

ProfLycee

Quelques *petites* commandes pour L^AT_EX (au lycée)

Cédric Pierquet

c pierquet - at - outlook . fr

Version 1.1.0 – 25 Avril 2022

Résumé :

Quelques commandes pour faciliter l'utilisation de L^AT_EX pour les enseignants de mathématiques en lycée.

Quelques commandes pour des courbes *lisses* avec gestion des extrema et des dérivées.

Quelques commandes pour simuler une fenêtre de logiciel de calcul formel, en TikZ.

Quelques environnements (tcbox) pour présenter du code python ou pseudocode.

Quelques environnements (tcbox) pour présenter des commandes dans un terminal (win ou mac ou linux).

Un cartouche (tcbox) pour présenter des codes de partage capytale.

Une commande pour tracer un pavé en droit, en TikZ, avec création des nœuds liés aux sommets.

L^AT_EX

pdfL^AT_EX

LuaL^AT_EX

TikZ

T_EXLive

MiK_TE_X

Table des matières

1	Introduction	3
1.1	« Philosophie » du package	3
1.2	Options du package	3
1.3	Le système de « clés/options »	4
1.4	Outils disponibles	4
1.5	Compilateur(s)	4
2	L'outil « splinetikz »	5
2.1	Courbe d'interpolation	5
2.2	Code, clés et options	5
2.3	Compléments sur les coefficients de « compensation »	5
2.4	Exemples	6
2.5	Avec une gestion plus fine des « coefficients »	7
2.6	Conclusion	7
3	L'outil « tangentetikz »	8
3.1	Définitions	8
3.2	Exemple et illustration	8
3.3	Exemple avec les deux outils, et « personnalisation »	9
4	L'outil « Calcul Formel »	10
4.1	Introduction	10
4.2	La commande « paramCF »	10
4.3	La commande « ligneCF »	10
4.4	Visualisation des paramètres	11
5	Code & Console Python	12
5.1	Introduction	12
5.2	Présentation de code Python via pythontex	12
5.3	Présentation de code Python via minted	14
5.4	Console d'exécution Python	15
6	Pseudo-Code	16
6.1	Introduction	16
6.2	Présentation de Pseudo-Code	16
6.3	Compléments	17
7	Terminal Windows/UNIX/OSX	18
7.1	Introduction	18
7.2	Commandes	18
8	Cartouche Capytale	20
8.1	Introduction	20
8.2	Commandes	20
9	Pavé droit « simple »	21
9.1	Introduction	21
9.2	Commandes	21
9.3	Influence des paramètres	22
10	Tétraèdre « simple »	23
10.1	Introduction	23
10.2	Commandes	23
10.3	Influence des paramètres	24
11	Historique	25

1 Introduction

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, dans le préambule, par `\usepackage{ProfLycee}`. Il charge quelques packages utiles, mais j'ai fait le choix de laisser l'utilisateur gérer ses autres packages, comment 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 les packages :

- `\xcolor` avec les options `[table,svgnames]`;
- `\tikz`, `\pgf`, `\xfp`;
- `\xparse`, `\xkeyval`, `\xstring`, `\simplekv`;
- `\listofitems`, `\xintexpr`;
- `\tabularray`, `\fontawesome5`, `\tcolorbox`.

💡 Idée(s)

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

🔗 Code \LaTeX

```
\documentclass{article}
\usepackage[french]{babel}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{ProfLycee}
...
```

1.2 Options du package

🔧 Information(s)

Par défaut, `\minted` est chargé et donc la compilation nécessite d'utiliser shell-escape. Cependant, si vous ne souhaitez pas utiliser les commandes nécessitant `\minted` vous pouvez charger le package `\ProfLycee` avec l'option `\nominted`.

🔗 Code \LaTeX

```
...
\usepackage[nominted]{ProfLycee}
...
```

🔧 Information(s)

En compilant (notamment avec les packages `\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.

Code L^AT_EX

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

Information(s)

Les options précédentes sont cumulables, et, pour info, elles conditionnent le chargement des packages avec les options :

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

1.3 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 (entre [...]) ou mandataires (entre {...}).

Information(s)

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

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

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

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

1.4 Outils disponibles

Idée(s)

Le `\package`, qui s'enrichira peut-être au fil du temps permet – pour le moment – de :

- tracer des splines cubiques avec gestion *assez fine* des tangentes;
- tracer des tangentes (ou portions) de tangentes sur la même base que pour les splines;
- simuler une fenêtre de logiciel formel (*à la manière de XCas*);
- mettre en forme du code python ou pseudocode;
- simuler une fenêtre de terminal (win/unix/osx).

Information(s)

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

1.5 Compilateur(s)

Information(s)

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

En ce qui concerne les codes python et/ou pseudocode, il faudra :

- compiler en chaîne pdflatex + pythontex + pdflatex pour les environnements avec `\pythontex`;
- compiler avec shell-escape (ou write18) pour les environnements avec `\minted`.

2 L'outil « splinetikz »

2.1 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 paramétrer les **coefficients** pour « affiner » les portions.

