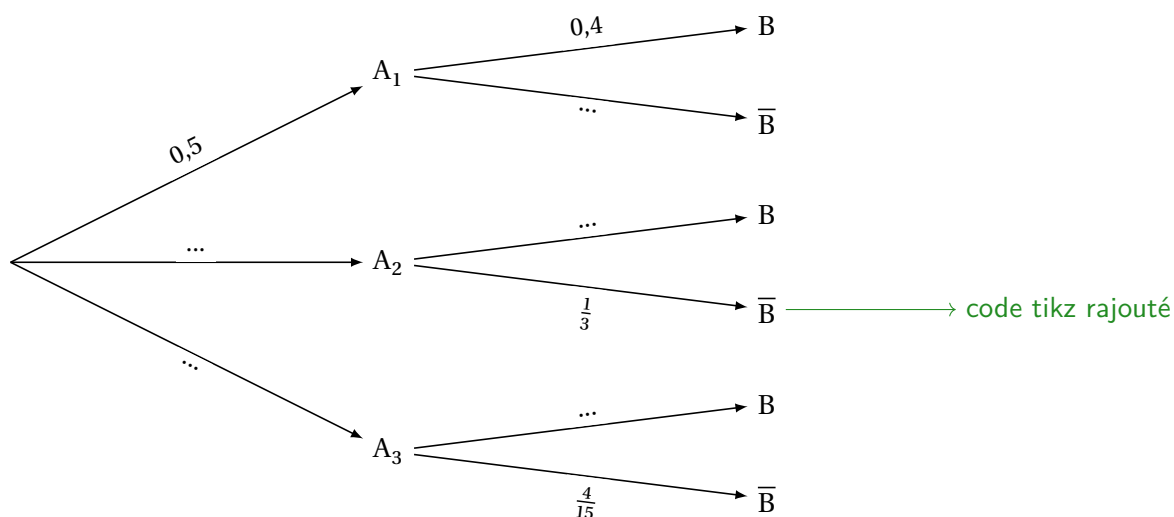
26.3 Exemples complémentaires

Code \LaTeX

```
\def\ArbreTroisDeux{
  $A_1$/\num{0.5}/above,
  $B$/\num{0.4}/above,
  $\overline{B}$/\dots/below,
  $A_2$/\dots/above,
  $B$/\dots/above,
  $\overline{B}$/$\frac{1}{3}$/$/below,
  $A_3$/\dots/below,
  $B$/\dots/above,
  $\overline{B}$/$\frac{4}{15}$/$/below
}

\begin{EnvArbreProbasTikz}[Type=3x2,Fleche,EspaceNiveau=5,EspaceFeuille=1.25]{\ArbreTroisDeux}
  \draw[ForestGreen,->] (A24)--($ (A24)+(2.5,0) $) node[right,font=\sffamily] {code tikz
    ↳ rajouté} ;
\end{EnvArbreProbasTikz}
```

Sortie \LaTeX



Code \LaTeX

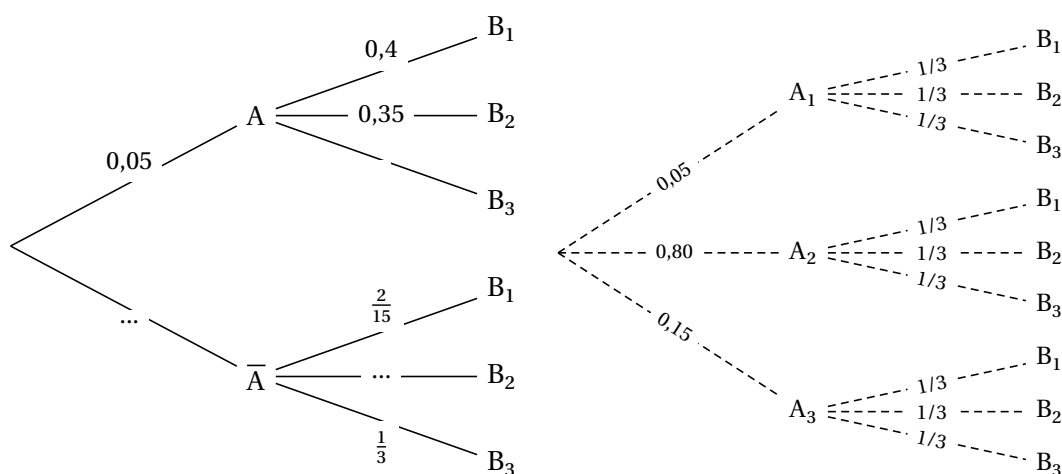
```
\def\ArbreDeuxTrois{
  $\$/\num{0.05}/above,
  $\$B_1$/\num{0.4}/above,$B_2$/\num{0.35}/,$B_3$/below,
  $\overline{A}$/\dots/below,
  $\$B_1$/\frac{2}{15}/above,$B_2$/\dots/,$B_3$/\frac{1}{3}/below
}

\ArbreProbasTikz[Type=2x3,InclineProbas=false,EspaceFeuille=1.15]{\ArbreDeuxTrois}

\def\ArbreTroisTrois{
  $A_1$/\num{0.05}/,$B_1$/\frac{1}{3}/,$B_2$/\frac{1}{3}/,$B_3$/\frac{1}{3}/,
  $A_2$/\num{0.80}/,$B_1$/\frac{1}{3}/,$B_2$/\frac{1}{3}/,$B_3$/\frac{1}{3}/,
  $A_3$/\num{0.15}/,$B_1$/\frac{1}{3}/,$B_2$/\frac{1}{3}/,$B_3$/\frac{1}{3}/
}

\ArbreProbasTikz[Type=3x3,StyleTrait={densely
  \dashed},EspaceFeuille=0.7,PoliceProbas=\scriptsize,Police=\small]{\ArbreTroisTrois}
```

Sortie \LaTeX



27 Petits schémas pour des probabilités continues

27.1 Idée

💡 Idée(s)

L'idée est de proposer des commandes pour illustrer, sous forme de schémas en TikZ, des probabilités avec des lois continues (normales et exponentielles).

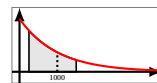
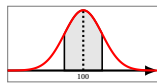
Ces « schémas » peuvent être insérés en tant que graphique explicatif, ou bien en tant que petite illustration rapide!

🔗 Code \LaTeX

```
\LoiNormaleGraphe[options]<options tikz>\m\{s\}{a}\{b\}
```

```
\LoiExpoGraphe[options]<options tikz>\l\{a\}\{b\}
```

➡ Sortie \LaTeX



🔑 Clés et options

Les probabilités *illustrables* sont donc des probabilités Cumulées ($P(a \leq X \leq b)$).

On peut utiliser $\text{\texttt{inf}}^*$ comme borne (a ou b), pour les probabilités du type $P(X \leq b)$ et $P(X \geq a)$.

27.2 Commandes et options

🔑 Clés et options

Quelques **Clés** sont disponibles pour ces commandes :

- | | |
|---|-------------------------|
| — la clé CouleurAire pour l'aire sous la courbe; | défaut LightGray |
| — la clé CouleurCourbe pour la courbe; | défaut red |
| — la clé Largeur qui sera la largeur (en cm) du graphique; | défaut 2 |
| — la clé Hauteur qui sera la hauteur (en cm) du graphique; | défaut 1 |
| — un booléen AfficheM qui affiche la moyenne; | défaut true |
| — un booléen AfficheCadre qui affiche un cadre pour délimiter le schéma. | défaut true |

📌 Information(s)

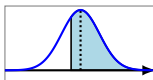
Les commandes sont donc des environnements TikZ, sans possibilité de « rajouter » des éléments. Ces petits schémas sont donc vraiment dédiés à *montrer* rapidement une probabilité continue, sans fioriture.

