Le package `tnsmath` : des formules plus sémantiques

Code source disponible sur `https://github.com/typensee-latex/tnsmath.git`.

Version `2.2.0-beta` développée et testée sur Mac OS X.

Christophe BAL

2021-03-03

Table des matières

Chapitre 1

Généralités

I. Introduction

LaTeX est un excellent langage, pour ne pas dire le meilleur, pour rédiger des documents contenant des formules mathématiques. Malheureusement toute la puissance de LaTeX permet d'écrire des codes très peu sémantiques. Le modeste but du package `tnsmath` est de fournir quelques macros sémantiques[1] pour la rédaction de documents mathématiques élémentaires. Considérons le code LaTeX suivant.

```
Sachant que $f^\prime(x) = cos(x^2)$ sur $[a ; b]$ , nous avons :

$\displaystyle \int_a^b cos(x^2) dx = \left[ f(x) \right]_{x=a}^{x=b}$.
```

Avec `tnsmath`, vous pouvez écrire le code suivant.

```
Sachant que $\sder{f}(x) = cos(x^2)$ sur $\intervalC{a}{b}$, nous avons :

$\dintegrate*{cos(x^2)}{x}{a}{b} = \hook{f(x)}{x}{a}{b}$.
```

Même si certaines commandes sont plus longues à écrire que ce que permet LaTeX, il y a des avantages à utiliser des commandes sémantiques.

1. La mise en forme du document devient consistante.
2. Il est facile de changer une mise en forme sur l'ensemble d'un document ou localement via certaines options.
3. `tnsmath` résout certains problèmes « complexes » pour vous.

II. Beta-dépendance

`tnscom` qui est disponible sur `https://github.com/typensee-latex/tnscom.git` est un package utilisé en coulisse.

III. Comment lire cette documentation ?

Le choix a été fait de fournir des exemples comme documentation du package et de donner des fiches techniques des macros-commandes dans le chapitre dédié 12. Les exemples se présentent comme

1. En fait l'aspect sémantique est l'objectif commun de tous les packages de la suite `tns`.

ci-dessous et sont généralement très courts.

```
$\dintegrate*{cos(x^2)}{x}{a}{b}
 =
 \hook*{f(x)}{x}{a}{b}$
```

$$\int_a^b cos(x^2)\,\mathrm{d}x = \big[\, f(x) \,\big]_a^b$$

IV. A propos des macros

1. Règles de nommage

Les macros de même « type »

Les macros partageant une même fonctionnalité mathématique suivent les règles suivantes.

1. Un nom de base explicite est choisi comme par exemple `dotproduct` pour *« produit scalaire »* en anglais ou `set` pour *« ensemble »* en anglais.
2. Si besoin on spécialise du point de vue sémantique avec un préfixe et/ou un suffixe. Voici deux exemples.
   (a) Dans `\vdotproduct`, le préfixe `v` est pour `v`-ecteur car cette macro s'utilise avec des noms de vecteurs `u` et `v` et non directement des vecteurs `\vect{u}` et `\vect{v}`, autrement dit c'est `\vdotproduct` qui se charge d'appliquer `\vect` à `u` et `v`.
   (b) Dans `\setproba`, le suffixe `proba` est pour `proba`-bilité car cette macro sert à écrire des ensembles munis d'une probabilité [2].
3. Si l'on propose différentes mises en forme pour une même signification sémantique alors ceci se fera via des versions étoilées et/ou par le biais d'option(s) comme dans `\dotproduct[r]` pour obtenir des produits scalaires utilisant des chevrons ⟨ et ⟩ *(`r` est pour `r`-after soit* « chevron » *en anglais).*

Les formes « négatives » des macros

Les formes « négatives » des macros auront un nom préfixé par la lettre `n` en référence à `n`-ot. C'est l'usage dans le monde LaTeX comme par exemple pour `\neq`.

Les macros en mode `displaystyle`

Les macros évitant d'avoir à taper `\displaystyle` auront un nom préfixé par la lettre `d` comme par exemple pour `\dintegrate`.

Les macros « textuelles »

Certains macros produisent du contenu de type texte. Ces dernières seront toutes préfixées par `txt`. Par exemple `\txtopdef` est un texte utilisé pour décorer le signe égal, ou `\txtfuncdef` permet de saisir rapidement la définition explicite d'une fonction pour l'avoir directement dans du texte et non dans une formule.

Les macros standards redéfinies

Certaines macros comme `\frac` sont un peu revues par `tnsmath`. Dans ce cas, les versions standard restent accessibles en utilisant le préfixe `std` ce qui donne ici la macro `\stdfrac`.

2. Ce choix est assumé même si on obtient un nom faisant penser à *« régler … »* au lieu de *« ensemble de type … »*.

Casse utilisée pour les lettres

Les macros à usage graphique utiliseront une casse en bosses de chameau comme c'est le cas par exemple pour `\ptreeFrame` qui trace des cadres sur des arbres de probabilité, ou pour `\graphSign` qui ajoute des graphiques de signe près d'une ligne d'un tableau de signe [3].

Les macros non graphiques n'utiliseront que des minuscules. L'auteur de `tnsmath` préfère cette convention car elle est plus efficace à utiliser lors de la saisie et qu'elle impose aussi aux concepteurs de ne pas proposer des noms de macro à rallonge [4].

2. Les arguments : deux conventions à connaître

Avec un nombre fixé d'arguments

Dans ce cas, c'est la syntaxe LaTeX usuelle qui sera à utiliser comme dans `\dotprod{u}{v}`.

Avec un nombre variable d'arguments

Certaines macros offrent la possibilité de fournir un nombre variable d'arguments comme dans `\coord{x | y | z | t}` et `\coord{x | y}`. Ceci se fait en utilisant un seul argument, au sens de LaTeX, dont le contenu est formé de morceaux séparés par des traits verticaux `|`. Ainsi dans `\coord{x | y | z | t}` l'unique argument `x | y | z | t`, au sens de LaTeX, sera analysé par `tnsmath` comme étant formé des quatre arguments `x` , `y` , `z` et `t`.

Des arguments pas toujours affichés

Certaines macros pourront demander des arguments non utilisés pour la mise en forme. Pourquoi cela ? Tout simplement pour permettre des copier-coller lors de la rédaction : voir par exemple le calcul du déterminant de deux vecteurs du plan pour vérifier leur colinéarité *(ceci est présenté dans la section* **??***)*.

V. Couleurs

Certaines macros utilisent de la couleur. Les choix faits sont tels que l'impression en noir et blanc ne soit pas impactée.

VI. Packages utilisés

La roue ayant déjà été inventée, le package `tnsmath` utilise les packages suivants sans aucun scrupule.

- `amssymb`
- `centernot`
- `circledsteps`
- `commado`
- `dsfont`
- `esvect`
- `forest`
- `forloop`
- `ifmtarg`
- `longtable`
- `mathrsfs`
- `mathtools`
- `nicematrix`
- `pgfplots`
- `relsize`
- `simplekv`
- `stackengine`
- `tcolorbox`
- `tkz-tab`
- `trimspaces`
- `upgreek`
- `witharrows`
- `xstring`

3. La raison qui a poussé à ce choix est expliqué dans la présentation des outils de décoration des tableaux de signe : voir la présentation de la macro `\backLine`.

4. Cette pratique n'est pas inutile en interne pour autant par exemple pour nommer de façon compréhensible des macros privées.

Chapitre 2

Théorie générale des ensembles

I. Ensembles

1. Ensembles versus accolades

Exemple 1

`$\setgene{1 ; 3 ; 5}$ .`	$\{1\,;3\,;5\}$.

Exemple 2

Dans l'exemple suivant on utilise l'option `sb` pour `s`-mall `b`-races soit *« petites accolades »* en anglais.

`$\setgene {\dfrac{1}{3} ; \dfrac{5}{7} ; \dfrac{9}{11}}$`	$\left\{\dfrac{1}{3}\,;\dfrac{5}{7}\,;\dfrac{9}{11}\right\}$
`$\setgene*{\dfrac{1}{3} ; \dfrac{5}{7} ; \dfrac{9}{11}}$`	$\{\dfrac{1}{3}\,;\dfrac{5}{7}\,;\dfrac{9}{11}\}$

2. Ensembles pour la géométrie

Exemple 1

`$\setgeo{C}$ ,`
`$\setgeo{D}$ ou`
`$\setgeo{d}$`
	$\mathscr{C}$, $\mathscr{D}$ ou $\mathscr{d}$

Remarque. Pour le moment, il n'est pas possible de taper `$\setgeo{ABC}$` avec plusieurs lettres.

Exemple 2 – Avec des indices

`$\setgeo*{C}{1}$ ou` `$\setgeo*{C}{2}$`	$\mathscr{C}_1$ ou $\mathscr{C}_2$

3. Ensembles probabilistes

Exemple 1

`$\setproba{E}$ ou` `$\setproba{G}$`	$\mathcal{E}$ ou $\mathcal{G}$

Remarque. Pour le moment, il n'est pas possible de taper `$\setproba{ABC}$` avec plusieurs lettres.

Exemple 2 – Avec des indices

`$\setproba*{E}{1}$ ou` `$\setproba*{E}{2}$`	$\mathcal{E}_1$ ou $\mathcal{E}_2$

4. Ensembles pour l'algèbre générale

Exemple 1

`$\setalge{A}$ ,`
`$\setalge{K}$ ,`
`$\setalge{h}$ ou`
`$\setalge{k}$`
	$\mathbb{A}$, $\mathbb{K}$, $\mathbb{h}$ ou $\mathbb{k}$

Remarque. Pour le moment, il n'est pas possible de taper `$\setalge{ABC}$` avec plusieurs lettres.

Exemple 2 – Avec des indices

`$\setalge*{k}{1}$ ou $\setalge*{k}{2}$`	$\mathbb{k}_1$ ou $\mathbb{k}_2$

5. Ensembles classiques en mathématiques et en informatique théorique

La liste complète

Dans l'exemple suivant, $\mathbb{P}$ désigne l'ensemble des nombres premiers, $\mathbb{H}$ celui des quaternions, $\mathbb{O}$ celui des octonions et $\mathbb{F}$ un ensemble de nombres flottants *(notation à préciser suivant le contexte)*.

`$\nullset$`	$\varnothing$
`$\NN$ , $\ZZ$ , $\PP$`	$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{P}$
`$\DD$ , $\QQ$ , $\RR$ , $\CC$`	$\mathbb{D}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$
`$\HH$ , $\OO$`	$\mathbb{H}$, $\mathbb{O}$
`$\FF$`	$\mathbb{F}$

Ensembles classiques suffixés

L'ensemble $\mathbb{R}$ nous permet de voir tous les cas possibles.

`$\RRn$` , `$\RRp$` , `$\RRs$`
 	
`$\RRsn$` , `$\RRsp$`
	$\mathbb{R}_-$, $\mathbb{R}_+$, $\mathbb{R}^*$
	$\mathbb{R}_-^*$, $\mathbb{R}_+^*$

Nous avons utilisé les suffixes `n` pour `n`-égatif, `p` pour `p`-ositif et `s` pour `s`-tar soit *« étoile »* en anglais. Il y a aussi les suffixes composites `sn` et `sp`.

Notez qu'il est interdit d'utiliser `$\CCn$` pour $\mathbb{C}_-$ car l'ensemble $\mathbb{C}$ ne possède pas de structure ordonnée standard. Jetez un oeil à la section suivante pour apprendre à taper $\mathbb{C}_-$ si vous en avez besoin. L'interdiction est ici purement sémantique !

Remarque. La table 2.1 de la présente page montre les associations autorisées entre ensembles classiques et suffixes.

TABLE 2.1 – Suffixes

	n	p	s	sn	sp
\NN			×		
\PP					
\ZZ \DD \QQ \RR	×	×	×	×	×
\CC \HH \OO			×		
\FF	×	×	×	×	×

Des suffixes à la carte

Dans l'exemple suivant, il faut savoir que le 2e argument ne peut prendre que les valeurs `n`, `p`, `s`, `sn` ou `sp`.

`$\setspecial{\CC}{n}$` , `$\setspecial{\HH}{sp}$` ou `$\setspecial*{\setproba{P}}{n}$`	$\mathbb{C}_-$, $\mathbb{H}_+^*$ ou $\mathcal{P}_{\leq 0}$

II. Intervalles

1. Intervalles réels – Notation française (?)

Exemple 1

Dans cet exemple, la syntaxe fait référence à `O`-pened et `C`-losed pour *« ouvert et fermé »* en anglais. Nous verrons que `CC` et `OO` sont contractés en `C` et `O`. Notez au passage que la macro utilisée résout un problème d'espacement vis à vis du signe = .

`$I = ]a ; b] = \intervalOC{a}{b}$`	$I =]a\,;b] =]a\,;b]$

Exemple 2

Les crochets s'étendent verticalement automatiquement. Pour empêcher cela, il suffit d'utiliser la version étoilée de la macro. Dans ce cas, les crochets restent tout de même un peu plus grands que des crochets utilisés directement. Voici un exemple.

```
$\displaystyle
 \intervalC{ \frac{1}{2} }{ 1^{2^{3}} }
 =
 [ \frac{1}{2} ; 1^{2^{3}} ]
 =
 \intervalC*{ \frac{1}{2} }{ 1^{2^{3}} }$
```

$$\displaystyle \left[\frac{1}{2}\,;1^{2^3}\right] = [\frac{1}{2}\,;1^{2^3}] = [\frac{1}{2}\,;1^{2^3}\,]$$

2. Intervalles réels – Notation américaine

Dans l'exemple suivant la syntaxe fait référence à P-arenthèse. Cette notation est utilisée aux États Unis.

```
$\intervalPC{a}{b} = \intervalOC{a}{b}$
et
$\intervalP{a}{b} = \intervalO{a}{b}$.
```

$(a\,;b] =]a\,;b]$ et $(a\,;b) =]a\,;b[$.

3. Intervalles discrets d'entiers

Dans l'exemple suivant la syntaxe fait référence à $\mathbb{Z}$ l'ensemble des entiers relatifs (les crochets doubles ⟦ et ⟧ sont accessibles via `\Zlbracket` et `\Zrbracket` directement [1]).

```
$\ZintervalC{-1}{4} =
 \{ -1 ; 0 ; 1 ; 2 ; 3 ; 4 \}$

$\ZintervalC{-1}{4} =
 \ZintervalO{-2}{5}$.
```

$\llbracket -1\,;4\rrbracket = \{-1\,;0\,;1\,;2\,;3\,;4\}$
$\llbracket -1\,;4\rrbracket = \rrbracket -2\,;5\llbracket$.

III. Unions et intersections en mode ligne

LaTeX permet d'afficher sans souci $\cup_{k=1}^{n}$ mais ne propose pas $\bigcup_{k=1}^{n}$. Les macros `\dcap`, `\dcup` et `\dsqcup` donnent accès à ce type de fonctionnalité pour $\cap$, $\cup$ et $\sqcup$ respectivement. Voici des exemples d'utilisation.

Exemple 1 – Les symboles « seuls »

```
$A \dcap B = C \cap D$

$A \dcup B = C \cup D$

$A \dsqcup B = C \sqcup D$
```

$A \dcap B = C \cap D$
$A \dcup B = C \cup D$
$A \dsqcup B = C \sqcup D$

1. Ces deux symboles « proviennent » du code source du package `stmaryrd`, lequel package n'est pas chargé.

Exemple 2 – Des intersections indicées

Ci-dessous est utilisée la macro `\bigcap` proposée par le package `amssymb`.

```
$\cap_{k=1}^{n}    A_k$ ,
$\dcap_{k=1}^{n}   B_k$ ,
$\bigcap_{k=1}^{n} C_k$ ,
$\displaystyle%
 \bigcap_{k=1}^{n} D_k$
```

$\cap_{k=1}^{n} A_k$, $\dcap_{k=1}^{n} B_k$, $\bigcap_{k=1}^{n} C_k$, $\displaystyle\bigcap_{k=1}^{n} D_k$

Exemple 3 – Des unions indicées

Ci-dessous est utilisée la macro `\bigcup` proposée par le package `amssymb`.

```
$\cup_{k=1}^{n}    A_k$ ,
$\dcup_{k=1}^{n}   B_k$ ,
$\bigcup_{k=1}^{n} C_k$ ,
$\displaystyle%
 \bigcup_{k=1}^{n} D_k$
```

$\cup_{k=1}^{n} A_k$, $\dcup_{k=1}^{n} B_k$, $\bigcup_{k=1}^{n} C_k$, $\displaystyle\bigcup_{k=1}^{n} D_k$

Exemple 4 – Des unions disjointes indicées

Ci-dessous sont utilisées les macros `\sqcup` et `\bigsqcup` proposée par le package `amssymb`.

```
$\sqcup_{k=1}^{n}    A_k$ ,
$\dsqcup_{k=1}^{n}   B_k$ ,
$\bigsqcup_{k=1}^{n} C_k$ ,
$\displaystyle%
 \bigsqcup_{k=1}^{n} D_k$
```

$\sqcup_{k=1}^{n} A_k$, $\dsqcup_{k=1}^{n} B_k$, $\bigsqcup_{k=1}^{n} C_k$, $\displaystyle\bigsqcup_{k=1}^{n} D_k$

IV. Des applications très spéciales

1. Fonction identité

```
$\id_A$ vérifie par définition
$\forall x \in A$, $\id_A(x) = x$.
```

Id_A vérifie par définition $\forall x \in A$, $\mathrm{Id}_A(x) = x$.

2. Fonction caractéristique

Exemple 1 – Version très utilisée

```
$\caract_A$ vérifie par définition
$\forall x \in A$, $\caract_A(x) = 1$ et
$\forall x \notin A$, $\caract_A(x) = 0$.
```

χ_A vérifie par définition $\forall x \in A$, $\chi_A(x) = 1$ et $\forall x \notin A$, $\chi_A(x) = 0$.

Exemple 2 – Version alternative

```
$\caractone_A = \caract_A$
```

$\mathbb{1}_A = \chi_A$

V. Applications

1. Cardinal, image et compagnie

Exemple 1 – Cardinal

```
$\card* E = \card E$
```

$\card* E = \card E$

Exemple 2 – Image et compagnie

Ci-dessous se trouve la macro `\ker` proposée par `amsmath` qui est importé par `tnssets`.

```
$\ker f$ , $\dom f$ ,
$\im f$ ou $\codom f$
```

$\ker f$, $\dom f$,
$\im f$ ou $\codom f$

2. Application totale, partielle, injective, surjective et/ou bijective

Voici des symboles qui, bien que très techniques, facilitent la rédaction de documents à propos des applications totales ou partielles [2] *(on parle aussi d'applications, sans qualificatif, et de fonctions)*.

Exemple 1 – Applications totales

```
$f: A \to B$ est une application totale, c'est à dire définie sur $A$ tout entier.

$i: C \onetoone D$ est une application totale injective.

$s: E \onto F$ est une application totale surjective.

$b: G \biject H$ est une application totale bijective.
```

$f: A \to B$ est une application totale, c'est à dire définie sur A tout entier.

$i: C \onetoone D$ est une application totale injective.

$s: E \onto F$ est une application totale surjective.

$b: G \biject H$ est une application totale bijective.

Exemple 2 – Applications partielles

```
$f: A \pto B$ est une application partielle, c'est à dire définie sur
un sous-ensemble de $A$.

$i: C \ponetoone D$ est une application partielle injective.

$s: E \ponto F$ est une application partielle surjective.

$b: G \pbiject H$ est une application partielle bijective.
```

$f: A \pto B$ est une application partielle, c'est à dire définie sur
un sous-ensemble de A.

$i: C \ponetoone D$ est une application partielle injective.

$s: E \ponto F$ est une application partielle surjective.

$b: G \pbiject H$ est une application partielle bijective.

2. $a : E \to F$ est une application totale si $\forall x \in E$, $\exists ! y \in F$ tel que $y = a(x)$. Plus généralement, $f : E \to F$ est une application partielle si $\forall x \in E$, $\exists_{\leq 1} y \in F$ tel que $y = f(x)$, autrement dit soit $f(x)$ existe dans F, soit f n'est pas définie en x.

$f : A \nrightarrow B$ est une application partielle, c'est à dire définie sur un sous-ensemble de A.
$i : C \rightarrowtail D$ est une application partielle injective.
$s : E \twoheadrightarrow F$ est une application partielle surjective.
$b : G \rightarrowtail\!\!\!\!\twoheadrightarrow H$ est une application partielle bijective.

3. Définition explicite d'une fonction

Exemple 1 – Écriture par défaut

```
$\funcdef{f}{x}{x^2}%
            {I}{J}$
```

$$f : \left| \begin{array}{l} I \to J \\ x \mapsto x^2 \end{array} \right.$$

Remarque. Même si cela est peu utile, vous pouvez utiliser la mise en forme dans du texte pour obtenir $f : \left| \begin{array}{l} I \to J \\ x \mapsto x^2 \end{array} \right.$ mais c'est un peu affreux.

Exemple 2 – Écriture alternative

On peut cacher le trait vertical via l'option `s` pour `s`-hort soit *« court »* en anglais.

```
$\funcdef[s]{f}{x}{x^2}%
              {I}{J}$
```

$$f : I \to J \\ x \mapsto x^2$$

Exemple 3 – Écriture en ligne

Pour avoir tout sur une ligne, ce qui est l'idéal pour une insertion dans du texte, il suffit d'utiliser l'option `h` pour `h`-orizontal.

```
Soit $\funcdef[h]{f}{x}{x^2}%
                    {I}{J}$ ...
```

Soit $f : x \in I \mapsto x^2 \in J$ ...

Exemple 4 – Écriture en ligne incomplète

En mode horizontal, les ensembles peuvent être de valeur vide pour ne pas les indiquer.

```
$\funcdef[h]{f}{x}{x^2}{}{J}$

$\funcdef[h]{f}{x}{x^2}{I}{}$

$\funcdef[h]{f}{x}{x^2}{}{}$
```

$f : x \mapsto x^2 \in J$
$f : x \in I \mapsto x^2$
$f : x \mapsto x^2$

Exemple 5 – Écriture textuelle

On peut enfin obtenir une version *« textuelle »* via la macro `\txtfuncdef` où `txt` est pour `texte`. Cette macro ne s'utilise qu'en mode texte[3] et elle accepte l'omission de l'un ou des deux ensembles.

3. Logic ! Isn't it ?

```
\txtfuncdef{f}{x}{x^2}{I}{J}

\txtfuncdef{f}{x}{x^2}{}{J}

\txtfuncdef{f}{x}{x^2}{I}{}

\txtfuncdef{f}{x}{x^2}{}{}
```

$f(x) = x^2$ pour $x \in I$ *(on sait que* $f(x) \in J$ *est toujours vérifié)*
$f(x) = x^2$ *(on sait que* $f(x) \in J$ *est toujours vérifié)*
$f(x) = x^2$ pour $x \in I$
$f(x) = x^2$

4. Composition

Exemple 1 – Opérateur

La macro `\compo` est juste une version un peu plus petite de `\circ`.

```
$f \compo g$ et non
$f \circ g$
```

$f \compo g$ et non $f \circ g$

Exemple 2 – Compositions successives

La macro `\multicompo` sert à indiquer la composition d'une application plusieurs fois de suite par elle-même[4]. Voici toutes les mises en forme disponibles où l'option `exp` nécessite que le nombre d'applications composées soit un naturel non nul connu. Vous noterez que l'écriture par défaut, qui n'est pas standard, n'est pas $f^{(p)}$ car cette notation est traditionnellement utilisée pour indiquer la dérivée p$^{\text{e}}$ d'une application.

```
 $\multicompo     {g}{5}
= \multicompo[exp]{g}{5}$

 $\multicompo     {f}{p}
= \multicompo[dot]{f}{p}$
```

$g^{\langle 5\rangle} = g \compo g \compo g \compo g \compo g$
$f^{\langle p\rangle} = f \compo \cdots \compo f$

Remarque. La convention retenue est analogue à ce que l'on fait avec les puissances de nombres réels. En particulier, $f^{\langle 1\rangle} = f$ et $f^{\langle 0\rangle} = \mathrm{Id}_{\mathrm{dom}\, f}$.

4. Une telle fonction f doit vérifier $\mathrm{im}\, f \subseteq \mathrm{dom}\, f$.

Chapitre 3

Méthodes formelles en logique

I. Espace après la négation logique

`\neg` a été redéfinie pour ajouter un peu d'espace après le symbole. Le comportement par défaut se retrouve en utilisant la macro `\stdneg`. Voici un exemple.

```
$\neg A = \stdneg A$

$\neg\neg A = \stdneg \stdneg A$
```

$\neg A = \neg A$
$\neg\neg A = \neg\neg A$

II. Personnaliser les opérateurs de comparaison

1. Décorer avec du texte

D'un point de vue pédagogique il peut être intéressant de disposer de différentes façons d'écrire une égalité, une non égalité ou une inégalité en lui ajoutant un texte descriptif. Bien entendu on tord les règles de typographie avec ce type de pratique mais c'est pour le bien de la communauté éducative. Pour cela on utilisera l'une des macros suivantes ou leur version négative obtenue en préfixant le nom d'un `n`.

1. `\eq` donne $=$ tandis que `\neq` donne $\neq$.
2. `\less` et `\gtr` donnent $<$ et $>$ tandis que`\nless` et `\ngtr` donnent $\not<$ et $\not>$.
3. `\leq` et `\geq` donnent $\leq$ et $\geq$ tandis que`\nleq` et `\ngeq` donnent $\not\leq$ et $\not\geq$.

Voici un exemple où l'utilisation d'arguments optionnels permet de décorer des opérateurs.

```
   $f(x) \eq[def]   x^3 + 1$
si $x    \leq[cond] 2$

    $f(x) \neq[cons] x^3 + 1$
car $x    \nleq[hyp] 2$
```

$f(x) \overset{\text{def}}{=} x^3 + 1 \text{ si } x \overset{\text{cond}}{\leq} 2$
$f(x) \overset{\text{cons}}{\neq} x^3 + 1 \text{ car } x \overset{\text{hyp}}{\not\leq} 2$

Remarque. La mise en forme du texte est faite par la macro `\txtdecoope` qui utilise `\coldecoope` pour la coloration *(il est donc facile de changer la couleur du texte mais aussi la mise en forme du texte si besoin).*

2. Quelques écritures symboliques nouvelles

Certaines décorations fournissent une écriture symbolique obtenue via la version étoilée de la macro d'un opérateur. Si une telle écriture n'existe pas, le package utilisera silencieusement la version non étoilée. Pour le moment, seule la macro `\equ` propose ceci ainsi que sa version négative. Voici tous les cas possibles.

```
$a \eq*[def]   1$ ou $a \neq*[def]   1$

$b \eq*[id]    2$ ou $b \neq*[id]    2$
```

$a \eq*[def] 1$ ou $a \neq*[def] 1$

$b \eq*[id] 2$ ou $b \neq*[id] 2$

III. Équivalences et implications

1. Des symboles logiques supplémentaires

Exemple 1 – Implication réciproque

En plus des opérateurs `\iff` et `\implies` proposés par LaTeX, il a été ajouté l'opérateur `\becauseof` un opérateur pour pour obtenir $\impliedby$ [1] ainsi que leurs versions négatives. Voici un exemple d'utilisation.

```
$(A \implies B)
 \iff (B \becauseof A)$

$(A \implies B)
 \niff (A \nimplies B)$
```

$(A \implies B) \iff (B \becauseof A)$

$(A \implies B) \niff (A \nimplies B)$

Exemple 2 – Des opérateurs décorés

Tout comme pour les comparaisons, pour décorer il suffit de passer via un argument optionnel.

```
$A \iff[ssi] B \niff[i.e.] C$

$A \implies[donc] B \nimplies[donne] C$

$A \becauseof[car] B \nbecauseof[d'après]
↪  C$
```

$A \iff[ssi] B \niff[i.e.] C$

$A \implies[donc] B \nimplies[donne] C$

$A \becauseof[car] B \nbecauseof[d'après] C$

2. Équivalences et implications verticales

À quoi cela sert-il ?

Les sections ?? et ?? présentent deux environnements pour détailler les étapes d'un raisonnement. Avec ces outils il devient utile d'avoir des versions verticales non décorées des symboles d'équivalence et d'implication. Voici comment les obtenir *(tous les cas possibles ont été indiqués)*. Bien entendu le préfixe `v` est pour `v`-ertical.

1. Penser aussi aux preuves d'équivalence par double implication.

```
\begin{tabular}{cccccc}
    $A$           & $B$
  & $C$           & $D$
  & $E$           & $F$
  \\
    $\viff$       & $\vimplies$
  & $\vbecauseof$  & $\nviff$
  & $\nvimplies$ & $\nvbecauseof$
  \\
    $A$           & $B$
  & $C$           & $D$
  & $E$           & $F$
\end{tabular}
```

$$\begin{array}{cccccc} A & B & C & D & E & F \\ \Updownarrow & \Downarrow & \Uparrow & \not\Updownarrow & \not\Downarrow & \not\Uparrow \\ A & B & C & D & E & F \end{array}$$

IV. Des versions alternatives du quantificateur existentiel

1. Quantifier l'existence

Voici deux versions, l'une classique, et l'autre beaucoup moins, permettant de préciser le quantificateur $\exists$.

```
$\existsone x \in E$ pour
\og il existe un seul $x$ \fg.

$\existmulti{\leq 1} y \in F$ pour
\og il existe au plus un $y$ \fg.

$\existmulti{=1} \eq*[def] \existsone$
```

$\exists! x \in E$ pour « il existe un seul x ».
$\exists_{\leq 1} y \in F$ pour « il existe au plus un y ».
$\exists_{=1} \coloneqq \exists!$

2. Versions négatives

```
$\nexistsone$ pour
\og il n'existe pas un unique \fg.

$\existmulti{\neq1} \eq*[def]
↪  \nexistsone$

$\existmulti{>4}$ pour
\og il n'existe pas plus de quatre \fg.

$\nexists$ vient de \verb+amssymb+.
```

$\nexists!$ pour « il n'existe pas un unique ».
$\exists_{\neq 1} \coloneqq \nexists!$
$\nexists_{>4}$ pour « il n'existe pas plus de quatre ».
$\nexists$ vient de `amssymb`.

V. Détailler un raisonnement simple

1. Version pour le lycée et après

Exemple 1 – Avec les réglages par défaut

L'environnement `stepcalc` permet de détailler les étapes principales d'un calcul ou d'un raisonnement simple en s'appuyant sur la macro `\explnext` dont le nom vient de *« expl-ain next step »* soit *« expliquer la prochaine étape »* en anglais [2]. On dispose aussi de `\explnext*` pour des explications descendantes et/ou montantes [3].

Ci-dessous se trouve un exemple, très farfelu vers la fin, où l'on utilise les réglages par défaut. Notons au passage que ce type de présentation n'est sûrement pas bien adaptée à un jeune public pour lequel une 2e façon de détailler des calculs et/ou un raisonnement simple est proposée plus bas dans la section 2..

```
\begin{stepcalc}
    (a + b)^2
    \explnext{On utilise $x^2 = x \cdot x$.}
    (a + b) (a + b)
        \explnext*{Double développement depuis la parenthèse gauche.}%
                  {Double factorisation pas facile.}
    a^2 + a b + b a + b^2
        \explnext*{}%
                  {Commutativité du produit.}
    a^2 + 2 a b + b^2
        \explnext*{Commutativité de l'addition.}%
                  {}
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

$$
\begin{aligned}
&(a+b)^2 \\
=\quad & \{\ \textit{On utilise } x^2 = x \cdot x.\ \} \\
&(a+b)(a+b) \\
=\quad & \left\{ \begin{array}{l} \downarrow \textit{ Double développement depuis la parenthèse gauche. } \downarrow \\ \uparrow \qquad\qquad \textit{Double factorisation pas facile.} \qquad\qquad \uparrow \end{array} \right\} \\
&a^2 + ab + ba + b^2 \\
=\quad & \{\uparrow \textit{ Commutativité du produit. } \uparrow\} \\
&a^2 + 2ab + b^2 \\
=\quad & \{\downarrow \textit{ Commutativité de l'addition. } \downarrow\} \\
&a^2 + b^2 + 2ab
\end{aligned}
$$

Remarque. Il faut savoir que la mise en forme est celle d'une formule ce qui peut rendre service comme dans l'exemple suivant.

2. Cet environnement utilise aussi le package `witharrows` qui est très sympathique pour expliquer des étapes de calcul.

3. Les explications données ne doivent pas être trop longues car ce serait contre-productif.

```
Un calcul avec un placement pouvant être
utile :
\begin{stepcalc}
    (a + b)^2
        \explnext{Identité remarquable.}
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

Un calcul avec un placement pouvant être utile :
$$(a+b)^2 \quad \{\textit{Identité remarquable.}\} \quad a^2+b^2+2ab$$

Avec un retour à la ligne, il faudra donc si besoin gérer l'espacement vertical.

```
Mon calcul pas trop proche.

\medskip
\begin{stepcalc}
    (a + b)^2
        \explnext{Identité remarquable.}
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

Mon calcul pas trop proche.
$$(a+b)^2 \quad \{\textit{Identité remarquable.}\} \quad a^2+b^2+2ab$$

Remarque. Voici des petites choses à connaître sur les macros `\explnext` et `\explnext*`.

1. `\expltxt` est utilisée par `\explnext` pour mettre en forme le texte d'explication.
2. `\expltxtup` et `\expltxtdown` sont utilisées par `\explnext*` décorer les textes d'explication juste avant leur mise en forme finale via `\expltxtupdown`.
3. `\explnext` et `\explnext*` utilisent la macro constante `\expltxtspacein` pour l'espacement entre le symbole et la courte explication. Par défaut, cette macro vaut `2em`.

Exemple 2 – Utiliser un autre symbole globalement

L'environnement `stepcalc` possède plusieurs options dont l'une est `ope` qui vaut `{=}` par défaut. Ceci permet de faire ce qui suit sans effort.

```
\begin{stepcalc}[ope = \viff]
    x^2 + 10 x + 25 = 0
        \explnext{Identité remarquable.}
    (x + 5)^2 = 0
        \explnext{$P^2 = 0$ si et
                  seulement si $P = 0$.}
    x = -5
\end{stepcalc}
```

$$x^2+10x+25=0 \quad \{\textit{Identité remarquable.}\} \quad (x+5)^2=0 \quad \{P^2=0 \textit{ si et seulement si } P=0.\} \quad x=-5$$

Exemple 3 – Juste utiliser des symboles

Si l'argument obligatoire de la macro `\explnext` est vide alors seul le symbole est affiché *(ne pas oublier les accolades vides)*. Voici un court exemple de ceci.

```
\begin{stepcalc}[ope = \viff]
    a^2 = b^2
        \explnext{}
    a = \pm b
\end{stepcalc}
```

$$a^2=b^2 \quad a=\pm b$$

Exemple 4 – Utiliser un autre symbole localement

La macro `\explnext` possède un argument optionnel qui utilise par défaut celui de l'environment. En utilisant cette option, on choisit alors localement le symbole à employer. Voici un exemple d'utilisation complètement farfelu bien que correct.