Pour déclarer les paramètres :

- liste des points de contrôle par : `liste=x1/y1/d1$x2/y2/d2$...`
 - il faut au-moins deux points;
 - 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$...`

2.2 Code, clés et options

Code L^AT_EX

```
\begin{tikzpicture}
...
\splinetikz[liste=...,coeffs=...,affpoints=...,couleur=...,epaisseur=...,%
            taillepoints=...,couleurpoints=...,style=...]
...
\end{tikzpicture}
```

Clés et options

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

- | | |
|---|------------------------------------|
| — la couleur de la courbe par la clé <code><couleur></code> ; | défaut <code><red></code> |
| — l'épaisseur de la courbe par la clé <code><epaisseur></code> ; | défaut <code><1.25pt></code> |
| — du style supplémentaire pour la courbe peut être rajouté, grâce à la clé <code><style=></code> ; | défaut <code><vide></code> |
| — les coefficients de <i>compensation</i> gérés par la clé <code><coeffs></code> ; | défaut <code><3></code> |
| — les points de contrôle ne sont pas affichés par défaut, mais clé booléenne <code><affpoints></code> permet de les afficher; | défaut <code><true></code> |
| — la taille des points de contrôle est géré par la clé <code><taillepoints></code> . | défaut <code><2pt></code> |

2.3 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 contrôles 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!

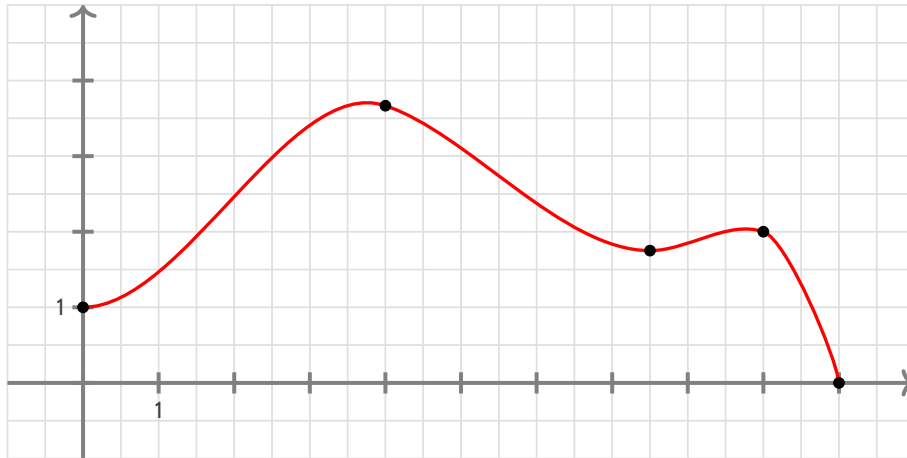
Pour des intervalles *étroits*, la *pente* peut paraître abrupte, et donc le(s) coefficient(s) peuvent être modifiés, de manière fine.

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

2.4 Exemples

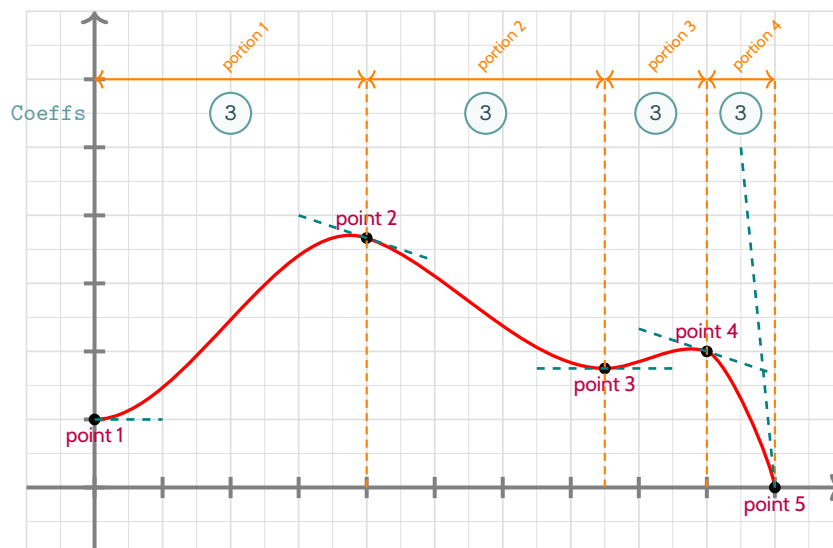
</> Code L^AT_EX

```
%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=0.3pt,lightgray!50] (\xmin,\ymin) grid (\xmax,\ymax);
\draw[xstep=\xgrilles,ystep=\ygrilles,line width=0.6pt,lightgray!50] (\xmin,\ymin) grid (\xmax,\ymax);
\draw[line width=1.5pt,->,gray] (\xmin,0)--(\xmax,0) ;
\draw[line width=1.5pt,->,gray] (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[list=\LISTE,affpoints=true,coeffs=3,couleur=red]
```



Information(s)