🔗 Code \LaTeX

Avec centrage vertical sur l'axe des abscisses :

```
\LoiNormaleGraphe  
[AfficheM=false,CouleurCourbe=Blue,CouleurAire=LightBlue]<baseline=0pt>\{1000\}\{100\}\{950\}\{*\}
```

Avec centrage vertical sur l'axe des abscisses :



</> Code \LaTeX

Avec quelques modifications :

```
\LoiNormaleGraphe[Largeur=4,Hauteur=2]{150}{12.5}{122}{160}
```

```
\medskip
```

Avec centrage vertical :

```
\LoiNormaleGraphe[Largeur=5,Hauteur=2.5]<baseline=(current bounding  
↪ box.center)>{200}{5}{204}{*}
```

```
\medskip
```

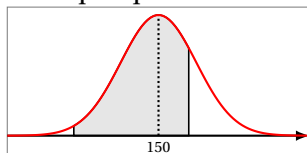
Avec centrage vertical sur l'axe des abscisses :

```
\LoiExpoGraphe  
[AfficheM=false,CouleurCourbe=Blue,CouleurAire=LightBlue]<baseline=0pt>{0.05}{*}{32}
```

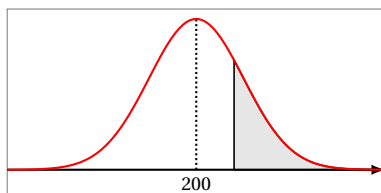
```
\medskip
```

```
\LoiExpoGraphe[Largeur=4,Hauteur=2]{0.00025}{5000}{*}
```

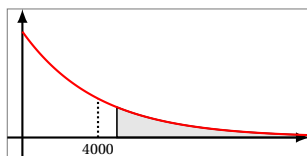
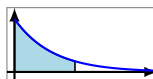
Avec quelques modifications :



Avec centrage vertical :



Avec centrage vertical sur l'axe des abscisses :



27.3 Remarques et compléments

Information(s)

Pour le moment, seules les lois (continues) exponentielles et normales sont disponibles, peut-être que d'autres lois seront ajoutées, mais il ne me semble pas très pertinent de proposer des schémas similaires pour des lois discrètes, qui ont des *représentations* assez variables...

28 Nombres aléatoires

28.1 Idée

💡 Idée(s)

2.0.9 L'idée est de proposer des commandes pour générer des nombres aléatoires, pour exploitation ultérieure :

- un entier ou un nombre décimal;
- des nombres entiers, avec ou sans répétitions.

🔧 Information(s)

Pour chacune des commandes, le ou les résultats sont stockés dans une macro dont le nom est choisi par l'utilisateur.

🔗 Code \LaTeX

```
%entier aléatoire entre a et b
\NbAlea{a}{b}{macro}

%nombre décimal (n chiffres après la virgule) aléatoire entre a et b+1 (exclus)
\NbAlea[n]{a}{b}{macro}

%création d'un nombre aléatoire sous forme d'une macro
\VarNbAlea{macro}{calculs}

%liste d'entiers aléatoires
\TirageAleatoireEntiers[options]{macro}
```

🔗 Code \LaTeX

```
%nombre aléatoire entre 1 et 50, stocké dans \PremierNbAlea
Entier entre 1 et 50 : \NbAlea{1}{50}{\PremierNbAlea}\PremierNbAlea
%nombre aléatoire créé à partir du 1er, stocké dans \DeuxiemeNbAlea
Entier à partir du précédent :
  ↳ \VarNbAlea{\DeuxiemeNbAlea}{\PremierNbAlea+randint(0,10)}\DeuxiemeNbAlea
%nombre aléatoire décimal (au millième) entre 0 et 10+1 (exclus), stocké dans \PremierDecAlea
Décimal entre 0 et $10,999\ldots$ : \NbAlea[3]{0}{10}{\PremierDecAlea}\PremierDecAlea
%liste de 6 nombres, sans répétitions, entre 1 et 50
Liste par défaut (6 entre 1 et 50) :
  ↳ \TirageAleatoireEntiers{\PremiereListeAlea}\PremiereListeAlea
```

Entier entre 1 et 50 : 12Entier à partir du précédent : 19Décimal entre 0 et 10,999... : 5.553Liste par défaut (6 entre 1 et 50) : 21,27,47,18,31,12

🔧 Information(s)

Les listes créées sont exploitables, *a posteriori*, par le package `listofitems` par exemple!

🔗 Code \LaTeX

```
Liste générée : \TirageAleatoireEntiers{\TestListeA}\TestListeA

Liste traitée : \readlist*\LISTEa{\TestListeA}\showitems{\LISTEa}
```

Liste générée : 37,16,12,36,38,11
Liste traitée :

37	16	12	36	38	11
----	----	----	----	----	----

28.2 Clés et options

Clés et options

Quelques clés sont disponibles pour la commande `\TirageAleatoireEntiers` :

- la clé **⟨ValMin⟩** pour préciser borne inférieure de l'intervalle; défaut **⟨1⟩**
- la clé **⟨ValMax⟩** pour préciser borne supérieure de l'intervalle; défaut **⟨50⟩**
- la clé **⟨NbVal⟩** qui est le nombre d'entiers à générer; défaut **⟨6⟩**
- la clé **⟨Sep⟩** pour spécifier le séparateur d'éléments; défaut **⟨,⟩**
- la clé **⟨Tri⟩** parmi **⟨non/croissant/décroissant⟩** pour trier les valeurs; défaut **⟨non⟩**
- le booléen **⟨Repetition⟩** pour autoriser la répétition d'éléments. défaut **⟨false⟩**

Code \LaTeX

Une liste de 15 valeurs (différentes), entre 10 et 100, stockée dans la macro MaListeA : `\\`
Liste : `\TirageAleatoireEntiers[ValMin=10,ValMax=100,NbVal=15]{\MaListeA}\MaListeA \\`

Une liste de 12 valeurs (différentes), entre 1 et 50, ordre croissant : `\\`
Liste : `\TirageAleatoireEntiers[ValMin=1,ValMax=50,NbVal=12,Tri=croissant]{\MaListeB}\MaListeB`
`~ \\`

Une liste de 12 valeurs (différentes), entre 1 et 50, ordre décroissant : `\\`
Liste :
`~ \TirageAleatoireEntiers[ValMin=1,ValMax=50,NbVal=12,Tri=décroissant]{\MaListeC}\MaListeC`
`~ \\`

15 tirages de dé à 6 faces : `\\`
`~ \TirageAleatoireEntiers[ValMin=1,ValMax=6,NbVal=15,Repetition]{\TestDes}\TestDes`

Une liste de 15 valeurs (différentes), entre 10 et 100, stockée dans la macro MaListeA :
Liste : 21,45,49,14,60,54,73,43,28,10,46,76,22,84,93

Une liste de 12 valeurs (différentes), entre 1 et 50, ordre croissant :
Liste : 3,5,7,9,13,14,16,22,25,35,41,49

Une liste de 12 valeurs (différentes), entre 1 et 50, ordre décroissant :
Liste : 50,48,42,35,34,25,22,21,20,14,3,2

15 tirages de dé à 6 faces :
5,6,4,2,3,4,6,3,4,5,5,5,6,4,2

Code \LaTeX

Une liste (10) pour le Keno \textcopyright , ordonnée, et séparée par des `\texttt{-}` :

```
\TirageAleatoireEntiers[ValMin=1,ValMax=70,NbVal=10,Tri=croissant,Sep={-}]{\ListeKeno}  
$\ListeKeno$  
  
\setsepchar{-}\readlist*\KENO{\ListeKeno}\showitems{\KENO}
```

Une liste (10) pour le Keno $\text{\textcircled{C}}$, ordonnée, et séparée par des `-` :

5 – 8 – 9 – 22 – 26 – 28 – 29 – 39 – 40 – 65

5	8	9	22	26	28	29	39	40	65
---	---	---	----	----	----	----	----	----	----

29 Combinatoire

29.1 Idée

💡 Idée(s)

L'idée est de proposer une commande pour calculer un arrangement ou une combinaison, en utilisant les capacités de calcul du package `\xint` (2.5.4).