```
\begin{stepcalc}[ope = \viff]
    0 \leq a < b
        \explnext[\vimplies]%
                 {Croissance de $x^2$
                  sur $R_{+}$.}
    a^2 < b^2
        \explnext{}
    a^2 - b^2 < 0
        \explnext{Identité remarquable.}
    (a - b)(a + b) < 0
        \explnext[\vimplies]{}
    a \neq b
\end{stepcalc}
```

$$
\begin{aligned}
& 0 \leq a < b \\
\Downarrow \quad & \{ \textit{Croissance de } x^2 \textit{ sur } R_+. \} \\
& a^2 < b^2 \\
\Updownarrow \quad & \\
& a^2 - b^2 < 0 \\
\Updownarrow \quad & \{ \textit{Identité remarquable.} \} \\
& (a-b)(a+b) < 0 \\
\Downarrow \quad & \\
& a \neq b
\end{aligned}
$$

Exemple 5 – Choisir la mise en forme des explications

Pour la mise en forme des explications à double sens, la macro `\explnext` fait appel à la macro `\expltxt`. Par défaut, le package utilise la définition suivante.

```
\newcommand\expltxt[1]{%
    \text{\color{blue}\footnotesize \{\,{\itshape #1}\,\} }%
}
```

Pour la mise en forme des explications à sens unique, la macro `\explnext*` fait appel aux macros `\expltxtup`, `\expltxtdown` et `\expltxtupdown`. Par défaut le package utilise les définitions suivantes.

```
\newcommand\expltxtup[1]{%
    $\uparrow$ #1 $\uparrow$%
}

\newcommand\expltxtdown[1]{%
    $\downarrow$ #1 $\downarrow$%
}

\newcommand\expltxtupdown[2]{{%
    \displaystyle\footnotesize\color{blue}%
    \left\{\,%
        \genfrac{}{}{0pt}{}{%
            \text{\itshape\expltxtdown{\samesizeas{#1}{#2}}}%
        }{%
            \text{\itshape\expltxtup{\samesizeas{#2}{#1}}}%
        }%
    \,\right\}%
}}
```

Nous allons expliquer comment obtenir l'affreux exemple ci-dessous montrant que l'on peut adapter si besoin la mise en forme.

```
\begin{stepcalc}
    (a + b) (a + b)
        \explnext{Se souvenir de $P\cdot
        ↪  P = P^2$.}%
    (a + b)^2
        \explnext*{Id. Rm. - Dév.}%
                  {Id. Rm. - Facto.}
    a^2 + 2 a b + b^2
\end{stepcalc}
```

$$
\begin{aligned}
&(a+b)(a+b)\\
&= \qquad \Downarrow \textit{ Se souvenir de } P\cdot P = P^2. \Uparrow\\
&(a+b)^2\\
&= \qquad \left\langle \begin{array}{c} \Downarrow \textit{ Id. Rm. - Dév. } \Downarrow \\ \hline \Uparrow \textit{ Id. Rm. - Facto. } \Uparrow \end{array} \right\rangle\\
&a^2+2ab+b^2
\end{aligned}
$$

La mise en forme a été obtenue en utilisant le code LaTeX suivant où la macro `\samesizeas{#1}{#2}` rend le texte `#1` aussi large que `#2` en ajoutant des espaces supplémentaires tout en centrant le résultat final si besoin *(ne pas oublier de passer en mode texte via `\text`)*.

```
\newcommand\myexpltxt[2]{%
    \text{\color{#1} \footnotesize \itshape \bfseries #2}%
}

\renewcommand\expltxt[1]{%
    \myexpltxt{gray}{$\Downarrow$ #1 $\Uparrow$}%
}

\renewcommand\expltxtup[1]{%
    \myexpltxt{orange}{$\Uparrow$ #1 $\Uparrow$}%
}

\renewcommand\expltxtdown[1]{%
    \myexpltxt{red}{$\Downarrow$ #1 $\Downarrow$}%
}

\renewcommand\expltxtupdown[2]{%
    \displaystyle\color{blue!20!black!30!green}%
    \genfrac{\langle}{\rangle}{1pt}{}{%
        \expltxtdown{\samesizeas{#1}{#2}}%
    }{%
        \expltxtup{\samesizeas{#2}{#1}}%
    }%
}
```

2. Version pour les collégiens

L'environnement `stepcalc`[4] avec les options `style = ar` et `style = ar*` imprime une rédaction plus classique tout en utilisant des flèches pour indiquer les explications *(`ar` est pour `ar`-row soit « flèche » en anglais)*. Dans ce cas d'utilisation, la macro `\explnext*` permet d'avoir une flèche unidirectionnelle, vers le haut ou le bas au choix, ou bien d'écrire deux indications dont l'une est montante et l'autre descendante.

4. Cet environnement utilise aussi le package `witharrows`.

Il existe aussi l'option `style = sar` lorsque la toute 1re étape n'est pas expliquée *(`s` est pour `s`-hort soit « court » en anglais)*. Voir l'exemple 4 page 29.

Exemple 1 – L'opérateur dans la marge... ou pas

Considérons le code suivant qui utilise `style = ar`.

```
\begin{stepcalc}[style = ar]
    (a + b)^2
        \explnext{}
    a^2 + 2 a b + b^2
\end{stepcalc}
```

$$\begin{aligned}&(a+b)^2\\=\ &a^2+2ab+b^2\end{aligned}$$

Intégré dans un paragraphe, on obtient le rendu suivant qui vaut ce qu'il vaut ici.

$$\begin{aligned}&(a+b)^2\\=\ &a^2+2ab+b^2\end{aligned}$$

En utilisant `style = ar*` au lieu de `style = ar`, on aboutit à ce qui suit qui semble mieux.

$$\begin{aligned}&(a+b)^2\\=\ &a^2+2ab+b^2\end{aligned}$$

Dans la suite, on va se concentrer sur le style `ar` mais tout ce qui est dit pour ce style s'applique aussi à `ar*`.

Exemple 2 – Des flèches à double sens

```
\begin{stepcalc}[style = ar]
    (a + b)^2
        \explnext{Identité remarquable}
    a^2 + 2 a b + b^2
        \explnext{}
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

$$\begin{aligned}&(a+b)^2\\=\ &a^2+2ab+b^2 \quad \text{Identité remarquable}\\=\ &a^2+b^2+2ab\end{aligned}$$

Exemple 3 – Des flèches unidirectionnelles

Ce qui suit est juste là comme démo. car les explications y sont un peu farfelues.

```
\begin{stepcalc}[style = ar]
    (a + b)^2
        \explnext*{Via $P^2 = P \cdot P$.}
                  {Via $P \cdot P = P^2$.}
    (a + b) (a + b)
        \explnext*{Double développement.}%
                  {Double factorisation (pas simple).}
    a^2 + a b + b a + b^2
        \explnext*{Commutativité du produit.}%
                  {}
    a^2 + 2 a b + b^2
        \explnext*{}%
                  {Commutativité de l'addition.}
```

```
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

$$
\begin{aligned}
&(a+b)^2 \\
&= (a+b)(a+b) && \text{Via } P^2 = P\cdot P. \quad \text{Via } P\cdot P = P^2. \\
&= a^2 + ab + ba + b^2 && \text{Double développement. Double factorisation (pas simple).} \\
&= a^2 + 2ab + b^2 && \text{Commutativité du produit.} \\
&= a^2 + b^2 + 2ab && \text{Commutativité de l'addition.}
\end{aligned}
$$

Exemple 4 – Ne pas expliquer le tout début

L'environnement `stepcalc` avec l'option `style = sar` débute différemment la mise en forme. Bien entendu ici le tout premier `\explnext` doit avoir un argument vide !

```
\begin{stepcalc}[style = sar]
    (a + b) (a + b)
        \explnext{}
    (a + b)^2
        \explnext{Identité remarquable.}
    a^2 + b^2 + 2 a b
\end{stepcalc}
```

$$
\begin{aligned}
(a+b)(a+b) &= (a+b)^2 \\
&= a^2 + b^2 + 2ab && \text{Identité remarquable.}
\end{aligned}
$$

Exemple 5 – Choisir son symbole

Voici comment faire où l'implication finale est juste là pour la démonstration *(on notera une petite bidouille un peu sale à faire pour avoir un alignement à peu près correct)*.

```
\begin{stepcalc}[style = ar, ope = \iff]
    a^2 + 2 a b + b^2 = 0
        \explnext{}
    (a + b)^2 = 0
        \explnext[\:\implies]%
                 {$P^2 = 0$ ssi $P = 0$.}
    a + b = 0
\end{stepcalc}
```

$$
\begin{aligned}
& a^2 + 2ab + b^2 = 0 \\
\iff\ & (a+b)^2 = 0 && P^2 = 0 \text{ ssi } P = 0. \\
\implies\ & a + b = 0
\end{aligned}
$$

Avec la version courte, on obtient ce qui suit.

```
\begin{stepcalc}[style = sar, ope = \iff]
    a^2 + 2 a b + b^2 = 0
        \explnext{}
    (a + b)^2 = 0
        \explnext[\:\implies]%
                 {$P^2 = 0$ ssi $P = 0$.}
    a + b = 0
\end{stepcalc}
```

$$\begin{aligned} a^2 + 2ab + b^2 = 0 &\iff (a+b)^2 = 0 \\ &\implies a + b = 0 \end{aligned}$$ $P^2 = 0$ *ssi* $P = 0$.

3. De courts commentaires

Exemple 1 – Sans alignement

Il est possible d'ajouter de petits commentaires via `\comthis` où `comthis` est pour `com`-ment `this` soit *« commenter ceci »* en anglais.

```
\begin{stepcalc}
    (a + b)^2
        \comthis{Forme facto.}
        \explnext*{Id.Rq. -- Dév.}%
                  {Id.Rq. -- Facto.}
    a^2 + 2 a b + b^2
        \comthis{Forme dév.}
\end{stepcalc}
```

$$\begin{aligned} &(a+b)^2 && [\textit{ Forme facto. }] \\ &= && \left\{ \begin{array}{l} \downarrow \textit{ Id.Rq. – Dév. } \downarrow \\ \uparrow \textit{ Id.Rq. – Facto. } \uparrow \end{array} \right\} \\ &a^2 + 2ab + b^2 && [\textit{ Forme dév. }] \end{aligned}$$

Remarque. La mise en forme du texte des commentaires est fait via la macro personnalisable `\explcom`. Quant à l'espacement ajouté entre le texte et son commentaire il est défini par la macro `\expltxtspacein` qui est égale à `2em` par défaut.

Exemple 2 – Tout aligner

Il peut être utile d'aligner tous les commentaires. Ceci s'obtient via l'option `com = al` où `al` est pour `al`-igné *(par défaut `com = nal` avec le préfixe `n` pour `n`-on).*

```
\begin{stepcalc}[com = al]
    (a + b)^2
        \comthis{Forme facto.}
        \explnext*{Id.Rq - Dév.}%
                  {Id.Rq - Facto.}
    a^2 + 2 a b + b^2
        \comthis{Forme dév.}
\end{stepcalc}
```

$$\begin{aligned} &(a+b)^2 && [\textit{ Forme facto. }] \\ &= && \left\{ \begin{array}{l} \downarrow \textit{ Id.Rq - Dév. } \downarrow \\ \uparrow \textit{ Id.Rq - Facto. } \uparrow \end{array} \right\} \\ &a^2 + 2ab + b^2 && [\textit{ Forme dév. }] \end{aligned}$$

Exemple 3 – Le meilleur des deux mondes

Dans d'autres situations, utilisez les deux types d'alignement peut faire sens. Ceci s'obtient via l'option `com = al` et l'emploi de la macro étoilée `\comthis*` à chaque fois que l'on souhaite "coller" un commentaire le plus à gauche possible.

```
\begin{stepcalc}[com = al]
    (a + b) (a + b)
        \comthis{Forme facto.}
        \explnext{Via $x^2 = x \cdot x$.}
    (a + b)^2
        \comthis*{Au passage...}
        \explnext*{Id.Rq - Dév.}%
                  {Id.Rq - Facto.}
    a^2 + 2 a b + b^2
        \comthis{Forme dév.}
\end{stepcalc}
```

$$
\begin{aligned}
&(a+b)(a+b) && [\ \textit{Forme facto.}\ ]\\
&= \quad \{\ \textit{Via } x^2 = x\cdot x.\ \} &&\\
&(a+b)^2 \qquad [\ \textit{Au passage...}\ ] &&\\
&= \quad \left\{\begin{array}{l}\downarrow\ \textit{Id.Rq - Dév.}\ \downarrow\\ \uparrow\ \textit{Id.Rq - Facto.}\ \uparrow\end{array}\right\} &&\\
&a^2+2ab+b^2 && [\ \textit{Forme dév.}\ ]
\end{aligned}
$$

Remarque. Si l'alignement n'est pas activé, les macros `\comthis*` et `\comthis` auront toutes les deux le même effet.

Exemple 4 – Ceci marche aussi avec le style « fléché »

Voici ce que donne le mode mixte lorsque des flèches sont utilisées pour les explications. Il semble moins pertinent ici de mixer les modes « alignement » et « non alignement » mais chacun pris séparément peut avoir son utilité.

```
\begin{stepcalc}[style = ar, com = al]
    (a + b) (a + b)
        \comthis{Forme facto.}
        \explnext{Via $x^2 = x \cdot x$.}
    (a + b)^2
        \comthis*{Au passage...}
        \explnext*{Id.Rq - Dév.}%
                  {Id.Rq - Facto.}
    a^2 + 2 a b + b^2
        \comthis{Forme dév.}
\end{stepcalc}
```

$$
\begin{aligned}
&(a+b)(a+b) && [\ \textit{Forme facto.}\ ]\\
&= (a+b)^2 \qquad [\ \textit{Au passage...}\ ] &&\\
&= a^2+2ab+b^2 && [\ \textit{Forme dév.}\ ]
\end{aligned}
\qquad
\begin{array}{l}\textit{Via } x^2 = x\cdot x.\\ \textit{Id.Rq - Dév.} \quad \textit{Id.Rq - Facto.}\end{array}
$$

Remarque. Bien entendu il est impossible de commenter le tout début en mode fléché court.

VI. Détailler un « vrai » raisonnement

1. Un tableau pour le post-bac

Exemple 1 – Le minimum avec les réglages par défaut

Prenons un exemple utile à la logique formelle en informatique théorique mais qui a complètement sa place en mathématiques plus classiques *(voir la section ?? pour un autre type de présentation plus adapté à un public de collège ou de lycée)*. Ci-dessous l'environnement `demotab` facilite la mise en page [5] et la macro étoilée `\explref*` permet d'indiquer une référence interne au raisonnement [6].

5. En coulisse est utilisé l'environnement `longtable` du package éponyme.
6. Les indications peuvent être numérotes jusqu'à 99 ce qui est bien au-delà des besoins pratiques.

Dans cet exemple en deux morceaux, pour montrer au passage comment continuer la numérotation là où elle s'était arrêtée, on utilise *« m.p. »* comme abréviation de *« modus ponens »*.

```
\begin{demotab}
    \demostep
        Hypothèse & $A$
    \demostep
        Axiome 1  & $A \implies B$
    \demostep
        m.p. sur
        \explref*{1} et \explref*{2}
      & $B$
    \demostep
        \explref*{1} et \explref*{3}
      & $A \wedge B$
\end{demotab}
```

1	Hypothèse	A
2	Axiome 1	$A \implies B$
3	m.p. sur 1 et 2	B
4	1 et 3	$A \wedge B$

Il est possible de couper sa démonstration en morceaux en indiquant à l'environnement la valeur du 1er numéro de justification via la clé `start` : la valeur spéciale `last` indique de continuer la numérotation à la suite.

```
\begin{demotab}[start = last]
    \demostep
        Axiome 3
      & $(A \wedge B) \implies C$
    \demostep
        m.p. sur
        \explref*{4} et \explref*{5}
      & $C$
\end{demotab}
```

5	Axiome 3	$(A \wedge B) \implies C$
6	m.p. sur 4 et 5	C

Exemple 2 – Référencer une indication

L'argument optionnel de `\demostep` permet de définir un label qui ensuite facilitera le référencement d'une justification de façon pérenne via la macro non étoilée `\explref`.

```
\begin{demotab}
    \demostep[demo-my-hyp]
        Hypothèse & $A$
    \demostep[demo-use-axiom-1]
        Axiome 1  & $A \implies B$
    \demostep
        m.p. sur
        \explref{demo-my-hyp}
        et
        \explref{demo-use-axiom-1}
      & $B$
\end{demotab}
```

1	Hypothèse	A
2	Axiome 1	$A \implies B$
3	m.p. sur 1 et 2	B

Remarque. Prendre bien garde au fait que ce mécanisme utilise les macros `\label` et `\ref` de LaTeX. On travaille donc avec des références globalement au document compilé.

Exemple 3 – Indiquer ce que l'on cherche à faire

Les clés optionnelles `hyps` pour plusieurs hypothèses, `hyp` pour une seule hypothèse et `ccl` pour la conclusion permettent d'expliquer ce que l'on démontre et sous quel contexte.

```
\begin{demotab}[hyp = $A$, ccl = $B$]
    \demostep
        Hypothèse & $A$
    \demostep
        Axiome 1  & $A \implies B$
    \demostep
        m.p. sur
        \explref*{1} et \explref*{2}
      & $B$
\end{demotab}
```

Démonstration sous l'hypothèse : A

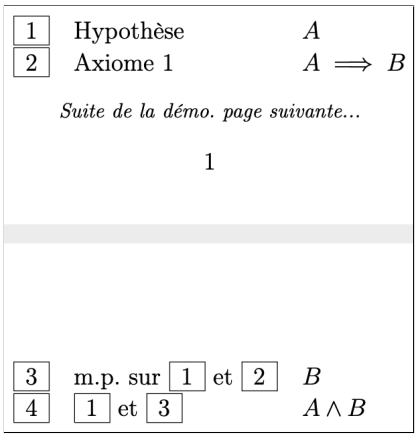
1	Hypothèse	A
2	Axiome 1	$A \implies B$
3	m.p. sur 1 et 2	B

Conclusion : B

Remarque. Aucune des clés `hyps`, `hyp` et `ccl` n'est obligatoire. Par contre il n'est pas possible d'utiliser à la fois les clés `hyps` et `hyp`.

2. Un tableau sur plusieurs pages

Un tableau devant utiliser plusieurs pages sera scindé comme ci-dessous sans perte d'information [7].



3. Un tableau pour le collège et le lycée

Exemple 1 – Avec les réglages par défaut

L'environnement étoilé `demotab*` est différent de l'environnement `demotab` puisqu'il sert à indiquer trois choses et non juste deux comme le montre l'exemple suivant [8]. Par contre, la syntaxe est très similaire. Notez au passage la possibilité d'utiliser `\newline` pour forcer un retour à la ligne dans une cellule et aussi la nécessité d'écrire les accolades de la macro sans argument `\demostep` lorsque la 1re case est vide *(ceci est inutile lorsque l'argument optionnel est renseigné comme nous allons le vérifier dans l'exemple juste après)*.

```
\begin{demotab*}
    \demostep
        $ABC$ est un triangle \newline équilatéral
      & Définition d'un triangle \newline équilatéral.
      & $AB = BC = AC$
```

7. Tout le travail est fait par l'environnement `longtable` du package éponyme.
8. C'est pour cela qu'est proposé une version étoilée de l'environnement et non l'utilisation d'une option de l'environnement non étoilé.

```
    \demostep{} % --> Ne pas oublier ici !
      & Voir l'énoncé.
      & $AB = 10 \, cm$
    \demostep
        Voir les conséquences \newline \explref*{1} et \explref*{2} .
      & Simple calcul.
      & $ABC$ a pour périmètre $30 \, cm$.
\end{demotab*}
```

Réf.	Je sais que...	Propriété ou fait utilisé	Conséquence
1	ABC est un triangle équilatéral	Définition d'un triangle équilatéral.	$AB = BC = AC$
2		Voir l'énoncé.	$AB = 10\,cm$
3	Voir les conséquences 1 et 2 .	Simple calcul.	ABC a pour périmètre $30\,cm$.

Exemple 2 – Avec toutes les options

Le système de référence marche ici aussi. Par contre `demotab*` ne propose que `start` comme clé optionnelle avec le même fonctionnement que pour `demotab`.

```
\begin{demotab*}[start = last]
    \demostep[demo-first-geo-fact]
        $ABC$ est un triangle \newline équilatéral
      & Définition d'un triangle \newline équilatéral.
      & $AB = BC = AC$
    \demostep[known-data]
      & Voir l'énoncé.
      & $AB = 10 \, cm$
    \demostep
        Voir les conséquences \newline
        \explref{demo-first-geo-fact} et \explref{known-data} .
      & Simple calcul.
      & $ABC$ a pour périmètre $30 \, cm$.
\end{demotab*}
```

Réf.	Je sais que...	Propriété ou fait utilisé	Conséquence
4	ABC est un triangle équilatéral	Définition d'un triangle équilatéral.	$AB = BC = AC$
5		Voir l'énoncé.	$AB = 10\,cm$
6	Voir les conséquences 4 et 5 .	Simple calcul.	ABC a pour périmètre $30\,cm$.

4. Un tableau sur plusieurs pages

Un tableau devant utiliser plusieurs pages sera scindé comme ci-dessous sans perte d'information [9].

Réf.	Je sais que...	Propriété ou fait utilisé	Conséquence
1	ABC est un triangle équilatéral	Définition d'un triangle équilatéral.	$AB = BC = AC$
2		Voir l'énoncé.	$AB = 10\,cm$
3	Voir les conséquences 1 et 2 .	Simple calcul.	ABC a pour périmètre $30\,cm$.

Suite de la démo. page suivante...

Réf.	Je sais que...	Propriété ou fait utilisé	Conséquence
4	ABC est un triangle équilatéral	Définition d'un triangle équilatéral.	$AB = BC = AC$
5		Voir l'énoncé.	$AB = 10\,cm$
6	Voir les conséquences 4 et 5 .	Simple calcul.	ABC a pour périmètre $30\,cm$.

9. Tout le travail est fait par l'environnement `longtable` du package éponyme.

Chapitre 4

Géométrie

I. Ensembles géométriques

Se reporter à la section 2. page 13 où est présentée la macro `\setgeo` pour indiquer des ensembles géométriques.

II. Utiliser des unités S.I.

Comme l'excellent package `siunitx` est chargé en coulisse il devient facile de travailler avec des unités de mesure tout en respectant **les conventions d'écriture qui seront françaises** [1] **dès lors que vous aurez chargé `babel` avec l'option `french`** comme c'est le cas pour cette documentation. Notez que les espaces dans `\num{123 000}` et `\num{1230 * 100}` sont inutiles mais qu'ils facilitent la relecture du code.

```
$\ang{180} = \SI{\pi}{\radian}$ ,
$\SI{1}{km} = \SI{1e3}{m}$ avec aussi
$\num{123 000} = \num{1230 * 100}$
```

$180^\circ = \pi\,\mathrm{rad}$, $1\,\mathrm{km} = 1\times10^{3}\,\mathrm{m}$ avec aussi $123\,000 = 1230\times100$

III. Points et lignes

1. Points

Exemple 1 – Sans indice

```
$\pt{I}$
```

I

Exemple 2 – Avec un indice

```
$\pt*{I}{1}$ ou
$\pt*{I}{2}$
```

I_1 ou I_2

1. Notez dans l'exemple l'écriture de 1230 n'utilise pas d'espace contrairement à celle de 12 300.

2. Lignes

Exemple 1 – Les droites

Dans l'exemple suivant, le préfixe `g` est pour `g`-éometrie tandis que `p` est pour `p`-oint [2].

```
$\gline{A}{B}$ ,
$\gline{\pt{A}}{\pt{B}}$ ou
$\gpline{A}{B}$
```

$\gline{A}{B}$, $\gline{\pt{A}}{\pt{B}}$ ou $\gpline{A}{B}$

Exemple 2 – Les segments

Les macros `\segment` et `\psegment` ont un comportement similaire à `\gline` et `\gpline`.

```
$\segment{A}{B}$ ,
$\segment{\pt{A}}{\pt{B}}$ ou
$\psegment{A}{B}$
```

$\segment{A}{B}$, $\segment{\pt{A}}{\pt{B}}$ ou $\psegment{A}{B}$

Exemple 3 – Les demi-droites

Dans l'exemple suivant, le préfixe `h` est pour `h`-alf soit *« moitié »* en anglais.

```
$\ghline{A}{B}$ ,
$\ghline{\pt{A}}{\pt{B}}$ ou
$\gphline{A}{B}$
```

$\ghline{A}{B}$, $\ghline{\pt{A}}{\pt{B}}$ ou $\gphline{A}{B}$

Exemple 4 – D'autres demi-droites

Ce qui suit nécessite d'utiliser l'argument optionnel de `\gline` et `\gpline`. La valeur `OC` provient de `O`-pened – `C`-losed soit *« ouvert – fermé »* en anglais.

```
$\gline[OC]{A}{B}$ ,
$\gline[OC]{\pt{A}}{\pt{B}}$ ou
$\gpline[OC]{A}{B}$
```

$\gline[OC]{A}{B}$, $\gline[OC]{\pt{A}}{\pt{B}}$ ou $\gpline[OC]{A}{B}$

Remarque. Les segments utilisent en fait l'option `C` et les demi-droites standard l'option `CO`. La valeur par défaut est `O`.

3. Droites parallèles ou non

Les opérateurs `\parallel` et `\nparallel` utilisent des obliques au lieu de barres verticales comme le montre l'exemple qui suit où `\stdnparallel` est un alias de `\nparallel` fourni par le package `amssymb`, et `\stdparallel` est un alias de la version standard de `\parallel` proposée par LaTeX.

2. Comme la macro `\line` est proposée par LaTeX, il a fallu ajouter le préfixe `g`.

```
$\gpline{A}{B} \parallel \gpline{C}{D}$
au lieu de
$\gpline{A}{B}
 \stdparallel \gpline{C}{D}$

$\gpline{E}{F} \nparallel \gpline{G}{H}$
au lieu de
$\gpline{E}{F}
 \stdnparallel \gpline{G}{H}$
```

(AB) ∥ (CD) au lieu de (AB) ∥ (CD)
(EF) ∦ (GH) au lieu de (EF) ∦ (GH)

IV. Vecteurs

1. Les écrire

Exemple 1

```
$\vect{UnNomLong}$  ,
$\vect*{e}{rot}$ ou
$\vect{e_{rot}}$
```

$\overrightarrow{UnNomLong}$, $\vec{e}_{rot}$ ou $\overrightarrow{e_{rot}}$

Exemple 2 – Sans point disgracieux

```
$\vect{i}$ ou
$\vect*{j}{2}$
```

$\vec{\imath}$ ou $\vec{\jmath}_2$

Exemple 3 – Se passer de la macro \pt

```
$\pvect{A}{B} = \vect{\pt{A}\pt{B}}$.
```

$\overrightarrow{\mathrm{AB}} = \overrightarrow{\mathrm{AB}}$.

2. Norme

```
 $\norm{\vect{i}} = \vnorm{i}$

 $\norm {\dfrac{2}{7} \vect*{e}{k}}
= \norm*{\dfrac{2}{7} \vect*{e}{k}}$
```

$\|\vec{\imath}\| = \|\vec{\imath}\|$
$\left\|\frac{2}{7}\vec{e}_k\right\| = \|\frac{2}{7}\vec{e}_k\|$

Remarque. Le code LaTeX pour des doubles barres extensibles ou non vient directement de ce message : `https://tex.stackexchange.com/a/43009/6880`.

3. Produit scalaire

Les 1ers exemples utilisent une syntaxe longue mais adaptables à toutes les situations. Voir l'exemple 5 un peu plus bas pour une écriture rapide utilisable dans certains cas.

Exemple 1 – Version classique

```
$\dotprod{\dfrac{1}{2} \vect{u}}%
         {\vect{v}}$
```

$\dotprod{\dfrac{1}{2} \vect{u}}{\vect{v}}$

Exemple 2 – Version « pédagogique mais pas écolo. »

Dans l'exemple suivant l'option `b` est pour `b`-ullet soit *« puce »* en anglais. Cette écriture peut être utile avec des débutants mais elle est peu pratique pour une écriture manuscrite.

```
$\dotprod[b]{\dfrac{1}{2} \vect{u}}%
            {\vect{v}}$
```

$\dotprod[b]{\dfrac{1}{2} \vect{u}}{\vect{v}}$

Exemple 3 – Écriture « universitaire »

Dans l'exemple suivant l'option `p` est pour `p`-arenthèse et dans `sp` le `s` est pour `s`-mall soit *« petit »* en anglais. On rencontre souvent cette écriture dans les cursus mathématiques universitaires.

```
$\dotprod[p]{\dfrac{1}{2} \vect{u}}%
            {\vect{v}}$

$\dotprod[sp]{\dfrac{1}{2} \vect{u}}%
             {\vect{v}}$
```

$\dotprod[p]{\dfrac{1}{2} \vect{u}}{\vect{v}}$

$\dotprod[sp]{\dfrac{1}{2} \vect{u}}{\vect{v}}$

Exemple 4 – Écriture « à la physicienne »

Dans l'exemple suivant `r` est pour `r`-after soit *« chevron »* en anglais. Les physiciens aiment bien cette notation.

```
$\dotprod[r]{\dfrac{1}{2} \vect{u}}%
            {\vect{v}}$

$\dotprod[sr]{\dfrac{1}{2} \vect{u}}%
             {\vect{v}}$
```

$\dotprod[r]{\dfrac{1}{2} \vect{u}}{\vect{v}}$

$\dotprod[sr]{\dfrac{1}{2} \vect{u}}{\vect{v}}$

Exemple 5 – Version courte mais restrictive

Dans l'exemple suivant le préfixe `v` est pour `v`-ecteur. Notons que dans ce cas les options `sp` et `sr` n'apportent rien de nouveau.

```
 $\vdotprod   {u}{v}
= \vdotprod[b]{u}{v}$

 $\vdotprod[r]{u}{v}
= \vdotprod[p]{u}{v}$
```

$\vdotprod{u}{v} = \vdotprod[b]{u}{v}$

$\vdotprod[r]{u}{v} = \vdotprod[p]{u}{v}$

4. 3D – Produit vectoriel

Écriture symbolique

Exemple 1 – Version classique en France

```
$\crossprod{\dfrac{1}{2} \vect{i}}%
           {\vect{j}}$
```

$\frac{1}{2}\vec{\imath} \wedge \vec{\jmath}$

Exemple 2 – Version alternative

La macro `\crossprod` possède un argument optionnel que l'on peut utiliser pour obtenir la mise en forme suivante.

```
$\crossprod[t]{\dfrac{1}{2} \vect{i}}%
             {\vect{j}}$
```

$\frac{1}{2}\vec{\imath} \times \vec{\jmath}$

Exemple 3 – Version courte mais restrictive

```
$\vcrossprod   {i}{j}$ ou
$\vcrossprod[t]{i}{j}$
```

$\vec{\imath} \wedge \vec{\jmath}$ ou $\vec{\imath} \times \vec{\jmath}$

Explication du mode de calcul

Exemple 4

Dans l'exemple suivant, le préfixe `calc` est pour `calc`-uler quant à `v` est pour `v`-ecteur pour une rédaction raccourcie pour les vecteurs.

```
$\calccrossprod{\vect{u}}{x}{y}{z}%
               {\vect{v}}{x'}{y'}{z'}$
ou
$\vcalccrossprod{AB}{x_B - x_A}%
                    {y_B - y_A}%
                    {z_B - z_A}%
                {CD}{x_D - x_C}%
                    {y_D - y_C}%
                    {z_D - z_C}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix} \text{ ou } \begin{vmatrix} \overrightarrow{AB} & \overrightarrow{CD} \\ x_B - x_A & x_D - x_C \\ y_B - y_A & y_D - y_C \\ z_B - z_A & z_D - z_C \\ x_B - x_A & x_D - x_C \end{vmatrix}$$

Remarque. Tous les exemples suivants se feront avec `\vcalccrossprod` mais bien entendu ils restent adaptables directement à `\calccrossprod`.

Exemple 5 – Pour un public averti

On peut juste proposer des croix fléchées ou non, voir juste les coordonnées « augmentées » sans les décorations comme ci-après via l'argument optionnel de `\calccrossprod` ou `\vcalccrossprod`.

```
$ \vcalccrossprod[arrows]{u}{x}{y}{z}%
                         {v}{x'}{y'}{z'}
= \vcalccrossprod[cross] {u}{x}{y}{z}%
                         {v}{x'}{y'}{z'}
= \vcalccrossprod[nodeco]{u}{x}{y}{z}%
                         {v}{x'}{y'}{z'}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix} = \begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix} = \begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix}$$

Exemple 6 – Sans les vecteurs

Si les vecteurs vous gênent il suffira d'utiliser l'option `novec` pour `no vec`-tor soit *« pas de vecteur »* en anglais. Voici deux cas d'utilisation [3].

```
$ \vcalccrossprod[novec]
      {u}{x}{y}{z}{v}{x'}{y'}{z'}
= \vcalccrossprod[novec,cross]
      {u}{x}{y}{z}{v}{x'}{y'}{z'}$
```

$$\begin{vmatrix} x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix} = \begin{vmatrix} x & x' \\ y & y' \\ z & z' \\ x & x' \end{vmatrix}$$

Exemple 7 – Les coordonnées « développées »

Parmi les options proposées par `\calccrossprod` et `\vcalccrossprod`, il y en existe quelques unes fournissant des coordonnées [4] détaillant les calculs. Indiquons que `exp` est pour `exp`-and soit *« développer »* en anglais, `c` pour `\cdot` et enfin `t` pour `\times` [5].