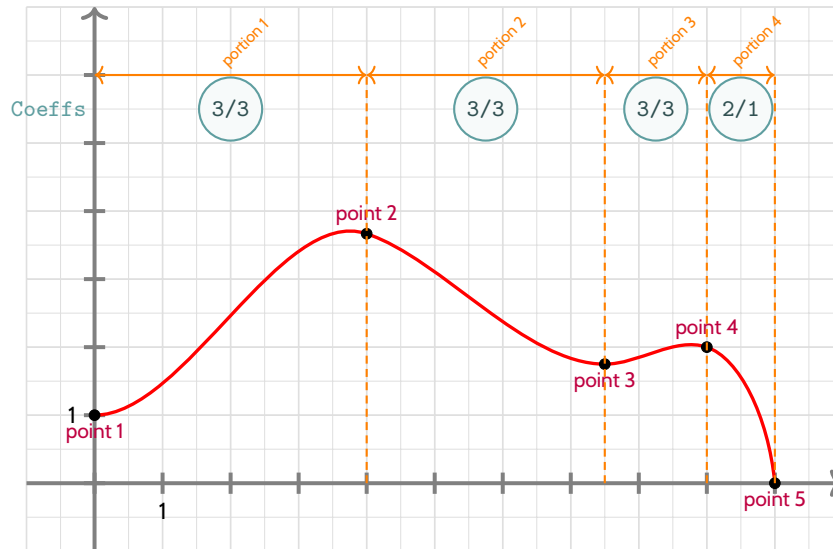
Avec des explications utiles à la compréhension :



2.5 Avec une gestion plus fine des « coefficients »

Information(s)

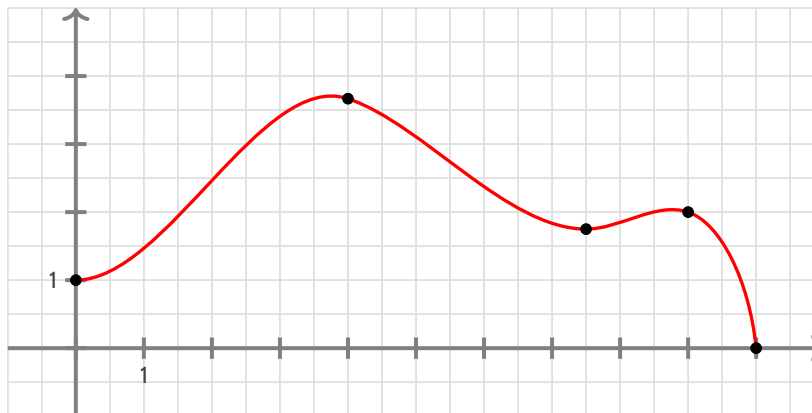
Dans la majorité des cas, le *coefficient* ③ permet d'obtenir une courbe (ou une portion) très satisfaisante!
 Dans certains cas, il se peut que la portion paraisse un peu trop « abrupte ».
 On peut dans ce cas *jouer* sur les coefficients de cette portion pour *arrondir* un peu tout cela (ie diminuer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



Code L^AT_EX

```
...
%splines
\def\LISTE{0/1/0$4/3.667/-0.333$7.5/1.75/0$9/2/-0.333$10/0/-10}
\splinetikz[liste=\LISTE,affpoints=true,coeffs=3$3$3$2/1]
...
```

Sortie L^AT_EX



2.6 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=\LISTE]`;
- d'ajuster les options et coefficients en fonction du rendu!

3 L'outil « tangentetikz »

3.1 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 L^AT_EX

```
\begin{tikzpicture}
...
\tangentetikz[liste=...,couleur=...,epaisseur=...,xl=...,xr=...,style=...,point=...]
...
\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 `epaisseur`, 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 .

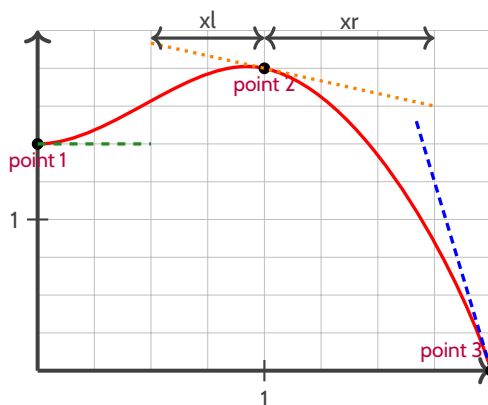
3.2 Exemple et illustration

</> Code L^AT_EX