🔗 Code L^AT_EX

```
\Arrangement(*)[option]{p}{n}
\Combinaison(*)[option]{p}{n}
\CalculAnp{p}{n} ou \CalculCnp{p}{n} dans un calcul via \xinteval{...}
```

29.2 Utilisation

🔑 Clés et options

Peu de paramétrage pour ces commandes qui permettent de calculer A_n^p et $\binom{n}{p}$:

- les versions étoilées ne formatent pas le résultat grâce à `\num` de `\sinuitx`;
- le booléen **<Notation>** pour avoir la notation au début; défaut **<false>**
- le booléen **<Formule>** permet de présenter la formule avant le résultat; défaut **<false>**
- le premier argument, *obligatoire*, est la valeur de p ;
- le second argument, *obligatoire*, est la valeur de n .

🔗 Code L^AT_EX

On a $A_{20}^3 = \text{\texttt{\textbackslash Arrangement}\{3\}\{20\}}$ en non formaté,
et $\text{\texttt{\textbackslash Arrangement}\{3\}\{20\}}$ en formaté avec la notation au début.

On a $A_{20}^3 = 6840$ en non formaté, et $A_{20}^3 = 6840$ en formaté avec la notation au début.

🔗 Code L^AT_EX

On a $\text{\texttt{\textbackslash displaystyle\binom}\{20\}\{3\}} = \text{\texttt{\textbackslash Combinaison}\{3\}\{20\}}$ en non formaté,~
et $\text{\texttt{\textbackslash displaystyle\binom}\{20\}\{3\}}$ en formaté avec la notation au début.~
Et $\text{\texttt{\textbackslash dbinom}\{20\}\{3\}} + \text{\texttt{\textbackslash dbinom}\{20\}\{4\}} = \text{\texttt{\textbackslash num}\{\text{\texttt{\textbackslash xinteval}\{\text{\texttt{\textbackslash CalculCnp}\{3\}\{20\}} + \text{\texttt{\textbackslash CalculCnp}\{4\}\{20\}}\}}}$.

On a $\binom{20}{3} = 1140$ en non formaté, et $\binom{20}{3} = 1140$ en formaté avec la notation au début.

Et $\binom{20}{3} + \binom{20}{4} = 5985$.

🔗 Code L^AT_EX

On a $\text{\texttt{\textbackslash displaystyle\Arrangement}\{3\}\{20\}}$.

On a $A_{20}^3 = \frac{20!}{17!} = 6840$.

🔗 Code L^AT_EX

On a $\text{\texttt{\textbackslash displaystyle\Combinaison}\{3\}\{20\}}$. %ancienne notation FR

On a $C_{20}^3 = \frac{20!}{3! \times 17!} = 1140$.

Outils pour l'arithmétique

30 Conversions binaire/hexadécimal/décimal

30.1 Idée

💡 Idée(s)

L'idée est de *compléter* les possibilités offertes par le package `xintbinhex`, en mettant en forme quelques conversions :

- décimal en binaire avec blocs de 4 chiffres en sortie;
- conversion binaire ou hexadécimal en décimal avec écriture polynomiale.

🔧 Information(s)

Le package `xintbinhex` est la base de ces macros, puisqu'il permet de faire des conversions directes!

Les macros présentées ici ne font que les intégrer dans un environnement adapté à une correction ou une présentation!

🔗 Code \LaTeX

```
\xintDecToHex{100}
\xintDecToBin{51}
\xintHexToDec{A4C}
\xintBinToDec{110011}
\xintBinToHex{11111111}
\xintHexToBin{ACDC}
\xintCHexToBin{3F}
```

➡ Sortie \LaTeX

```
64
110011
2636
51
FF
1010110011011100
00111111
```

30.2 Conversion décimal vers binaire

🔗 Code \LaTeX

```
\ConversionDecBin(*)[clés]{nombre}
```

🔑 Clés et options

Concernant la commande en elle même, peu de paramétrage :

- la version *étoilée* qui permet de ne pas afficher de zéros avant pour « compléter »;
- le booléen `\AffBase` qui permet d'afficher ou non la base des nombres; défaut `(true)`
- l'argument, *obligatoire*, est le nombre entier à convertir.

Le formatage est géré par `\sinuitx`, le mieux est donc de positionner la commande dans un environnement mathématique.

Les nombres écrits en binaire sont, par défaut, présentés en bloc(s) de 4 chiffres.

</> Code \LaTeX

```
% Conversion avec affichage de la base et par bloc de 4
 $\backslash\text{ConversionDecBin}\{415\}$ 
% Conversion avec affichage de la base et sans forcément des blocs de 4
 $\backslash\text{ConversionDecBin}^*\{415\}$ 
% Conversion sans affichage de la base et par bloc de 4
 $\backslash\text{ConversionDecBin}[\text{AffBase=false}]\{415\}$ 
% Conversion sans affichage de la base et sans forcément des blocs de 4
 $\backslash\text{ConversionDecBin}^*[\text{AffBase=false}]\{415\}$ 
```

⊕ Sortie \LaTeX

```
41510 = 0001 1001 11112
41510 = 1 1001 11112
415 = 0001 1001 1111
415 = 1 1001 1111
```

30.3 Conversion binaire vers hexadécimal

✚ Information(s)

L'idée est ici de présenter la conversion, grâce à la conversion « directe » par blocs de 4 chiffres :

- la macro rajoute éventuellement les zéros pour compléter;
- elle découpe par blocs de 4 chiffres binaires;
- elle présente la conversion de chacun des blocs de 4 chiffres binaires;
- elle affiche la conversion en binaire.


</> Code \LaTeX

```
 $\backslash\text{ConversionBinHex}[\text{clés}]\{\text{nombre}\}$ 
```

🔗 Clés et options

Quelques **<clés>** sont disponibles pour cette commande :

- le booléen **<AffBase>** qui permet d'afficher ou non la base des nombres; défaut **<true>**
- le booléen **<Details>** qui permet d'afficher ou le détail par bloc de 4. défaut **<true>**

Le formatage est géré par le package  `sinuitx`, le mieux est de positionner la commande dans un environnement mathématique.

</> Code \LaTeX

```
%conversion avec détails et affichage de la base
 $\backslash\text{ConversionBinHex}\{110011111\}$ 
%conversion sans détails et affichage de la base
 $\backslash\text{ConversionBinHex}[\text{Details=false}]\{110011111\}$ 
%conversion sans détails et sans affichage de la base
 $\backslash\text{ConversionBinHex}[\text{AffBase=false,Details=false}]\{110011111\}$ 
```

⊕ Sortie \LaTeX

```
1 1001 11112 = 0001 1001 1111 =  $\frac{0001}{1} \frac{1001}{9} \frac{1111}{F} = 19F_{16}$ 
1 1001 11112 =  $19F_{16}$ 
1 1001 1111 = 19F
```

30.4 Conversion binaire ou hexadécimal en décimal

Information(s)

L'idée est ici de présenter la conversion, grâce à l'écriture polynômiale :

- écrit la somme des puissances ;
- convertir si besoin les *chiffres* hexadécimal ;
- peut ne pas afficher les monômes de coefficient 0.

Code \LaTeX

```
\ConversionVersDec[clés]{nombre}
```

Clés et options

Quelques **clés** sont disponibles pour cette commande :

- la clé **<BaseDep>** qui est la base de départ (2 ou 16!) ; défaut **<2>**
- le booléen **<AffBase>** qui permet d'afficher ou non la base des nombres ; défaut **<true>**
- le booléen **<Details>** qui permet d'afficher ou le détail par bloc de 4 ; défaut **<true>**
- le booléen **<Zeros>** qui affiche les chiffres 0 dans la somme. défaut **<true>**

Le formatage est toujours géré par le package `sinuitx`, le mieux est de positionner la commande dans un environnement mathématique.