```
$\vcalccrossprod[exp]%
                {u}{\dfrac{1}{2}x}{y}{z}%
                {v}{x'}{y'}{z'}$

$\vcalccrossprod[cexp,vb]%
                {u}{\dfrac{1}{2}x}{y}{z}%
                {v}{x'}{y'}{z'}$

$\vcalccrossprod[texp,sp]%
                {u}{\dfrac{1}{2}x}{y}{z}%
                {v}{x'}{y'}{z'}$
```

$$\left(y\,z' - z\,y' \,;\, z\,x' - \frac{1}{2}x\,z' \,;\, \frac{1}{2}x\,y' - y\,x' \right)$$

$$\begin{bmatrix} y \cdot z' - z \cdot y' \\ z \cdot x' - \frac{1}{2}x \cdot z' \\ \frac{1}{2}x \cdot y' - y \cdot x' \end{bmatrix}$$

$$(y \times z' - z \times y' \,;\, z \times x' - \frac{1}{2}x \times z' \,;\, \frac{1}{2}x \times y' - y \times x')$$

Voici les options disponibles. Nous expliquons ensuite comment les utiliser.

1. `p` vient de `p`-arenthèses. Ceci donnera une écriture horizontale.
2. `b` vient de `b`-rackets soit *« crochets »* en anglais. Ceci donnera une écriture horizontale.
3. `sp` et `sb` produisent des délimiteurs non extensibles en mode horizontal. Ici `s` vient de `s`-mall soit *« petit »* en anglais.
4. `vp` et `vb` produisent des écritures verticales. Ici `v` vient de `v`-ertical.
5. `exp` tout seul demande d'utiliser un espace pour séparer les facteurs de chaque produit.
6. `texp` tout seul demande d'utiliser `\times` comme opérateur de multiplication.
7. `cexp` tout seul demande d'utiliser `\cdot` comme opérateur de multiplication.

3. Il peut sembler un peu lourd d'avoir des arguments pour des vecteurs non affichés mais ce choix permet à l'usage de faire des copier-coller redoutables d'efficacité !

4. En coulisse on utilise la macro `\coord` présentée dans la section 1. page 45 dont on reprend les options de mise en forme.

5. De nouveau on obtient ici une méthode très pratique à l'usage car elle permet de faire des copier-coller.

Attention ! Les produits sont rédigés stupidement. Autrement dit ce sera à vous d'ajouter des parenthèses là où il y en aura besoin sinon vous obtiendrez des horreurs comme celle ci-dessous.

```
$\vcalccrossprod[exp,vb]%
 {AB}%
 {x_B - x_A}{y_B - y_A}{z_B - z_A}%
 {CD}%
 {x_D - x_C}{y_D - y_C}{z_D - z_C}$
```

$$\begin{bmatrix} y_B - y_A\,z_D - z_C - z_B - z_A\,y_D - y_C \\ z_B - z_A\,x_D - x_C - x_B - x_A\,z_D - z_C \\ x_B - x_A\,y_D - y_C - y_B - y_A\,x_D - x_C \end{bmatrix}$$

Ici nous n'avons pas d'autre choix que de corriger le tir nous-même. Ceci étant indiqué, ce genre de situation est très rare dans la vraie vie mathématique où l'on évite d'avoir à calculer un produit vectoriel avec des expressions compliquées.

```
$\vcalccrossprod[exp,vb]%
 {AB}%
 {(x_B - x_A)}{(y_B - y_A)}{(z_B - z_A)}%
 {CD}%
 {(x_D - x_C)}{(y_D - y_C)}{(z_D - z_C)}$
```

$$\begin{bmatrix} (y_B - y_A)\,(z_D - z_C) - (z_B - z_A)\,(y_D - y_C) \\ (z_B - z_A)\,(x_D - x_C) - (x_B - x_A)\,(z_D - z_C) \\ (x_B - x_A)\,(y_D - y_C) - (y_B - y_A)\,(x_D - x_C) \end{bmatrix}$$

5. 2D – Déterminant de deux vecteurs

Exemple 1 – Version décorée avec une boucle

Dans l'exemple suivant, le préfixe `calc` est pour `calc`-uler quant à `v` est pour `v`-ecteur pour une rédaction raccourcie pour les vecteurs.

```
$\calcdetplane{\vect{u}}{x}{y}%
              {\vect{v}}{x'}{y'}$
ou
$\calcdetplane{AB}%
              {x_B - x_A}{y_B - y_A}%
              {CD}%
              {x_D - x_C}{y_D - y_C}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \end{vmatrix} \text{ ou } \begin{vmatrix} \overrightarrow{AB} & \overrightarrow{CD} \\ x_B - x_A & x_D - x_C \\ y_B - y_A & y_D - y_C \end{vmatrix}$$

Remarque. Tous les exemples suivants se feront avec `\vcalcdetplane` mais bien entendu ils restent adaptables directement à `\calcdetplane`.

Exemple 2 – Version décorée avec une croix fléchée

L'argument optionnel de `\calcdetplane` ou `\vcalcdetplane` permet d'obtenir une croix fléchée au lieu d'une boucle.

```
$\vcalcdetplane[arrows]{u}{x}{y}%
                       {v}{x'}{y'}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \end{vmatrix}$$

Exemple 3 – Version décorée avec une croix non fléchée

```
$\vcalcdetplane[cross]{u}{x}{y}%
                      {v}{x'}{y'}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \end{vmatrix}$$

Exemple 4 – Version non décorée

```
$\vcalcdetplane[nodeco]{u}{x}{y}%
                       {v}{x'}{y'}$
```

$$\begin{vmatrix} \vec{u} & \vec{v} \\ x & x' \\ y & y' \end{vmatrix}$$

Exemple 5 – Sans les vecteurs

```
 $\vcalcdetplane[novec]%
                {u}{x}{y} {v}{x'}{y'}
= \vcalcdetplane[novec,cross]%
                {u}{x}{y} {v}{x'}{y'}$
```

$$\begin{vmatrix} x & x' \\ y & y' \end{vmatrix} = \begin{vmatrix} x & x' \\ y & y' \end{vmatrix}$$

Exemple 6 – Calcul développé

Grâce à l'argument optionnel de `\calcdetplane` ou de `\vcalcdetplane` il est aussi possible d'obtenir le résultat développé du calcul comme ci-après où `exp` est pour `exp`-and soit *« développer »* en anglais, `c` pour `\cdot` et enfin `t` pour `\times`[6].

```
$\vcalcdetplane[exp]{u}{x}{y}%
                    {v}{x'}{y'}$

$\vcalcdetplane[cexp]{u}{x}{y}%
                     {v}{x'}{y'}$

$\vcalcdetplane[texp]{u}{x}{y}%
                     {v}{x'}{y'}$
```

$$x\,y' - x'\,y$$
$$x \cdot y' - x' \cdot y$$
$$x \times y' - x' \times y$$

Attention ! Le développement effectué est stupide. Autrement dit ce sera à vous d'ajouter des parenthèses là où il y en aura besoin sinon vous obtiendrez des horreurs comme celle qui suit.

```
$\vcalcdetplane[exp]{AB}%
                    {x_B - x_A}%
                    {y_B - y_A}%
                    {CD}%
                    {x_D - x_C}%
                    {y_D - y_C}$
```

$$x_B - x_A\,y_D - y_C - x_D - x_C\,y_B - y_A$$

Ici nous n'avons pas d'autre choix que de régler le problème à la main. Ce genre de situation n'est pas rare dans la vraie vie mathématique.

```
$\vcalcdetplane[exp]{AB}%
                    {(x_B - x_A)}%
                    {(y_B - y_A)}%
                    {CD}%
                    {(x_D - x_C)}%
                    {(y_D - y_C)}$
```

$$(x_B - x_A)\,(y_D - y_C) - (x_D - x_C)\,(y_B - y_A)$$

6. Même si les vecteurs ne sont pas utilisés pour la mise en forme, on obtient ici une méthode très pratique à l'usage car elle permet de faire des copier-coller.

V. Géométrie cartésienne

1. Coordonnées

Exemple 1 – Des coordonnées seules

`tnsgeo` propose, via un argument optionnel, six façons différentes de rédiger des coordonnées seules *(nous verrons après des macros pour les coordonnées d'un point et celles d'un vecteur afin de produire un code LaTeX plus sémantique)*. Commençons par les écritures horizontales où vous noterez l'utilisation de `|` pour séparer les coordonnées dont le nombre peut être quelconque.

```
$\coord    {\dfrac{1}{3} | -4 | 0}$ ou
$\coord[sp]{\dfrac{1}{3} | -4 | 0}$

$\coord[b] {\dfrac{1}{3} | -4 | 0}$ ou
$\coord[sb]{\dfrac{1}{3} | -4 | 0}$
```

$\left(\dfrac{1}{3}\,;-4\,;0\right)$ ou $(\dfrac{1}{3}\,;-4\,;0)$
$\left[\dfrac{1}{3}\,;-4\,;0\right]$ ou $[\dfrac{1}{3}\,;-4\,;0]$

Il existe en plus deux versions verticales.

```
$\coord[vp]{3 | -4}$ ou
$\coord[vb]{3 | -4}$
```

$\begin{pmatrix}3\\-4\end{pmatrix}$ ou $\begin{bmatrix}3\\-4\end{bmatrix}$

Voici d'où viennent les noms des options.

1. `p`, qui est aussi la valeur par défaut, vient de `p`-arenthèses.
2. `b` vient de `b`-rackets soit *« crochets »* en anglais.
3. `s` pour `s`-mall soit *« petit »* en anglais permet d'avoir des délimiteurs non extensibles en mode horizontal car par défaut ils le sont.
4. `v` pour `v`-ertical demande de produire une écriture verticale.

Exemple 2 – Coordonnées d'un point

La macro `\pcoord` avec `p` pour `p`-oint prend un argument supplémentaire avant les coordonnées qui est le nom d'un point qui sera mis en forme par la macro `\pt`. Si vous ne souhaitez pas que `\pt` soit appliquée, il suffit de passer via la version étoilée `\pcoord*`.

```
$\pcoord{A}{3 | -4 | 0 | -1}$ ou
$\pcoord*{\Sigma}{7 | 9 | 8}$
```

$\mathrm{A}\,(\,3\,;-4\,;0\,;-1\,)$ ou $\Sigma\,(\,7\,;9\,;8\,)$

Toutes les options disponibles avec `\coord` le sont aussi avec `\pcoord`.

```
$\pcoord[b]{A}{3 | -4 | 0 | -1}$ ou
$\pcoord*[b]{\Sigma}{7 | 9 | 8}$
```

$\mathrm{A}\,[\,3\,;-4\,;0\,;-1\,]$ ou $\Sigma\,[\,7\,;9\,;8\,]$

Exemple 3 – Coordonnées d'un vecteur

Le fonctionnement de `\vcoord` est similaire à celui de `\pcoord` si ce n'est que c'est la macro `\vect` qui sera appliquée si besoin.

```
$\vcoord{u}{3 | -4}$ ou
$\vcoord*{\dfrac{1}{2} \vect{u}}%
         {3 | -4}$

$\vcoord[vp]{u}{3 | -4}$ ou
$\vcoord*[vp]{\dfrac{1}{2} \vect{u}}%
             {3 | -4}$
```

$\vec{u}\,(3\,;-4)$ ou $\frac{1}{2}\vec{u}\,(3\,;-4)$
$\vec{u}\begin{pmatrix}3\\-4\end{pmatrix}$ ou $\frac{1}{2}\vec{u}\begin{pmatrix}3\\-4\end{pmatrix}$

2. Nommer un repère

Exemple 1 – La méthode basique

Commençons par la manière la plus basique d'écrire un repère *(nous verrons d'autres méthodes qui peuvent être plus efficaces)*.

```
$\axes{\pt{O} %
     | \pt{I} | \pt{J}}$
```

$(\mathrm{O}\,;\mathrm{I},\mathrm{J})$

Exemple 2 – La méthode basique en version étoilée

Dans l'exemple ci-dessous, on voit que la version étoilée produit des petites parenthèses.

```
$\axes{\pt{O} %
     | \dfrac{7}{3} \vect{i} %
     | \vect{j}}$
ou
$\axes*{\pt{O} %
     | \dfrac{7}{3} \vect{i} %
     | \vect{j}}$
```

$\left(\mathrm{O}\,;\frac{7}{3}\vec{\imath},\vec{\jmath}\right)$ ou $\left(\mathrm{O}\,;\frac{7}{3}\vec{\imath},\vec{\jmath}\,\right)$

Exemple 3 – La méthode basique en dimension quelconque

Il faut au minimum deux "morceaux" séparés par des barres `|`, cas de la dimension 1, mais il n'y a pas de maximum, cas d'une dimension quelconque $n > 0$.

```
$\axes{\pt{O} %
     | \vect*{i}{1} %
     | \vect*{i}{2} %
     | \vect*{i}{3} %
     | \dots %
     | \vect*{i}{9} %
     | \vect*{i}{10} %
     | \vect*{i}{11} %
     | \vect*{i}{12}}$
```

$(\mathrm{O}\,;\vec{\imath}_1,\vec{\imath}_2,\vec{\imath}_3,\ldots,\vec{\imath}_9,\vec{\imath}_{10},\vec{\imath}_{11},\vec{\imath}_{12})$

Exemple 4 – Repère affine

Dans l'exemple suivant, le préfixe `p` est pour p-oint.

```
$\paxes{O | I | J | K}$
au lieu de
$\axes{\pt{O} %
     | \pt{I} | \pt{J} | \pt{K}}$
```

$(\mathrm{O}\,;\mathrm{I},\mathrm{J},\mathrm{K})$ au lieu de $(\mathrm{O}\,;\mathrm{I},\mathrm{J},\mathrm{K})$

Exemple 5 – Repère vectoriel (méthode 1)

Dans l'exemple suivant, le préfixe `v` est pour `v`-ecteur.

```
$\vaxes{\pt{O} | i | j}$
au lieu de
$\axes{\pt{O} | \vect{i} | \vect{j}}$
```

$(\mathrm{O}\,;\vec{\imath},\vec{\jmath})$ au lieu de $(\mathrm{O}\,;\vec{\imath},\vec{\jmath})$

Exemple 6 – Repère vectoriel (méthode 2)

Dans l'exemple suivant, le préfixe `pv` permet de combiner ensemble les fonctionnalités proposées par les préfixes `p` et `v`.

```
$\pvaxes{O | i | j}$
au lieu de
$\axes{\pt{O} | \vect{i} | \vect{j}}$
```

$(\mathrm{O}\,;\vec{\imath},\vec{\jmath})$ au lieu de $(\mathrm{O}\,;\vec{\imath},\vec{\jmath})$

VI. Arcs circulaires

Exemple 1

L'arc extensible utilise une macro que l'on trouve dans le package `yhmath`.

```
$\circarc{UnNomLong}$ ,
$\circarc*{A}{rot}$ ou
$\circarc{A_{rot}}$
```

$\overset{\frown}{UnNomLong}$, $\widehat{A}_{rot}$ ou $\overset{\frown}{A_{rot}}$

Exemple 2

```
$\circarc{i}$ ou
$\circarc*{j}{2}$
```

$\widehat{\imath}$ ou $\widehat{\jmath}_2$

VII. Angles

1. Angles géométriques « intérieurs »

Exemple 1

Le chapeau extensible utilise une macro que l'on trouve dans le package `yhmath`.

```
$\anglein{UnNomLong}$ ,
$\anglein*{A}{rot}$ ou
$\anglein{A_{rot}}$
```

$\overset{\frown}{UnNomLong}$, $\widehat{A}_{rot}$ ou $\overset{\frown}{A_{rot}}$

Exemple 2 – Cacher les points du i et du j

```
$\anglein{i}$ ou
$\anglein*{j}{2}$
```

$\widehat{i}$ ou $\widehat{j_2}$

2. Angles orientés de vecteurs

Sans chapeau - Version longue

L'option par défaut est `p` pour `p`-arenthèse. Dans `sp` le `s` est pour `s`-mall soit *« petit »* en anglais.

```
$\angleorient    {\dfrac{1}{2} \vect{i}}%
                 {\vect{j}}$

$\angleorient[sp]{\dfrac{1}{2} \vect{i}}%
                 {\vect{j}}$
```

$\left(\frac{1}{2}\vec{i}\,;\vec{j}\right)$
$(\frac{1}{2}\vec{i}\,;\vec{j}\,)$

Sans chapeau - Version courte mais restrictive

Dans l'exemple suivant, le préfixe `v` est pour `v`-ecteur qui permet de simplifier la saisie quand l'on a juste des vecteurs nommés avec des lettres *(notez que l'option* ***sp*** *n'apporte rien de nouveau)*.

```
$\vangleorient    {i}{j}$ comme
$\vangleorient[sp]{i}{j}$
```

$(\vec{i}\,;\vec{j}\,)$ comme $(\vec{i}\,;\vec{j}\,)$

Avec un chapeau

Dans l'exemple suivant, `h` est pour `h`-at soit *« chapeau »* en anglais. Notez au passage que `sh` produit juste des parenthèses petites mais ce choix de nom simplifie l'utilisation de la macro *(c'est mieux que* ***hsp*** *par exemple)*.

```
$\angleorient[h] {\dfrac{1}{2} \vect{i}}%
                 {\vect{j}}$

$\angleorient[sh]{\dfrac{1}{2} \vect{i}}%
                 {\vect{j}}$

$\vangleorient[h] {i}{j}$ comme
$\vangleorient[sh]{i}{j}$
```

$\widehat{\left(\frac{1}{2}\vec{i}\,;\vec{j}\right)}$
$\widehat{(\frac{1}{2}\vec{i}\,;\vec{j}\,)}$
$\widehat{(\vec{i}\,;\vec{j}\,)}$ comme $\widehat{(\vec{i}\,;\vec{j}\,)}$

Chapitre 5

Analyse

I. Constantes et paramètres

1. Constantes classiques ajoutées

Exemple 1

Les macros ci-dessous ne nécessitent pas d'accolades.

```
$\ggamma$ , $\ppi$ , $\ttau$ ,
$\ee$ , $\ii$ , $\jj$
et $\kk$ où $\ttau = 2 \ppi$
```

γ , π , τ , $\mathbf{e}$, $\mathbf{i}$, $\mathbf{j}$ et $\mathbf{k}$ où $\tau = 2\,\pi$

Remarque. Faites attention car `{\Large $\ppi \neq \pi$}` produit $\boldsymbol{\pi} \neq \pi$. Comme vous le constatez, les symboles ne sont pas identiques. Ceci est vraie pour toutes les constantes grecques.

Exemple 2 – Gestion de l'espace

Le choix a été fait d'ajouter de l'espace autour des constantes ajoutées.

```
$2 \ppi R$ ou $2 \pi R$
```

$2\,\pi\,R$ ou $2\pi R$

2. Constantes latines personnelles

La macro `\param` est surtout là pour une utilisation pédagogique. Noter au passage qu'ici aussi un espace est ajouté autour des paramètres.

```
$\param{a} x^2 + \param{b} x + \param{c}$
ou
$a x^2 + b x + c$
```

$\mathbf{a}\,x^2 + \mathbf{b}\,x + \mathbf{c}$ ou $ax^2 + bx + c$

II. Une variable « symbolique »

Le package `tnscom`, chargé en coulisse, propose la macro `\symvar` qui produit différents symboles donnés ci-dessous.

```
$\symvar = \symvar[1]$ ou $\symvar[2]$

$\symvar[3]$ ou $\symvar[4]$
```

$\bullet = \bullet$ ou $\circ$
$\blacksquare$ ou $\square$

Le nom `symvar` vient de `symb`-olic `var`-iable soit *« variable symbolique »* en anglais. Ces symboles sont utiles pour indiquer un argument symboliquement sans faire référence précisément à une ou des variables nommées.

III. La fonction valeur absolue

Exemple

```
$\abs{2}$ ,
$\abs {\dfrac{3}{5}}$ ou
$\abs*{\dfrac{3}{5}}$
```

$|2|$, $\left|\dfrac{3}{5}\right|$ ou $|\dfrac{3}{5}|$

Remarque. Le code LaTeX vient directement de ce poste : `https://tex.stackexchange.com/a/43009/6880`.

IV. Fonctions nommées spéciales

1. Sans paramètre

Quelques fonctions nommées ont été ajoutées en plus de celles fournies par le package `amsmath` qui est chargé en coulisse et propose par exemple $\lg x$ et $\ln x$ via les macros `\lg` et `\ln`. Ci-dessous le `f` dans `fch` est pour `f`-rench soit *« français »* en anglais *(ce choix sert à éviter des incompatibilités avec d'autres packages)*. La liste complète des fonctions nommées est donnée plus bas dans la section 3.

```
$\fch x = \cosh x$ ou $\acos x$
```

$\operatorname{ch} x = \cosh x$ ou $\operatorname{acos} x$

2. Avec un paramètre

Voici des fonctions utilisables avec un paramètre optionnel.

```
$\exp[6] y = 6^y$   ,
$\log[2] x = \lg x$ ,
$\lg[3] x$ et $\ln[4] x$
```

$\exp_6 y = 6^y$, $\log_2 x = \lg x$, $\lg_3 x$ et $\ln_4 x$

3. Toutes les fonctions nommées en plus

`\acos x` : $\operatorname{acos} x$	`\asin x` : $\operatorname{asin} x$	`\atan x` : $\operatorname{atan} x$
`\arccosh x` : $\operatorname{arccosh} x$	`\arcsinh x` : $\operatorname{arcsinh} x$	`\arctanh x` : $\operatorname{arctanh} x$
`\acosh x` : $\operatorname{acosh} x$	`\asinh x` : $\operatorname{asinh} x$	`\atanh x` : $\operatorname{atanh} x$
`\fch x` : $\operatorname{ch} x$	`\fsh x` : $\operatorname{sh} x$	`\fth x` : $\operatorname{th} x$
`\afch x` : $\operatorname{ach} x$	`\afsh x` : $\operatorname{ash} x$	`\afth x` : $\operatorname{ath} x$
`\exp x` : $\exp x$	`\exp[p] x` : $\exp_p x$	
`\lg x` : $\lg x$	`\lg[p] x` : $\lg_p x$	

`\ln x`	: $\ln x$	`\ln[p] x`	: $\ln_p x$
`\log x`	: $\log x$	`\log[p] x`	: $\log_p x$

V. Limite

Exemple 1 – Cas minimaliste

Notez ci-dessous que le rendu de la limite via `\limit` se fait toujours en mode `\displaystyle` pour l'opérateur de limite.

```
$\limit{f(x)}{x}{0} = \frac{1}{2}$
```

$$\lim_{x \to 0} f(x) = \frac{1}{2}$$

Remarque. `\lim\limits_{x \rightarrow 0} f(x)` produit $\lim\limits_{x \to 0} f(x)$ contre $\lim_{x \to 0} f(x)$ ci-dessus. La version `\limit` utilise un petit peu plus d'espace sous le mot lim.

Exemple 2 – Avec des conditions

Ce 2e exemple devrait rendre intéressante la macro `\limit` qui permet d'ajouter facilement plusieurs conditions [1] comme le montre la 2e limite ci-après.

```
$\limit{f(x)}{x}{0 | x > 0}$ ou
$\limit{f(z)}{z}%
      {0 | z \in \Omega | \abs{z} < 3}$
```

$$\lim_{\substack{x \to 0 \\ x > 0}} f(x) \text{ ou } \lim_{\substack{z \to 0 \\ z \in \Omega \\ |z| < 3}} f(z)$$

Exemple 3 – Ajout de parenthèses

Les options `p` et `sp` permettent d'ajouter facilement des parenthèses extensibles ou non autour de la fonction. Indiquons que `sp` est pour `s`-mall `p`-arenthesis soit *« petites parenthèses »* en anglais.

```
$\limit[p] {\dfrac{1}{x}}{x}%
           {0 | x > 0}$
ou
$\limit[sp]{\dfrac{1}{x}}{x}%
           {0 | x > 0}$
```

$$\lim_{\substack{x \to 0 \\ x > 0}} \left(\frac{1}{x}\right) \text{ ou } \lim_{\substack{x \to 0 \\ x > 0}} (\frac{1}{x})$$

VI. Calcul différentiel

1. Les opérateurs ∂ et d

Voici deux opérateurs utiles aussi bien pour du calcul différentiel que du calcul intégral.

```
$\dd{t} = \dd[1]{t}$ ou $\dd[n]{x}$

$\pp{t} = \pp[1]{t}$ ou $\pp[n]{x}$
```

$$\mathrm{d}t = \mathrm{d}^1 t \text{ ou } \mathrm{d}^n x$$
$$\partial t = \partial^1 t \text{ ou } \partial^n x$$

1. En pratique, on utilise généralement au maximum une condition.

2. Dérivations totales d'une fonction – Version longue avec une variable

Exemple 1 – Différentes écritures possibles

La macro `\der` est stricte du point de vue sémantique car on doit lui fournir la fonction, l'ordre de dérivation et la variable de dérivation *(voir la section 3. qui présente la macro `\sder` permettant une rédaction efficace pour obtenir $f^{(1)}$ ou f')*. Voici plusieurs mises en forme faciles à taper via l'option de `\der`. Attention bien entendu à n'utiliser l'option par défaut `u` ou l'option `d` qu'avec un ordre de dérivation de valeur naturelle connue !

```
 $\der   {f}{x}{3}
= \der[e]{f}{x}{3}
= \der[d]{f}{x}{3}$

 $\der[i] {u}{x}{k}
= \der[f] {u}{x}{k}
= \der[sf]{u}{x}{k}$
```

$$f'''=f^{(3)}=\dddot{f}$$
$$\mathrm{d}_x^k u=\frac{\mathrm{d}^k u}{\mathrm{d}x^k}=\frac{\mathrm{d}^k u}{\mathrm{d}x^k}$$

On peut aussi ajouter autour de la fonction des parenthèses extensibles ou non sauf même si cela n'est pas utile pour le mode `d` *(voir juste après le mode `bd`)*. Ci-dessous on montre au passage une écriture du type *« opérateur fonctionnel »* : voir la section 4. page 53 à ce sujet.

```
 $\der[osf,sp]{\frac{1}{2}  uv}{x}{k}
= \der[of,p]  {\dfrac{1}{2} uv}{x}{k}$
```

$$\frac{\mathrm{d}^k}{\mathrm{d}x^k}\left(\tfrac{1}{2}uv\right)=\frac{\mathrm{d}^k}{\mathrm{d}x^k}\left(\frac{1}{2}uv\right)$$

Avec l'option `d` les parenthèses seules sont sans utilité car on peut obtenir des choses non souhaitées comme $\ddot{(u+v)}$. À la place on utilisera l'option `bd` où le `b` est pour `b`-racket soit *« crochet »* en anglais. Notez au passage que `p` et `sp` restent utilisables.

```
 $\der[bd]   {\frac{1}{2}  uv}{x}{2}
= \der[bd,p] {\frac{1}{2}  uv}{x}{2}
= \der[bd,sp]{\dfrac{1}{2} uv}{x}{2}$
```

$$\ddot{\tfrac{1}{2}uv}=\ddot{\left(\frac{1}{2}uv\right)}=\ddot{(\frac{1}{2}uv)}$$

Remarque. Expliquons les valeurs des options.

1. `u`, la valeur par défaut, est pour `u`-suel soit l'écriture avec les primes. Cette option ne marchera pas avec un nombre symbolique de dérivations.
2. `e` est pour `e`-xposant.
3. `i` est pour `i`-ndice.
4. `d` est pour `d`-ot soit *« point »* en anglais.
5. `bd` est pour `b`-racket `d`-ot où *« bracket »* est pour *« crochet »* en anglais.
6. `f` est pour `f`-raction avec aussi `sf` pour une écriture réduite où `s` est pour `s`-mall soit *« petit »* en anglais.
7. `of` et `osf` utilisent le préfixe `o` pour `o`-pérateur.
8. `p` est pour `p`-arenthèse : dans ce cas les parenthèses seront extensibles. Le fonctionnement est différent avec l'option `d` comme nous l'avons vu avant.
9. `sp` est pour des parenthèses non extensibles. Là aussi le fonctionnement est différent avec l'option `d`.

Exemple 2 – Pas de uns inutiles

```
 $\der[i ]{u}{x}{1}
= \der[f ]{u}{x}{1}
= \der[sf]{u}{x}{1}
= \der[of]{u}{x}{1}$
```

$$\mathrm{d}_x u = \frac{\mathrm{d}u}{\mathrm{d}x} = \frac{\mathrm{d}u}{\mathrm{d}x} = \frac{\mathrm{d}}{\mathrm{d}x}u$$

Remarque. Voici comment forcer les exposants 1 si besoin. Fonctionnel mais très moche...

```
 $\der[i ]{u}{x}{\,\!1}
= \der[f ]{u}{x}{\,\!1}
= \der[sf]{u}{x}{\,\!1}
= \der[of]{u}{x}{\,\!1}$
```

$$\mathrm{d}^1_x u = \frac{\mathrm{d}^1 u}{\mathrm{d}x^1} = \frac{\mathrm{d}^1 u}{\mathrm{d}x^1} = \frac{\mathrm{d}^1}{\mathrm{d}x^1}u$$

3. Dérivations totales d'une fonction – Version courte sans variable

Dans l'exemple suivant le code manque de sémantique car on n'indique pas la variable de dérivation. Ceci étant dit à l'usage la macro `\sder` rend de grands services. Ici le préfixe `s` est pour `s`-imple voire `s`-impliste... Voici des exemples où de nouveau l'option par défaut `u` et l'option `d` ne seront fonctionnelles qu'avec un ordre de dérivation de valeur naturelle connue !

Exemple 1

```
 $\sder{f}{1} = \der{f}{x}{1}$

 $\sder   {f}{3}
= \sder[e]{f}{3}
= \sder[d]{f}{3}$
```

$$f' = f'$$
$$f''' = f^{(3)} = \dddot{f}$$

Exemple 2

```
 $\sder[sp] {\dfrac{1}{2} uv}{2}
= \sder[e,p]{\dfrac{1}{2} uv}{2}
= \sder[bd] {\dfrac{1}{2} uv}{2}$
```

$$(\frac{1}{2}uv)'' = \left(\frac{1}{2}uv\right)^{(2)} = \overbrace{\frac{1}{2}uv}^{..}$$

Remarque. Ici les seules options disponibles sont `u`, `e`, `b`, `bd`, `p` et `sp`.

4. L'opérateur de dérivation totale

Ce qui suit peut rendre service au niveau universitaire. Les options possibles sont `f`, valeur par défaut, `sf` et `i` avec les mêmes significations que pour la macro `\der`.

```
 $\derope    {x}{k}
= \derope[sf]{x}{k}
= \derope[i] {x}{k}$
```

$$\frac{\mathrm{d}^k}{\mathrm{d}x^k} = \frac{\mathrm{d}^k}{\mathrm{d}x^k} = \mathrm{d}^k_x$$

Ici non plus il n'y a pas de uns inutiles mais l'astuce `\,\!1` reste utilisable.

```
$\derope    {x}{1}
= \derope[sf]{x}{1}
= \derope[i] {x}{1}$
```

$$\frac{\mathrm{d}}{\mathrm{d}x} = \tfrac{\mathrm{d}}{\mathrm{d}x} = \mathrm{d}_x$$

5. Dérivations partielles

Exemple 1 – Différentes écritures

La macro `\pder` [2] avec `p` pour `p`-artielle permet de rédiger des dérivées partielles en utilisant facilement plusieurs mises en forme via une option qui vaut `f` par défaut. Cette macro attend une fonction, les dérivées partielles effectuées et l'ordre total de dérivation. Voici deux types de mise en forme classiques où vous noterez comment `x | y^2` est interprété.

```
$\pder    {f}{x | y^2}{3}
= \pder[sf]{f}{x | y^2}{3}$
```

$$\frac{\partial^3 f}{\partial x\,\partial y^2} = \tfrac{\partial^3 f}{\partial x\,\partial y^2}$$

Il existe en plus deux notations indicielles données en exemple ci-dessous. Notez qu'avec l'option `ei` l'exposant total n'est pas imprimé et que les exposants partiels doivent être des naturels connus.

```
$\pder[i] {f}{x | y^2}{3}
= \pder[ei]{f}{x | y^2}{3}$
```

$$\partial^3_{x,y(2)} f = f'_{x\,y\,y}$$

Les options `i` et `ei` marchent avec des variables indicées à condition de bien mettre les dites variables entre des accolades.

```
$\pder[i] {f}{x_1 | {x_2}^2}{3}
= \pder[ei]{f}{x_1 | {x_2}^2}{3}$
```

$$\partial^3_{x_1,x_2(2)} f = f'_{x_1\,x_2\,x_2}$$

On peut aussi ajouter autour de la fonction à différencier des parenthèses extensibles ou non via `p` et `sp` respectivement. Ci-dessous on montre aussi une écriture du type *« opérateur fonctionnel »* : voir la section 6. page 55 à ce sujet.

```
$\pder[of,sp] {u + v}{x | y^2}{}
= \pder[osf,sp]{u + v}{x | y^2}{}$

$\pder[i,sp]  {u + v}{x | y^2}{}
= \pder[ei,sp] {u + v}{x | y^2}{}$
```

$$\frac{\partial}{\partial x\,\partial y^2}(u+v) = \tfrac{\partial}{\partial x\,\partial y^2}(u+v)$$
$$\partial_{x,y(2)}(u+v) = (u+v)'_{x\,y\,y}$$

Remarque. Les options disponibles sont `f`, `sf`, `of`, `osf`, `p` et `sp` avec des significations similaires à celles pour la macro `\der` auxquelles s'ajoutent `i` et `ei` pour les écritures indicielles où le `e` dans `ei` est pour `e`-xpand soit *« développer »* en anglais.

Exemple 2 – Pas de uns inutiles

```
$\pder    {u}{x}{1}
= \pder[sf]{u}{x}{1}
= \pder[i] {u}{x}{1}$
```

$$\frac{\partial u}{\partial x} = \tfrac{\partial u}{\partial x} = \partial_x u$$

Remarque. Rappelons que pour obtenir $\partial^1_x u$ on peut taper `\pder[i]{u}{x}{\,\!1}`.

2. `\partial` existe déjà pour obtenir ∂.

6. L'opérateur de dérivation partielle

Ce qui suit peut rendre service au niveau universitaire. Les options possibles sont `f`, valeur par défaut, `sf` et `i` avec les mêmes significations que pour la macro `\pder`.

```
 $\pderope    {x | y^2}{3}
= \pderope[sf]{x | y^2}{3}
= \pderope[i] {x | y^2}{3}$
```

$$\frac{\partial^3}{\partial x\,\partial y^2} = \frac{\partial^3}{\partial x\,\partial y^2} = \partial^3_{x,y(2)}$$

VII. Tableaux de variation et de signe

1. Les bases de `tkz-tab`

Comment ça marche ?

Pour les tableaux de variation et de signe non décorés, tout le boulot est fait par le package `tkz-tab`. Ce package est utilisé avec les réglages par défaut *« maison »* suivants via l'utilisation de `\tkzTabSetup`.

1. `arrowstyle = triangle 60` permet d'obtenir des pointes de flèche plus visibles.
2. `doubledistance = 3pt` améliore la visibilité des doubles barres pour les valeurs interdites.

Nous donnons quelques exemples classiques d'utilisation proches ou identiques de certains proposés dans la documentation de `tkz-tab` *(les codes ont été mis en forme pour faciliter la compréhension de la syntaxe à suivre)*. Reportez vous à la documentation de `tkz-tab` pour des compléments d'information : vous y trouverez des réglages très fins.