```
\begin{tikzpicture}
...
\def\LISTE{0/1.5/0$1/2/-0.333$2/0/-5}
% spline
\splinetikz[liste=\LISTE,affpoints=true,coeffs=3$2,couleur=red]
% tangente
\tangentetikz[liste=\LISTE,xl=0,xr=0.5,couleur=ForestGreen,style=dashed]
\tangentetikz[liste=\LISTE,xl=0.5,xr=0.75,couleur=orange,style=dotted,point=2]
\tangentetikz[liste=\LISTE,xl=0.33,xr=0,couleur=blue,style=densely dashed,point=3]
...
\end{tikzpicture}
```

🖨 Sortie L^AT_EX

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



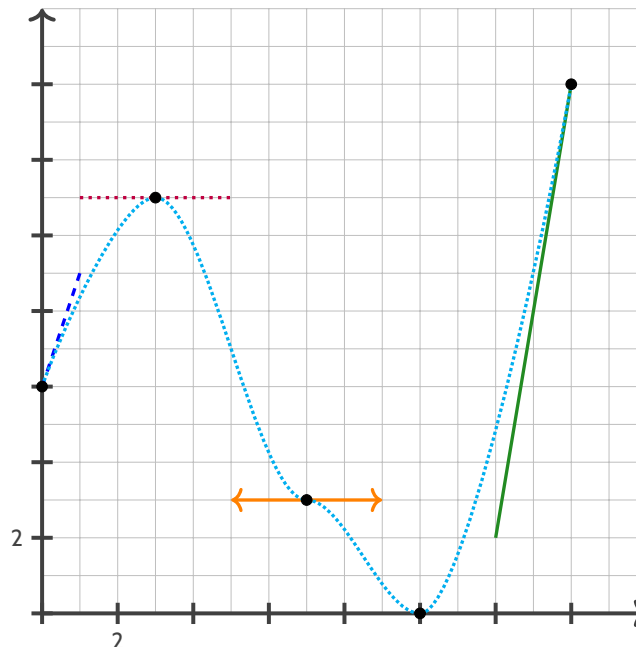
3.3 Exemple avec les deux outils, et « personnalisation »

</> Code L^AT_EX

```
\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] (\xmin,0)--(\xmax,0) ;
  \draw[line width=1.5pt,->,darkgray] (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 "stylisées"
  \tangentetikz[liste=\liste,xl=0,xr=1,couleur=blue,style=dashed]
  \tangentetikz[liste=\liste,xl=2,xr=2,couleur=purple,style=dotted,point=2]
  \tangentetikz[liste=\liste,xl=2,xr=2,couleur=orange,style=<->,point=3]
  \tangentetikz[liste=\liste,xl=2,xr=0,couleur=ForestGreen,point=5]
  %la courbe en elle-même
  \splinetikz[liste=\liste,affpoints=true,coeffs=3,couleur=cyan,style=densely dotted]
\end{tikzpicture}
```

⌚ Sortie L^AT_EX



4 L'outil « Calcul Formel »

4.1 Introduction

💡 Idée(s)

L'idée des commandes suivantes est de définir, dans un environnement `TikZ`, une présentation proche de celle d'un logiciel de calcul formel comme `XCas` ou `Geogebra`.

Les sujets d'examens, depuis quelques années, peuvent comporter des *captures d'écran* de logiciel de calcul formel, l'idée est ici de reproduire, de manière autonome, une telle présentation.

À la manière du package `tkz-tab`, l'environnement de référence est un environnement `TikZ`, dans lequel les lignes sont créées petit à petit, à l'aide de nœuds qui peuvent être réutilisés à loisir ultérieurement.

4.2 La commande « paramCF »

🧩 Information(s)

La première chose à définir est l'ensemble des paramètres *globaux* de la fenêtre de calcul formel, à l'aide de `<Clés>`.

🔗 Code L^AT_EX

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

🔑 Clés et options

Les `<Clés>` disponibles sont :

— <code><larg></code> : largeur de l'environnement;	défaut <code><16></code>
— <code><esplg></code> : espacement vertical entre les lignes;	défaut <code><2pt></code>
— <code><premcold></code> & <code><hpremcold></code> : largeur et hauteur de la case du <i>petit numéro</i> ;	défaut <code><0.3></code> & <code><0.4></code>
— <code><taille></code> : taille du texte;	défaut <code><\normalsize></code>
— <code><couleur></code> : couleur des traits de l'environnement;	défaut <code><darkgray></code>
— <code><titre></code> : booléen pour l'affichage d'un bandeau de titre;	défaut <code><false></code>
— <code><tailletitre></code> : taille du titre;	défaut <code><\normalsize></code>
— <code><poscmd></code> : position horizontale de la commande d'entrée;	défaut <code><gauche></code>
— <code><posres></code> : position horizontale de la commande de sortie;	défaut <code><centre></code>
— <code><couleurcmd></code> : couleur de la commande d'entrée;	défaut <code><ed></code>
— <code><couleurres></code> : couleur de la commande de sortie;	défaut <code><blue></code>
— <code><sep></code> : booléen pour l'affichage du trait de séparation E/S;	défaut <code><true></code>
— <code><menu></code> : booléen pour l'affichage du bouton MENU;	défaut <code><true></code>
— <code><labeltitre></code> : libellé du titre.	défaut <code><Résultats obtenus avec un logiciel de Calcul Formel></code>

4.3 La commande « ligneCF »

🧩 Information(s)

Une fois les paramètres déclarés, il faut créer les différentes lignes, grâce à la commande `\ligneCF`.

🔗 Code L^AT_EX