Code \LaTeX

```
%conversion 16->10 avec détails et affichage de la base et zéros
$\ConversionVersDec[BaseDep=16]{19F}$
%conversion 2->10 avec détails et affichage de la base et zéros
$\ConversionVersDec{110011}$
%conversion 2->10 avec détails et affichage de la base et sans zéros
$\ConversionVersDec[Zeros=false]{110011}$
%conversion 16->10 sans détails et affichage de la base et avec zéros
$\ConversionVersDec[BaseDep=16,Details=false]{AC0DC}$
%conversion 16->10 avec détails et sans affichage de la base et sans zéros
$\ConversionVersDec[Eeros=false,Basedep=16]{AC0DC}$
```

Sortie \LaTeX

```
19F16 = 1 × 162 + 9 × 161 + 15 × 160 = 41510
1100112 = 1 × 25 + 1 × 24 + 0 × 23 + 0 × 22 + 1 × 21 + 1 × 20 = 5110
1100112 = 1 × 25 + 1 × 24 + 1 × 21 + 1 × 20 = 5110
AC0DC16 = 70473210
AC0DC16 = 10 × 164 + 12 × 163 + 13 × 161 + 12 × 160 = 70473210
```

31 Conversion « présentée » d'un nombre en base décimale

31.1 Idée

💡 Idée(s)

L'idée est de proposer une « présentation » par divisions euclidiennes pour la conversion d'un entier donné en base 10 dans une base quelconque.

Les commandes de la section précédente donne *juste* les résultats, dans cette section il y a en plus la présentation de la conversion.

La commande utilise – par défaut – du code TikZ en mode `\tikz[overlay]`, donc on pourra déclarer – si ce n'est pas fait – dans le préambule, la commande qui suit.

🔗 Code \LaTeX

```
...
\tikzstyle{every picture}+=[remember picture]
...
```

31.2 Code et clés

🔗 Code \LaTeX

```
%conversion basique
\ConversionDepuisBaseDix{78}{2}
```

$$\left\{ \begin{array}{l} 78 = 2 \times 39 + 0 \\ 39 = 2 \times 19 + 1 \\ 19 = 2 \times 9 + 1 \\ 9 = 2 \times 4 + 1 \\ 4 = 2 \times 2 + 0 \\ 2 = 2 \times 1 + 0 \\ 1 = 2 \times 0 + 1 \end{array} \right. \Rightarrow 78_{10} = 1001110_2$$

📌 Information(s)

La « tableau », qui est géré par `\array` est inséré dans un `\ensuremath`, donc les `\$...\$` ne sont pas utiles.

🔗 Code \LaTeX

```
\ConversionDepuisBaseDix[options]{nombre en base 10}{base d'arrivée}
```

🔑 Clés et options

Quelques options pour cette commande :

- la clé **<Couleur>** pour la couleur du « rectangle » des restes; défaut **<red>**
- la clé **<DecalH>** pour gérer le décalage H du « rectangle », qui peut être donné soit sous la forme **<Esp>** ou soit sous la forme **<espgauche/espdroite>**; défaut **<2pt>**
- la clé **<DecalV>** pour le décalage vertical du « rectangle »; défaut **<3pt>**
- la clé **<Noeud>** pour le préfixe du nœud du premier et du dernier reste (pour utilisation en TikZ); défaut **<EEE>**
- le booléen **<Rect>** pour afficher ou non le « rectangle » des restes; défaut **<true>**
- le booléen **<CouleurRes>** pour afficher ou non la conversion en couleur (identique au rectangle). défaut **<false>**

</> Code L^AT_EX

```
%conversion avec changement de couleur
\ConversionDepuisBaseDix[Couleur=DarkBlue]{45}{2}

%conversion sans le rectangle
Par divisions euclidiennes successives, \ConversionDepuisBaseDix[Rect=false]{54}{3}.

%conversion avec gestion du decalh pour le placement précis du rectangle
\ConversionDepuisBaseDix[Couleur=Goldenrod,DecalH=6pt/2pt]{1012}{16}

%conversion avec nœud personnalisé et réutilisation
\ConversionDepuisBaseDix[Couleur=ForestGreen,CouleurRes,Noeud=TEST]{100}{9}.
\begin{tikzpicture}
  \draw[overlay,ForestGreen,thick,->] (TEST2.south east) to[bend right] ++ (3cm,-1cm)
  \node[right] {test } ;
\end{tikzpicture}
```

➔ Sortie L^AT_EX

$$\left\{ \begin{array}{l} 45 = 2 \times 22 + 1 \\ 22 = 2 \times 11 + 0 \\ 11 = 2 \times 5 + 1 \\ 5 = 2 \times 2 + 1 \\ 2 = 2 \times 1 + 0 \\ 1 = 2 \times 0 + 1 \end{array} \right. \Rightarrow 45_{10} = 101101_2$$

$$\text{Par divisions euclidiennes successives, } \left\{ \begin{array}{l} 54 = 3 \times 18 + 0 \\ 18 = 3 \times 6 + 0 \\ 6 = 3 \times 2 + 0 \\ 2 = 3 \times 0 + 2 \end{array} \right. \Rightarrow 54_{10} = 2000_3.$$

$$\left\{ \begin{array}{l} 1012 = 16 \times 63 + 4 \\ 63 = 16 \times 3 + 15 \\ 3 = 16 \times 0 + 3 \end{array} \right. \Rightarrow 1012_{10} = 3F4_{16}$$

$$\text{On obtient donc } \left\{ \begin{array}{l} 100 = 9 \times 11 + 1 \\ 11 = 9 \times 1 + 2 \\ 1 = 9 \times 0 + 1 \end{array} \right. \Rightarrow 100_{10} = 121_9.$$

→ test

32 Algorithme d'Euclide pour le PGCD

32.1 Idée

💡 Idée(s)

L'idée est de proposer une « présentation » de l'algorithme d'Euclide pour le calcul du PGCD de deux entiers. Le package `xintgcd` permet déjà de le faire, il s'agit ici de travailler sur la *mise en forme*.

🔗 Code \LaTeX

```
\PresentationPGCD[options]{a}{b}
```

🔗 Code \LaTeX

```
\tikzstyle{every picture}+=[remember picture]
...
\PresentationPGCD{150}{27}
```

➡ Sortie \LaTeX

$$\left\{ \begin{array}{l} 150 = 27 \times 5 + 15 \\ 27 = 15 \times 1 + 12 \\ 15 = 12 \times 1 + \textcolor{red}{3} \\ 12 = 3 \times 4 + 0 \end{array} \right. \Rightarrow \text{PGCD}(150;27) = 3$$

⚠ Attention

La mise en valeur du dernier reste non nul est géré par du code TikZ, en mode `overlay`, donc il faut bien penser à déclarer dans le préambule : `\tikzstyle{every picture}+=[remember picture]`

32.2 Options et clés

🔗 Clés et options

Quelques options disponibles pour cette commande :

- la clé **<Couleur>** qui correspond à la couleur pour la mise en valeur; défaut **<red>**
- la clé **<DecalRect>** qui correspond à l'écartement du rectangle de mise en valeur; défaut **<2pt>**
- le booléen **<Rectangle>** qui gère l'affichage ou non du rectangle de mise en valeur; défaut **<true>**
- la clé **<Noeud>** qui gère le préfixe du nom du nœud TikZ du rectangle (pour exploitation ultérieure); défaut **<FFF>**
- le booléen **<CouleurResultat>** pour mettre ou non en couleur de PGCD; défaut **<false>**
- le booléen **<AfficheConclusion>** pour afficher ou non la conclusion; défaut **<true>**
- le booléen **<AfficheDelimitateurs>** pour afficher ou non les délimiteurs (accolade gauche et trait droit). défaut **<true>**

Le rectangle de mise en valeur est donc un nœud TikZ qui sera nommé, par défaut `FFF1`.