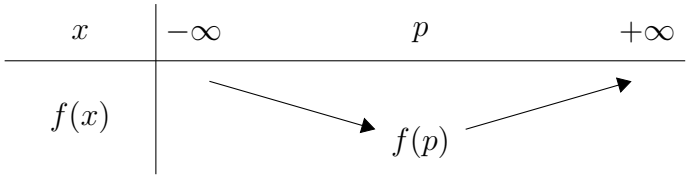
Exemple 1 – Avec des signes

x	0		$\frac{\pi}{2}$		π
$\cos(x)$		$+$	0	$-$	

Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit{
        $x$       / 0.75 , % Facteur d'échelle de 0.75 pour la hauteur de la 1er ligne.
        $\cos(x)$ / 1      % Facteur d'échelle de  1   pour la hauteur de la 2e ligne.
    }{
                $0$     , $\frac{\pi}{2}$     , $\pi$  % Valeurs utiles de x.
    }
    \tkzTabLine{    , + , z               , - ,      } % Signes et zéro.
\end{tikzpicture}
```

Exemple 2 – Avec des variations (sans cadre)



Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[nocadre]{
        $x$    / 0.75 ,
        $f(x)$ / 1.5
    }{
              $-\infty$ , $p$        , $+\infty$
    }
    \tkzTabVar{+ /       , - / $f(p)$ , + /      } % Variations via position / valeur.
\end{tikzpicture}
```

Exemple 3 – Variations via une dérivée (sans cadre)

x	0		$\frac{\pi}{2}$		π
$\cos(x)$		$+$	0	$-$	
$\sin(x)$	0	↗	1	↘	0

Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[nocadre]{
        $x$       / 0.75 ,
        $\cos(x)$ / 1    ,
        $\sin(x)$ / 1.5
    }{
              $0$       , $\frac{\pi}{2}$ ,     $\pi$
    }
    \tkzTabLine{      , + , z               , - ,      }
    \tkzTabVar {- / 0     , + / 1               , - / 0}
\end{tikzpicture}
```

Exemple 4 – Une image intermédiaire avec une seule flèche

x	$-\infty$		0		$+\infty$
$3x^2$		$+$	0	$+$	
x^3	$-\infty$	↗	0	↗	$+\infty$

Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit{
        $x$     / 0.75 ,
        $3 x^2$ / 1    ,
        $x^3$   / 1.5
```

```
    }{
                $-\infty$          , $0$      , $+\infty$
    }
    \tkzTabLine{                , + , 0    , + ,               }
    \tkzTabVar {- / $-\infty$      , R        , + / $+\infty$}
    %
    \tkzTabIma{1}{3}{2} % Position entre les 1re et 3e valeurs puis rang relatif.
               {$0$}     % Valeur de l'image.
\end{tikzpicture}
```

Exemple 5 – Valeurs interdites et valeurs supplémentaires

x	0	1	$\mathbf{e}$	$+\infty$
$\frac{1}{x}$	‖	$+$		
$\ln$	‖ $-\infty$	0	1	$+\infty$

Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[espcl = 6]{    % Largeur entre les valeurs du tableau.
        $x$             / 0.75 ,
        $\frac{1}{x}$ / 1.25 ,
        $\ln$           / 1.75
    }{
                $0$                  , $+\infty$
    }
    \tkzTabLine{d                , + ,                }
    \tkzTabVar {D- / $-\infty$      , + / $+\infty$}
    %
    \tkzTabVal{1}{2}{0.35} % Position entre les 1re et 2e valeurs puis en proportion.
               {1}{0}       % x_1 et f(x_1)
    \tkzTabVal{1}{2}{0.65} % Position entre les 1re et 2e valeurs puis en proportion.
               {$\ee$}{1}   % x_2 et f(x_2)
\end{tikzpicture}
```

Voici un autre exemple pour comprendre comment utiliser `\tkzTabVal` avec en plus l'option `draw` qui peut rendre service.

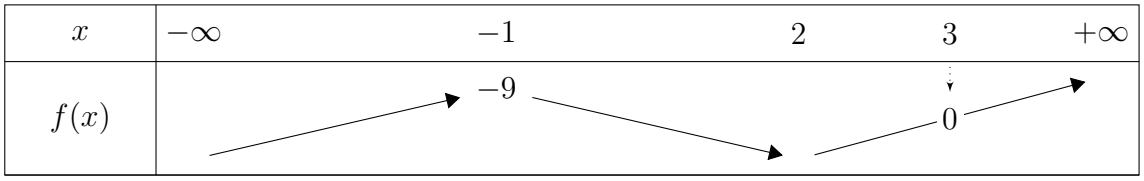
x	0	1	$\mathbf{e}$	$\mathbf{e}^2$	$+\infty$
$f'(x)$	‖	$+$	0	$-$	
$f(x)$	‖ $-\infty$	$\frac{1}{\mathbf{e}}$	$\mathbf{e}$	1	0

Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[espcl = 4]{
        $x$     / 0.75 ,
        $f'(x)$ / 1    ,
        $f(x)$  / 1.5
    }{
                $0$                 , $\ee$     , $+\infty$
    }
    \tkzTabLine{d               , + , 0         , - ,          }
    \tkzTabVar {D- / $-\infty$     , + / $\ee$ , - / $0$  }
    %
    \tkzTabVal[draw]{1}{2}{0.5} % Position entre les 1re et 2e valeurs au milieu.
                    {$1$}{$\frac{1}{\ee}$}
    \tkzTabVal[draw]{2}{3}{0.5} % Position entre les 2e et 3e valeurs au milieu.
                    {$\ee^2$}{$1$}
\end{tikzpicture}
```

Exemple 6 – Signe à partir des variations (un peu de pédagogie...)

Il est assez facile de produire des choses très utiles pédagogiquement comme ce qui suit en se salissant un peu les mains avec du code TikZ.

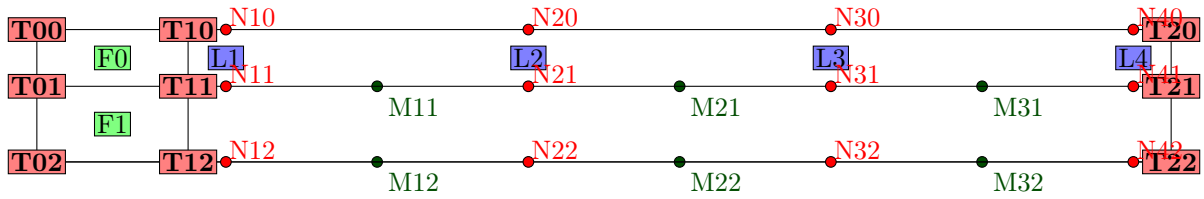


On déduit du tableau précédent le signe de la fonction f.

Voici le code du 1er tableau où le placement de la valeur 3 au milieu entre 2 et $+\infty$ va nous simplifier le travail pour le 2e tableau.

```
\begin{tikzpicture}
    \tkzTabInit[espcl = 4]{
        $x$    / 0.75 ,
        $f(x)$ / 1.5
    }{
               $-\infty$ , $-1$     , $2$ , $+\infty$
    }
    \tkzTabVar{- /       , + / $-9$ , - / , + /      }
    %
    \tkzTabVal[draw]{3}{4}{0.5} % Position entre les 3e et 4e valeurs au milieu.
               {$3$}{$0$}
\end{tikzpicture}
```

Pour produire le code du 2e tableau il a fallu utiliser au préalable ce qui suit en activant l'option `help` qui demande à `tkz-tab` d'afficher les noms de noeuds au sens TikZ qui ont été créés. Ceci permet alors d'utiliser ces noeuds pour des dessins TikZ faits maison [3].



Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[
        espcl = 4,
        help       % <-- Pour voir les noms des noeuds.
    ]{
        $x$    / 0.75 ,
        $f(x)$ / 1
    }{
       $-\infty$ , $-1$ , $2$ , $+\infty$
    }
\end{tikzpicture}
```

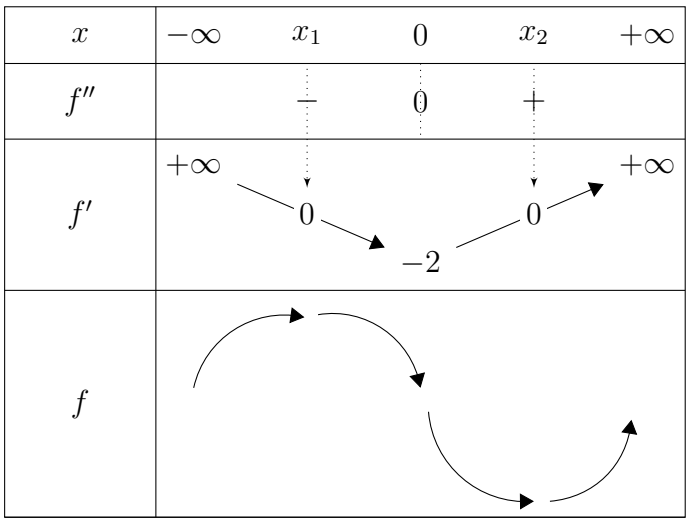
Maintenant que les noms des noeuds sont connus, il devient facile de produire le code ci-après. Bien noter l'usage de valeurs utiles « vides » de x ainsi que les mystiques `\node at ($(A)!0.5!(B)$)` permettant de placer un noeud au milieu entre les deux noeuds `A` et `B`.

```
\begin{tikzpicture}
    \tkzTabInit[
        espcl = 4,
%        help       % <-- Pour voir les noms des noeuds.
    ]{
        $x$    / 0.75 ,
        $f(x)$ / 1
    }{
        $-\infty$ , , , $+\infty$ % ATTENTION ! Ne pas oublier de virgules.
    }
    %
    \node at ($(N31)!0.5!(N40)$){$3$};
    \node at ($(M31)!0.5!(M32)$){$0$};
    \node at ($(M31)!0.5!(N42)$){$+$};
    \node at ($(N11)!0.5!(M32)$){$-$};
\end{tikzpicture}
```

Exemple 7 – Convexité et concavité symbolisées dans les variations

Voici un autre exemple s'utilisant la machinerie TikZ afin d'indiquer dans les variations la convexité et la concavité via des flèches incurvées *(cette convention est proposée dans la sous-section* « Exemple utilisant l'option `\help` » *de la section* « Gallerie » *de la documentation de* ***tkz-tab****)*.

3. C'est grâce à ce mécanisme que `tnsana` peut proposer des outils explicatifs des tableaux de signe : voir la section suivante.



Le rendu précédent a été obtenu via le code suivant.

```
\begin{tikzpicture}
    \tkzTabInit[
        espcl = 3,
%        help       % <-- Pour voir les noms des noeuds.
    ]{
        $x$   / 0.75 ,
        $f''$ / 1    ,
        $f'$  / 2    ,
        $f$   / 3
    }{
              $-\infty$          , $0$          , $+\infty$
    }
    \tkzTabLine{               , - , z        , + ,              }
    \tkzTabVar {+ / $+\infty$      , - / $-2$     , + / $+\infty$}

    \tkzTabVal[draw]{1}{2}{.5}{$x_1$}{$0$}
    \tkzTabVal[draw]{2}{3}{.5}{$x_2$}{$0$}

    \begin{scope}[->, > = triangle 60]
        \coordinate (Middle1) at ($(N13)!0.5!(N14)$);
        \coordinate (Middle2) at ($(N23)!0.5!(N24)$);
        \coordinate (Middle3) at ($(N33)!0.5!(N34)$);
        \path (N13) -- (N23) node[midway,below=6pt](N){};
        %
        \draw ([above=6pt]Middle1)
              to [bend left=45] ([left=1pt]N);
        \draw ([right=3pt]N)
              to [bend left=45] ([above=6pt]Middle2) ;
        \draw ([below right=3pt]Middle2)
              to [bend left=-45] ([above=6pt]M24) ;
        \draw ([above right=6pt]M24)
              to [bend right=40] ([below left=6pt]Middle3);
    \end{scope}
\end{tikzpicture}
```

2. Décorer facilement un tableau

Motivation

Considérons le tableau suivant et imaginons que nous voulions l'expliquer à un débutant.

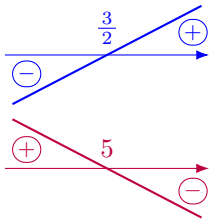
x	$-\infty$		$\frac{3}{2}$		5		$+\infty$
Signe de $2x-3$		$-$	0	$+$		$+$	
Signe de $-x+5$		$+$		$+$	0	$-$	
Signe de $(2x-3)(-x+5)$		$-$	0	$+$	0	$-$	

Deux options s'offrent à nous pour justifier comment a été rempli le tableau.

1. Classiquement on résout par exemple juste les deux inéquations $2x-3>0$ et $-x+5>0$ puis on complète les deux premières lignes [4] pour en déduire la dernière via la règle des signes d'un produit.
2. On peut proposer une méthode moins sujette à la critique qui s'appuie sur la représentation graphique d'une fonction affine en produisant le tableau suivant.

x	$-\infty$		$\frac{3}{2}$		5		$+\infty$
Signe de $2x-3$		$-$	0	$+$		$+$	
Signe de $-x+5$		$+$		$+$	0	$-$	
Signe de $(2x-3)(-x+5)$		$-$	0	$+$	0	$-$	

← Valeurs utiles de x



← Signe du produit.

Pour produire le 2e tableau, en plus du code `tkz-tab` pour le tableau de signe [5], il a fallu ajouter les lignes données ci-dessous où sont utilisées les macros `\backLine`, `\graphSign` et `\comLine` proposées par `tnsana` *(la syntaxe simple à suivre sera expliquée dans les trois sections suivantes)*. Indiquons que les lignes pour les signes doivent utiliser un coefficient minimal de `1.5` pour la hauteur afin d'éviter la superposition des graphiques.

```
\begin{tikzpicture}
    % ------------------------------------------------------ %
    % -- Code tkz-tab pour les signes non reproduit ici -- %
    % ------------------------------------------------------ %
```

4. Notons que cette approche est un peu scandaleuse car il faudrait en toute rigueur aussi résoudre $2x-3<0$, $-x+5<0$, $2x-3=0$ et $-x+5=0$. Personne ne le fait car l'on pense aux variations d'une fonction affine. Dans ce cas pourquoi ne pas juste utiliser ce dernier argument ? C'est ce que propose la 2e méthode.

5. Nous avons utilisé les réglages optionnels `lgt = 3.5` et `espcl = 2.5` de `\tkzTabInit` pour avoir de la place dans la 1re colonne pour le dernier produit et aussi réduire la largeur des colonnes pour les signes.

```
    \backLine{0, 3}

    \comLine[gray]{0}{$\leftarrow$ Valeurs utiles de $x$}

    \graphSign        {1}{ax+b, ap}{$\frac{3}{2}$}
    \graphSign[purple]{2}{ax+b, an}{$5$}

    \comLine[gray]{3}{$\leftarrow$ Signe du produit.}
\end{tikzpicture}
```

Remarque. Il est aussi possible de décorer une ligne de variation comme cela sera montré dans l'exemple 3 page 66.

`\backLine` – Ajouter une couleur de fond à une ligne

La modification de la couleur de fond d'une ligne se fait via la macro `\backLine`[6] pour `back`-ground `of the line` soit *« fond de la ligne »* en anglais. Cette macro possède un argument optionnel et un obligatoire.

- *L'argument optionnel : choix de la couleur de fond.*
  Ci-dessus nous avons utilisé la couleur par défaut qui est `gray!30`.
- *L'argument obligatoire : les numéros de ligne séparés par des virgules.*
  La numérotation des lignes commence à 0 comme en informatique. Ainsi `\backLine{0,3}` ajoute une couleur de fond à la ligne des valeurs utiles de x et à la 3^{e} ligne de signes, ou moins intuitivement à la $(3+1)^{\text{e}}$ ligne du tableau.

`\comLine` – Commenter une ligne

L'ajout de commentaires courts se fait via la macro `\comLine` pour `com`-ment a `line` soit *« commenter une ligne »* en anglais. Cette macro possède un argument optionnel et deux obligatoires.

- *L'argument optionnel : choix de la couleur du texte.*
  Ci-dessus nous avons utilisé `\comLine[gray]{0}{...}` pour avoir un texte en gris.
- *Le 1^{er} argument : le numéro de ligne* (comme pour `\backLine`).
- *Le 2^{e} argument : texte du commentaire.*
  Par défaut aucun retour à la ligne n'est possible. Si besoin se reporter à l'exemple 3, page 66 section 2., où est montré comment écrire sur plusieurs lignes.

`\graphSign` – Graphiques pour expliquer des signes

Pour le moment, la macro `\graphSign` propose différents types de graphiques de fonctions dites de référence. Avant de voir ce qui est proposé rappelons que la convention est de prendre 0 pour numéro de la toute 1^{re} ligne contenant les valeurs utiles de la variable.

1. ***Fonctions affines*** $ax+b$**.**
   Pour les fonctions du type $f(x) = ax + b$ avec $a \neq 0$, nous devons connaître le signe de a et la racine r de f. Le codage est simple : considérons par exemple `\graphSign{2}{ax+b, an}{$5$}`.

6. L'auteur de `tnsana` n'est absolument pas un fan de la casse en bosses de chameau mais par souci de cohérence avec ce que propose `tkz-tab` le nom `\backLine` a été proposé à la place de `\backline`.

- *1er argument* 2.

  Ceci indique d'ajouter le graphique dans la 2e ligne de signes.
- `ax+b` *dans le 2e argument.*

  Ce code sans espace indique une fonction affine.
- `an` *dans le 2e argument.*

  Ce code venant de `a négatif` ajoute la condition $a < 0$.
- *3e argument* 5.

  Ceci donne la racine.

Donc pour ajouter dans la 4e ligne de signe le graphique de $f(x) = 3x$, on utilisera dans ce cas `\graphSign{4}{ax+b, ap}{$0$}` où `ap` pour `a positif` code la condition $a > 0$.

2. ***Fonctions trinômiales du 2e degré*** $ax^2 + bx + c$**.**

   Pour les fonctions du type $f(x) = ax^2 + bx + c$ avec $a \neq 0$, en plus du signe de a nous devons connaître celui du discriminant $\Delta = b^2 - 4ac$, ce dernier pouvant être nul, sans oublier les racines réelles éventuelles. Voyons comment coder ce genre de chose via par exemple `\graphSign{5}{ax2+bx+c, an, dp}{$r_1$}{$r_2$}`.

   - *1er argument* 5.

     On indique la 5e ligne de signes.
   - `ax2+bx+c` *dans le 2e argument.*

     Ce code sans espace indique un trinôme du 2e degré.
   - `an` *dans le 2e argument* (comme avant).
   - `dp` *dans le 2e argument.*

     Ce code venant de `discriminant positif` ajoute la condition $\Delta > 0$. En plus de `dn` et `dp` il y a aussi `dz` pour `discriminant zéro`.
   - *3e et 4e arguments* r_1 *et* r_2.

     Ceci donne les deux racines réelles avec obligatoirement $r_1 < r_2$.

   Ainsi pour indiquer dans la 3e ligne de signe la courbe relative à $f(x) = -4x^2$, on utilisera `\graphSign{3}{ax2+bx+c, an, dz}{$0$}`.

   Enfin le graphique associé au trinôme $f(x) = 7x^2 + 3$, qui est sans racine réelle, s'obtiendra dans la 4e ligne de signe via `\graphSign{4}{ax2+bx+c, ap, dn}`.

3. ***Fonctions de référence.***

   Voici ce qui est disponible via `\graphSign{noligne}{codefonc}` où les valeurs possibles de `codefonc` sont les suivantes.

`codefonc`	`x2`	`sqrt`	`1/x`	`abs`	`exp`	`ln`
$f(x)$	x^2	$\sqrt{x}$	$\frac{1}{x}$	$\lvert x\rvert$	$\exp x$	$\ln x$

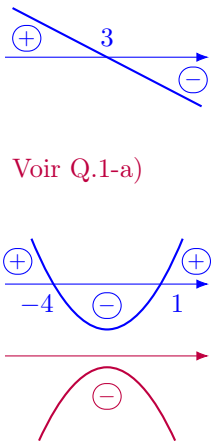
Quelques exemples

Exemple 1 – Avec une parabole

Il devient très facile de proposer un tableau décoré comme le suivant.

x	$-\infty$		-4		1		3		$+\infty$
Signe de $-x+3$		$+$		$+$		$+$	0	$-$	
Signe de $f(x)$		$-$	0	$+$	0	$+$	0	$-$	
Signe de x^2+3x-4		$+$	0	$-$	0	$+$		$+$	
Signe de $-x^2+x-4$		$-$		$-$		$-$		$-$	
Signe du produit		$+$	0	$+$	0	$-$	0	$-$	

Schémas



Voir Q.1-a)

← Conclusion

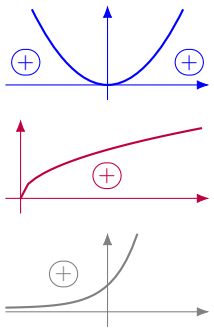
En plus des deux exemples de schémas de paraboles, il faut noter dans le code supplémentaire ajouté l'utilisation de `\kern1.75em` dans `\comLine[gray]{0}{\kern1.75em Schémas}` afin de mettre un espace horizontal précis pour centrer à la main le texte *« Schémas » (un peu sâle mais ça marche)*.

```
\begin{tikzpicture}
    % ------------------------------------------------- %
    % -- Code tkz-tab pour les signes non reproduit ici -- %
    % ------------------------------------------------- %

    \backLine{0, 5}

    \comLine[gray]{0}{\kern1.75em Schémas}

    \graphSign         {1}{ax+b, an}{$3$}
    \comLine[purple]   {2}{Voir Q.1-a)}
    \graphSign         {3}{ax2+bx+c, ap, dp}{$-4$}{$1$} % Deux racines réelles.
    \graphSign[purple]{4}{ax2+bx+c, an, dn}             % Aucune racine réelle.

    \comLine[gray]{5}{$\leftarrow$ Conclusion}
\end{tikzpicture}
```

Exemple 2 – Avec des fonctions sans paramètre

Voici un 1er tableau avec certaines des fonctions sans paramètre.

x	0	$+\infty$
x^2	0 $+$	
$\sqrt{x}$	0 $+$	
$\exp x$	$+$	
$x^2\sqrt{x}\exp x$	0 $+$	

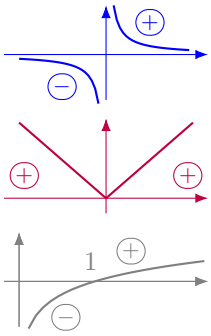


Le code correspondant est le suivant.

```
\begin{tikzpicture}
    % ------------------------------------------------------ %
    % -- Code tkz-tab pour les signes non reproduit ici -- %
    % ------------------------------------------------------ %

    \graphSign         {1}{x2}
    \graphSign[purple]{2}{sqrt}
    \graphSign[gray]   {3}{exp}
\end{tikzpicture}
```

Voici un 2e tableau avec les fonctions sans paramètre manquantes ci-dessus.

x	0	1	$+\infty$
$\frac{1}{x}$	$+$	$+$	
$\lvert x\rvert$	0 $+$	$+$	
$\ln x$	$-$	0 $+$	
$\frac{\lvert x\rvert}{x\ln x}$	$-$	$+$	



Le code correspondant est le suivant.

```
\begin{tikzpicture}
    % ------------------------------------------------------ %
    % -- Code tkz-tab pour les signes non reproduit ici -- %
    % ------------------------------------------------------ %

    \graphSign         {1}{1/x}
```

```
    \graphSign[purple]{2}{abs}
    \graphSign[gray]  {3}{ln}
\end{tikzpicture}
```

Exemple 3 – Commenter des variations

Pour finir, indiquons que les outils de décoration marchent aussi pour les tableaux de variation. Voici un exemple possible d'utilisation où les retours à la ligne ont été obtenus affreusement, ou pas, via `\parbox{11.5em}{...}`.

x	0		$\mathbf{e}$		$+\infty$
$f'(x)$	‖	$+$	0	$-$	
$f(x)$	‖	$-\infty$ ↗	$\mathbf{e}$	↘	0

Sur un intervalle le signe de $f'(x)$ implique les variations de $f(x)$.

Les limites sont hors programme pour cette année.

VIII. Calcul intégral

1. Le symbole standard revisité

Commençons par un point important : le package réduit les espacements entres des symboles $\int$ successifs. Voici un exemple.

```
$\displaystyle
 \int \int \int
 F(x;y;z) \dd{x} \dd{y} \dd{z}$

$\displaystyle
 \int_{a}^{b} \int_{c}^{d} \int_{e}^{f}
 F(x;y;z) \dd{x} \dd{y} \dd{z}$
```

$$\displaystyle \int\int\int F(x\,;y\,;z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$$
$$\displaystyle \int_a^b\int_c^d\int_e^f F(x\,;y\,;z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$$

Remarque. Par défaut, LaTeX affiche $\int\int\int F(x\,;y\,;z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$ et $\int_a^b\int_c^d\int_e^f F(x\,;y\,;z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z$. Nous avons obtenu ce résultat en utilisant `\stdint` qui est l'opérateur proposé de façon standard par LaTeX.

2. Un opérateur d'intégration clés en main

Exemple 1 – À quoi bon ?

Le 1er exemple qui suit semblera être une hérésie pour les habitués de LaTeX mais rappelons que le but de `tnsana` est de rendre les documents facilement modifiables globalement ou localement comme le montre le 2e exemple.

```
 $\displaystyle
  \integrate{f(x)}{x}{a}{b}
= \int_{x=a}^{x=b} f(x) \dd{x}$

 $\displaystyle
  \integrate*{f(x)}{x}{a}{b}
= \integrate{f(x)}{x}{a}{b}$
```

$$\int_{x=a}^{x=b} f(x)\,\mathrm{d}x = \int_{x=a}^{x=b} f(x)\,\mathrm{d}x$$
$$\int_{a}^{b} f(x)\,\mathrm{d}x = \int_{x=a}^{x=b} f(x)\,\mathrm{d}x$$

Exemple 2 – Le mode `displaystyle`

La macro `\dintegrate*` présentée ci-dessous possède aussi une version non étoilée `\dintegrate`.

```
 $\dintegrate*{f(x)}{x}{a}{b}
= \integrate*{f(x)}{x}{a}{b}$
```

$$\int_{a}^{b} f(x)\,\mathrm{d}x = \int_{a}^{b} f(x)\,\mathrm{d}x$$

3. L'opérateur « crochet »

Exemple 1

```
 $\hook{F(x)}{x}{a}{b}
= F(b) - F(a)$

 $\dintegrate*{f(x)}{x}{a}{b}
= \hook*{F(x)}{x}{a}{b}$
```

$$\big[\,F(x)\,\big]_{x=a}^{x=b} = F(b) - F(a)$$
$$\int_{a}^{b} f(x)\,\mathrm{d}x = \big[\,F(x)\,\big]_{a}^{b}$$

Remarque. Il faut savoir que `\hook` signifie *« crochet »* en anglais mais la bonne traduction du terme mathématique est en fait *« square bracket »*. Ceci étant dit l'auteur de `tnsana` trouve plus efficace d'utiliser `\hook` comme nom de macro.

Exemple 2 – Des crochets non extensibles

Dans l'exemple suivant, on utilise l'option `sb` pour `s`-mall `b`-rackets soit *« petits crochets »* en anglais. Les options sont disponibles à la fois pour `\hook` et `\hook*`.

```
 $\hook*{\dfrac{x - 1}{5 + x^2}}{x}%
       {a}{b}
= \hook*[sb]%
       {\dfrac{x - 1}{5 + x^2}}{x}%
       {a}{b}$
```

$$\left[\,\frac{x-1}{5+x^2}\,\right]_{a}^{b} = \big[\,\frac{x-1}{5+x^2}\,\big]_{a}^{b}$$

Exemple 3 – Un trait vertical épuré

Via les options `r` et `sr` pour `s`-mall et `r`-ull soit *« petit »* et *« trait »* en anglais, on obtient ce qui suit.

```
 $\hook[r]  {\dfrac{x - 1}{5 + x^2}}{x}%
           {a}{b}
= \hook*[sr]{\dfrac{x - 1}{5 + x^2}}{x}%
           {a}{b}$
```

$$\frac{x-1}{5+x^2}\,\Big|_{x=a}^{x=b} = \frac{x-1}{5+x^2}\,\big|_{a}^{b}$$

Chapitre 6

Suites

I. Des notations complémentaires pour des suites spéciales

Voici trois types de suites avec deux ou quatre indices.

```
$\seqplus{F}{1}{2}$

$\seqhypergeo{F}{1}{2}$

$\seqsuprageo{F}{1}{2}{3}{4}$
pour les fous\dots :-)
```

${}_{1}^{}F_{2}^{2}$
${}_{1}^{}F_{2}$
${}_{1}^{4}F_{2}^{3}$ pour les fous… :-)

II. Sommes et produits en mode ligne

Pour limiter l'espace, LaTeX affiche $\sum_{k=0}^{n}$ et non $\displaystyle\sum_{k=0}^{n}$ sauf si l'on utilise la commande `\displaystyle`. Les macros `\dsum` et `\dprod` permettent de se passer de `\displaystyle`. Voici un exemple.

```
 $\dsum_{k=0}^{n} 2^k
= \sum_{k=0}^{n} 2^k$

 $\dprod_{k=1}^{n} k
= \prod_{k=1}^{n} k$
```

$\displaystyle\sum_{k=0}^{n} 2^k = \sum_{k=0}^{n} 2^k$
$\displaystyle\prod_{k=1}^{n} k = \prod_{k=1}^{n} k$

Remarque. On peut taper $\displaystyle\sum_{k=0}^{n} \frac{1}{k}$ où la fraction n'est pas en mode `\displaystyle`.

III. Comparaison asymptotique de suites et de fonctions

1. Les notations $\mathcal{O}$ et o

Exemple 1

Les notations suivantes sont dues à Landau.

```
$\bigO$ ou $\smallO$
```

$\mathcal{O}$ ou o

Exemple 2

`$\bigO(x) \neq \smallO(x)$ ou` `$e^{t + \smallO(t)} = e^{\bigO(t)}$`	$\mathcal{O}(x) \neq o(x)$ ou $e^{t+o(t)} = e^{\mathcal{O}(t)}$

2. La notation Ω

Exemple 1

La notation suivante est due à Hardy et Littlewood.

`$\bigomega$`	Ω

Exemple 2

Dans l'exemple suivant, $f(n) = \Omega(g(n))$ signifie : $\exists (m, n_0)$ tel que $n \geq n_0$ implique $f(n) \geq mg(n)$.

`$f(n) = \bigomega(g(n))$`	$f(n) = \Omega(g(n))$

3. La notation Θ

Exemple 1

`$\bigtheta$`	Θ

Exemple 2

Dans l'exemple suivant, $f(n) = \Theta(g(n))$ signifie : $\exists (m, M, n_0)$ tel que $mg(n) \leq f(n) \leq Mg(n)$ dès que $n \geq n_0$.

`$f(n) = \bigtheta(g(n))$`	$f(n) = \Theta(g(n))$

Chapitre 7

Probabilité

I. Ensembles classiques de nombres

Se reporter à la section 3. page 13 où est présentée la macro `\setproba` pour indiquer des ensembles de type probabiliste.

II. Juste pour rédiger

1. Probabilité « simple »

Exemple 1

`$\proba{A}$`	$\mathrm{P}(A)$

Exemple 2 – Choisir le nom de la probabilité

`$\proba[p]{A}$`	$\mathrm{p}(A)$

2. Probabilité conditionnelle

Exemple 1 – Les deux écritures classiques

La 1re notation, qui est devenue standard, permet de comprendre l'ordre des arguments.

`$\probacond {B}{A}` `= \probacond*{B}{A}$`	$\mathrm{P}_{\mathrm{B}}(A) = \mathrm{P}(A \mid B)$

Exemple 2 – Obtenir la formule de définition

Le préfixe `e` est pour `e`-xpand soit *« développer »* en anglais [1].

`$\eprobacond {B}{A}` `= \eprobacond*{B}{A}$`	$\dfrac{\mathrm{P}(A \cap B)}{\mathrm{P}(B)} = \frac{\mathrm{P}(A \cap B)}{\mathrm{P}(B)}$

1. Pour ne pas alourdir l'utilisation de `\probacond`, il a été choisi d'utiliser un préfixe au lieu d'un système de multi-options.

Exemple 3 – Choisir le nom de la probabilité

```
 $\probacond  [p]{B}{A}
= \probacond* [p]{B}{A}
= \eprobacond*[p]{B}{A}
= \eprobacond [p]{B}{A}$
```

$$\mathrm{p}_B(A) = \mathrm{p}(A \mid B) = \frac{\mathrm{p}(A \cap B)}{\mathrm{p}(B)} = \frac{\mathrm{p}(A \cap B)}{\mathrm{p}(B)}$$

3. Évènement contraire

`\nevent` vient de `n`-ot `event` qui est une pseudo-traduction de *« évènement contraire »* en anglais.

```
$\nevent{A}$
```

$$\overline{A}$$

4. Espérance, variance et écart-type

Exemple 1 – Espérance

`\expval` vient de `exp`-ected `val`-ue soit *« espérance »* en anglais.

```
$\expval{X}$
```

$$\mathrm{E}(X)$$

Exemple 2 – Choisir le nom de l'espérance

```
$\expval[E_1]{X}$
```

$$\mathrm{E}_1(X)$$

Exemple 3 – Variance

Notez la possibilité d'utiliser `\mathit` pour obtenir des lettres en italique.

```
$\var   {X}$ ou
$\var[v]{X}$ ou
$\var[\mathit{v}]{X}$
```

$$\mathrm{V}(X) \text{ ou } \mathrm{v}(X) \text{ ou } v(X)$$

Exemple 4 – Écart-type

`\stddev` vient de `st`-andar-`d` `dev`-iation soit *« écart-type »* en anglais.

```
$\stddev   {X}$ ou
$\stddev[s]{X}$ ou
$\stddev[\mathit{s}]{X}$
```

$$\sigma(X) \text{ ou } \mathrm{s}(X) \text{ ou } s(X)$$

III. Calculer l'espérance – Cas fini

Il est possible de définir facilement les valeurs d'une variable aléatoire finie (v.a.f.) et d'obtenir si souhaiter le calcul détaillé de son espérance [2].

2. Pour l'affichage une limitation importante est que le tableau et les calculs doivent tenir sur la largeur de la ligne. Concrètement pour un usage pédagogique cette limitation ne devrait jamais poser de problème.

1. Pour un usage direct – Notations par défaut

Dans la section 2. page 75 nous verrons qu'il est possible de définir une loi à un endroit puis de la réutiliser à d'autres[3]. Nous verrons aussi dans la section 3. page 76 que certaines notations sont modifiables.

Dans la présente section nous reproduirons à chaque fois les paramètres de la v.a.f. même si ceci produit des exemples un peu lourds à lire.

Exemple 1 – Loi d'une v.a.f.

La définition de la loi d'une v.a.f. avec la macro `\calcexpval` se fait avec une syntaxe semblable à celle d'un tableau. Notez dans l'exemple suivant que par défaut un tableau centré est affiché ce qui correspond à l'option par défaut `disp = table`. Ce n'est pas le seul comportement possible comme vont le montrer les autres exemples à venir.