```
\begin{tikzpicture}[...]
  \paramCF[.....]
  \ligneCF[...]
  ...
\end{tikzpicture}
```

Clés et options

Les (quelques) **Clés** disponibles sont :

- **hc** : hauteur de la ligne de commande d'entrée;
- **hr** : hauteur de la ligne de commande de sortie;
- deux arguments, celui de la commande d'entrée et celui de la commande de sortie.

défaut **0.75**

défaut **0.75**

Chaque argument **COMMANDE** & **RÉSULTAT** peut être formaté (niveau police) de manière indépendante.

Code L^AT_EX

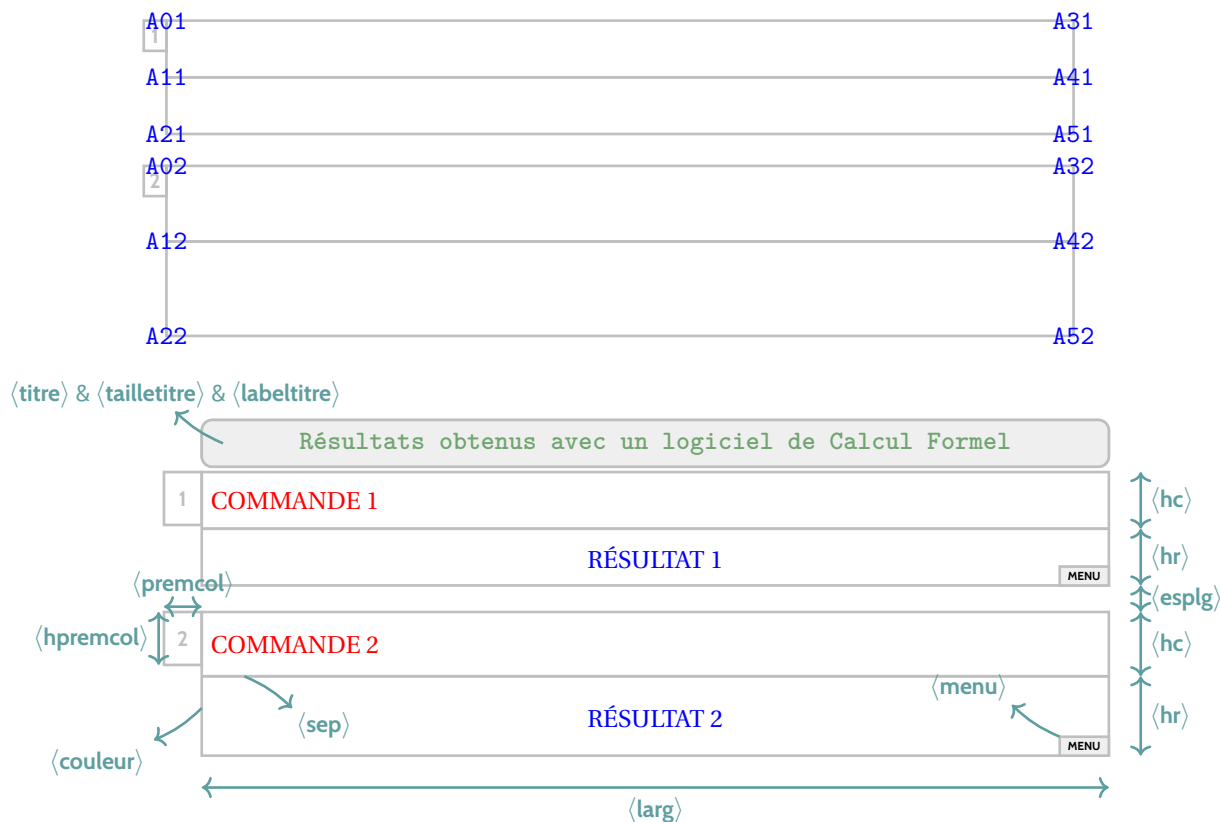
```
%code tikz
\paramCF[titre=true,couleurcmd=olive,couleurres=orange]
\ligneCF{COMMANDE 1}{RÉSULTAT 1}
\ligneCF[hc=0.75,hr=1]{\texttt{(x+1)\CFchap2}}{\mathhtt{x^2+2x+1}} %\CFchap := ^ en mathtt
```

Résultats obtenus avec un logiciel de Calcul Formel	
1	COMMANDE 1
	RÉSULTAT 1
2	(x+1) ²
	x ² + 2x + 1

4.4 Visualisation des paramètres

Information(s)

Pour *illustrer* un peu les **clés**, un petit schéma, avec les différents nœuds créés par les macros.



5 Code & Console Python

5.1 Introduction

💡 Idée(s)

Le package `pythontex` permet d'insérer et d'exécuter du code Python. On peut :

- présenter du code python;
- exécuter du code python dans un environnement type « console »;
- charger du code python, et éventuellement l'utiliser dans la console.

🔧 Information(s)

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* de gérer les options (par paramètres/clés...) des *tcbx* et des *fancyvrb*, le style est « fixé » tel quel, et seules la taille et la position de la *tcbx* 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* !

5.2 Présentation de code Python via `pythontex`

💡 Idée(s)

L'environnement `\envcodepythontex` (chargé par `ProfLycee`, avec l'option *autogobble*) permet de présenter du code python, dans une `colorbox` avec un style particulier.

🔗 Code L^AT_EX

```
\begin{envcodepythontex}[largeur=...,centre=...,lignes=...]  
...  
\end{envcodepythontex}
```

🔗 Clés et options

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

- | | | |
|--------------------|--|--------------------------------|
| — largeur : | largeur de la <i>tcbx</i> ; | défaut <code>\linewidth</code> |
| — centre : | booléen pour centrer ou non la <i>tcbx</i> ; | défaut <code>true</code> |
| — lignes : | booléen pour afficher ou non les numéros de ligne. | défaut <code>true</code> |

🔗 Code L^AT_EX