La présentation est dans un environnement `ensuremath` donc les `$...$` ne sont pas indispensables.

🔗 Code \LaTeX

```
\PresentationPGCD[CouleurResultat]{150}{27}
```

$$\left\{ \begin{array}{l} 150 = 27 \times 5 + 15 \\ 27 = 15 \times 1 + 12 \\ 15 = 12 \times 1 + \textcolor{red}{3} \\ 12 = 3 \times 4 + 0 \end{array} \right. \Rightarrow \text{PGCD}(150;27) = \textcolor{red}{3}$$

Code \LaTeX

```
\PresentationPGCD[CouleurResultat,Couleur=ForestGreen]{1250}{450}.

\PresentationPGCD[CouleurResultat,Couleur=DarkBlue]{13500}{2500}.

\PresentationPGCD[Rectangle=false]{420}{540}. \\\

D'après l'algorithme d'Euclide, on a  $\left[ \begin{array}{l} 123456789 \\ 9876 \end{array} \right]$ 
\begin{tikzpicture}
  \draw[overlay,LightSkyBlue,thick,<-] (FFF1.east) to[bend right] ++ (2cm,0.75cm) node[right]
  {dernier reste non nul} ;
\end{tikzpicture}
```

$$\left\{ \begin{array}{l} 1250 = 450 \times 2 + 350 \\ 450 = 350 \times 1 + 100 \\ 350 = 100 \times 3 + 50 \\ 100 = 50 \times 2 + 0 \end{array} \right\} \Rightarrow \text{PGCD}(1250; 450) = 50.$$

$$\left\{ \begin{array}{l} 13500 = 2500 \times 5 + 1000 \\ 2500 = 1000 \times 2 + 500 \\ 1000 = 500 \times 2 + 0 \end{array} \right\} \Rightarrow \text{PGCD}(13500; 2500) = 500.$$

$$\left\{ \begin{array}{l} 420 = 540 \times 0 + 420 \\ 540 = 420 \times 1 + 120 \\ 420 = 120 \times 3 + 60 \\ 120 = 60 \times 2 + 0 \end{array} \right\} \Rightarrow \text{PGCD}(420; 540) = 60.$$

D'après l'algorithme d'Euclide, on a

$$\begin{array}{rcl} 123456789 & = & 9876 \times 12500 + 6789 \\ 9876 & = & 6789 \times 1 + 3087 \\ 6789 & = & 3087 \times 2 + 615 \\ 3087 & = & 615 \times 5 + 12 \\ 615 & = & 12 \times 51 + 3 \\ 12 & = & 3 \times 4 + 0 \end{array}$$

dernier reste non nul

32.3 Compléments

Information(s)

La présentation des divisions euclidiennes est gérée par un tableau du type \LaTeX `array`, avec alignement vertical de symboles \LaTeX `=` et \LaTeX `+`.

Par défaut, les délimiteurs choisis sont donc l'accolade gauche et le trait droit, mais la clé booléenne $\langle \text{AfficheDelimiteurs}=\text{false} \rangle$ permet de choisir des délimiteurs différents.

Code \LaTeX

```
 $\left[ \begin{array}{l} 1234 = 5 \times 246 + 4 \\ 5 = 4 \times 1 + 1 \\ 4 = 1 \times 4 + 0 \end{array} \right]$ 
```

Écritures, simplifications

33 Simplification sous forme d'une fractions

33.1 Idée

💡 Idée(s)

L'idée est d'obtenir une commande pour *simplifier* un calcul sous forme de fraction irréductible.

🔗 Code \LaTeX

```
\ConversionFraction(*)[option de formatage]{calcul}
```

33.2 Commande et options

🔑 Clés et options

Quelques explications sur cette commande :

- `\ConversionFraction` la version *étoilée* force l'écriture du signe « - » sur le numérateur;
- le premier argument, *optionnel* et entre [...] permet de spécifier un formatage du résultat :
 - `<t>` pour l'affichage de la fraction en mode tfrac;
 - `<d>` pour l'affichage de la fraction en mode dfrac;
 - `<n>` pour l'affichage de la fraction en mode nicefrac;
 - `<dec>` pour l'affichage du résultat en mode décimal (sans arrondi!);
 - `<dec=k>` pour l'affichage du résultat en mode décimal arrondi à 10^{-k} ;
- le second argument, *obligatoire*, est quant à lui, le calcul en syntaxe xint.

À noter que la macro est dans un bloc `\ensuremath` donc les `\$...` ne sont pas nécessaires.

🔗 Code \LaTeX

```
\ConversionFraction{-10+1/3*(-5/16)}           %sortie par défaut
\ConversionFraction*{-10+1/3*(-5/16)}          %sortie fraction avec - sur numérateur
\ConversionFraction[d]{-10+1/3*(-5/16)}         %sortie en displaystyle
\ConversionFraction[n]{-10+1/3*(-5/16)}         %sortie en nicefrac
\ConversionFraction[dec=4]{-10+1/3*(-5/16)}     %sortie en décimal arrondi à 0,0001
\ConversionFraction{2+91/7}                     %entier formaté
\ConversionFraction{111/2145}
\ConversionFraction{111/3}
```

➡ Sortie \LaTeX

```
- 485
- 48
- 485
- 48
- 485
- 48
- 485/48
- 10,104 2
15
37
715
37
```

</> Code \LaTeX

```
 $\frac{111}{2145}=\text{\ConversionFraction{111/2145}}$ 

 $\smallskip$ 

 $\frac{3}{15}=\text{\ConversionFraction[]{3/15}}$ 

 $\smallskip$ 

 $\text{\tfrac{3}{15}}=\text{\ConversionFraction[t]{3/15}}$ 

 $\smallskip$ 

 $\text{\dfrac{3}{15}}=\text{\ConversionFraction[d]{3/15}}$ 

 $\smallskip$ 

 $\text{\dfrac{0,42}{0,015}}=\text{\ConversionFraction[d]{0.42/0.015}}$ 

 $\smallskip$ 

 $\text{\dfrac{0,41}{0,015}}=\text{\ConversionFraction[d]{0.41/0.015}}$ 

 $\smallskip$ 

 $\text{\dfrac{1}{7}}-\text{\dfrac{3}{8}}=\text{\ConversionFraction[d]{1/7-3/8}}$ 

 $\smallskip$ 

 $\text{\ConversionFraction[d]{1+1/2}}$ 

 $\smallskip$ 

 $\text{\ConversionFraction{0.1/0.7+30/80}}$ 
```

$$\begin{aligned}\frac{111}{2145} &= \frac{37}{715} \\ \frac{3}{15} &= \frac{1}{5} \\ \frac{3}{15} &= \frac{1}{5} \\ \frac{3}{15} &= \frac{1}{5} \\ \frac{0,42}{0,015} &= 28 \\ \frac{0,41}{0,015} &= \frac{82}{3} \\ \frac{1}{7} - \frac{3}{8} &= -\frac{13}{56} \\ \frac{3}{2} &= \frac{29}{56}\end{aligned}$$

Information(s)

A priori le package `xint` permet de s'en sortir pour des calculs « simples », je ne garantis pas que tout calcul ou toute division donne un résultat *satisfaisant* !

34 Ensembles

34.1 Idée

💡 Idée(s)

L'idée est d'obtenir une commande pour simplifier l'écriture d'un ensemble d'éléments, en laissant gérer les espaces.

Les délimiteurs de l'ensemble créé sont toujours $\{ \}$.