```
\calcexpval{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

Remarque. Il est possible de ne rien afficher comme ci-dessous où `disp` est pour `disp`-lay soit *« afficher »* en anglais. Cette option sera utile pour définir une loi à un endroit et l'utiliser à d'autres.

```
\calcexpval[disp = none]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

Exemple 2 – Calculs expliqués et décorés de l'espérance d'une v.a.f.

La macro `\calcexpval` propose un système de clé valeur pour des réglages personnalisé. L'exemple ci-dessous montre comment obtenir un calcul détaillé et décoré de l'espérance.

```
\calcexpval[disp = all]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

$$\mathrm{E}(X) = \sum_{k=1}^{7} p_k \cdot x_k$$

$$\mathrm{E}(X) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$$

3. La réutilisation se fera avec un argument vide de `\calcexpval`. C'est pour cela qu'une macro est proposée et non un environnement.

L'exemple suivant montre qu'il est facile de changer le couleurs qui sont utilisées de façon cyclique *(on peut indiquer une seule couleur)*.

```
\calcexpval[disp   = all,
            colors = orange - olive]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

$$\mathrm{E}(X) = \sum_{k=1}^{7} p_k \cdot x_k$$

$$\mathrm{E}(X) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$$

Exemple 3 – Juste détailler les calculs de l'espérance d'une v.a.f.

Ci-dessous `exp` est pour `exp`-and soit *« développer »* en anglais. Cette option sera très utile lors de la réutilisation d'une loi définie précédemment.

```
\calcexpval[disp   = exp]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

$$\mathrm{E}(X) = \sum_{k=1}^{7} p_k \cdot x_k$$

$$\mathrm{E}(X) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$$

Exemple 4 – Des bouts du calcul

Lors de la réutilisation d'une loi définie précédemment les deux options `formal` et `eval` peuvent rendre service.

```
Somme formelle :
\calcexpval[disp = formal]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}

Somme développée :
\calcexpval[disp = eval]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

Somme formelle :

$$\sum_{k=1}^{7} p_k \cdot x_k$$

Somme développée :
$0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$

Exemple 5 – Calculs expliqués sans Σ

L'option `nosigma` permet de ne pas afficher le symbole Σ pour des raisons pédagogiques ou d'efficacité.

```
\calcexpval[disp = all,
            nosigma]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}

\medskip

Juste les calculs :
\calcexpval[disp = exp,
            nosigma]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
} .
```

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

$$\mathrm{E}(X) = \boxed{0.2000 \cdot 0} + 0.1 \cdot 1 + \boxed{0.2 \cdot 2} + 0.05 \cdot 3 + \boxed{0.15 \cdot 4} + 0.1 \cdot 5 + \boxed{0.2 \cdot 6}$$

Juste les calculs :
$\mathrm{E}(X) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$.

2. Pour un usage différé – Notations par défaut

Rien de bien compliqué à utiliser comme le montre l'exemple suivant. Notez bien qu'il faut indiquer un argument vide à la macro `\calcexpval` lors de la réutilisation.

```
Voici ma loi de dingue.

\calcexpval[name = maloidedingue]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}

Je dis des choses, bla, bla...

Et puis finalement j'indique le calcul de l'espérance.

\calcexpval[disp  = exp,
            reuse = maloidedingue]{}
```

Voici ma loi de dingue.

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

Je dis des choses, bla, bla...
Et puis finalement j'indique le calcul de l'espérance.

$$\mathrm{E}(X) = \sum_{k=1}^{7} p_k \cdot x_k$$

$$\mathrm{E}(X) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$$

Remarque. Si cela vous convient, il est possible d'utiliser des noms de type « chemin de fichiers » comme ci-après [4].

```
Définissons de façon cachée la loi...
\calcexpval[disp = none,
            name = ma/loi/de/dingue]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}

Bla, bla... Affichons la loi.

\calcexpval[disp  = table,
            reuse = ma/loi/de/dingue]{}
```

Définissons de façon cachée la loi...
Bla, bla... Affichons la loi.

x_k	0	1	2	3	4	5	6
p_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

3. Notations personnalisées

Exemple 1 – Toutes les options

L'exemple ci-après utilise toutes les options liées à l'affichage textuel. Il faut noter que la clé `com` sert juste à ajouter un court texte sur une seule ligne [5].

```
\calcexpval[disp = all,
            E    = Esp, X  = Y,
            k    = i  , xk = y, pk = q,
            com  = Le tableau précédent aboutit aux calculs suivants.,
            ope  = \times]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}
```

4. Ceci permet d'utiliser des espaces de noms très pratiques à l'usage.
5. Si besoin utiliser séparément l'affichage par défaut avec `name = maloi` suivi d'un long texte puis enfin employer `disp = exp, reuse = maloi`.

y_i	0	1	2	3	4	5	6
q_i	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

Le tableau précédent aboutit aux calculs suivants.

$$\mathrm{Esp}(Y) = \sum_{i=1}^{7} q_i \times y_i$$

$$\mathrm{Esp}(Y) = 0.2000 \times 0 + 0.1 \times 1 + 0.2 \times 2 + 0.05 \times 3 + 0.15 \times 4 + 0.1 \times 5 + 0.2 \times 6$$

Exemple 2 – En réutilisant une loi définie précédemment

Lors de la création d'une loi nommée via la clé `name` les options textuelles, autre que le commentaire court, sont mémorisées. Ceci est très utile comme le montre le 1er affichage de l'exemple suivant tout en étant modifiable à tout moment comme dans le dernier affichage ci-après.

```
DÉFINITION CACHÉE.
\calcexpval[name = option/texte/loi,
            disp = none,
            E    = Esp, X  = Y,
            k    = i  , xk = y, pk = q,
            ope  = \times]{
    0      & 1   & 2   & 3    & 4    & 5   & 6   \\
    0.2000 & 0.1 & 0.2 & 0.05 & 0.15 & 0.1 & 0.2
}

\bigskip

RÉUTILISATION DIRECTE.
\calcexpval[reuse = option/texte/loi,
            disp  = all]{}

\bigskip

RÉUTILISATION AVEC DES OPTIONS MODIFIÉES (E, k et ope = $\cdot$).
\calcexpval[reuse = option/texte/loi,
            disp  = all,
            E     = E,
            k     = k,
            com   = Le tableau précédent aboutit aux calculs suivants.,
            ope   = \cdot]{}
```

DÉFINITION CACHÉE.

RÉUTILISATION DIRECTE.

y_i	0	1	2	3	4	5	6
q_i	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

$$\mathrm{Esp}(Y) = \sum_{i=1}^{7} q_i \times y_i$$

$$\mathrm{Esp}(Y) = 0.2000 \times 0 + 0.1 \times 1 + 0.2 \times 2 + 0.05 \times 3 + 0.15 \times 4 + 0.1 \times 5 + 0.2 \times 6$$

RÉUTILISATION AVEC DES OPTIONS MODIFIÉES (E, k et ope = ·).

y_k	0	1	2	3	4	5	6
q_k	0.2000	0.1	0.2	0.05	0.15	0.1	0.2

Le tableau précédent aboutit aux calculs suivants.

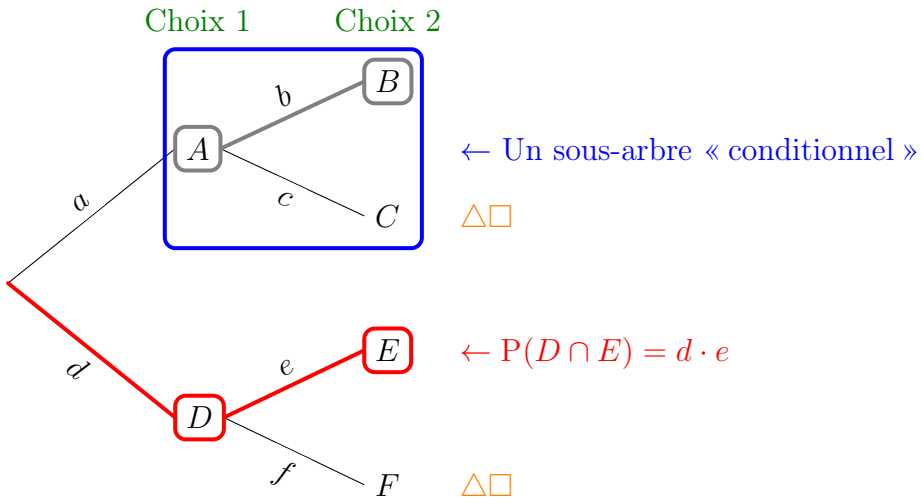
$$\mathrm{E}(Y) = \sum_{k=1}^{7} q_k \cdot y_k$$

$$\mathrm{E}(Y) = 0.2000 \cdot 0 + 0.1 \cdot 1 + 0.2 \cdot 2 + 0.05 \cdot 3 + 0.15 \cdot 4 + 0.1 \cdot 5 + 0.2 \cdot 6$$

IV. Arbres pondérés

1. Au commencement était la forêt...

Le gros du travail est fait par le package `forest` qui s'appuie `TikZ` dont on peut utiliser toute la machinerie afin d'obtenir des choses sympathiques comme ci-dessous et ceci à moindre coût neuronal comme vont le montrer les explications données dans les sections suivantes.



Le rendu précédent a été obtenu via le code suivant.

```
\begin{probatree}
    [{}, name = nU              % Noeud racine sans texte via {} mais avec un nom.
        [A, apweight = a,
            name     = nA,
            pframe   = blue     % Encadrement d'ici à la fin.
            [B, name     = nB,
                apweight = b]
            [C, name     = nC,
                bpweight = c]   % Pas besion de nommer ce noeud.
        ]
        [D, bpweight = d,
            name     = nD
            [E, apweight = e,
                name     = nE]
            [F, name     = nF,
                bpweight = f]   % Pas besion de nommer ce noeud.
```

```
        ]
    ]
    % Étiquettes pour les niveaux.
    \ptreeTagLevel[tcol = green!50!black, dy = .75mm]
                  {nA}{Choix 1}
    \ptreeTagLevel[tcol = green!50!black, dy = .75mm]
                  {nB}{Choix 2}
    % Mettre en valeur un chemin.
    \ptreeFocus[lcol = gray, frame = start]
               {nA | nB}
    % Mettre en valeur un chemin et le commenter.
    \ptreeComment[tcol = blue]
                 {nA}{$\leftarrow$ Un sous-arbre \og conditionnel \fg}
    % Comme avant.
    \ptreeFocus[lcol = red]
               {nU | nD | nE}
    \ptreeComment[tcol = red]
                 {nE}{$\leftarrow \proba{D \cap E} = d \cdot e$}
    % Décorer des feuilles.
    \ptreeTagLeaf[tcol = orange]
                 {nC | nF}{$\triangle \square$}
\end{probatree}
```

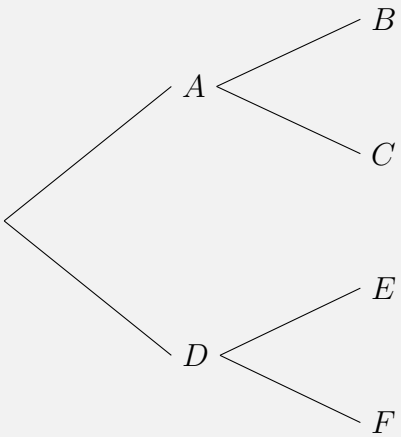
Remarque. Jusqu'à la section 9. page 95, nous nommerons à la main les noeuds des arbres via `name = ...` lorsque cela sera nécessaire. Dans la section indiquée nous verrons comment utiliser les noms automatiques donnés par le package `forest`.

2. Les bases

Exemple 1 – Le cas type

Commençons par un arbre nu pour voir comment utiliser l'environnement `probatree` qui s'appuie en coulisse sur celui nommé `forest` du package éponyme. L'exemple qui suit utilise juste les réglages spécifiques de mise en forme de l'arbre qui sont propres à `probatree`.

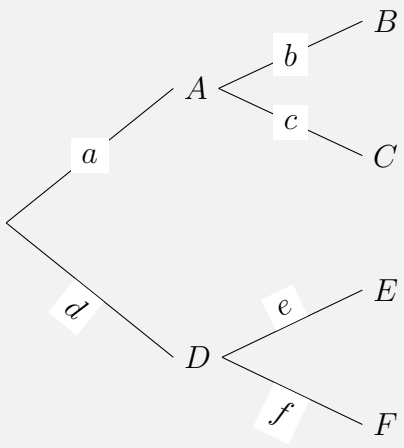
```
\begin{probatree}
    [           % Noeud racine sans texte
        [A      % Sous-noeud nommé
            [B] % Sous-sous-noeud nommé
            [C] % Sous-sous-noeud nommé
        ]
        [D      % Sous-noeud nommé
            [E] % Sous-sous-noeud nommé
            [F] % Sous-sous-noeud nommé
        ]
    ]
\end{probatree}
```



Exemple 2 – Ajouter des pondérations

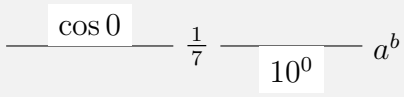
Dans le code suivant, ce sont les clés [6] `pweight`, `apweight` et `bpweight` qui facilitent l'écriture des pondérations sur les branches. Indiquons que `pweight` vient de `p`-robability et `weight` soit *« probabilité »* et *« poids »* en anglais. Quant au `a` et au `b` au début de `apweight` et `bpweight` respectivement, ils viennent de `a`-bove et `b`-elow soit *« dessus »* et *« dessous »* en anglais.

```
\begin{probatree}
    [
        [A, pweight = a
            [B, pweight = b]
            [C, pweight = c]
        ]
        [D, bpweight = d
            [E, apweight = e]
            [F, bpweight = f]
        ]
    ]
\end{probatree}
```



Remarque. Notez que le mode mathématique est activé par défaut pour les noms des noeuds et les poids comme le montre l'exemple improbable [7] ci-dessous.

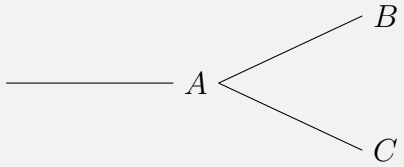
```
\begin{probatree}
    [
        [\frac{1}{7}, apweight = \cos 0
            [a^b, bpweight = 10^0]
        ]
    ]
\end{probatree}
```



Exemple 3 – Des poids cachés partout

On peut cacher tous les poids sans avoir à les effacer partout dans le code LaTeX *(ceci peut être utile lors de la rédaction d'exercices)*. Il suffit pour cela d'utiliser une option `hideall` de l'environnement `probatree`. Comme les codes des arbres utilisent des crochets, l'option s'indique via `<hideall>` et non le traditionnel `[hideall]` [8]. L'option utilisée par défaut est `asit` qui demande de respecter les indications données pour les poids dans le code de l'arbre.

```
\begin{probatree}<hideall>
    [
        [A, pweight = a
            [B, apweight = b]
            [C, bpweight = c]
        ]
    ]
\end{probatree}
```

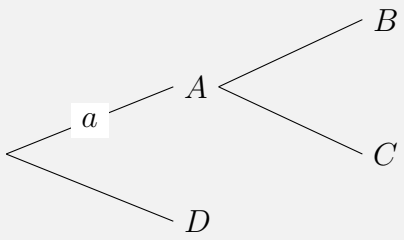


6. En fait du point de vue de TikZ, ce sont des styles.
7. Quoique...
8. Ceci a été rendu possible grâce à un code proposé dans l'excellent livre *« Apprendre à programmer en TeX »* de Christian Tellechea.

Exemple 4 – Des poids cachés localement

Pour ne cacher que certains poids afin de produire par exemple un arbre à compléter, il faudra utiliser localement le style `pweight*` comme dans l'exemple ci-dessous.

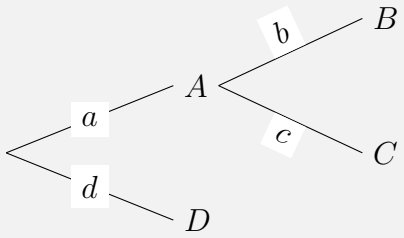
```
\begin{probatree}
    [
        [A, pweight = a
            [B, apweight* = b]
            [C, bpweight* = c]
        ]
        [D, pweight* = d]
    ]
\end{probatree}
```



Exemple 5 – Forcer l'affichage des poids cachés localement

Imaginons que nous voulions donner l'arbre précédent avec tous ses poids visibles pour une correction par exemple. Il suffit dans ce cas de passer via l'option `showall` de l'environnement `probatree` qui affichera absolument tous les poids *(on tape une fois et on réutilise le même code dans deux contextes différents)*.

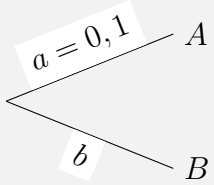
```
\begin{probatree}<showall>
    [
        [A, pweight = a
            [B, apweight* = b]
            [C, bpweight* = c]
        ]
        [D, pweight* = d]
    ]
\end{probatree}
```



Exemple 6 – Un signe = et/ou une virgule dans les étiquettes

Vous ne pouvez pas utiliser directement un signe = ou une virgule dans les étiquettes des branches [9]. Pour contourner cette limitation, il suffit de mettre le contenu de l'étiquette entre des accolades.

```
\begin{probatree}
    [
        [A, apweight = {a = 0,1}]
        [B, bpweight = b]
    ]
\end{probatree}
```



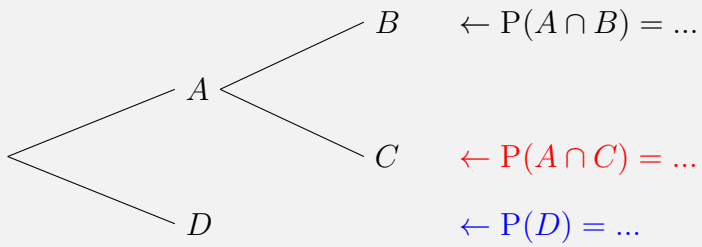
9. Ces deux symboles font partie de la syntaxe TikZ.

3. Commenter les feuilles

Exemple 1 – Tout aligner

Que ce soit pour expliquer un arbre de probabilité, ou bien pour raisonner sur ce dernier, l'effet suivant est très utile [10]. Noter l'utilisation possible de la clé `tcol` pour `t`-ext et `col`-or afin d'indiquer la couleur du texte au format TikZ. La couleur par défaut est le noir.

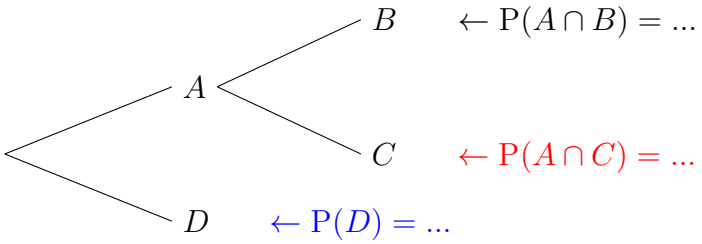
```
\begin{probatree}
    [
        [A
            [B, name = nB]
            [C, name = nC]
        ]
        [D, name = nD]
    ]
    %
    \ptreeComment              {nB}{$\leftarrow \proba{A \cap B} = ...$}
    \ptreeComment[tcol = red] {nC}{$\leftarrow \proba{A \cap C} = ...$}
    \ptreeComment[tcol = blue]{nD}{$\leftarrow \proba{D} = ...$}
\end{probatree}
```



Remarque. Commenter un noeud interne ne provoquera pas d'erreur même si `\ptreeComment` n'a pas été conçu pour ceci. Ceci a été utilisé dans l'exemple d'introduction mais ça reste un petit hack.

Exemple 2 – Coller au plus près

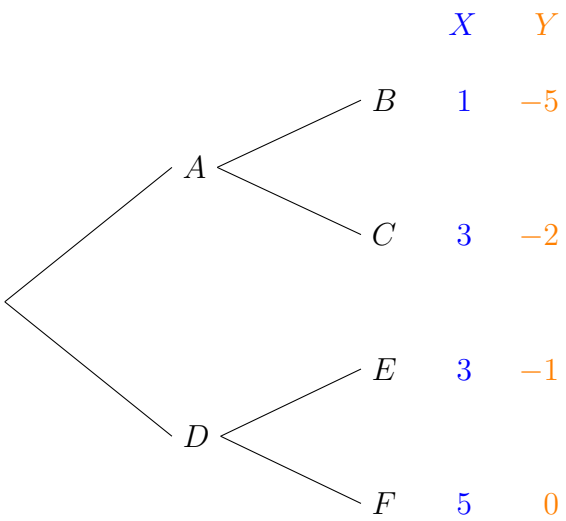
En utilisant `\ptreeComment*` au lieu de `\ptreeComment`, les commentaires seront proches des noeuds et donc non alignés verticalement. Avec l'exemple précédent on obtient la mise en forme qui suit.



Exemple 3 – Décalages horizontal et vertical – Avec des variables aléatoires

Grâce aux clés `dx` et `dy` il est possible d'ajouter des décalages horizontal et vertical. Ceci permet d'obtenir ce qui suit sans trop se fatiguer les méninges.

10. Le package `forest` permet d'indiquer directement des mises en forme dans le code de l'arbre. L'auteur du présent package trouve bien plus efficace à l'usage de ne pas toucher au code minimal d'un arbre. Ceci explique donc le choix retenu de donner les décorations supplémentaires après le code de l'arbre.



Le code utilisé est le suivant. Notez qu'ici il y a des réglages à faire au doigt mouillé. Dans l'exemple suivant nous allons voir comment se passer des horribles copier-coller.

```
\begin{probatree}
    % ------------------------------------------- %
    % -- Code de l'arbre seul non reproduit ici -- %
    % ------------------------------------------- %
    % Valeurs de X
    \ptreeComment[tcol = blue,
                  dx   = -.25em,
                  dy   = 1cm]{nB}{$X$}
    %
    \ptreeComment[tcol = blue]{nB}{$1$}
    \ptreeComment[tcol = blue]{nC}{$3$}
    \ptreeComment[tcol = blue]{nE}{$3$}
    \ptreeComment[tcol = blue]{nF}{$5$}
    %
    % Valeurs de Y
    \ptreeComment[tcol = orange,
                  dx   = 2em+.5em,
                  dy   = 1cm]{nB}{$Y$}
    %
    \ptreeComment[tcol = orange,
                  dx   = 2em]{nB}{$-5$}
    \ptreeComment[tcol = orange,
                  dx   = 2em]{nC}{$-2$}
    \ptreeComment[tcol = orange,
                  dx   = 2em]{nE}{$-1$}
    \ptreeComment[tcol = orange,
                  dx   = 2em]{nF}{$\phantom{-}0$}
\end{probatree}
```

Exemple 4 – Commenter via une boucle – Avec des variables aléatoires

Dans le code précédent nous avons dû faire des copier-coller. La macro `\foreach` de TikZ permet d'éviter cela afin d'obtenir un code très facile à maintenir et à comprendre comme celui ci-après. Ceci étant indiqué, il y a des pièges à éviter comme nous l'expliquons juste après.

```
\begin{probatree}
    % ------------------------------------------ %
    % -- Code de l'arbre seul non reproduit ici -- %
    % ------------------------------------------ %
    % Valeurs de X
    \ptreeComment[tcol = blue,
                  dx = -.25em, dy = 1cm] {nB}{$X$}
    %
    \foreach \name/\val in {
        nB/$1$,
        nC/$3$,
        nE/$3$,
        nF/$5$
    }{
        \ptreeComment[tcol = blue]{\name}{\val}
    }
    % Valeurs de Y
    \ptreeComment[tcol = orange,
                  dx   = 2em+.5em,
                  dy   = 1cm]{nB}{$Y$}
    %
    \foreach \name/\val in {
        nB/$-5$,
        nC/$-2$,
        nE/$-1$,
        nF/$\phantom{-}0$
    }{
        \ptreeComment[tcol = orange,
                      dx   = 2em]{\name}{\val}
    }
\end{probatree}
```

Voici les pièges à éviter.

1. `\foreach` ignore les espace initiaux mais pas les finaux. Si vous utilisez `nB /$1$` au lieu de `nB/$1$` alors la macro croira que le nom se finit par un espace et `forest` ne pourra rien faire.
2. Comme les noms `\color`, `\tcol`, `\dx` et `\dy` sont utilisés en coulisse, il n'est pas possible de les utiliser dans les boucles.

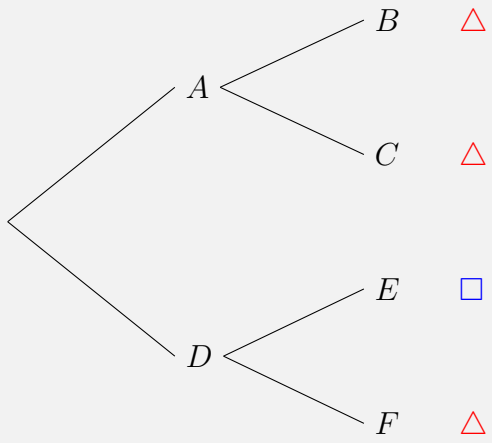
4. Décorer les feuilles

Il peut être utile de décorer de la même façon différents chemins pour indiquer des évènements étudiés. C'est la raison d'être de `\ptreeTagLeaf` et `\ptreeTagLeaf*` qui produisent le même commentaire pour différents noeuds. Notez dans les exemples que les noms des noeuds sont séparés par le symbole `|` avec la possibilité d'ajouter des espaces pour améliorer la lisibilité du code.

Exemple 1 – Décorer au même niveau

```
\begin{probatree}
    [
        [A
            [B, name = nB]
```

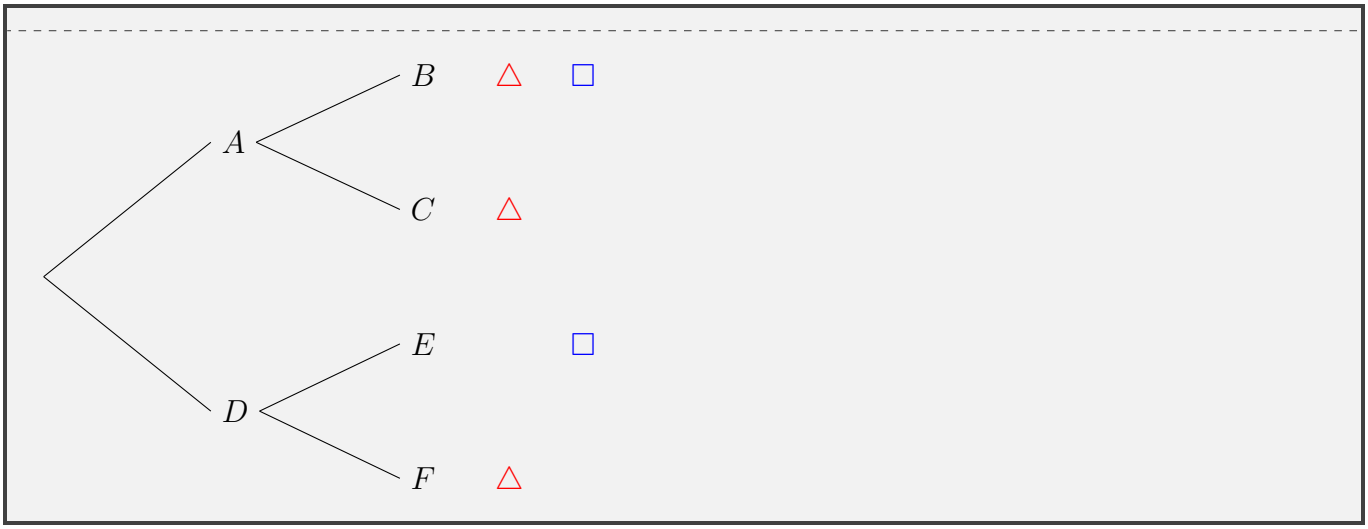
```
            [C, name = nC]
        ]
        [D
            [E, name = nE]
            [F, name = nF]
        ]
    ]
    %
    \ptreeTagLeaf[tcol = red]
                 {nB | nC | nF}{$\triangle$}
    \ptreeTagLeaf[tcol = blue]
                 {nE}{$\square$}
\end{probatree}
```



Exemple 2 – Décorer sur des niveaux décalés

La démo. suivante n'utilise que `dx` mais bien entendu `dy` est aussi disponible.

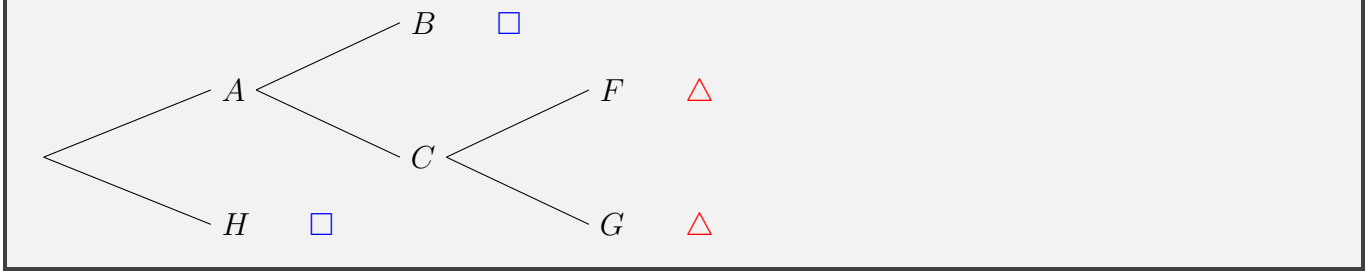
```
\begin{probatree}
    [
        [A
            [B, name = nB]
            [C, name = nC]
        ]
        [D
            [E, name = nE]
            [F, name = nF]
        ]
    ]
    %
    \ptreeTagLeaf[tcol = red]
                 {nB | nC | nF}{$\triangle$}
    \ptreeTagLeaf[tcol = blue,
                  dx   = 1cm]
                 {nB | nE}{$\square$}
\end{probatree}
```



Exemple 3 – Décorer au plus près

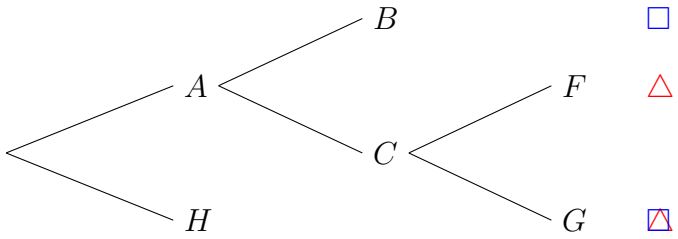
Voici un exemple avec un arbre dissymétrique[11].

```
\begin{probatree}
    [
        [A
            [B, name = nB]
            [C,
                [F, name = nF]
                [G, name = nG]
            ]
        ]
        [H, name = nH]
    ]
    \ptreeTagLeaf*[tcol = red]
                  {nF | nG}{$\triangle$}
    \ptreeTagLeaf*[tcol = blue]
                  {nB | nH}{$\square$}
\end{probatree}
```



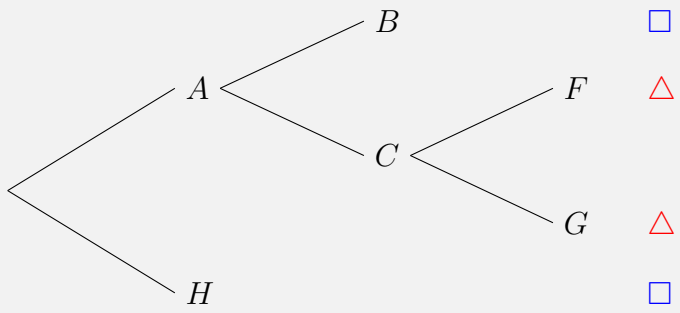
Remarque. Sans utiliser la version étoilée de `\ptreeTagLeaf`, on obtient ce qui suit qui est mauvais.

11. Est-ce vraiment pertinent comme usage ?



Le package `forest` propose `fit = band` qui est très utile dans ce type de situation. Voici comment utiliser ceci.

```
\begin{probatree}
    [, s sep = 0.35cm     % <-- Un peu de design aux doigts mouillés.
        [A
            [B, name = nB]
            [C,
                [F, name = nF]
                [G, name = nG]
            ]
        ]
        [H, name = nH,
            fit  = band]  % <-- Pour placer H seul sur son niveau.
    ]
    \ptreeTagLeaf[tcol = red]
                 {nF | nG}{$\triangle$}
    \ptreeTagLeaf[tcol = blue]
                 {nB | nH}{$\square$}
\end{probatree}
```



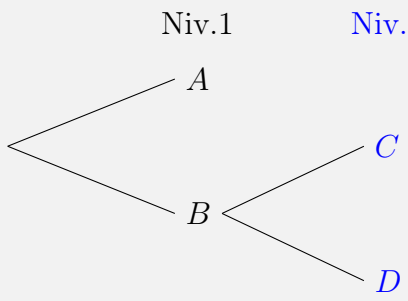
5. Expliquer les niveaux

Grâce aux deux sections précédentes les exemples devraient se suffire à eux-mêmes pour comprendre les fonctionnements de `\ptreeTagLevel` et `\ptreeTagLevel*`.

Exemple 1 – Expliquer au-dessus

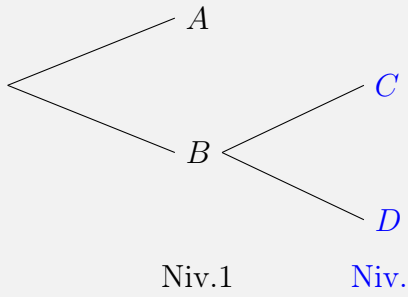
```
\begin{probatree}
    [
        [A, name = nA]
        [B
            [C, name = nC,
                blue]
```

```
            [D, blue]
        ]
    ]
    %
    \ptreeTagLevel            {nA}{Niv.1}
    \ptreeTagLevel[tcol=blue]{nC}{Niv.2}
\end{probatree}
```



Exemple 2 – Expliquer en dessous

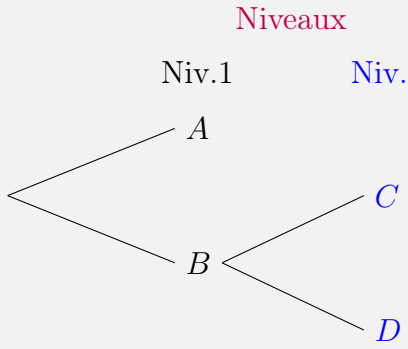
```
\begin{probatree}
    [
        [A, name = nA]
        [B
            [C, name = nC,
                blue]
            [D, blue]
        ]
    ]
    %
    \ptreeTagLevel*            {nA}{Niv.1}
    \ptreeTagLevel*[tcol=blue]{nC}{Niv.2}
\end{probatree}
```



Exemple 3 – Décalages horizontal et vertical

```
\begin{probatree}
    [
        [A, name = nA]
        [B
            [C, name = nC,
```

```
                blue]
            [D, blue]
        ]
    ]
    %
    \ptreeTagLevel[tcol = purple,
                   dx   = 3em,
                   dy   = 1.75em]{nA}{Niveaux}
    %
    \ptreeTagLevel           {nA}{Niv.1}
    \ptreeTagLevel[tcol=blue]{nC}{Niv.2}
\end{probatree}
```

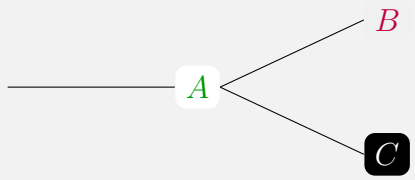


6. Textes des noeuds

Exemple 1 – Changer les couleurs

On peut a posteriori changer les couleurs du texte et du fond d'un noeud via `\ptreeNodeColor` présentée ci-dessous.