```
\begin{envcodepythontex}[largeur=12cm]  
#environnement Python(tex) centré avec numéros de ligne  
def f(x) :  
    return x**2  
\end{envcodepythontex}
```

🔗 Sortie L^AT_EX

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

Code Python

</> Code L^AT_EX

```
\begin{envcodepythontex}[largeur=12cm,lignes=false,centre=false]
#environnement Python(tex) non centré sans numéro de ligne
def f(x) :
    return x**2
\end{envcodepythontex}
```

⊖ Sortie L^AT_EX

```
#environnement Python(tex) non centré sans numéro de ligne
def f(x) :
    return x**2
```

Code Python

5.3 Présentation de code Python via minted

Information(s)

Pour celles et ceux qui ne sont pas à l'aise avec le package `pythontex` et notamment sa spécificité pour compiler, il existe le package `minted` qui permet de présenter du code, et notamment python (il nécessite quand même une compilation avec l'option `-shell-escape` ou `-write18`).

Idée(s)

L'environnement `\envcodepythonminted` permet de présenter du code python, dans une `colorbox` avec un style (*minted*) particulier.

Code L^AT_EX

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

Clés et options

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

- la version *étoilée* qui permet de pas afficher les numéros de lignes ;
- le premier argument optionnel concerne la **largeur** de la `tcbox` ; défaut **12cm**
- le second argument optionnel concerne les **options** de la `tcbox` en *langage tcolorbox*. défaut **vide**

Code L^AT_EX

```
\begin{envcodepythonminted}[12cm][center]
#environnement Python(minted) centré avec numéros, de largeur 12cm
def f(x) :
    return x**2
\end{envcodepythonminted}
```

Sortie L^AT_EX

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

Code Python

Code L^AT_EX

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

Sortie L^AT_EX

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

Code Python

5.4 Console d'exécution Python

💡 Idée(s)

`\pythonx` permet également de *simuler* (en exécutant également!) du code python dans une *console*. C'est l'environnement `\envconsolepythonx` qui permet de le faire.

</> Code L^AT_EX

```
\begin{envconsolepythonx}[largeur=...,centre=...,label=...]  
...  
\end{envconsolepythonx}
```

🔑 Clés et options

Les **Clés** disponibles sont :

- | | |
|---|--|
| — <code><largeur></code> : largeur de la <i>console</i> ; | défaut <code><\linewidth></code> |
| — <code><centre></code> : booléen pour centrer ou non la <i>console</i> ; | défaut <code><true></code> |
| — <code><label></code> : booléen pour afficher ou non le titre. | défaut <code><true></code> |

</> Code L^AT_EX

```
\begin{envconsolepythonx}[largeur=14cm,centre=false]  
#console Python(tex) non centrée avec label  
from math import sqrt  
1+1  
sqrt(12)  
\end{envconsolepythonx}
```

⊖ Sortie L^AT_EX

```
----- Début de la console python -----  
  
>>> #console Python(tex) non centrée avec label  
>>> from math import sqrt  
>>> 1+1  
2  
>>> sqrt(12)  
3.4641016151377544  
  
----- Fin de la console python -----
```

</> Code L^AT_EX

```
\begin{envconsolepythonx}[largeur=14cm,label=false]  
#console Python(tex) centrée sans label  
table = [[1,2],[3,4]]  
table[0][0]  
\end{envconsolepythonx}
```

⊖ Sortie L^AT_EX

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

6 Pseudo-Code

6.1 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>`, car l'environnement `tcblisting` est un peu différent...

6.2 Présentation de Pseudo-Code

Idee(s)

L'environnement `\envpseudocode` permet de présenter du (pseudo-code) dans une `tcolorbox`.

Information(s)

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 L^AT_EX

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

Clés et options

Plusieurs `<arguments>` (optionnels) sont disponibles :

- la version *étoilée* qui permet de pas afficher les numéros de lignes;
- le premier argument optionnel concerne la `<largeur>` de la `tcbbox`; défaut `<12cm>`
- le second argument optionnel concerne les `<options>` de la `tcbbox` en *langage tcolorbox*. défaut `<vide>`

Code L^AT_EX

```
\begin{envpseudocode} %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{envpseudocode}
```

Sortie L^AT_EX

```
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
```

Pseudo-Code

</> Code L^AT_EX

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

⌕ Sortie L^AT_EX

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

 Pseudo-Code

6.3 Compléments

🧩 Information(s)

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 libre de code pseudo-code.

💡 Idée(s)

Évidemment, le code source est récupérable et adaptable à volonté, en utilisant les possibilités du package `listings`. Celles et ceux qui sont déjà à l'aise avec les packages `listings` ou `minted` doivent déjà avoir leur environnement personnel prêt! Il s'agit ici de présenter une version « clé en main ».

7 Terminal Windows/UNIX/OSX

7.1 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 `termwin`, 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.

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

7.2 Commandes

</> Code L^AT_EX

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

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

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

🔑 Clés et options

Peu d'options pour ces commandes :

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

📄 Information(s)

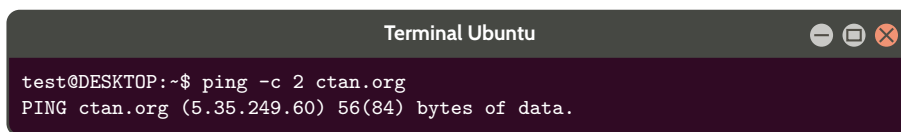
Le code n'est pas formaté, ni mis en coloration syntaxique.

De ce fait tout 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 L^AT_EX