🔗 Code \LaTeX

```
\EcritureEnsemble[clés]{liste}
```

34.2 Commande et options

🔗 Clés et options

Peu d'options pour ces commandes :

- le premier argument, *optionnel*, permet de spécifier les **⟨Clés⟩** :
 - clé **⟨Sep⟩** qui correspond au délimiteur des éléments de l'ensemble; défaut **⟨;⟩**
 - clé **⟨Option⟩** qui est un code (par exemple `strut...`) inséré avant les éléments; défaut **⟨vide⟩**
 - un booléen **⟨Mathpunct⟩** qui permet de préciser si on utilise l'espacement mathématique `mathpunct`. défaut **⟨true⟩**
- le second, *obligatoire*, est la liste des éléments, séparés par `/`.

🔗 Code \LaTeX

```
$\EcritureEnsemble{a/b/c/d/e}$  
$\EcritureEnsemble[Mathpunct=false]{a/b/c/d/e}$  
$\EcritureEnsemble[Sep=,]{a/b/c/d/e}$  
$\EcritureEnsemble[Option={\strut}]{a/b/c/d/e}$ % \strut pour "augmenter"  
— un peu la hauteur des {}  
$\EcritureEnsemble{\frac{1}{1+\frac{1}{3}} / b / c / d / \frac{1}{2}}$
```

➡ Sortie \LaTeX

```
{a;b;c;d;e}  
{a;b;c;d;e}  
{a,b,c,d,e}  
{a;b;c;d;e}  
 $\left\{ \frac{1}{1+\frac{1}{3}}; b; c; d; \frac{1}{2} \right\}$ 
```

📌 Information(s)

Attention cependant au comportement de la commande avec des éléments en mode mathématique, ceux-ci peuvent générer une erreur si `displaystyle` n'est pas utilisé...

35 Écriture d'un trinôme, trinôme aléatoire

35.1 Idée

💡 Idée(s)

L'idée est de proposer une commande pour écrire, sous forme développée réduite, un trinôme en fonction de ses coefficients a , b et c (avec $a \neq 0$), avec la gestion des coefficients nuls ou égaux à ± 1 .

En combinant avec le package `\xfp` et fonction de générateur d'entiers aléatoires, on peut de ce fait proposer une commande pour générer aléatoirement des trinômes à coefficients entiers (pour des fiches d'exercices par exemple).

L'affichage des monômes est géré par le package `\siunitx` et le tout est dans un environnement `\ensuremath`.

</> Code \LaTeX

```
\EcritureTrinome[options]{a}{b}{c}
```

</> Code \LaTeX

```
\EcritureTrinome{1}{7}{0}\  
\EcritureTrinome{1.5}{7.3}{2.56}\  
\EcritureTrinome{-1}{0}{12}\  
\EcritureTrinome{-1}{-5}{0}
```

$x^2 + 7x$
 $1,5x^2 + 7,3x + 2,56$
 $-x^2 + 12$
 $-x^2 - 5x$

35.2 Clés et options

🔑 Clés et options

Quelques clés et options sont disponibles :

- la clé booléenne **<Alea>** pour autoriser les coefficients aléatoires; défaut **<false>**
- la clé booléenne **<Anegatif>** pour autoriser a à être négatif. défaut **<true>**

📌 Information(s)

La clé **<Alea>** va modifier la manière de saisir les coefficients, il suffira dans ce cas de préciser les bornes, sous la forme `\valmin, valmax`, de chacun des coefficients. C'est ensuite le package `\xfp` qui va se charger de générer les coefficients.

Code \LaTeX

Avec a entre 1 et 5 (et signe aléatoire) puis b entre -2 et 7 puis c entre -10 et 20
↪ :

```
 $f(x)=\text{\EcritureTrinome[Alea]\{1,5\}\{-5,5\}\{-10,10\}\$\\}$   
 $g(x)=\text{\EcritureTrinome[Alea]\{1,5\}\{-5,5\}\{-10,10\}\$\\}$   
 $h(x)=\text{\EcritureTrinome[Alea]\{1,5\}\{-5,5\}\{-10,10\}\$\\}$ 
```

Avec a entre 1 et 10 (forcément positif) puis b entre -2 et 2 puis c entre 0 et 4 :

```
 $\text{\EcritureTrinome[Alea,Anegatif=false]\{1,10\}\{-2,2\}\{0,4\}\\}$   
 $\text{\EcritureTrinome[Alea,Anegatif=false]\{1,10\}\{-2,2\}\{0,4\}\\}$   
 $\text{\EcritureTrinome[Alea,Anegatif=false]\{1,10\}\{-2,2\}\{0,4\}}$ 
```

Avec a entre 1 et 5 (et signe aléatoire) puis b entre -2 et 7 puis c entre -10 et 20 :

$$f(x) = x^2 - 4x + 2$$

$$g(x) = -3x^2 + x + 6$$

$$h(x) = -4x^2 + x - 4$$

Avec a entre 1 et 10 (forcément positif) puis b entre -2 et 2 puis c entre 0 et 4 :

$$3x^2 + x + 3$$

$$5x^2 + x + 2$$

$$9x^2 - x + 3$$

36 Simplification de racines

36.1 Idée

💡 Idée(s)

2.1.0 L'idée est de proposer une commande pour simplifier *automatiquement* une racine carrée, sous la forme $\frac{a\sqrt{b}}{c}$ avec $\frac{a}{c}$ irréductible et b le « plus petit possible ».

🔗 Code \LaTeX

```
\SimplificationRacine{expression ou calcul}
```

🔗 Code \LaTeX

```
\SimplificationRacine{48} \ \ \SimplificationRacine{100/34} \ \ \SimplificationRacine{99999} \ \ \SimplificationRacine{1500*0.31*(1-0.31)} \ \ \
```

$$\begin{array}{l} 4\sqrt{3} \\ \frac{5\sqrt{34}}{17} \\ 3\sqrt{11111} \\ \frac{3\sqrt{3565}}{10} \end{array}$$

📌 Information(s)

C'est – comme souvent – le package `xint` qui s'occupe en interne des calculs, et qui devrait donner des résultats satisfaisants dans la majorité des cas (attention aux *grands nombres*...)

La commande ne fait pas office de *calculatrice*, elle ne permet *que* de simplifier *une* racine carrée (donc transformer si besoin!).

36.2 Exemples

🔗 Code \LaTeX

```
%Simplification d'un module de complexe
$\left| 4+6\text{i}\right| = \sqrt{4^2+6^2} =
\quad \sqrt{\xinteval{4**2+6**2}}=\SimplificationRacine{4**2+6**2}$

%Simplification n°1
$\frac{1}{\sqrt{6}}=\left(\sqrt{\frac{1}{6}}\right)=\SimplificationRacine{1/6}$

%Simplification n°2
$\frac{42}{\sqrt{5}}=\left(\sqrt{\frac{42^2}{5}}\right)=\SimplificationRacine{(42*42)/5}$

%Écart-type d'une loi binomiale
$\sqrt{\num{150}\times\num{0.35}\times(1-\num{0.35})} =
\quad \displaystyle\SimplificationRacine{150*0.35*(1-0.35)}$
```

$$\begin{array}{l} |4 + 6i| = \sqrt{4^2 + 6^2} = \sqrt{52} = 2\sqrt{13} \\ \frac{1}{\sqrt{6}} = \left(\sqrt{\frac{1}{6}}\right) = \frac{\sqrt{6}}{6} \\ \frac{42}{\sqrt{5}} = \left(\sqrt{\frac{42^2}{5}}\right) = \frac{42\sqrt{5}}{5} \\ \sqrt{150 \times 0,35 \times (1 - 0,35)} = \frac{\sqrt{546}}{4} \end{array}$$

37 Mesure principale d'un angle

37.1 Idée

💡 Idée(s)

2.1.2 L'idée est de proposer (sur une suggestion de Marylyne Vignal) une commande pour déterminer la mesure principale d'un angle en radian.

🔗 Code \LaTeX

```
\MesurePrincipale[booléens]{angle} %dans un mode mathématique
```

📌 Information(s)

La commande est à insérer dans un environnement mathématique, via $\$...\$$ ou $\backslash[...]$.
L'angle peut être donné sous forme *explicite* avec la chaîne π .