```
\begin{probatree}
    [
        [A, name = nA
            [B, name = nB]
            [C, name = nC]
        ]
    ]
    \ptreeNodeColor{nA}{tcol = green!60!black}
    \ptreeNodeColor{nB}{tcol = purple,
                        bcol = black!6!white}
    \ptreeNodeColor{nC}{tcol = white,
                        bcol = black}
\end{probatree}
```



Voici ce qu'il faut retenir du code précédent.

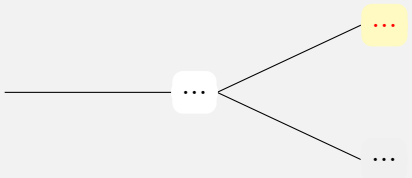
1. Par défaut le blanc sert de couleur de fond. Ceci se voit dans la mise en forme du noeud A pour lequel `tcol` change juste la couleur du texte qui par défaut sera le noir.
2. Pour changer la couleur de fond, on passe via `bcol`. Ceci permet d'avoir le rendu souhaité pour la mise en forme du noeud B [12].
3. Les préfixes `t` et `b` de `bcol` et `tcol` sont pour `t`-exte et `b`-ackground, ce dernier mot signifiant *« fond »* en anglais. Quant à `col` c'est pour `col`-or soit *« couleur »* en anglais.

Remarque. Techniquement toutes les macros présentées dans cette section cachent l'ancien texte d'un noeud par superposition de ce texte en utilisant une couleur identique pour le texte et le fond.

Exemple 2 – Changer le texte

On peut a posteriori changer le texte d'un noeud, avec un choix des couleurs, via `\ptreeNodeNewText` présentée ci-après. Noter que les couleurs par défaut du texte et du fond restent le noir et le blanc respectivement.

```
\begin{probatree}
    [
        [A, name = nA
            [B, name = nB]
            [C, name = nC]
        ]
    ]
    \ptreeNodeNewText{nA}{...}
    \ptreeNodeNewText[tcol = red,
                      bcol = yellow!30!white]%
                     {nB}{...}
    \ptreeNodeNewText[bcol = black!6!white]%
                     {nC}{...}
\end{probatree}
```



Exemple 3 – Récupérer le texte d'un noeud

Considérons l'arbre un peu plat suivant.

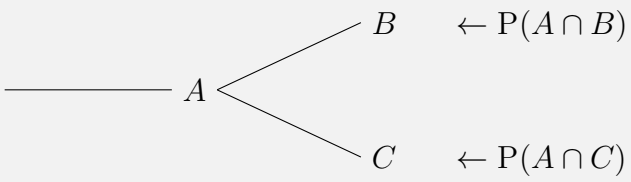
```
\begin{probatree}
    [
        [texte.de.A, name = nA
            [B]
        ]
    ]
\end{probatree}
```

texte.de.A — B

12. Ainsi le fond du noeud et celui du cadre ont la même couleur.

Une fois ce code inséré il est possible de récupérer *texte.de.A* juste en tapant `\ptreeTextOf{nA}`. Voici un exemple concret [13] d'utilisation.

```
\begin{probatree}
    [
        [A, name = nA
            [B, name = nB]
            [C, name = nC]
        ]
    ]
    %
    \ptreeComment{nB}%
                 {$\leftarrow \proba{\ptreeTextOf{nA} \cap \ptreeTextOf{nB}}$}
    \ptreeComment{nC}%
                 {$\leftarrow \proba{\ptreeTextOf{nA} \cap \ptreeTextOf{nC}}$}
\end{probatree}
```



Remarque. Comme la macro `\ptreeNodeNewText` utilise le dernier noeud rencontré, il faudra veiller à ne pas vouloir utiliser les textes de noeuds présents dans deux arbres différents et qui ont en même temps le même nom.

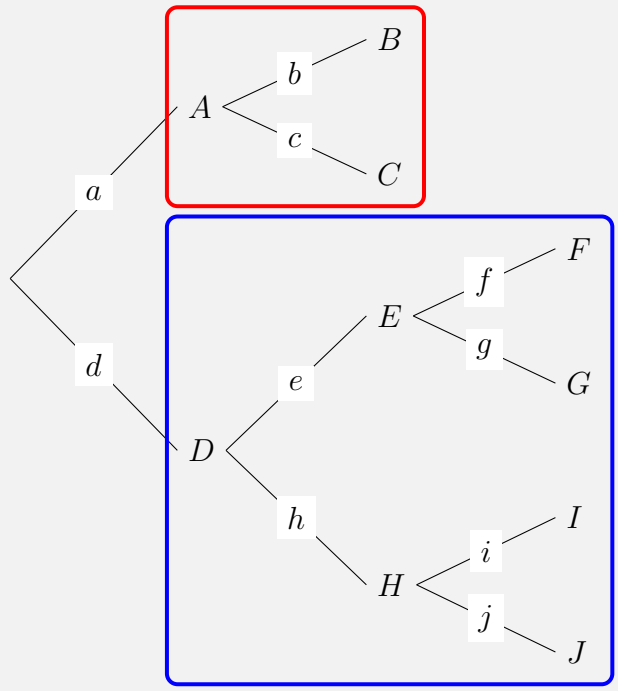
7. Avec des cadres

Exemple 1 – Des cadres finaux

Via la clé `pframe` il est très aisé d'encadrer un sous-arbre final [14] comme le montre l'exemple suivant [15]. Dans l'exemple ci-après nous utilisons la bidouille `{},s sep = 1.3cm` qui évite que les cadres se superposent.

13. Mais pas forcément pertinent... L'exemple peut être intéressant dans le cadre de contenus produits de façon automatisée.
14. Un sous-arbre sera dit final si toutes ses feuilles correspondent à des feuilles de l'arbre initial.
15. Ce type de cadre est très utile d'un point de vue pédagogique.

```
\begin{probatree}
    [{}, s sep = 1.3cm
     % Espacment pour éviter que
     % les cadres se superposent.
        [A, pweight = a,
            pframe  = red
            [B, pweight = b]
            [C, pweight = c]
        ]
        [D, pweight = d,
            pframe  = blue
            [E, pweight = e
                [F, pweight = f]
                [G, pweight = g]
            ]
            [H, pweight = h
                [I, pweight = i]
                [J, pweight = j]
            ]
        ]
    ]
\end{probatree}
```

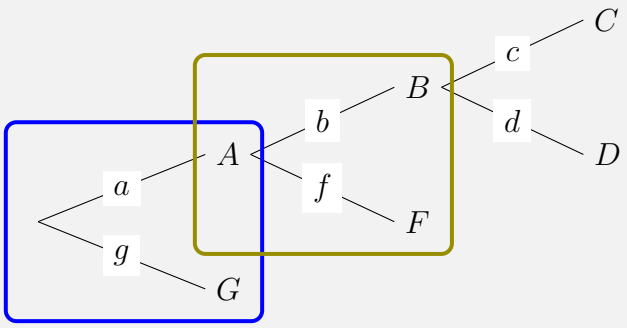


Remarque. La clé `pframe` est un cas particulier de décoration car les autres décorations se font en dehors de la définition de l'arbre

Exemple 2 – Des cadres non finaux

La macro `\ptreeFrame` permet facilement d'encadrer un sous-arbre non final. Ceci nécessite d'utiliser des noms de noeuds. Voici un exemple où la macro `\ptreeFrame` attend les noms de la racine et des deux noeuds finaux le plus haut et le plus bas.

```
\begin{probatree}
    [{}, name = nU
     % La racine a un texte vide {}.
        [A, pweight = a,
            name    = nA
            [B, pweight = b,
                name    = nB
                [C, pweight = c]
                [D, pweight = d]
               ]
            [F, pweight = f,
                name    = nF]
        ]
        [G, pweight = g,
            name    = nG]
    ]
    %
    \ptreeFrame                {nU}{nA}{nG}
    \ptreeFrame[lcol = olive]{nA}{nB}{nF}
\end{probatree}
```

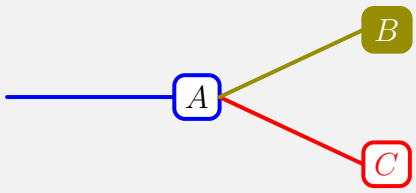


8. Mettre en valeur des chemins

Exemple 1 – Avec deux noeuds – Choix des couleurs

La macro `\ptreeFocus` rend facile la mise en valeur d'un chemin particulier comme le montre l'exemple ci-après qui est une simple démo. Notez que les noms des noeuds sont séparés par des barres verticales `|` et qu'il est possible d'utiliser des espaces pour améliorer la lisibilité du code.

```
\begin{probatree}
    [{}, name = nU
        [A, name = nA
            [B, name = nB]
            [C, name = nC]
        ]
    ]
    %
    \ptreeFocus{nU | nA}
    \ptreeFocus[tcol = red,
                lcol = red]{nA | nC}
    \ptreeFocus[tcol = white,
                bcol = olive,
                lcol = olive]{nA | nB}
\end{probatree}
```



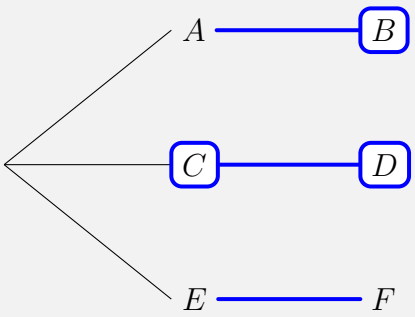
Voici ce qu'il faut retenir pour les couleurs qui doivent être du type TikZ.

1. `lcol` sert à indiquer la couleur des arêtes et des cadres éventuels. Par défaut `lcol = blue`. Indiquons que le préfixe `l` est pour `l`-igne.
2. `tcol` sert à indiquer la couleur du texte. Par défaut `lcol = black`. Indiquons que le préfixe `t` est pour `t`-exte.
3. `bcol` sert à indiquer la couleur du fond. Par défaut `lcol = white`. Indiquons que le préfixe `b` est pour `b`-ackground soit *« fond »* en anglais.

Exemple 2 – Avec deux noeuds – Choix des cadres

Par défaut le 1er noeud n'est pas encadré car il est courant de vouloir indiquer un noeud partant de la racine qui traditionnellement ne contient aucun texte. Il est possible d'obtenir deux autres mises en forme comme ci-après.

```
\begin{probatree}
    [
        [A, name = nA
            [B, name = nB]
        ]
        [C, name = nC
            [D, name = nD]
        ]
        [E, name = nE
            [F, name = nF]
        ]
    ]
    %
    \ptreeFocus                {nA | nB}
    \ptreeFocus[frame = start]{nC | nD}
    \ptreeFocus[frame = none] {nE | nF}
\end{probatree}
```



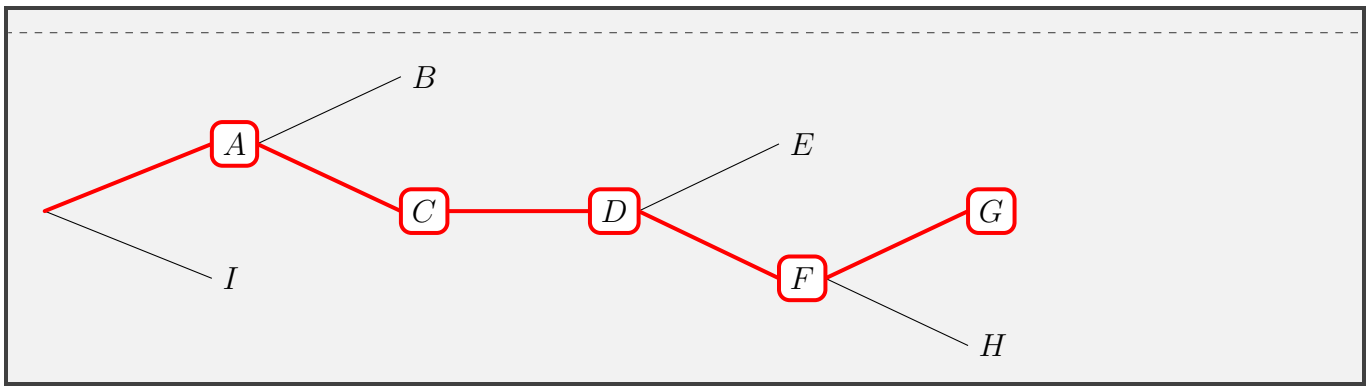
Voici ce qu'il faut retenir à propos des encadrements.

1. `frame = nostart`, le réglage par défaut, demande d'encadrer tous les noeuds sauf le 1er.
2. `frame = start` demande d'encadrer tous les noeuds.
3. `frame = none` demande de n'encadrer aucun noeud.

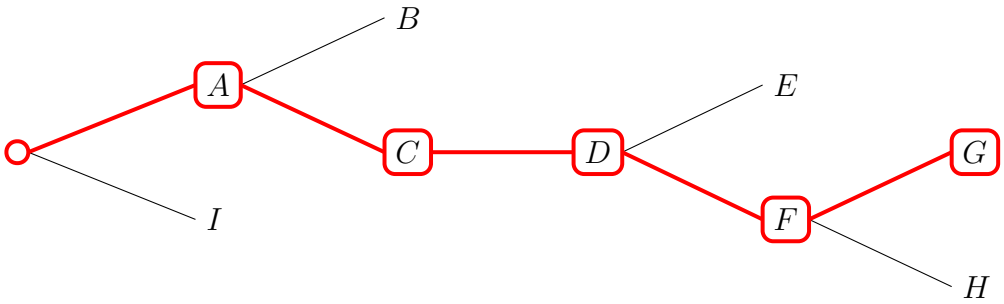
Exemple 3 – Plusieurs noeuds d'un coup

Rien de bien compliqué à condition de bien respecter l'ordre de saisie des noeuds.

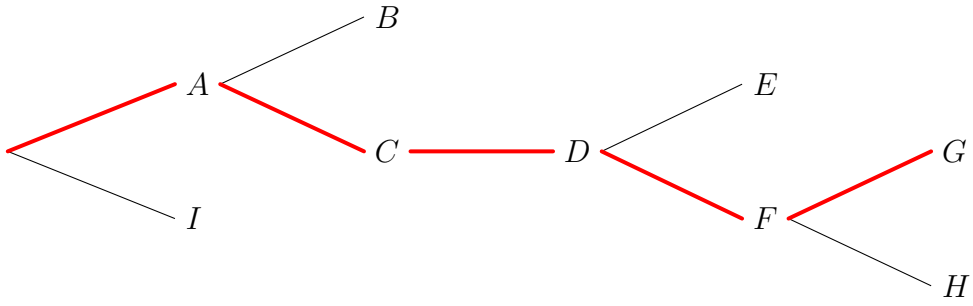
```
\begin{probatree}
    [{}, name = nU
        [A, name = nA
            [B]
            [C, name = nC
                [D, name = nD
                    [E]
                    [F, name = nF
                        [G, name = nG]
                        [H]
                    ]
                ]
            ]
        ]
        [I]
    ]
    %
    \ptreeFocus[lcol = red]{nU | nA | nC | nD | nF | nG}
\end{probatree}
```



Avec `frame = start` on obtient l'arbre suivant où le mini disque initial [16] n'est pas forcément souhaité.

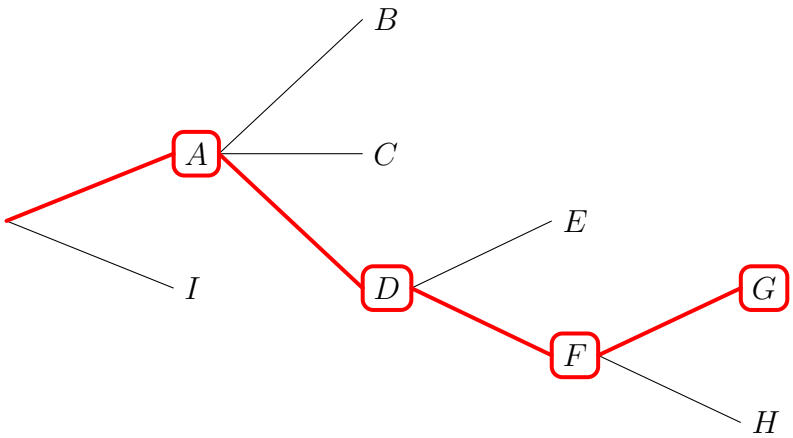


Avec `frame = none` on obtient l'arbre ci-dessous.



9. Utiliser les noms automatiques donnés par `forest`

Voyons comment obtenir le résultat suivant en indiquant tous les noeuds via les noms automatiques fabriqués par `forest`.



Le rendu précédent a été obtenu via le code suivant.

16. Ce disque est en fait un carré aux coins arrondis autour d'un texte vide.

```
\begin{probatree}
    [
        [A
            [B]
            [C]
            [D
                [E]
                [F
                    [G]
                    [H]
                ]
            ]
        ]
        [I]
    ]
    %
    \aptreeFocus[lcol = red]{ | 1 | 13 | 132 | 1321}
\end{probatree}
```

Voici comment s'y prendre.

1. On utilise `\aptreeFocus` au lieu de `\ptreeFocus` où le préfixe `a` est pour `a`-uto.

   ATTENTION ! *Il n'est pas possible de modifier les couleurs du texte et du fond des noeuds car* `\aptreeTextOf`*,* `\aptreeNodeColor` *et* `\aptreeNodeNewText` *n'ont pas pu être implémentées. Par contre les macros* `\aptreeComment` *et* `\aptreeFrame` *ainsi que* `\aptreeTagLevel` *et* `\aptreeTagLeaf` *existent sans limitation.*

2. Chaque nom automatique[17] fabriqué par `forest` commence par `!` . Ce caractère spécial n'est pas à indiquer car il sera ajouté automatiquement en coulisse.
3. La racine est nommée `!` par `forest` d'où le `|` seul au début de l'argument de `\aptreeFocus*` ci-dessus afin d'indiquer un texte vide comme nom du tout premier noeud.
4. Pour voir ce qu'il faut faire pour un noeud autre que la racine, considérons par exemple `1321`. On indique en fait le chemin à suivre après la racine pour arriver au noeud voulu.

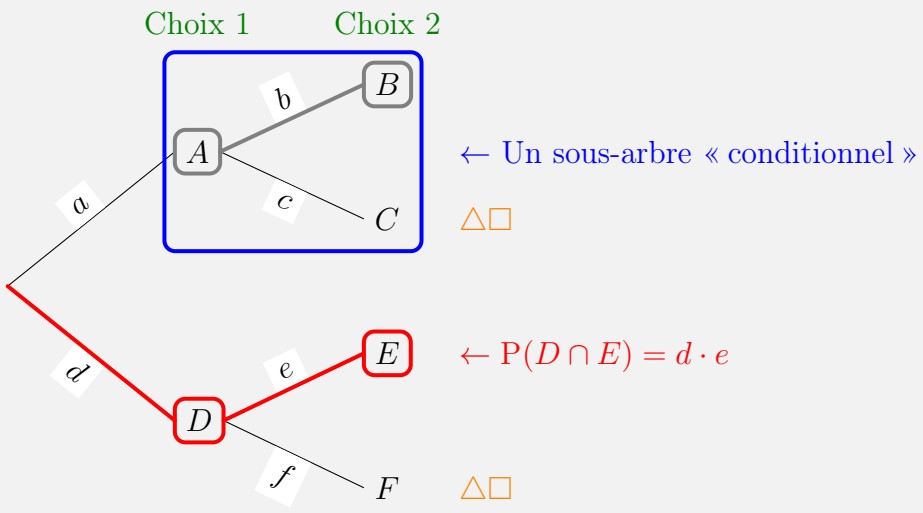
   - Aller d'abord au $\boxed{1}^{\text{er}}$ noeud du niveau 1 qui ici est A.
   - Aller ensuite au $\boxed{3}^{\text{e}}$ noeud du niveau 2 qui ici est D.
   - Aller après au $\boxed{2}^{\text{e}}$ noeud du niveau 3 qui ici est F.
   - Aller enfin au $\boxed{1}^{\text{er}}$ noeud du niveau 4 qui ici est G.
   - On obtient finalement notre noeud nommé $\boxed{1321}$.

Remarque. Dans la mesure du possible, utiliser les noms automatiques facilite la maintenance des arbres sur le long terme. Si on reprend le tout premier exemple d'arbre décoré, il est bien plus simple de faire comme suit car on ne touche pas à la structure minimale du code de l'arbre. On a même utilisé `\ptreeFrame` au lieu de la clé `pframe` dans l'arbre.

```
\begin{probatree}
    [
        [A, apweight = a
            [B, apweight = b]
            [C, bpweight = c]
        ]
```

17. Ce sont en fait des noms relativement à la racine de l'arbre.

```
        [D, bpweight = d
            [E, apweight = e]
            [F, bpweight = f]
        ]
    ]
    % Étiquettes pour les niveaux.
    \aptreeTagLevel[tcol = green!50!black, dy = .75mm]
                   {1}{Choix 1}
    \aptreeTagLevel[tcol = green!50!black, dy = .75mm]
                   {11}{Choix 2}
     % Mettre en valeur un chemin.
    \aptreeFocus[lcol = gray,frame = start]
                {1 | 11}
   % Mettre en valeur un chemin et le commenter.
    \aptreeFrame{1}{11}{12}
    \aptreeComment[tcol = blue]%
                  {1}{$\leftarrow$ Un sous-arbre \og conditionnel \fg}
    % Comme avant.
    \aptreeFocus[lcol = red]
                { | 2 | 21}
    \aptreeComment[tcol = red]
                  {21}{$\leftarrow \proba{D \cap E} = d \cdot e$}
     % Décorer des feuilles.
    \aptreeTagLeaf[tcol = orange]
                  {12 | 22}{$\triangle \square$}
\end{probatree}
```



← Un sous-arbre « conditionnel »

△□

← $\mathrm{P}(D \cap E) = d \cdot e$

△□

Chapitre 8

Arithmétique

I. Ensembles classiques

Se reporter à la section 5. page 14 où sont présentées les macros `\NN`, `\ZZ`, `\QQ` ainsi que `\PP` pour indiquer l'ensemble des naturels, celui des entiers relatifs, celui des fractions rationnelles et enfin celui des nombres premiers.

II. Opérateurs de base

Pour des raisons d'expressivité des codes LaTeX, les opérateurs binaires `\divides`, `\ndivides` et `\modulo` ont été ajoutés comme alias respectifs de `\mid`, `\nmid` et `\bmod` qui sont proposés par le package `amssymb`. Un opérateur `\nequiv` a été aussi ajouté.

```
$10 \divides 150$ au lieu de
$10 | 150$

$10 \ndivides 154$ au lieu de
$10 \not| 154$

$    a \nequiv b \modulo p
\iff p \ndivides (a - b)$.
```

$10 \mid 150$ au lieu de $10|150$
$10 \nmid 154$ au lieu de $10 \not| 154$
$a \nequiv b \bmod p \iff p \nmid (a-b)$.

III. Fonctions nommées spéciales

Deux fonctions nommées `\pgcd` et `\ppcm` utiles au francophone ont été ajoutées ainsi que la fonction `\lcm` pour les anglophones car cette dernière n'est pas disponible par défaut.

```
$\pgcd x = \gcd x$ et $\ppcm x = \lcm x$
```

$\pgcd x = \gcd x$ et $\ppcm x = \lcm x$

IV. Fractions continuées

1. Fractions continuées standard

Exemple 1 – Version longue

Dans l'exemple suivant, la notation en ligne semble être due à Alfred Pringsheim. La notation à gauche utilise toujours le maximum d'espace pour améliorer la lisibilité.

```
 $\contfrac {u_0 | u_1 | u_2 | \dots | u_n}
= \contfrac*{u_0 | u_1 | u_2 | \dots | u_n}$
```

$$u_0 + \cfrac{1}{u_1 + \cfrac{1}{u_2 + \cfrac{1}{\dots + \cfrac{1}{u_n}}}} = u_0 + \cfrac{1}{\left|u_1\right|} + \cfrac{1}{\left|u_2\right|} + \cfrac{1}{\left|\dots\right|} + \cfrac{1}{\left|u_n\right|}$$

Exemple 2 – Version Courte

Les macros `\scontfrac` et `\scontfrac*` donnent directement la fraction. Le `s` est pour `s`-hort soit *« court »* en anglais.

```
 $\scontfrac {u_0 | u_1 | u_2 | \dots | u_n}
= \scontfrac*{u_0 | u_1 | u_2 | \dots | u_n}$
```

$$\cfrac{1}{u_0 + \cfrac{1}{u_1 + \cfrac{1}{u_2 + \cfrac{1}{\dots + \cfrac{1}{u_n}}}}} = \cfrac{1}{\left|u_0\right|} + \cfrac{1}{\left|u_1\right|} + \cfrac{1}{\left|u_2\right|} + \cfrac{1}{\left|\dots\right|} + \cfrac{1}{\left|u_n\right|}$$

2. Fractions continuées généralisées

Exemple 1 – Version longue

Voici comment écrire une fraction continuée généralisée.

```
 $\displaystyle
  \contfracgene {a | b | c | d | e | f | \dots | y | z}
= \contfracgene*{a | b | c | d | e | f | \dots | y | z}$
```

$$a + \cfrac{b}{c + \cfrac{d}{e + \cfrac{f}{\dots + \cfrac{y}{z}}}} = a + \cfrac{b}{\left|c\right|} + \cfrac{d}{\left|e\right|} + \cfrac{f}{\left|\dots\right|} + \cfrac{y}{\left|z\right|}$$

Exemple 2 – Version courte

Les macros `\scontfracgene` et `\scontfracgene*` donnent directement la fraction. Le `s` est de nouveau pour `s`-hort.

```
$\displaystyle
 \scontfracgene {a | b | c | d | e | \dots | y | z}
= \scontfracgene*{a | b | c | d | e | \dots | y | z}$
```

$$\displaystyle \scontfracgene {a | b | c | d | e | \dots | y | z} = \scontfracgene*{a | b | c | d | e | \dots | y | z}$$

3. L'opérateur $\mathcal{K}$

Exemple 1

La notation suivante est proche de celle qu'utilisait Carl Friedrich Gauss.

```
$\displaystyle
 \contfracope_{k=1}^{n} (b_k:c_k)
= \scontfracgene{b_1 | c_1 | b_2 | c_2 | b_3 | \dots | b_n | c_n}$
```

$$\displaystyle \contfracope_{k=1}^{n} (b_k:c_k) = \scontfracgene{b_1 | c_1 | b_2 | c_2 | b_3 | \dots | b_n | c_n}$$

Remarque. La lettre $\mathcal{K}$ vient de "kettenbruch" qui signifie "fraction continuée" en allemand.

Exemple 2

```
$\displaystyle
 u_0 + \contfracope_{k=1}^{n} (1:u_k)
= \contfrac{u_0 | u_1 | u_2 | \dots | u_n}$
```

$$\displaystyle u_0 + \contfracope_{k=1}^{n} (1:u_k) = \contfrac{u_0 | u_1 | u_2 | \dots | u_n}$$

Chapitre 9

Algèbre linéaire

I. Matrices via `nicematrix`

Le gros du boulot est fait par l'excellent package `nicematrix`[1]. `tnslinalg` propose en plus une macro à but pédagogique : voir la section 2. page 105. Veuillez vous reporter à la documentation de `nicematrix` pour savoir comment s'y prendre en général.

1. Quelques exemples pour bien démarrer

Exemple 1 – Vu dans la documentation de `nicematrix`

```
$\begin{pmatrix}
    1      & \cdots & \cdots & 1      \\
    0      & \ddots &        & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0      & \cdots & 0      & 1
\end{pmatrix}$
```

$$\begin{pmatrix}
    1   & \cdots & \cdots & 1   \\
    0   & \ddots &   & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0   & \cdots & 0   & 1
\end{pmatrix}$$

Exemple 2

```
$\begin{vmatrix}
    1      & \cdots & \cdots & 1      \\
    0      & \ddots &        & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0      & \cdots & 0      & 1
\end{vmatrix}$
```

$$\begin{vmatrix}
    1   & \cdots & \cdots & 1   \\
    0   & \ddots &   & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0   & \cdots & 0   & 1
\end{vmatrix}$$

Exemple 3

```
$\begin{bmatrix}
    1      & \cdots & \cdots & 1      \\
    0      & \ddots &        & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0      & \cdots & 0      & 1
\end{bmatrix}$
```

$$\begin{bmatrix}
    1   & \cdots & \cdots & 1   \\
    0   & \ddots &   & \vdots \\
    \vdots & \ddots & \ddots & \vdots \\
    0   & \cdots & 0   & 1
\end{bmatrix}$$

1. On impose l'option `transparent`.

Exemple 4 – Vu dans la documentation de `nicematrix`

```
$\begin{pNiceMatrix}[name = mymatrix]
     1 & 2 & 3 \\
     4 & 5 & 6 \\
     7 & 8 & 9
 \end{pNiceMatrix}$

 \tikz[remember picture,
       overlay]
 \draw[red]
     (mymatrix-2-2) circle (2.5mm);
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Exemple 5 – Vu dans la documentation de `nicematrix`

```
$\left(
     \begin{NiceArray}{cccc:c}
         1  & 2  & 3  & 4  & 5  \\
         6  & 7  & 8  & 9  & 10 \\
         11 & 12 & 13 & 14 & 15
     \end{NiceArray}
 \right)$
```

$$\left(\begin{array}{cccc:c} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{array}\right)$$

Exemple 6 – Proposition de l'auteur de `nicematrix` suite à une discussion par mail

```
% Besoin du package ``ifthen``.
\newcommand\aij{%
  a_{\arabic{iRow}\arabic{jCol}}%
}

$\begin{bNiceArray}{*{5}{>{%
     \ifthenelse{\value{iRow}>0}{\aij}{}%
 }c}}[
     first-col,
     first-row,
     code-for-first-row
       = \text{\textbf{\arabic{jCol}}},
     code-for-first-col
       = \text{\textbf{\arabic{iRow}}}
   ]
       & & & & & \\
       & & & & & \\
       & & & & &
 \end{bNiceArray}$
```

$$\begin{array}{c} \\ \mathbf{1} \\ \mathbf{2} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{bmatrix} \end{array}$$

Exemple 7 – Avec des calculs automatiques

```
\newcounter{cntaij}
\newcommand\aij{%
    \setcounter{cntaij}{\value{iRow}}%
    \addtocounter{cntaij}{\value{jCol}}%
    \addtocounter{cntaij}{-1}%
    \arabic{cntaij}%
}

Si $a_{ij} = i + j - 1$ alors

$(a_{ij})_{1 \leq i \leq 3 , 
           1 \leq j \leq 5}
 =
 \begin{bNiceArray}{*{5}{>{\aij}c}}
     & & & & \\
     & & & & \\
     & & & &
 \end{bNiceArray}$
```

Si $a_{ij} = i + j - 1$ alors

$$(a_{ij})_{1 \leq i \leq 3, 1 \leq j \leq 5} = \begin{bNiceArray}{*{5}{>{\aij}c}} & & & & \\ & & & & \\ & & & & \end{bNiceArray}$$

2. Calcul expliqué d'un déterminant 2×2

Exemple 1 – Versions matricielles

```
$\calcdettwo{a}{b}%
            {c}{d}$
ou
$\calcdettwo[loop]{a}{b}%
                  {c}{d}$
ou
$\calcdettwo[arrows]{a}{b}%
                    {c}{d}$
ou
$\calcdettwo[cross]{a}{b}%
                   {c}{d}$
```

$\calcdettwo{a}{b}{c}{d}$ ou $\calcdettwo[loop]{a}{b}{c}{d}$ ou $\calcdettwo[arrows]{a}{b}{c}{d}$ ou $\calcdettwo[cross]{a}{b}{c}{d}$

Exemple 2 – Versions développées

Ci-dessous `exp` est pour `exp`-and soit *« développer »* en anglais, `c` pour `\cdot` et enfin `t` pour `\times`.

```
$\calcdettwo[exp]{a}{b}%
                 {c}{d}$

$\calcdettwo[texp]{a}{b}%
                  {c}{d}$

$\calcdettwo[cexp]{a}{b}%
                  {c}{d}$
```

$\calcdettwo[exp]{a}{b}{c}{d}$

$\calcdettwo[texp]{a}{b}{c}{d}$

$\calcdettwo[cexp]{a}{b}{c}{d}$

Chapitre 10

Polynômes et séries formelles

I. Ensembles classiques de nombres

Se reporter aux sections 4. page 14 et 5. page 14 où sont présentées diverses macros pour indiquer des ensembles classiques en algèbre.

II. Polynômes

Exemple 1 – Polynômes

```
$\setpoly{R}{X}$ ou
$\setpoly{R}{X | Y | Z}$
```

$R[X]$ ou $R[X;Y;Z]$

Exemple 2 – Fractions polynômiales

```
$\setpolyfrac{Q}{T}$ ou
$\setpolyfrac{Q}%
             {S_1 | S_2 | \dots | S_k}$
```

$Q(T)$ ou $Q(S_1;S_2;\dots;S_k)$

III. Séries formelles classiques

Exemple 1 – Séries formelles

```
$\setserie{C}{X}$ ou
$\setserie{C}{T | O | P}$
```

$C[[X]]$ ou $C[[T;O;P]]$

Exemple 2 – Corps des fractions de séries formelles

```
$\setseriefrac{Z}{X}$ ou
$\setseriefrac{Z}{Z | T | O | P}$
```

$Z((X))$ ou $Z((Z;T;O;P))$

IV. Polynômes et séries formelles de Laurent

Exemple 1 – Polynômes de Laurent

Ci-dessous, la notation $R\{X_1\,;X_2\}$ n'est pas standard.

```
$\setpolylaurent{R}{X} =
 \setpoly{R}{X | X^{-1}}$

$\setpolylaurent{R}{X_1 | X_2} =
 \setpoly{R}{X_1 | X_1^{-1} %
           | X_2 | X_2^{-1}}$
```

$$R\{X\} = R[X\,;X^{-1}]$$
$$R\{X_1\,;X_2\} = R[X_1\,;X_1^{-1}\,;X_2\,;X_2^{-1}]$$

Exemple 2 – Séries formelles de Laurent

Ci-dessous, la notation $Q\{\{X_1\,;X_2\}\}$ n'est pas standard.

```
$\setserielaurent{Q}{X} =
 \setserie{Q}{X | X^{-1}}$

$\setserielaurent{Q}{X_1 | X_2} =
 \setserie{Q}{X_1 | X_1^{-1} %
            | X_2 | X_2^{-1}}$
```

$$Q\{\{X\}\} = Q[[X\,;X^{-1}]]$$
$$Q\{\{X_1\,;X_2\}\} = Q[[X_1\,;X_1^{-1}\,;X_2\,;X_2^{-1}]]$$

Chapitre 11

Historique

Nous ne donnons ici qu'un très bref historique récent [1] de `tnsmath` à destination de l'utilisateur principalement. Pour une vision précise de tous les changements, il faudra se référer sur `github` aux dossiers `change-log` des codes sources des différents packages agrégés dans `tnsmath`.