```
\begin{PLtermunix}[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{PLtermunix}
```

🖥️ Sortie L^AT_EX



Code L^AT_EX

```
\begin{PLtermwin}[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{PLtermwin}

\begin{PLtermosx}[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{PLtermosx}
```

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.
```

8 Cartouche Capytale

8.1 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.

8.2 Commandes

🔗 Code L^AT_EX

```
\liencapytale(*)[options]{code}
```

🔗 Clés et options

Peu d'options pour ces commandes :

- la version *étoilée* qui permet de passer de la police `<sfamily>` à la police `<ttfamily>`, et donc dépendante des fontes du document;
- le deuxième, optionnel, permet de rajouter des caractères après le code (comme un espace); défaut `<vide>`
- le troisième, mandataire, est le code à afficher.

🔗 Code L^AT_EX

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

🔗 Sortie L^AT_EX

abcd-12345 🔗

abcd-12345 🔗

abcd-12345 🔗

abcd-12345 🔗

🔗 Information(s)

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

🔗 Code L^AT_EX

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

🔗 Sortie L^AT_EX

abcd-12345 🔗

9 Pavé droit « simple »

9.1 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.

9.2 Commandes

🔗 Code L^AT_EX

```
...  
\begin{tikzpicture}[<options>]  
  \paveCF[<options>]  
  ...  
\end{tikzpicture}
```

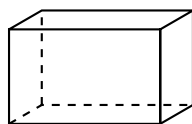
🔑 Clés et options

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> |
| — <epaisseur> : épaisseur des arêtes (en <i>langage simplifié</i> <i>TikZ</i>); | 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> |

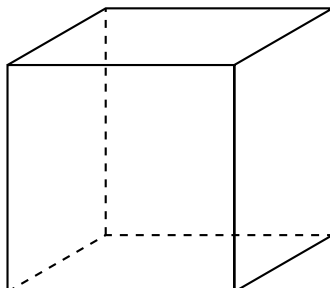
🔗 Code L^AT_EX

```
%code tikz  
\pavePL
```



🔗 Code L^AT_EX

```
%code tikz  
\pavePL[cube,largeur=3]
```



Information(s)

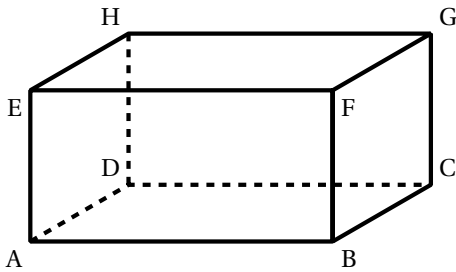
La ligne est de ce fait à insérer dans un environnement `TikZ`, avec les options au choix pour cet environnement. Le code crée les nœuds relatifs aux sommets, et les nomme comme les sommets, ce qui permet de les réutiliser pour éventuellement compléter la figure!

9.3 Influence des paramètres

Code \LaTeX

```
\begin{tikzpicture}[line join=bevel]
  \pavePL[aff,largeur=4,profondeur=3,hauteur=2,epaisseur={ultra thick}]
\end{tikzpicture}
```

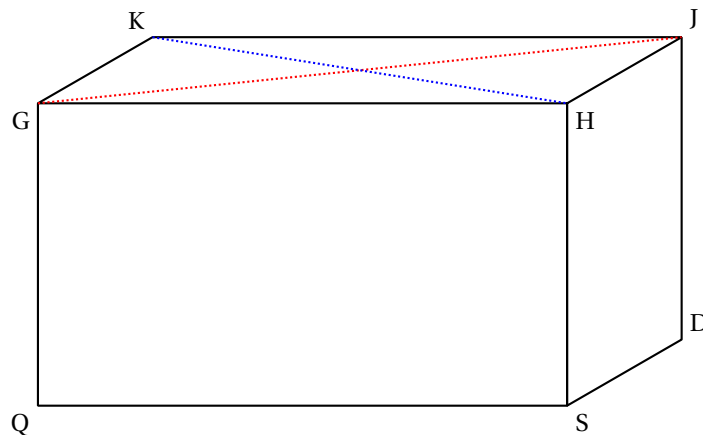
Sortie \LaTeX



Code \LaTeX

```
\begin{center}
  \begin{tikzpicture}[line join=bevel]
    \pavePL[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



10 Tétraèdre « simple »

10.1 Introduction

💡 Idée(s)

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

10.2 Commandes

</> Code \LaTeX