37.2 Exemples

🔑 Clés et options

Pour cette commande :

- le booléen **<d>** permet de forcer l'affichage en \displaystyle ; défaut **<false>**
- le booléen **<Crochets>** permet d'afficher le *modulo* entre crochets plutôt qu'entre parenthèses; défaut **<false>**
- l'argument *obligatoire* est en écriture *en ligne*.

🔗 Code \LaTeX

```
\$ \MesurePrincipale[d]{54\pi/7}$  
\$ \MesurePrincipale[d]{-128\pi/15}$  
\$ \MesurePrincipale{3\pi/2}$  
\$ \MesurePrincipale[Crochets]{5\pi/2}$  
\$ \MesurePrincipale{-13\pi}$  
\$ \MesurePrincipale{28\pi}$  
\$ \MesurePrincipale[d]{14\pi/4}$  
\$ \MesurePrincipale[Crochets]{14\pi/7}$
```

➡ Sortie \LaTeX

$$\frac{54\pi}{7} = \frac{-2\pi}{7} (2\pi)$$
$$\frac{-128\pi}{15} = \frac{-8\pi}{15} (2\pi)$$
$$\frac{3\pi}{2} = \frac{-\pi}{2} (2\pi)$$
$$\frac{5\pi}{2} = \frac{\pi}{2} [2\pi]$$
$$-13\pi = \pi (2\pi)$$
$$28\pi = 0 (2\pi)$$
$$\frac{14\pi}{4} = \frac{-\pi}{2} (2\pi)$$
$$\frac{14\pi}{7} = 0 [2\pi]$$

Onzième partie

Jeux et récréations

38 SudoMaths, en TikZ

38.1 Introduction

💡 Idée(s)

L'idée est de *proposer* un environnement TikZ, une commande permettant de tracer des grilles de SudoMaths.

L'environnement créé, lié à TikZ, trace la grille de SudoMaths (avec les blocs démarqués), et peut la remplir avec une liste d'éléments.

🔗 Code \LaTeX

```
%grille classique non remplie, avec légendes H/V, {} nécessaires pour préciser que les cases  
  seront "vides"  
\SudoMaths{}
```

🔗 Sortie \LaTeX

	a	b	c	d	e	f	g	h	i
A									
B									
C									
D									
E									
F									
G									
H									
I									

🔗 Information(s)

La commande `\SudoMaths` crée donc la grille (remplie ou non), dans un environnement TikZ, c'est *c'est tout* !
On peut également utiliser l'*environnement* `\EnvSudoMaths` dans lequel on peut rajouter du code TikZ !

🔗 Code \LaTeX

```
%grille "toute seule"  
\SudoMaths[clés]{liste}  
  
%grille avec ajout de code  
\begin{EnvSudoMaths}[clés]{grille}  
  %commandes tikz  
\end{EnvSudoMaths}
```

38.2 Clés et options

Clés et options

Quelques **clés** sont disponibles pour cette commande :

- la clé **<Epaisseur>** pour gérer l'épaisseur des traits épais; défaut **<1.5pt>**
- la clé **<Epaisseur>** pour gérer l'épaisseur des traits fins; défaut **<0.5pt>**
- la clé **<Unite>** qui est l'unité graphique de la figure; défaut **<1cm>**
- la clé **<CouleurCase>** pour la couleur (éventuelles) des cases; défaut **<LightBlue !50>**
- la clé **<CouleurTexte>** pour gérer la couleur du label des cases; défaut **<blue>**
- la clé **<NbCol>** qui est le nombre de colonnes; défaut **<9>**
- la clé **<NbSubCol>** qui est le nombre de sous-colonnes; défaut **<3>**
- la clé **<NbLig>** qui est le nombre de lignes; défaut **<9>**
- la clé **<NbSubLig>** qui est le nombre de sous-colonnes; défaut **<3>**
- la clé **<Police>** qui formate le label des cases; défaut **<\normalfont\normalsize>**
- le booléen **<Legendes>** qui affiche ou non les légendes (H et V) des cases; défaut **<true>**
- la clé **<PoliceLeg>** qui formate le label des légendes; défaut **<\normalfont\normalsize>**
- la clé **<ListeLegV>** qui est la liste de la légende verticale; défaut **<ABCD...WXYZ>**
- la clé **<ListeLegH>** qui est la liste de la légende horizontale; défaut **<abcd...wxyz>**
- la clé **<DecalLegende>** qui est le décalage de la légende par rapport à la grille. défaut **<0.45>**

Information(s)

La liste éventuelle des éléments à rentrer dans le tableau est traitée par le package `listofitems`, et se présente sous la forme suivante : `/ / / ... / / $ / / / ... / / $... $ / / / ... / /`

Il peut donc être intéressant de *déclarer* la liste au préalable pour simplifier la saisie de la commande!

Information(s)

La **<CouleurCase>** est gérée – en interne – par le caractère `*` qui permet de préciser qu'on veut que la case soit coloriée.

Code L^AT_EX

```
%grille 6x6 avec blocs 2x3, avec coloration de cases (présentée sous forme de "cases")
\def\grilleSuMa{%
  (a)* / (b)* /      /      / (c)* / (d)* $%
  (e)* /      /      / (f)* / (g)* / (h)* $%
      /      / (i)* /      /      / (j)* $%
      /      / (k)* /      / (l)* / (m)* $%
  (n)* /      / (o)* /      /      / (p)* $%
      /      /      / (q)* /      /      $%
}

\SudoMaths [Unite=0.75cm,NbCol=6,NbSubCol=2,NbLig=6,NbSubLig=3,%
  Police=\small\bfseries\ttfamily,CouleurTexte=red,CouleurCase=yellow!50,Legendes=false]{\grilleSuMa}
```

Sortie L^AT_EX

(a)	(b)			(c)	(d)
(e)			(f)	(g)	(h)
		(i)			(j)
		(k)		(l)	(m)
(n)		(o)			(p)
			(q)		

Information(s)

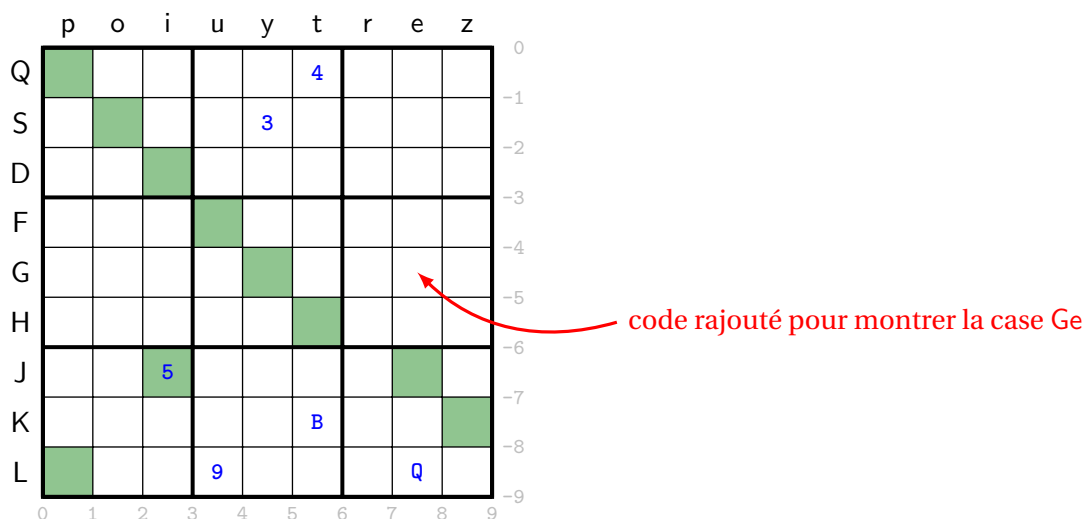
La grille, créée en TikZ, est portée par le rectangle de « coins » $(0; 0)$ et $(\text{nbc} ; -\text{nblg})$, de sorte que les labels des cases sont situés au nœuds de coordonnées $(x, 5; -y, 5)$.