2021-03-03 Nouvelle version mineure `2.2.0-beta`.

Géométrie [0.6.0-beta]

- **Points et lignes :** `\pgline`, `\hgline` et `\pghline` ont été renommées `\gpline`, `\ghline` et `\gphline` respectivement.

Suites [0.2.0-beta]

- **Changements internes :** les packages `bm` et `yhmath` étaient chargés à tort.

2021-03-02 Nouvelle version mineure `2.1.0-beta`.

Théorie générale des ensembles [0.4.0-beta]

- **Crochets doubles :** ajout des symboles `\Zlbracket` et `\Zrbracket` [2].

2021-03-01 Nouvelle version majeure `2.0.0-beta`.

Géométrie [0.5.0-beta]

- **Vecteur :** suppression des macros `\colicriteria` et `\vcolicriteria` car elles sont inutiles.

Algèbre linéaire [0.2.0-beta]

1. On ne va pas au-delà de un an depuis la dernière version.
2. En coulisse, on commence à tenter de gérer plus proprement les symboles particuliers utilisés.

- **Mise à jour imposée par nicematrix :** les options `renew-dots` et `renew-matrix` remplacent l'option dépréciée `transparent`.

Méthodes formelles en logique [1.0.0-beta]

- **Calculs et démonstrations pas à pas.**
  - L'environnement `demoexplain` a été renommé `demotab`.
  - L'environnement `explain` a été renommé `stepcalc`.
  - Le nouvel environnement `stepcalc` dispose d'un nouveau style `ar*` pour placer les opérateurs dans la marge.
- **Logique.**
  - `\liesimp` a été renommée `\becauseof`.
  - La façon de décorer les opérateurs de comparaison ou de logique a changé et est plus généraliste.
  - Les macros `\eq*` et `\neq*` proposent des versions symboliques pour certains textes de décoration.

2020-09-02 Nouvelle version mineure `1.10.0-beta`.

Géométrie [0.4.0-beta]

- **Vecteur :** retour de la macro `\norm*`.

Algèbre linéaire [0.1.0-beta]

- **Déterminant 2×2 :** changement de l'API.
  - `\calcdettwo` sert à obtenir au choix les versions développée ou bien celles matricielles avec pour décorations supplémentaires une croix fléchée ou non.
  - Suppression de `\calcdettwo*`.

Probabilité [0.9.0-beta]

- **Calculs de l'espérance dans le cas fini :** `\calcexpval` rend facile la définition d'une variable aléatoire finie avec la possibilité de détailler le calcul de son espérance.
- **Arbre de probabilités :** pas mal de changements dans l'API et quelques nouveautés.
  - `\begin{probatree}<hideall> ... \end{probatree}` remplace l'usage de l'environnement `probatree*`.
  - `\begin{probatree}<showall> ... \end{probatree}` remplace l'usage de l'environnement `probatree**`.

— De nouvelles macros permettent d'agir sur le texte des noeuds.

1. `\ptreeTextOf` renvoie le texte d'un noeud.
2. `\ptreeNodeColor` change les couleurs du texte et/ou du fond d'un noeud.
3. `\ptreeNodeNewText` change le texte en plus éventuellement les couleurs du texte et/ou du fond d'un noeud.

— `\ptreeFocus` a évolué [3].

1. `\ptreeFocus[frame = start]` remplace l'ancien `\ptreeFocus`.
2. `\ptreeFocus[frame = none]` remplace `\ptreeFocus**` qui n'existe plus.
3. `\ptreeFocus[frame = nostart]` est utilisé par défaut et remplace `\ptreeFocus*` qui n'existe plus. On obtient dans ce cas une mise en valeur encadrant tous les noeuds sauf le tout 1er.
4. Les clés optionnelles `lcol`, `tcol` et `bcol` permettent de choisir la couleur des arrêtes et des cadres, celle du texte et enfin celle du fond.

— `tcol` remplace l'ancien `col` de `\ptreeComment` et `\aptreeComment`.

— `lcol` remplace l'ancien `col` de `\ptreeFrame`, `\aptreeFrame` et `\aptreeFocus`.

Probabilité [0.10.0-beta]

- **Généralités.**
  — P est le nom par défaut d'une probabilité.
  — Les noms des probabilités, des espérances, des variances et des écarts-types utilisent tous une police droite via `\mathrm` en coulisse.
- **Espérance d'une variable aléatoire finie.**
  — `\expval` est utilisée en coulisse pour rédiger les espérances dans les détails des calculs.
  — L'ordre des produits dans le calcul numérique est le même que celui dans la somme formelle.

2020-08-26 Nouvelle version mineure `1.9.0-beta`.

Analyse [0.8.0-beta]

- **Fonctions avec un paramètre optionnel :** les macros `\lg` et `\ln` peuvent s'utiliser avec une base.

Arithmétique [0.2.0-beta]

- **Fractions continuées :** ajout de `\scontfrac`, `\scontfrac*`, `\scontfracgene` et `\scontfracgene*` qui donnent juste la partie fractionnaire.

Géométrie [0.3.0-beta]

3. Malheureusement ces changements n'ont pas pu être faits pour `\aptreeFocus` principalement car l'équivalent automatique de `\ptreeTextOf` n'a pu être implémenté.

- **Vecteur :** ajout de la macro `\pvect` pour ne pas avoir à taper `\pt`.
- **Produit vectoriel et déterminant de deux vecteurs :** nouveau changement de l'API.
  — `\calccrossprod`, `\vcalccrossprod`, `\calcdetplane` et `\vcalcdetplane` permettent de tracer une croix fléchée ou non à la place de la boucle fléchée.
  — `\coordcrossprod` a été supprimé. Il faut à la place utiliser l'une des options `exp`, `texp` et `cexp` de `\vcalccrossprod` et `\calccrossprod`.
  — Plus aucune version étoilée simple ou double pour `\calccrossprod`, `\vcalccrossprod`, `\calcdetplane` et `\vcalcdetplane`.

Probabilité [0.8.0-beta]

- **Arbre de probabilités :** le mode mathématique est activé par défaut pour les noms des noeuds et les poids *(plus besoin de taper plein de `$`)*.

2020-08-09 Nouvelle version mineure `1.8.0-beta`.

Méthodes formelles en logique [0.1.0-beta]

- **Macros « textuelles » :** le préfixe `txt` remplace l'ancien `text`.

Probabilité [0.6.0-beta]

- **Arbre de probabilités.**
  — `\aptreeFocus`, `\aptreeFocus*` et `\aptreeFocus**` permettent d'utiliser le système de nommage automatique des noeuds proposé par `forest`.
  — Il en va de même pour `\aptreeComment` et `\aptreeFrame`.

Probabilité [0.7.0-beta]

- **Arbre de probabilités.**
  — Utilisation obligatoire de `col=...` pour indiquer une couleur à toutes les macros de décoration.
  — Les macros `\ptreeComment`, `\ptreeComment*`, `\aptreeComment` et `\aptreeComment*` ont deux clés `dx` et `dy` pour indiquer un décalage relatif.
  — Ajout de l'environnement `probatree**` qui force l'affichage de tous les poids !

Suites [0.1.0-beta]

- **Comparaison asymptotique :** ce sont de vrais opérateurs mathématiques qui sont définis en coulisse *(du coup les macros `\bigO`, `\smallO`, `\bigOmega` et `\bigTheta` n'ont plus d'argument)*.

2020-08-06 Nouvelle version mineure `1.7.0-beta`.

Analyse [0.6.0-beta]

- **Définition explicite d'une fonction.**
  - Une nouvelle macro `\txfuncdef` produit une version textuelle courte.
  - Omission possible des ensembles, via des arguments vides, quand on utilise `\funcdef[h]` ou `\txfuncdef`.
- **Fonctions avec un paramètre :** les macros `\expb` et `\logb` ont été remplacées par `\exp` et `\log` qui ont un argument optionnel pour indiquer éventuellement une base.
- **Dérivation partielle :** `\pder[ei]` fonctionne maintenant aussi avec des variables indexées.

Analyse [0.7.0-beta]

- **Définition explicite d'une fonction :** les macros `\funcdef` et `\txtfuncdef` ont été déplacées dans `tnssets` qui est disponible sur `https://github.com/typensee-latex/tnssets.git`.

Géométrie [0.2.0-beta]

- **Critère de colinéarité :** ajout de la macro `\colicriteria`.
- **Produit vectoriel :** changement de l'API.
  - `\vcalccrossprod*` devient `\vcalccrossprod**`.
  - `\vcalccrossprod*` dessine des produits en croix à la place des boucles.

Probabilité [0.4.0-beta]

- **Arbre :** possibilité de mettre en valeur un chemin via `\ptreeFocus`, `\ptreeFocus*` ou `\ptreeFocus**`.

Probabilité [0.5.0-beta]

- **Arbre de probabilités.**
  - `\ptreeFocus`, `\ptreeFocus*` et `\ptreeFocus**` fonctionnent avec un multi-argument pour pourvoir indiquer un chemin sur plusieurs noeuds.
  - Suppression de la clé `\pcomment`.
  - Ajout des macros `\ptreeComment` et `\ptreeComment*` qui simplifient la saisie.

Théorie générale des ensembles [0.2.0-beta]

- **Composition d'applications.**
  - `\compo` est un opérateur de composition de deux applications.
  - `\multicomp` permet d'indiquer des compositions successives d'une application par elle-même.

Théorie générale des ensembles [0.3.0-beta]

- **Définition explicite d'une fonction :** intégration de deux macros proposées avant par `tnsana` disponible sur `https://github.com/typensee-latex/tnsana.git`.
  - `\funcdef` peut produire trois versions symboliques.
  - `\txtfuncdef` produit une version textuelle courte.
- **Fonctions spéciales.**
  - Ajout de `\id` pour la fonction identité.
  - Ajout de `\caract` et `\caractone` pour deux versions de la fonction caratéristque d'un ensemble.

2020-07-25 Nouvelle version mineure `1.6.0-beta`.

Probabilité [0.3.0-beta]

- **Arbre.**
  - Ajout du style `pcomment` pour placer du texte à la droite d'une feuille.
  - Le style `frame` a été renommé `pframe`.

2020-07-23 Nouvelle version mineure `1.5.0-beta`.

Probabilité [0.2.0-beta]

- **Arbre :** ajout de la macro `\ptreeFrame` pour tracer facilement des sous cadres non « finaux ».

2020-07-22 Nouvelle version mineure `1.4.0-beta`.

Analyse [0.5.0-beta]

- **Définition explicite d'une fonction :** ajout de `\funcdef`.

Probabilité [0.1.0-beta]

- **Probabilité conditionnelle :** `\probacondexp` renommée en `\eprobacond`.
- **Évènement contraire :** ajout de `\nevent`.
- **variance et écart-type :** ajout de `\var` et `\stddev`.

2020-07-21 Nouvelle version mineure `1.3.0-beta` : passage de `lymath` à `tnsmath`.

Analyse [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Analyse [0.1.0-beta]

- **Fonctions nommées :** `\ppcm` et `\pgcd` ont été déplacées dans `tnsarith` disponible sur `https://github.com/typensee-latex/tnsarith.git`.

Analyse [0.2.0-beta]

- **Symboles :** les nouvelles macros `\symvar` et `\symvar*` produisent un disque plein et un carré plein permettant par exemple d'indiquer symboliquement une ou des variables.

Analyse [0.3.0-beta]

- **Dérivation.**
  - — Dérivation pointée à la physicienne via `d` et `bd` deux nouvelles options de `\der`.
  - — Dérivation partielle indexée du type u_{xxy} à la physicienne via `ei` une nouvelle option de `\pder`.
- **Tableaux de signe et de variation.**
  - — Ajout de `\backLine` pour changer la couleur de fond d'une ou plusieurs lignes.
  - — `\graphSign` propose des fonctions de référence *(sans paramètre)*.

Analyse [0.4.0-beta]

- **Limite :** ajout de `\limit` pour l'écriture de limites de fonctions à une seule variable.
- **Dérivation :** par souci de cohérence, il faudra taper `\der{f}{x}{n}` au lieu de l'ancien `\der{f}{n}{x}`.
- **Intégration :** par souci de cohérence, il faudra taper `\integrate{f}{x}{a}{b}` au lieu de l'ancien `\integrate{a}{b}{f}{x}`. Il en va de même pour `\hook`.

Arithmétique [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Arithmétique [0.1.0-beta]

- **Fonctions nommées :** ajout de `\pgcd`, `\ppcm` et `\lcm`.

Géométrie [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Géométrie [0.1.0-beta]

- **Produit scalaire :** trois nouvelles options pour `\dotprod` et `\vdotprod`.
  - `p` et `sp` donnent une écriture parenthésée.
  - `b` utilise une puce au lieu d'un point.
- **Produit vectoriel :** un nouvel argument optionnel pour `\crossprod` et `\vcrossprod` afin d'obtenir aussi une mise en forme avec le symbole $\times$.

Algèbre linéaire [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Méthode formelle en logique [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Polynômes et séries formelles [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Probabilité [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Suites [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

Théorie générale des ensembles [0.0.0-beta]

- Simple migration depuis l'ancien code de `lymath`.

2020-07-05 Nouvelle version mineure `1.2.0-beta`.

- **Tableaux de signe et de variation :** il est maintenant possible d'ajouter des graphiques expliquant le signe d'une fonction affine ou d'une fonction trinômiale du 2^{e} degré.

2020-06-27 Nouvelle version mineure `1.1.0-beta`.

- **Logique.**
  - — Suppression des environnements `aexplain` et `aexplain*`.

    Leurs mises en forme restent accessibles respectivement via les options `style = ar` et `style = sar` de l'environnement `explain`.
  - — L'environnement `explain` propose des options de type `clé-valeur`.
  - — Ajout de petits commentaires pour les étapes via `\comthis` et si besoin `\comthis*`.
  - — Modification de `\explnext*` pour le mode universitaire sans flèche via l'ajout de `\exptxtupdown` qui gère la mise en forme de deux explications non vides. Ceci a pour effet l'alignement du rendu avec l'opérateur.
  - — Les environnements `demoexplain` et `demoexplain*` utilisent `longtable` en coulisse afin de pouvoir écrire un tableau sur plusieurs pages.

2020-06-21 Nouvelle version majeure `1.0.0-beta`.

- Le changement vers une nouvelle version majeure se justifie par de nouvelles règles strictes pour définir les signatures des macros. À l'avenir ces règles seront appliquées tout le temps sauf dans de très rares cas.

  Ceci a créé beaucoup de nouvelles façons de rédiger.

- **Algèbre linéaire :** `\calcdettwo` est un outil pédagogique pour expliquer le calcul d'un déterminant 2×2.

- **Analyse.**
  - — Calcul intégral.
    - → `\integrate` et `\dintegrate` servent à rédiger des intégrales simples.
    - → `\hook` s'utilise différemment : on doit taper `\hook{a}{b}{F(x)}{x}` avec l'obligation de donner la variable.
    - → `\hook` marche avec des options. Du coup `\vhook` and `\vhook*` ont été supprimées mais les mises en forme correspondantes existent toujours via `\hook[r]` et `\hook[sr]`.
  - — Dérivées totales.
    - → Il ne reste plus que trois macros : `\sder`, `\der` et `\derope`.
    - → `\sder` et `\sder[e]` remplacent `\derpow*` et `\derpow` avec en plus la possibilité d'ajout automatique de parenthèses pour faire comme avec `\derpar*`, `\derpar`, `\sderpar*` et `\sderpar` avant.
    - → `\der` s'utilise en indiquant la variable de dérivation. Cette macro propose différentes options pour différentes mises en forme *(on peut toujours obtenir la même chose que ce que proposaient `\derfrac`, `\derfrac*` et `\dersub`)*.
    - → `\derope` sert à écrire un opérateur fonctionnel.
  - — Dérivées partielles.
    - → Il ne reste plus que deux macros : `\pder` et `\pderope`.

→ `\pder` possède des options permettant d'obtenir le même résultat qu'avec les anciennes macros `\partialfrac` et `\partialsub`.

→ La mise en forme proposée par `\partialprime` n'a pas été gardée.

→ `\pderope` sert à écrire un opérateur fonctionnel.

- **Arithmétique.**
  - — `\notdivides` a été renommée `\ndivides`.
  - — Ajout de `\nequiv`.

- **Géométrie.**
  - — `\calcdetplane`, `\calcdetplane*`, `\vcalcdetplane` et `\vcalcdetplane*` permettent de détailler le calcul du déterminant de deux vecteurs en dimension 2 *(utile pour le critère de colinéarité).*
  - — `\calccrossprod`, `\calccrossprod*`, `\vcalccrossprod` et `\vcalccrossprod*` permettent de détailler le calcul du produit vectoriel de deux vecteurs en dimension 3.
  - — `\coordcrossprod` permet d'obtenir formellement les coordonnées d'un produit vectoriel avec des formats du type $(yz' - zy', zx' - xz', xy' - yx')$.
  - — Angles orientés.
    - → `\angleorient` devient l'unique macro pour rédiger des angles orientés de différentes façons via des options.
    - → `\angleorient*` a été remplacée par `\angleorient[sp]`.
    - → `\hangleorient` a été remplacée par `\angleorient[h]`.
    - → `\hangleorient*` a été remplacée par `\angleorient[sh]`.
  - — Produit scalaire.
    - → `\dotprod` devient l'unique macro pour rédiger des produits scalaires de différentes façons grâce à des options.
    - → `\adotprod` a été remplacée par `\dotprod[a]`.
    - → `\adotprod*` a été remplacée par `\dotprod[sa]`.
  - — Coordonnées.
    - → Il faut passer via l'une des macros : `\coord`, `\pcoord`, `\pcoord*`, `\vcoord` et `\vcoord*`. Toutes ces macros proposent des options pour choisir la mise en forme : des parenthèses en mode horizontal, des crochets en mode vertical...
    - → `\coord` est pour des coordonnées seules.
    - → `\pcoord` et `\pcoord*` sont pour un point avec ses coordonnées.
    - → `\vcoord` et `\vcoord*` sont pour un vecteur avec ses coordonnées.
    - → La version étoilée `\coord*` a été supprimée.
  - — Norme.
    - → `\norm` fonctionne maintenant avec des options. Du coup `\norm*` a été supprimée mais la mise en forme correspondante existe toujours via `\norm[s]`.

→ `\vnorm` évite d'avoir à utiliser `\vect` pour des vecteurs juste nommés.

- **Logique.**
  - La macro `\explain` a été supprimée pour être remplacée par l'environnement `explain` qui est redoutable d'efficacité pour détailler un calcul ou un raisonnement simple.
  - `explain` est complété par les deux environnements `aexplain` et `aexplain*` qui utilisent des flèches pour les indications.
  - Les environnements `demoexplain` et `demoexplain*` permettent de rédiger de *« vraies »* démonstrations via des tableaux efficaces.
  - Toutes les macros négatives avec pour préfixe `not` auparavant utilisent maintenant juste le préfixe `n`.
  - Tous les opérateurs de comparaison ont une version négative.

- **Probabilités.**
  - `\probacond**` et `\dprobacond**` sont devenues `\probacondexp*` et `\probacondexp` respectivement.
  - `\expval` est une nouvelle macro pour l'écriture symbolique de l'espérance d'une variable aléatoire.

2020-06-08 Nouvelle version mineure `0.7.0-beta`.

- **Analyse.**
  - Pour éviter des conflits avec d'autres packages les renommages suivants ont dû être faits.
    → `\ch` et `\ach` sont devenus `\fch` et `\afch` où `f` est pour `f`-rench.
    → `\sh` et `\ash` sont devenus `\fsh` et `\afsh`.
    → `\th` et `\ath` sont devenus `\fth` et `\afth`.
  - Les macros `\acosh`, `\asinh` et `\atanh` ont été ajoutées.
  - `\derpar` et `\derpar*` servent à rédiger des dérivées avec des parenthèses extensibles. En coulisse, `\derpow` et `\derpow*` sont appelées.
    Pour utiliser des parenthèses non extensibles, on passera par `\sderpar` et `\sderpar*` où `s` est pour `s`-mall.
  - Ajout de `\stdint` pour rendre public l'opérateur intégral proposé par défaut par LaTeX.
- **Géométrie.**
  - La macro `\pts` a été supprimée car sans signification sémantique puisqu'un point peut être nommé avec deux lettres.
  - Les macros `\gline` et `\pgline` servent à indiquer des droites définies par deux points.
  - La macro `\hgline`, avec `h` pour `h`-alf, est pour les demi-droites définies par deux points.
  - La macro `\segment` est utile pour les segments définis par deux points.

- **Logique.**
  - Pour les inégalités, on peut maintenant utiliser le décorateur `plot`.
  - Ajout des versions négatives des opérateurs logiques verticaux.
- **Probabilités.**
  - Ajout de `\proba` pour écrire des probabilités.
  - Le comportement de `\probacond` a été modifié pour le rendre plus logique.

Chapitre 12

Toutes les fiches techniques

I. Théorie générale des ensembles

1. Ensembles

Ensembles versus accolades

`\setgene[#opt]{#1}`

— `Option:` la valeur par défaut est `b`. Voici les différentes valeurs possibles.

1. `b` : on utilise des accolades extensibles.
2. `sb` : on utilise des accolades non extensibles.

— `Argument:` la définition de l'ensemble.

— `Argument:` la définition de l'ensemble.

2. Ensembles

Ensembles pour la géométrie

`\setgeo{#1}`

— `Argument:` un seul caractère `ASCII` indiquant un ensemble géométrique.

`\setgeo*{#1..#2}`

— `Argument 1:` un seul caractère `ASCII` indiquant $\mathscr{U}$ dans le nom $\mathscr{U}_d$ d'un ensemble géométrique.

— `Argument 2:` un texte donnant d dans le nom $\mathscr{U}_d$ d'un ensemble géométrique.

3. Ensembles

Ensembles probabilistes

`\setproba{#1}`

— `Argument:` un seul caractère `ASCII` majuscule indiquant un ensemble probabiliste.

`\setproba*{#1..#2}`

— `Argument 1:` un seul caractère `ASCII` majuscule indiquant $\mathcal{U}$ dans le nom $\mathcal{U}_d$ d'un ensemble probabiliste.

— `Argument 2:` un texte donnant d dans le nom $\mathcal{U}_d$ d'un ensemble probabiliste.

4. Ensembles

Ensembles pour l'algèbre générale

`\setalge{#1}`

— `Argument:` soit l'une des lettres `h` et `k`, soit un seul caractère `ASCII` majuscule indiquant un ensemble de type anneau ou corps.

`\setalge*{#1..#2}`

— `Argument 1:` un seul caractère `ASCII` indiquant $\mathbb{U}$ dans le nom $\mathbb{U}_d$ d'un ensemble de type anneau ou corps.

— `Argument 2:` un texte donnant d dans le nom $\mathbb{U}_d$ d'un ensemble de type anneau ou corps.

5. Ensembles

Ensembles classiques en mathématiques et en informatique théorique

Ensembles classiques suffixés

`\NN` `\NNs`

`\PP`

`\ZZ` `\ZZn` `\ZZp` `\ZZs` `\ZZsn` `\ZZsp`

`\DD` `\DDn` `\DDp` `\DDs` `\DDsn` `\DDsp`

`\QQ` `\QQn` `\QQp` `\QQs` `\QQsn` `\QQsp`

`\RR` `\RRn` `\RRp` `\RRs` `\RRsn` `\RRsp`

`\CC` `\CCs`

`\HH` `\HHs`

`\OO` `\OOs`

6. Ensembles

Ensembles classiques en mathématiques et en informatique théorique

Des suffixes à la carte

```
\setspecial {#1..#2}
\setspecial*{#1..#2}
```

— `Argument 1:` l'ensemble à "suffixer".

— `Argument 2:` l'un des suffixes `n`, `p`, `s`, `sn` ou `sp`.

7. Intervalles

Intervalles réels – Notation française (?)

Pour toutes les macros ci-dessous, la version non étoilée produit des délimiteurs qui s'étirent si besoin verticalement, tandis que la version étoilée ne le fait pas.

```
\intervalCO {#1..#2}
\intervalCO*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $[a\,;b[$.

— `Argument 2:` borne supérieure b de l'intervalle $[a\,;b[$.

```
\intervalC {#1..#2}
\intervalC*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $[a\,;b]$.

— `Argument 2:` borne supérieure b de l'intervalle $[a\,;b]$.

```
\intervalO {#1..#2}
\intervalO*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $]a\,;b[$.

— `Argument 2:` borne supérieure b de l'intervalle $]a\,;b[$.

```
\intervalOC {#1..#2}
\intervalOC*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $]a\,;b]$.

— `Argument 2:` borne supérieure b de l'intervalle $]a\,;b]$.

8. Intervalles

Intervalles réels – Notation américaine

Pour toutes les macros ci-dessous, la version non étoilée produit des délimiteurs qui s'étirent si besoin verticalement, tandis que la version étoilée ne le fait pas.

```
\intervalCP {#1..#2}
\intervalCP*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $[a\,;b)$.

— `Argument 2:` borne supérieure b de l'intervalle $[a\,;b)$.

```
\intervalP {#1..#2}
\intervalP*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $(a\,;b)$.

— `Argument 2:` borne supérieure b de l'intervalle $(a\,;b)$.

```
\intervalPC {#1..#2}
\intervalPC*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $(a\,;b]$.

— `Argument 2:` borne supérieure b de l'intervalle $(a\,;b]$.

9. Intervalles

Intervalles discrets d'entiers

Pour toutes les macros ci-dessous, la version non étoilée produit des délimiteurs qui s'étirent si besoin verticalement, tandis que la version étoilée ne le fait pas.

```
\ZintervalCO {#1..#2}
\ZintervalCO*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $[\![a\,;b[\![$.

— `Argument 2:` borne supérieure b de l'intervalle $[\![a\,;b[\![$.

```
\ZintervalC {#1..#2}
\ZintervalC*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $[\![a\,;b]\!]$.

— `Argument 2:` borne supérieure b de l'intervalle $[\![a\,;b]\!]$.

```
\ZintervalO {#1..#2}
\ZintervalO*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $]\!]a\,;b[\![$.

— `Argument 2:` borne supérieure b de l'intervalle $]\!]a\,;b[\![$.

```
\ZintervalOC {#1..#2}
\ZintervalOC*{#1..#2}
```

— `Argument 1:` borne inférieure a de l'intervalle $]\!]a\,;b]\!]$.

— `Argument 2:` borne supérieure b de l'intervalle $]\!]a\,;b]\!]$.

10. Unions et intersections en mode ligne

```
\dcap
\dcup
\dsqcup
```

11. Des applications très spéciales

Fonction identité

```
\id
```

12. Des applications très spéciales

Fonction caractéristique

```
\caract
\caractone
```

13. Applications

Cardinal, image et compagnie

```
\card
\card*
```

```
\dom
\codom
\im
```

14. Applications

Application totale, partielle, injective, surjective et/ou bijective

```
\to
\onetoone
\onto
\biject
```

```
\pto
\ponetoone
\ponto
\pbiject
```

15. Applications

Définition explicite d'une fonction

`\funcdef[#opt]{#1..#5}`

— `Option:` la valeur par défaut `u`.

1. `u` : écriture empilée avec un trait vertical.
2. `s` : écriture empilée sans trait vertical.
3. `h` : écriture horizontale en ligne.

— `Argument 1:` la fonction.

— `Argument 2:` la variable.

— `Argument 3:` la formule explicite de définition.

— `Argument 4:` l'ensemble de départ. Cet argument peut être vide si le mode `h` est activé

— `Argument 5:` l'ensemble d'arrivée.

`\txtfuncdef{#1..#5}`

— `Arguments 1..5:` voir les explications données ci-dessus pour la macro `\funcdef`.

16. Applications

Composition

`\compo` `compo = compo-sition`

`\multicompo[#opt]{#1..#2}`

— `Option:` la valeur par défaut est `r`. Voici les différentes valeurs possibles.

1. `r` : écriture utilisant des chevrons. `r = r-after.`
2. `exp` : écriture développée. `exp = exp-and.`
3. `dot` : écriture faussement développée utilisant des points de suspension.

— `Argument 1:` l'application.

— `Argument 2:` le nombre d'applications composées.

II. Méthodes formelles en logique

1. Espace après la négation logique

`\neg`
`\stdneg` (pour retrouver le symbole par défaut)

2. Personnaliser les opérateurs de comparaison

`\eq [#opt]`
`\neq[#opt]`

— `Option:` un texte de décoration éventuel. On obtient des versions symboliques si l'on utilise l'un des textes suivants.

1. `?` 2. `appli` 3. `cons` 4. `def` 5. `id`

`\less [#opt]` `\leq [#opt]` `\gtr [#opt]` `\geq [#opt]`
`\nless[#opt]` `\nleq[#opt]` `\ngtr[#opt]` `\ngeq[#opt]`

— `Option:` un texte de décoration éventuel.

3. Équivalences et implications

Des symboles logiques supplémentaires

```
\iff [#opt]          \implies [#opt]      \becauseof [#opt]
\niff[#opt]          \nimplies[#opt]      \nbecauseof[#opt]
```

— `Option:` un texte de décoration éventuel.

4. Équivalences et implications

Équivalences et implications verticales

```
\viff                \vimplies            \vbecauseof
\nviff               \nvimplies           \nvbecauseof
```

5. Des versions alternatives du quantificateur existentiel

```
\existmulti {#1}
\nexistmulti{#1}
```

— `Argument 1:` une écriture mathématique servant à préciser la portée du quantificateur.

```
\existsone
\nexistsone
```

6. Détailler un raisonnement simple

Détailler un raisonnement simple

```
\begin{stepcalc}[#opt]
   ...
\end{stepcalc}
```

— `Option:` la valeur utilise une syntaxe de type clé-valeur. Voici les différentes clés disponibles.

1. `ope` sert à définir l'opérateur utilisé dans tout l'environnement qui sera rédigé en mode mathématique. La valeur par défaut est `{=}` *(et non juste = attention)*.
2. `style` sert à définir le style de mise en forme. Voici les différentes valeurs possibles.
   (a) `u`, la valeur par défaut, est pour `u`-niversity.
   (b) `ar` est pour `ar`-row.
   (c) `ar*` est similaire à `ar` mais avec l'opérateur dans la marge.
   (d) `sar` est pour `s`-hort `ar`-row.
3. `com` permet de demander l'alignement ou non des commentaires non étoilés entre eux.
   (a) `nal`, la valeur par défaut, est pour `n`-ot `al`-igned.
   (b) `al` est pour `al`-igned.

`\explnext[#opt]{#1}` `expl = expl-ain`

— `Option:` le symbole à utiliser pour une explication, la valeur par défaut étant celle du symbole de l'environnement `stepcalc` où `\explnext` est utilisé.

— `Argument:` le texte de l'explication qui peut être vide si aucune explication n'est à afficher.

ATTENTION ! *La macro* `\explnext` *est à utiliser sans argument ni option au tout début du contenu de l'environnement* `stepcalc` *en cas d'utilisation du style* `sar`*.*

`\explnext*[#opt]{#1..#2}`

— `Option:` le symbole à utiliser pour une explication, la valeur par défaut étant celle du symbole de l'environnement `stepcalc` où `\explnext` est utilisé.

— `Argument 1:` le texte de l'explication pour la 1re ligne. Ce texte peut être vide *(voir l'environnement* `astepcalc` *pour la raison de ceci).*

— `Argument 2:` le texte de l'explication pour la 2e ligne. Ce texte peut être vide *(voir l'environnement* `astepcalc` *pour la raison de ceci).*

`\comthis {#1}` `com = com-ment`
`\comthis*{#1}`

— `Argument:` le texte d'un court commentaire.

Détailler un raisonnement simple – Mise en forme du texte

Les macros suivantes sont juste utilisées par l'environnement `stepcalc`.

`\expltxtspacein{}`

`\expltxt{#1}`

— `Argument:` le texte de l'explication que l'on veut mettre en forme.

`\expltxtdown{#1}`

— `Argument:` le texte de l'explication du haut vers le bas que l'on veut mettre en forme.

`\expltxtup{#1}`

— `Argument:` le texte de l'explication du bas vers le haut que l'on veut mettre en forme.

`\expltxtupdown{#1..#2}`

— `Argument 1:` le texte de l'explication du haut vers le bas que l'on veut mettre en forme.

— `Argument 1:` le texte de l'explication du bas vers le haut que l'on veut mettre en forme.

`\explcom{#1}`

— `Argument:` le texte d'un court commentaire.

7. Détailler un « vrai » raisonnement

Détailler un « vrai » raisonnement via un tableau

```
\begin{demotab}[#opts]
   ...
\end{demotab}
```

— `Clé "start":` le début de la numérotation des identifiants des justifications. La valeur par défaut est `1` et la valeur spéciale `last` permet de reprendre la numérotation là où elle s'était arrêtée le dernier environnement `demotab` ou `demotab*` utilisé.

— `Clé "hyps ":` les hypothèses, au format texte, vérifiées au départ. Cet argument peut être vide et ne doit pas rentrer en conflit avec l'option `hyp`.

— `Clé "hyp ":` une unique hypothèse, au format texte, vérifiée au départ. Cet argument peut être vide et ne doit pas rentrer en conflit avec l'option `hyps`.

— `Clé "ccl ":` la conclusion, au format texte, du raisonnement détaillé. Cet argument peut être vide.

```
\begin{demotab*}[#opt]
   ...
\end{demotab*}
```

— `Clé "start":` le début de la numérotation des identifiants des justifications. La valeur par défaut est `1` et la valeur spéciale `last` permet de reprendre la numérotation là où elle s'était arrêtée le dernier environnement `demotab` ou `demotab*` utilisé.

`\demostep[#opt]`

— `Option:` un texte qui sera utilisé comme label global référençant le numéro d'une justification.

`\explref{#1}` `ref = ref-erence`

— `Argument:` un numéro de 1 ou 2 chiffres qui sera encadré comme le sont les numérotations des indications.

`\explref*{#1}`

— `Argument:` un texte correspondant à un label global référençant le numéro d'une justification.

Détailler un « vrai » raisonnement via un tableau - Textes utilisés

```
\txtdemoID{}                                   ID = ID-entifier
\txtdemoKNOWN{}
\txtdemoPROP{}                               PROP = PROP-osition
\txtdemoCONS{}                               CONS = CONS-equence
```

`\txtdemoHYPS{}` HYPS = HYP-othesis (S-everal ones)
`\txtdemoHYP{}` HYP = HYP-othesis (just one)
`\txtdemoCCL{}` CCL = C-on-CL-usion
`\txtdemoNEXTPAGE{}`

III. Géométrie

1. Points et lignes

Points

`\pt{#1}`

— `Argument:` un texte donnant le nom d'un point.