```
...  
\begin{tikzpicture}[<options>]  
  \tetraCF[<options>]  
  ...  
\end{tikzpicture}
```

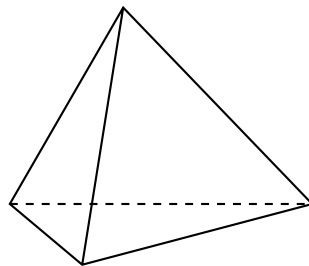
🔑 Clés et options

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

- | | |
|--|--|
| — <code><largeur></code> : <i>largeur</i> du tétraèdre; | défaut <code><4></code> |
| — <code><profondeur></code> : <i>profondeur</i> du tétraèdre; | défaut <code><1.25></code> |
| — <code><hauteur></code> : <i>hauteur</i> du tétraèdre; | défaut <code><3></code> |
| — <code><alpha></code> : angle <i>du sommet de devant</i> ; | défaut <code><40></code> |
| — <code><beta></code> : angle <i>du sommet du haut</i> ; | défaut <code><60></code> |
| — <code><sommets></code> : liste des sommets (avec délimiteur <code>\$!</code>); | défaut <code><A\$B\$C\$D></code> |
| — <code><epaisseur></code> : épaisseur des arêtes (en <i>langage simplifié TikZ</i>); | défaut <code><thick></code> |
| — <code><aff></code> : booléen pour afficher les noms des sommets; | défaut <code><false></code> |
| — <code><plein></code> : booléen pour ne pas afficher l'arête <i>invisible</i> . | défaut <code><false></code> |

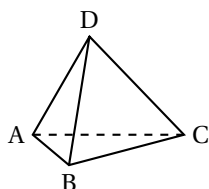
</> Code \LaTeX

```
%code tikz  
\tetraPL
```



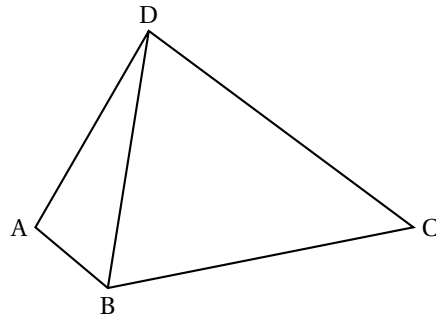
</> Code \LaTeX

```
%code tikz  
\tetraPL[aff,largeur=2,profondeur=0.625,hauteur=1.5]
```



Code L^AT_EX

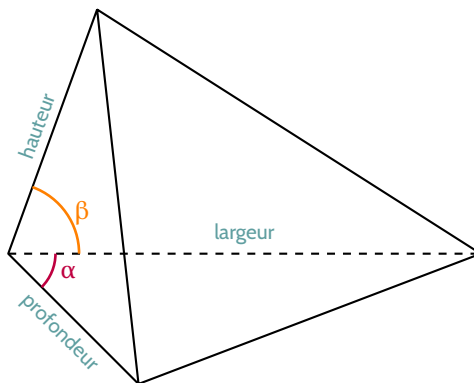
```
%code tikz
\tetraPL[plein,aff,largeur=5,beta=60]
```



10.3 Influence des paramètres

Information(s)

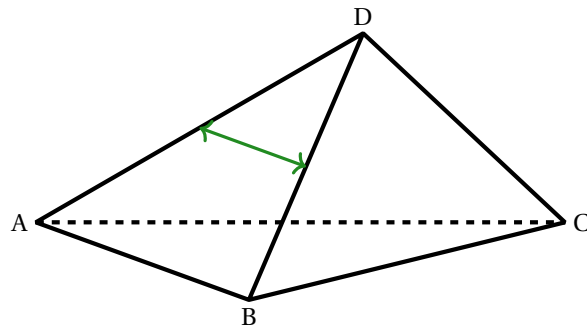
Pour *illustrer* un peu les <clés>, un petit schéma, avec les différents paramètres utiles.





Code L^AT_EX

```
\begin{center}
\begin{tikzpicture}[line join=bevel]
\tetraPL[aff,largeur=7,profondeur=3,hauteur=5,epaisseur={ultra thick},alpha=20,beta=30]
\draw[very thick,ForestGreen,<->] ($ (A)!0.5!(D)$)--($ (B)!0.5!(D)$) ;
\end{tikzpicture}
\end{center}
```

Sortie L^AT_EX



11 Historique

- v1.1.0: Ajout d'une commande tatraPL pour créer des tétraèdres (avec nœuds) en TikZ
- v1.0.9: Ajout d'une commande pavePL pour créer des pavés droits (avec nœuds) en TikZ
- v1.0.8: Ajout d'une commande liencapitale pour créer des cartouches de lien "comme capytale"
- v1.0.7: Ajout d'une option build pour placer certains fichiers auxiliaires dans un répertoire ./build
- v1.0.6: Ajout d'une option nominted pour ne pas charger  minted (pas besoin de compiler avec shell-escape)
- v1.0.5: Ajout d'un environnement pour Python (minted)
- v1.0.4: Ajout des environnements pour Terminal (win, osx, unix)
- v1.0.3: Ajout des environnements pour PseudoCode
- v1.0.2: Ajout des environnements pour Python (pythontex)
- v1.0.1: Modification mineure liée au chargement de  xcolor
- v1.0 : Version initiale