Code L^AT_EX

```
%grille classique avec coloration de cases et commande tikz
%graduations rajoutées pour la lecture des coordonnées
\def\grilleSuMaB{%
  *////4///S%
  /*///3///S%
  //*/////S%
  ///*/////S%
  ////*/////S%
  /////*///S%
  //5*/////*/S%
  ////B///*S%
  *///9///Q/S%
}

\begin{EnvSudoMaths}[%
  Unite=0.66cm,Police=\footnotesize\bfseries\ttfamily,CouleurCase=ForestGreen!50,%
  ListeLegV=QSDFGHJKL,ListeLegH=poiuytrez]{\grilleSuMaB}
  \draw[red,very thick,<-,>=latex] (7.5,-4.5) to[bend right] ++ (4,-1) node[right] {code
    ↪ rajouté...} ;
\end{EnvSudoMaths}
```

Sortie L^AT_EX



Douzième partie

Historique

- v 2.5.8 : Ajout d'un style Alt pour les codes (pages 37 et 45)
 - : Modification de la syntaxe des commandes avec Pythontex et PseudoCode (pages 44 et 48)
 - v 2.5.7 : Ajout de clés pour les codes Piton + Console via Pyluatex (page 41)
 - v 2.5.6 : Ajout d'une clé **<Trigo>** pour l'axe (Ox) (page 20)
 - v 2.5.5 : Externalisation de la fenêtre XCas (dans la package FentreCas)
 - v 2.5.4 : Modification des calculs (via xint) en combinatoire (page 91)
 - v 2.5.3 : Modification du traitement des tests dans les arbres de probas (page 84)
 - v 2.5.2 : Correction d'un dysfonctionnement avec tcolorbox 6.0
 - v 2.5.1 : Ajout d'une version étoilée pour la conversion en fraction (page 99)
 - v 2.5.0 : Système de bibliothèques pour certains packages/commandes (page 7)
 - v 2.2.0 : Ajout d'une clé **<Notation>** pour les arrangements et combinaisons (page 91)
 - v 2.1.9 : Correction d'un bug (et ajout d'une version étoilée) pour les petits schémas « de signe » (page 31)
 - v 2.1.8 : Suppression des commandes de PixelArt, désormais dans le package PixelArtTikz
 - v 2.1.7 : Ajout d'une clé Math pour les sommets des figures de l'espace (pages 54 et 56)
 - v 2.1.6 : Correction d'un bug lié au chargement de hvlogos, remplacé par hologo
 - v 2.1.5 : Combinatoire avec arrangements et combinaisons (page 91)
 - v 2.1.4 : Résolution approchée d'équations $f(x) = k$ (page 14)
 - v 2.1.3 : Améliorations dans les présentations Piton (page 41)
 - v 2.1.2 : Ajout d'une commande pour la mesure principale d'un angle (page 105)
 - v 2.1.1 : Ajout d'une section pour des repères en TikZ (page 20)
 - v 2.1.0 : Calcul du seuil, en interne désormais (page 18)
 - : Commande pour simplifier une racine carrée (page 104)
 - : Option [pythontex] pour charger le nécessaire pour pythontex
 - v 2.0.9 : Nombres aléatoires, tirages aléatoires d'entiers (page 89)
 - v 2.0.8 : Ajout d'un environnement pour présenter du code \LaTeX (page 53)
 - v 2.0.7 : Ajout d'options pour stretch et fonte env python(s) (pas tous...)
 - v 2.0.6 : Changement de taille de la police des codes Python (page 37)
 - v 2.0.5 : Correction d'un bug avec les calculs de suites récurrentes (page 18)
 - v 2.0.4 : Ajout d'une commande pour une présentation de solution par TVI (page 16)
 - v 2.0.3 : Commandes pour des suites récurrentes *simples* (page 18)
 - v 2.0.2 : Option left-margin=auto pour le package piton (page 41)
 - v 2.0.1 : Chargement du package piton uniquement si compilation en Lua \LaTeX (page 41)
 - v 2.0.0 : Refonte du code source avec modification des commandes, et de la documentation
-
- v 1.3.7 : Commandes pour du code python via piton, en compilation Lua \LaTeX (page 41)
 - : Corrections et modifications mineures de la documentation
 - v 1.3.6 : Présentation de l'algorithme d'Euclide pour le PGCD (page 97)
 - : Affichage d'un trinôme par coefficients, aléatoires ou non (page 102)
 - v 1.3.5 : Correction d'un bug avec la loi géométrique (page 79)
 - v 1.3.4 : Ajout de petits schémas, en TikZ, de lois normales et exponentielles (page 87)
 - : Calculs de probas avec les lois géométriques et hypergéométriques (page 79)
 - v 1.3.3 : Ajout d'un environnement pour des arbres de probas classiques, en TikZ (page 84)
 - v 1.3.2 : Correction d'un bug sur les conversions bintohex avec lualatex (page 92)
 - v 1.3.1 : Ajout d'une option pour ne pas afficher les bordures des corrections de pixelart
 - v 1.3.0 : Commande pour présenter une conversion depuis la base 10 (page 95)
 - v 1.2.9 : Correction des commandes avec simplekv
 - v 1.2.7 : Ajout de commandes pour des calculs de probabilités (page 79)
 - v 1.2.6 : Ajout d'un environnement pour des SudoMaths (page 106)
 - v 1.2.5 : Ajout de commandes pour des boîtes à moustaches (page 76)
 - v 1.2.4 : Correction de quelques bugs mineurs, et mise à jour de la doc
 - v 1.2.3 : Commandes pour du code python "simple", sans compilation particulière (page 37)
 - v 1.2.2 : Commandes pour travailler sur des stats à 2 variables (page 67)
 - v 1.2.1 : Amélioration de la gestion du csv pour Pixelart
 - v 1.1.9 : Pixelart en TikZ
 - v 1.1.8 : Style "Mainlevée" basique pour TikZ (page 61)
 - v 1.1.7 : Conversions bin/hex/dec (basées sur xintbinhex) avec quelques détails (page 92)
 - v 1.1.6 : Ajout d'une commande pour déterminer les paramètres d'une régression linéaire par moindres carrés (page 62)
 - v 1.1.5 : Ajout de deux commandes pour, en TikZ, créer des petits schémas « de signe » (page 31)

- v 1.1.4 : Ajout d'une commande pour, en TikZ, créer facilement un cercle trigo avec *options* (page 58)
- v 1.1.3 : Ajout des commandes pour fractions, ensembles et récurrence (pages 99, 101 et 34)
- v 1.1.1 : Modification mineure de l'environnement calcul formel, avec prise de charge de la taille du texte
- v 1.1.0 : Ajout d'une commande pour créer des tétraèdres (avec nœuds) en TikZ (page 56)
- v 1.0.9 : Ajout d'une commande pour créer des pavés droits (avec nœuds) en TikZ (page 54)
- v 1.0.8 : Ajout d'une commande pour créer des cartouches de lien "comme capytale" (page 52)
- v 1.0.7 : Ajout d'une option build pour placer certains fichiers auxiliaires dans un répertoire externe
- v 1.0.6 : Ajout d'une option nominted pour ne pas charger (pas besoin de compiler avec shell-escape)
- v 1.0.5 : Ajout d'un environnement pour Python (minted) (page 45)
- v 1.0.4 : Ajout des environnements pour Terminal (win, osx, unix) (page 50)
- v 1.0.3 : Ajout des environnements pour PseudoCode (page 48)
- v 1.0.2 : Ajout des environnements pour Python (pythontex) (page 44)
- v 1.0 : Version initiale