`\pt*{#1..#2}`

— `Argument 1:` un texte indiquant UP dans le nom UP_{down} d'un point.

— `Argument 2:` un texte indiquant *down* dans le nom UP_{down} d'un point.

2. Points et lignes

Lignes

`\gline [#opt]{#1..#2}` g = g-eometry
`\gpline[#opt]{#1..#2}` p = p-oint

— `Option:` pour indiquer les parenthèses ou crochets à utiliser, les valeurs possibles étant `O`, valeur par défaut, `C`, `CO` et `OC`.

— `Argument 1:` le 1er point géométrique.

— `Argument 2:` le 2e point géométrique.

`\ghline {#1..#2}` h = h-alf
`\gphline {#1..#2}` gph = g + p + h
`\segment {#1..#2}`
`\psegment{#1..#2}`

— `Argument 1:` le 1er point géométrique.

— `Argument 2:` le 2e point géométrique.

3. Points et lignes

Droites parallèles ou non

`\parallel`
`\nparallel`
`\stdparallel` (pour utiliser le symbole par défaut)
`\stdnparallel` (pour utiliser le symbole proposé par `amssymb`)

4. Vecteurs

Les écrire

`\vect{#1}`

— `Argument:` un texte donnant le nom d'un vecteur.

`\vect*{#1..#2}`

— `Argument 1:` un texte indiquant *up* dans le nom $\overrightarrow{up}_{down}$ d'un vecteur.

— `Argument 2:` un texte indiquant *down* dans le nom $\overrightarrow{up}_{down}$ d'un vecteur.

`\pvect{#1..#2}`

— `Argument 1:` le nom du 1er point qui sera mis en forme via la macro `\pt`.

— `Argument 2:` le nom du 2e point qui sera mis en forme via la macro `\pt`.

5. Vecteurs

Norme

`\norm {#1}`
`\norm*{#1}`

— `Argument:` le vecteur sur lequel appliquer la norme.

`\vnorm{#1}` `v = v-ector`

— `Argument:` le nom du vecteur sur lequel appliquer la norme.

6. Vecteurs

Produit scalaire

`\dotprod[#opt]{#1..#2}`

— `Option:` la valeur par défaut est `u` pour `u`-sual soit *« habituel »* en anglais. Voici les différentes valeurs possibles.

1. `u` : écriture habituelle avec un point.
2. `b` : écriture habituelle mais avec une puce.
3. `p` : écriture « universitaire » avec des parenthèses extensibles.
4. `sp` : écriture « universitaire » avec des parenthèses non extensibles.
5. `r` : écriture « à la physicienne » avec des chevrons extensibles.
6. `sr` : écriture « à la physicienne » avec des chevrons non extensibles.

— `Argument 1:` le 1er vecteur qu'il faut taper via la macro `\vect`.

— `Argument 2:` le 2e vecteur qu'il faut taper via la macro `\vect`.

`\vdotprod[#opt]{#1..#2}` `v = v-ector`

— `Option:` voir les explications précédentes données pour `\dotprod`.

— `Argument 1:` le nom du 1er vecteur sans utiliser la macro `\vect`.

— `Argument 2:` le nom du 2e vecteur sans utiliser la macro `\vect`.

7. Vecteurs

3D – Produit vectoriel

`\crossprod[#opt]{#1..#2}`

— `Option:` la valeur par défaut est `w` pour `w`-edge soit *« coin »* en anglais. Voici les valeurs possibles.

1. `w` : écriture classique en France.
2. `t` : écriture alternative avec le symbole `\times`.

— `Argument 1:` le 1er vecteur qu'il faut taper via la macro `\vect`.

— `Argument 2:` le 2e vecteur qu'il faut taper via la macro `\vect`.

`\vcrossprod[#opt]{#1..#2}` `v = v-ector`

— `Option:` voir les explications précédentes données pour `\crossprod`.

— `Argument 1:` le nom du 1er vecteur sans utiliser la macro `\vect`.

— `Argument 2:` le nom du 2e vecteur sans utiliser la macro `\vect`.

`\calccrossprod[#opt]{#1..#8}` `calc = calc-ulate`

— `Option:` la valeur par défaut est `vec,loop`. Voici les différentes valeurs possibles.

1. `vec` : pour afficher les vecteurs.
2. `novec` : pour ne pas afficher les vecteurs.
3. `arrows` : des croix fléchées indiquent comment effectuer les calculs.
4. `cross` : des croix non fléchées indiquent comment effectuer les calculs.
5. `loop` : des boucles indiquent comment effectuer les calculs.
6. `nodeco` : rien n'indique comment effectuer les calculs.
7. `exp` : ceci demande d'afficher les formules développées de chaque coordonnée en utilisant un espace pour séparer les facteurs de chaque produit.
8. `cexp` : comme `exp` mais avec le symbole · obtenu via `\cdot`.
9. `texp` : comme `exp` mais avec le symbole ×.
10. `p` : écriture horizontale avec des parenthèses extensibles.
11. `sp` : écriture horizontale avec des parenthèses non extensibles.
12. `vp` : écriture verticale avec des parenthèses.
13. `b` : écriture horizontale avec des crochets extensibles.
14. `sb` : écriture horizontale avec des crochets non extensibles.
15. `vb` : écriture verticale avec des crochets.

On peut combiner deux types de choix cohérents en les séparant par une virgule comme dans `vec,loop` la valeur par défaut de l'option.

— `Argument 1:` le 1er vecteur qu'il faut taper via la macro `\vect`.

— `Arguments 2..4:` les coordonnées du 1er vecteur.

— `Argument 5:` le 2e vecteur qu'il faut taper via la macro `\vect`.

— `Arguments 6..8:` les coordonnées du 2e vecteur.

`\vcalccrossprod[#opt]{#1..#8}` `calc = calc-ulate` et `v = v-ector`

— `Argument 1:` le 1er vecteur sans utiliser la macro `\vect`.

— `Argument 5:` le 2e vecteur sans utiliser la macro `\vect`.

Pour le reste, voir les indications données ci-dessus pour `\calccrossprod`.

8. Vecteurs

2D – Déterminant de deux vecteurs

`\calcdetplane[#opt]{#1..#6}` `calc = calc-ulate`

— `Option:` la valeur par défaut est `vec,loop`. Voici les différentes valeurs possibles.

1. `vec` : les vecteurs sont affichés si besoin.
2. `novec` : les vecteurs ne sont jamais affichés.
3. `arrows` : des croix fléchées indiquent comment effectuer les calculs.
4. `cross` : des croix non fléchées indiquent comment effectuer les calculs.
5. `loop` : des boucles indiquent comment effectuer les calculs.
6. `nodeco` : rien n'indique comment effectuer les calculs.
7. `exp` : ceci demande d'afficher une formule développée en utilisant un espace pour séparer les facteurs de chaque produit.
8. `cexp` : comme `exp` mais avec le symbole · obtenu via `\cdot`.
9. `texp` : comme `exp` mais avec le symbole ×.

On peut combiner deux types de choix en les séparant par une virgule comme dans `vec,loop` la valeur par défaut de l'option.

— `Argument 1:` le 1er vecteur qu'il faut taper via la macro `\vect`.

— `Arguments 2..3:` les coordonnées du 1er vecteur.

— `Argument 4:` le 2e vecteur qu'il faut taper via la macro `\vect`.

— `Arguments 5..6:` les coordonnées du 2e vecteur.

`\vcalcdetplane[#opt]{#1..#6}` `calc = calc-ulate` et `v = v-ector`

— `Argument 1:` le 1er vecteur sans utiliser la macro `\vect`.

— `Argument 4:` le 2e vecteur sans utiliser la macro `\vect`.

Pour le reste, voir les indications données ci-dessus pour `\calcdetplane`.

9. Géométrie cartésienne

Coordonnées

`\coord[#opt]{#1}`

— `Option:` la valeur par défaut est `p`. Voici les différentes valeurs possibles.

1. `p` : écriture horizontale avec des parenthèses extensibles.

2. sp : écriture horizontale avec des parenthèses non extensibles.
3. vp : écriture verticale avec des parenthèses.
4. b : écriture horizontale avec des crochets extensibles.
5. sb : écriture horizontale avec des crochets non extensibles.
6. vb : écriture verticale avec des crochets.

— Argument: l'argument est une suite de "morceaux" séparés par des barres | , chaque morceau étant une coordonnée. Il peut n'y avoir qu'un seul morceau.

`\pcoord[#opt]{#1..#2}` p = p-oint

— Option: voir les indications données pour la macro \coord.

— Argument 1: le point auquel sera appliqué automatiquement la macro \pt.

— Argument 2: voir les indications données pour l'unique argument obligatoire de la macro \coord.

`\pcoord*[#opt]{#1..#2}`

— Option: voir les indications données pour la macro \coord.

— Argument 1: le point auquel ne sera pas appliqué automatiquement la macro \pt.

— Argument 2: voir les indications données pour l'unique argument obligatoire de la macro \coord.

`\vcoord[#opt]{#1..#2}` v = v-ertical

— Option: voir les indications données pour la macro \coord.

— Argument 1: le vecteur sans utiliser la macro \vect.

— Argument 2: voir les indications données pour l'unique argument obligatoire de la macro \coord.

`\vcoord*[#opt]{#1..#2}`

— Option: voir les indications données pour la macro \coord.

— Argument 1: le vecteur qu'il faut taper via la macro \vect.

— Argument 2: voir les indications données pour l'unique argument obligatoire de la macro \coord.

10. Géométrie cartésienne

Nommer un repère

```
\axes {#1}
\axes*{#1}
```

— Argument: l'argument est une suite de "morceaux" séparés par des barres |.

- Le premier morceau est l'origine du repère.
- Les morceaux suivants sont des points ou des vecteurs qui "définissent" chaque axe.

`\paxes[#opt]{#1..# }` p = p-oint

— Argument: l'argument est une suite de "morceaux" séparés par des barres |.

- Le premier morceau est le nom de l'origine du repère sur laquelle la macro-commande `\pt` sera automatiquement appliquée.
- Viennent ensuite les noms des points "définissant" chaque axe. Pour chacun de ces points la macro-commande `\pt` sera automatiquement appliquée.

`\vaxes[#opt]{#1..# }` `v = v-ector`

— `Argument:` l'argument est une suite de "morceaux" séparés par des barres `|`.

- Le premier morceau est l'origine du repère.
- Viennent ensuite les noms des vecteurs "définissant" chaque axe. Pour chacun de ces vecteurs la macro-commande `\vect` sera automatiquement appliquée.

`\pvaxes[#opt]{#1..# }` `pv = p + v`

— `Argument:` l'argument est une suite de "morceaux" séparés par des barres `|`.

- Le premier morceau est le nom de l'origine du repère sur laquelle la macro-commande `\pt` sera automatiquement appliquée.
- Viennent ensuite les noms des vecteurs "définissant" chaque axe. Pour chacun de ces vecteurs la macro-commande `\vect` sera automatiquement appliquée.

11. Arcs circulaires

`\circarc{#1}` `circ = circ-ular`

— `Argument:` un texte donnant le nom d'un arc circulaire.

`\circarc*{#1..#2}`

— `Argument 1:` un texte indiquant *up* dans le nom $\widehat{up}_{down}$ d'un arc circulaire.

— `Argument 2:` un texte indiquant *down* dans le nom $\widehat{up}_{down}$ d'un arc circulaire.

12. Angles

Angles géométriques « intérieurs »

`\anglein{#1}` `in = in-terior`

— `Argument:` un texte donnant le nom d'un angle intérieur.

`\anglein*{#1..#2}`

— `Argument 1:` un texte indiquant *up* dans le nom $\widehat{up}_{down}$ d'un angle intérieur.

— `Argument 2:` un texte indiquant *down* dans le nom $\widehat{up}_{down}$ d'un angle intérieur.

13. Angles

Angles orientés de vecteurs

`\angleorient[#opt]{#1..#2}`

— `Option:` la valeur par défaut est `p`. Voici les différentes valeurs possibles.

1. p : écriture habituelle avec des parenthèses extensibles.
2. sp : écriture habituelle avec des parenthèses non extensibles.
3. h : écriture avec un chapeau et des parenthèses extensibles.
4. sh : écriture avec un chapeau et des parenthèses non extensibles.

— Argument 1: le premier vecteur qu'il faut taper via la macro \vect.

— Argument 2: le second vecteur qu'il faut taper via la macro \vect.

\vangleorient[#opt]{#1..#2} v = v-ector

— Option: voir les explications précédentes données pour \angleorient.

— Argument 1: le nom du premier vecteur sans utiliser la macro \vect.

— Argument 2: le nom du second vecteur sans utiliser la macro \vect.

IV. Analyse

1. Constantes et paramètres

Constantes classiques ajoutées

\ggamma
\ppi
\ttau
\ee
\ii
\jj
\kk

2. Constantes et paramètres

Constantes latines personnelles

\param{#1}

— Argument: un texte utilisant l'alphabet latin.

3. Une variable « symbolique »

\symvar[#opt]

— Option: le numéro du symbole qui vaut 1 par défaut.

- 1 donne • .
- 2 donne ◦ .
- 3 donne ▪ .
- 4 donne ▫ .

4. La fonction valeur absolue

\abs {#1}
\abs*{#1}

— Argument: l'expression sur laquelle appliquer la fonction valeur absolue.

5. Fonctions nommées spéciales

Sans paramètre

```
\acos
\asin
\atan
```

```
\arccosh
\arcsinh
\arctanh
```

```
\acosh
\asinh
\atanh
```

```
\fch                                                    f = f-rench
\fsh                                                    f = f-rench
\fth                                                    f = f-rench
```

```
\afch
\afsh
\afth
```

6. Fonctions nommées spéciales

Avec un paramètre

`\exp[#opt]`

— `Option:` la base de l'exponentielle

```
\lg [#opt]
\ln [#opt]
\log[#opt]
```

— `Option:` la base du logarithme

7. Limite

`\limit[#opt]{#1..#3}`

— `Option:` la valeur par défaut `asit` n'a pas vocation à être utilisée.

1. `asit` : rien n'est appliqué sur la fonction qui reste donc tel quelle.
2. `p` : ajout de parenthèses extensibles autour de la fonction.
3. `sp` : ajout de parenthèses non extensibles autour de la fonction.

— `Argument 1:` la fonction dont on indique la limite.

— `Argument 2:` la variable qui va tendre vers quelque chose.

— `Argument 3:` la valeur limite suivie éventuellement de conditions en utilisant la barre verticale `|` pour séparer ces différentes informations.

8. Calcul différentiel

Les opérateurs ∂ et d

`\dd [#opt] {#1}`
`\pp [#opt] {#1}`

— `Option:` utilisée, cette option sera mise en exposant du symbole ∂ ou d.

— `Argument:` la variable de différentiation à droite du symbole ∂ ou d.

Dérivations totales d'une fonction – Version longue avec une variable

`\der [#opt] {#1..#3}`

— `Option:` la valeur par défaut est `u`.

1. `u` : écriture usuelle avec des primes *(ceci nécessite d'avoir une valeur entière naturelle connue du nombre de dérivations successives)*.
2. `e` : écriture via un exposant entre des parenthèses.
3. `i` : écriture via un indice.
4. `d` : écriture pointée à la physicienne *(cf. la dérivation par rapport au temps)*.
5. `bd` : écriture pointée avec un crochet entre les points et la fonction.
6. `f` : écriture via une fraction en mode display.
7. `sf` : écriture via une fraction en mode non display.
8. `of` : écriture via une fraction en mode display sous la forme d'un opérateur *(la fonction est à côté de la fraction)*.
9. `osf` : écriture via une fraction en mode non display sous la forme d'un opérateur *(la fonction est à côté de la fraction)*.
10. `p` : ajout de parenthèses extensibles autour de la fonction.
11. `sp` : ajout de parenthèses non extensibles autour de la fonction.

— `Argument 1:` la fonction à dériver.

— `Argument 2:` la variable de dérivation.

— `Argument 3:` l'ordre de dérivation.

Dérivations totales d'une fonction – Version courte sans variable

`\sder [#opt] {#1..#2}` `s = s-imple`

— `Option:` la valeur par défaut est `u`. Les options disponibles sont `u`, `e`, `d`, `bd`, `p` et `sp` : voir la fiche technique de `\sder` ci-dessus.

— `Argument 1:` la fonction à dériver.

— `Argument 2:` l'ordre de dérivation.

L'opérateur de dérivation totale

`\derope[#opt]{#1..#2}` `ope = ope-rator`

— `Option:` la valeur par défaut est `f`. Les options disponibles sont `f`, `sf` et `i` : voir la fiche technique de `\der` donnée un peu plus haut.

— `Argument 1:` la fonction à dériver.

— `Argument 2:` l'ordre de dérivation.

Dérivations partielles

`\pder[#opt]{#1..#2}` `p = p-artial`

— `Option:` la valeur par défaut est `f`.

1. `f` : écriture via une fraction en mode display.
2. `sf` : écriture via une fraction en mode non display.
3. `of` : écriture via une fraction en mode display sous la forme d'un opérateur *(la fonction est à côté de la fraction)*.
4. `osf` : écriture via une fraction en mode non display sous la forme d'un opérateur *(la fonction est à côté de la fraction)*.
5. `i` : écriture via un indice.
6. `ei` : écriture via un indice mais en *« développant »*.
7. `p` : ajout de parenthèses extensibles autour de la fonction.
8. `sp` : ajout de parenthèses non extensibles autour de la fonction.

— `Argument 1:` la fonction à dériver.

— `Argument 2:` les variables utilisées avec leur ordre de dérivation pour la dérivation partielle en utilisant une syntaxe du type `x | y^2 | ...` qui indique de dériver suivant x une fois, puis suivant y deux fois... etc.

— `Argument 3:` l'ordre total de dérivation.

L'opérateur de dérivation partielle

`\pderope[#opt]{#1..#2}` `p = p-artial` et `ope = ope-rator`

— `Option:` la valeur par défaut est `f`. Les options disponibles sont `f`, `sf` et `i` : voir la fiche technique de `\pder` juste avant.

— `Argument 1:` les variables utilisées avec leur ordre de dérivation via la syntaxe indiquée ci-dessus.

— `Argument 2:` l'ordre total de dérivation.

9. Tableaux de variation et de signe

Coloriser le fond

`\backLine[#opt]{#1}` `back = back-ground`

— `Option:` couleur au format TikZ. La valeur par défaut est `gray!30`.

— `Argument 1:` les numéros de ligne séparés par des virgules, 0 étant le 1er numéro.

Commentaires

`\comLine[#opt]{#1..#2}` `com = com-ment`

— `Option:` couleur au format TikZ. La valeur par défaut est `blue`.

— `Argument 1:` le numéro de ligne, 0 étant le 1er numéro.

— `Argument 2:` le texte du commentaire.

Graphiques explicatifs

`\graphSign[#opt]{#1..#4}`

— `Option:` couleur au format TikZ. La valeur par défaut est `blue`.

— `Argument 1:` le numéro de ligne, 0 étant le 1er numéro.

— `Argument 2:` le type de fonctions avec des contraintes éventuelles en utilisant la virgule comme séparateur d'informations.

1. `x2` sans espace indique $f(x) = x^2$.
2. `srqt` sans espace indique $f(x) = \sqrt{x}$.
3. `1/x` sans espace indique $f(x) = \frac{1}{x}$.
4. `abs` sans espace indique $f(x) = |x|$.
5. `exp` sans espace indique $f(x) = \exp x$.
6. `ln` sans espace indique $f(x) = \ln x$.
7. `ax+b` sans espace indique $f(x) = ax + b$ avec $a \neq 0$ à caractériser.
8. `ax2+bx+c` sans espace indique $f(x) = ax^2 + bx + c$ avec $a \neq 0$ et le discriminant d à caractériser.
9. `ap` et `an` indiquent respectivement les conditions $a > 0$ et $a < 0$.
10. `dp`, `dz` et `dn` indiquent respectivement les conditions $d > 0$, $d = 0$ et $d < 0$.

— `Argument 3 supplémentaire pour ax+b:` la racine réelle de $ax + b$.

— `Arguments supplémentaires éventuels pour ax2+bx+c:` si $ax^2 + bx + c$ admet une ou deux racines réelles, on donnera toutes les racines de la plus petite à la plus grande [1].

10. Calcul intégral

Le symbole standard revisité

`\stdint` (pour retrouver le comportement par défaut)

11. Calcul intégral

Un opérateur d'intégration clés en main

```
\integrate  {#1..#4}
\integrate* {#1..#4}
\dintegrate {#1..#4}                                    d = d-isplaystyle
\dintegrate*{#1..#4}
```

— `Argument 1:` la fonction intégrée.

— `Argument 2:` la variable d'intégration.

1. Notant $\Delta = b^2 - 4ac$, si $\Delta < 0$ il n'y aura pas d'argument supplémentaire, si $\Delta = 0$ il y en aura un seul et enfin si $\Delta > 0$ il faudra en donner deux, le 1er étant le plus petit.

— `Argument 3:` valeur initiale d'intégration qui apparait en bas du symbole $\int_{\bullet}$.

— `Argument 4:` valeur finale d'intégration qui apparait en haut du symbole $\int^{\bullet}$.

12. Calcul intégral

L'opérateur « crochet »

```
\hook [#opt]{#1..#4}
\hook* [#opt]{#1..#4}
```

— `Option:` la valeur par défaut est `b`. Voici les différentes valeurs possibles.

1. `b` : des crochets extensibles sont utilisés.
2. `sb` : des crochets non extensibles sont utilisés.
3. `r` : un unique trait vertical extensible est utilisé à droite.
4. `sr` : un unique trait vertical non extensible est utilisé à droite.

— `Argument 1:` la fonction sur laquelle effectuer le calcul.

— `Argument 2:` la variable à affecter pour les calculs.

— `Argument 3:` valeur initiale à substituer qui apparait en bas du crochet fermant ou de la barre verticale.

— `Argument 4:` valeur finale à substituer qui apparait en haut du crochet fermant ou de la barre verticale.

V. Suites

1. Des notations complémentaires pour des suites spéciales

```
\seqplus{#1..#2}
```

— `Argument 1:` l'exposant à droite.

— `Argument 2:` l'indice à droite.

```
\seqhypergeo{#1..#2}
```

— `Argument 1:` l'indice à gauche.

— `Argument 2:` l'indice à droite.

```
\seqsuprageo{#1..#4}
```

— `Argument 1:` l'indice à gauche.

— `Argument 2:` l'indice à droite.

— `Argument 3:` l'exposant à droite.

— `Argument 4:` l'exposant à gauche.

2. Sommes et produits en mode ligne

Les opérateurs suivants ont un comportement spécifique vis à vis des mises en index et en exposant.

```
\dprod
\dsum
```

3. Comparaison asymptotique de suites et de fonctions

Les notations $\mathcal{O}$ et o

```
\bigO
\smallO
```

La notation Ω

```
\bigomega
```

La notation Θ

```
\bigtheta
```

VI. Probabilité

1. Juste pour rédiger

Probabilité « simple »

```
\proba[#opt]{#1}
```

— `Option:` le nom de la probabilité. La valeur par défaut est `P`.

— `Argument:` l'ensemble dont on veut calculer la probabilité.

2. Juste pour rédiger

Probabilité conditionnelle

```
\probacond  [#opt]{#1..#2}
\probacond* [#opt]{#1..#2}
\eprobacond [#opt]{#1..#2}
\eprobacond*[#opt]{#1..#2}
```

— `Option:` le nom de la probabilité. La valeur par défaut est `P`.

— `Argument 1:` l'ensemble qui donne la condition.

— `Argument 2:` l'ensemble dont on veut calculer la probabilité.

3. Juste pour rédiger

Évènement contraire

`\nevent{#1}`

— `Argument:` l'ensemble dont on veut indiquer le contraire.

4. Juste pour rédiger

Espérance, variance et écart-type

`\expval[#opt]{#1}`

— `Option:` le nom de la fonction espérance. valeur par défaut est `E`.

— `Argument:` la variable aléatoire dont on veut calculer l'espérance.

`\var[#opt]{#1}`

— `Option:` le nom de la fonction variance. valeur par défaut est `V`.

— `Argument:` la variable aléatoire dont on veut calculer la variance.

`\stddev[#opt]{#1}`

— `Option:` le nom de la fonction écart-type. La valeur par défaut est `\sigma`.

— `Argument:` la variable aléatoire dont on veut calculer l'écart-type.

5. Calculer l'espérance – Cas fini

`\calcexpval[#opt]{#1}` `calc = calc-ulate` `expval = exp-ected val-ue`

— `Option:` un système de type `clé = valeur`.

1. `disp` indique ce qu'il faut afficher. La valeur par défaut est `table`. Voici toutes les valeurs possibles.
   (a) `table` : un tableau donnant la loi est affiché seul.
   (b) `all` : un tableau donnant la loi est affiché suivi du début du calcul de l'espérance.
   (c) `none` : rien n'est affiché.
   (d) `exp` : le début du calcul de l'espérance sans le tableau donnant la loi.
   (e) `formal` : la formule avec Σ .
   (f) `eval` : la somme des produits du type $p_k \cdot x_k$.
2. `nosigma` : utilisée seule cette option demande de ne pas afficher la formule avec Σ .
3. `E` : la macro utilisée pour écrire l'espérance. Par défaut `E = \expval` .
4. `X` : le nom de la variable aléatoire. Par défaut `X = X` .
5. `k` : l'indice. Par défaut `k = k` .
6. `xk` : la lettre à indicer pour les valeurs de la variable aléatoire. Par défaut `xk = x` .
7. `pk` : la lettre à indicer pour les valeurs des probabilités. Par défaut `pk = p` .
8. `com` : un texte court éventuel à ajouter entre le tableau donnant la loi et le début du calcul de l'espérance.

9. `ope` : le symbole de multiplication. Par défaut `ope = \cdot` .
10. `name` : un nom éventuel de la loi pour une utilisation ultérieure.
11. `reuse` : un nom indiquant ce qui doit être réutilisé.
12. `colors` : le cycle des couleurs au format TikZ séparées par des tirets.
Par défaut `colors = red - blue - orange - green!70!black` .

— `Argument:` définition de la loi de la variable aléatoire finie avec une syntaxe semblable à celle d'un tableau.

6. Arbres pondérés

```
\begin{probatree}[#opt]
   ...
\end{probatree}
```

— `Option:` la valeur par défaut étant `asit`.

ATTENTION ! *L'option s'indique via `<...>` et non le traditionnel `[...]`.*

1. `asit` : pour les poids, on suit les indications données dans le code.
2. `hideall` : on cache tous les poids quelque soient les indications données dans le code.
3. `showall` : on montre tous les poids quelque soient les indications données dans le code.

— `Contenu:` un arbre codé en utilisant la syntaxe supportée par le package `forest`.

— `Clé "pweight":` pour écrire un poids sur le milieu d'une branche.

— `Clé "apweight":` pour écrire un poids au-dessus le milieu d'une branche.

— `Clé "bpweight":` pour écrire un poids en-dessous du milieu d'une branche.

— `Clé "pweight*":` pour indiquer un poids sans l'imprimer. Avec l'option `showall` le poids sera affiché comme si on avait utilisé `pweight`.

— `Clé "apweight*":` pour indiquer un poids sans l'imprimer. Avec l'option `showall` le poids sera affiché comme si on avait utilisé `apweight`.

— `Clé "bpweight*":` pour indiquer un poids sans l'imprimer. Avec l'option `showall` le poids sera affiché comme si on avait utilisé `bpweight`.

— `Clé "pframe":` pour encadrer un sous-arbre depuis un noeud vers toutes les feuilles de celui-ci.

```
\ptreeComment [#opt]{#1..#2}
\ptreeComment*[#opt]{#1..#2}
```

— `Option:` un système de type `clé = valeur`.

1. `tcol` : la couleur au format TikZ du texte. La valeur par défaut est `black`.
2. `dx` : une distance horizontale relative de décalage. La valeur par défaut est `0cm`.
3. `dy` : une distance verticale relative de décalage. La valeur par défaut est `0cm`.

— `Argument 1:` le nom de la feuille.

— `Argument 2:` le texte du commentaire.

```
\aptreeComment [#opt]{#1..#2}
```

`\aptreeComment* [#opt] {#1..#2}`

Voir les indications précédentes excepté qu'ici on utilise le système de nommage automatisé dérivé de celui de `forest`.

`\ptreeFocus [#opt] {#1}`

— `Option:` un système de type `clé = valeur`.

1. `lcol` : la couleur au format TikZ des lignes des arrêtes et des cadres éventuels. La valeur par défaut est `blue`.
2. `tcol` : la couleur au format TikZ du texte. La valeur par défaut est `black`.
3. `bcol` : la couleur au format TikZ du fond. La valeur par défaut est `white`.
4. `frame` : le type d'encadrement souhaité. La valeur par défaut est `nostart`. Voici les valeurs disponibles.
   (a) `nostart` : le 1er noeud n'est pas encadré mais les autres le sont.
   (b) `start` : tous les noeuds sont encadrés.
   (c) `none` : aucun des noeuds n'est encadré.

— `Argument:` les noms des noeuds dans le bon ordre séparés par des barres verticales `|` .

`\aptreeFocus [#opt] {#1}`

— `Option:` un système de type `clé = valeur` plus réduit que celui de `\ptreeFocus`.

1. `lcol` : la couleur au format TikZ des lignes des arrêtes et des cadres éventuels. La valeur par défaut est `blue`.
2. `frame` : voir les indications précédentes.

— `Argument:` : voir les indications précédentes excepté qu'ici on utilise le système de nommage automatisé dérivé de celui de `forest`.

`\ptreeFrame [#opt] {#1..#3}` `p = p-robabilty`

— `Option:` un système de type `clé = valeur`.

1. `lcol` : la couleur au format TikZ du cadre. La valeur par défaut est `blue`.

— `Arguments 1..3:` les noms de la sous-racine (à gauche), du noeud final en haut (à droite) et du noeud final en bas (à droite) tous indiqués via `name = ...` *(en fait l'ordre n'est pas important ici)*.

`\aptreeFrame [#opt] {#1..#3}` `a = a-auto`

Voir les indications précédentes excepté qu'ici on utilise le système de nommage automatisé dérivé de celui de `forest`.

`\ptreeTagLeaf [#opt] {#1..#2}`
`\ptreeTagLeaf* [#opt] {#1..#2}`

— `Option:` un système de type `clé = valeur`.

1. `tcol` : la couleur au format TikZ du texte. La valeur par défaut est `black`.
2. `dx` : une distance horizontale relative de décalage. La valeur par défaut est `0cm`.
3. `dy` : une distance verticale relative de décalage. La valeur par défaut est `0cm`.

— `Argument 1:` les noms des noeuds séparés par des barres verticales `|` .

— `Argument 2:` le texte commun décrivant chaque feuille.

```
\aptreeTagLeaf [#opt]{#1..#2}
\aptreeTagLeaf*[#opt]{#1..#2}
```

Voir les indications précédentes excepté qu'ici on utilise le système de nommage automatisé dérivé de celui de `forest`.

```
\ptreeTagLevel [#opt]{#1..#2}
\ptreeTagLevel*[#opt]{#1..#2}
```

— `Option:` un système de type `clé = valeur`.

1. `tcol` : la couleur au format TikZ du texte. La valeur par défaut est `black`.
2. `dx` : une distance horizontale relative de décalage. La valeur par défaut est `0cm`.
3. `dy` : une distance verticale relative de décalage. La valeur par défaut est `0cm`.

— `Argument 1:` le nom d'un noeud du niveau souhaité.

— `Argument 2:` le texte décrivant le niveau.

```
\aptreeTagLevel [#opt]{#1..#2}
\aptreeTagLevel*[#opt]{#1..#2}
```

Voir les indications précédentes excepté qu'ici on utilise le système de nommage automatisé dérivé de celui de `forest`.

```
\ptreeNodeColor{#1..#2}
```

— `Argument 1:` le nom du noeud.

— `Argument 2:` chacun des paramètres présentés ci-dessous est optionnel et par défaut la macro utilise `tcol = black` et `bcol = white`.

1. `tcol = ...` sert à indiquer la couleur du texte.
2. `bcol = ...` pour à indiquer la couleur du fond *(`b = b-ackground`)*.

```
\ptreeNodeNewText[#opt]{#1..#2}
```

— `Option:` chacun des paramètres présentés ci-dessous est optionnel et par défaut la macro utilise `tcol = black` et `bcol = white`.

1. `tcol = ...` sert à indiquer la nouvelle couleur du texte.
2. `bcol = ...` pour à indiquer la nouvelle couleur du fond *(`b = b-ackground`)*.

— `Argument 1:` le nom du noeud.

— `Argument 2:` le nouveau texte.

```
\ptreeTextOf{#1}
```

— `Argument:` le nom du noeud.

VII. Arithmétique

1. Opérateurs de base

```
\divides
\ndivides
\nequiv
\modulo
```

2. Fonctions nommées spéciales

```
\pgcd
\ppcm
```

```
\lcm
```

3. Fractions continuées

Fractions continuées standard

```
\contfrac  {#1}
\contfrac* {#1}
\scontfrac {#1}
\scontfrac*{#1}
```

— `Argument:` tous les éléments de la fraction continuée « courte » séparés par des `|`.

4. Fractions continuées

Fractions continuées généralisées

```
\contfracgene  {#1}
\contfracgene* {#1}
\scontfracgene {#1}
\scontfracgene*{#1}
```

— `Argument:` tous les éléments de la fraction continuée généralisée séparés par des `|`.

5. Fractions continuées

L'opérateur $\mathcal{K}$

```
\contfracope
```

Cette macro sans argument se comporte comme $\sum$ vis à vis des mises en index et en exposant.

VIII. Algèbre linéaire

1. Matrices via `nicematrix`

Calcul expliqué d'un déterminant 2×2

`\calcdettwo[#opt]{#1..#4}` `c = c-alculate`

— `Option:` la valeur par défaut est `nodeco`. Voici les différentes valeurs possibles.

1. `arrows` : des croix fléchées indiquent comment effectuer les calculs.
2. `cross` : des croix non fléchées indiquent comment effectuer les calculs.
3. `loop` : des boucles indiquent comment effectuer les calculs.
4. `nodeco` : rien n'indique comment effectuer les calculs.
5. `exp` : ceci demande d'afficher une formule développée en utilisant un espace pour séparer les facteurs de chaque produit.
6. `cexp` : comme `exp` mais avec le symbole $\cdot$ obtenu via `\cdot`.
7. `texp` : comme `exp` mais avec le symbole $\times$.

— `Argument 1:` l'entrée à la position $(1, 1)$

— `Argument 2:` l'entrée à la position $(1, 2)$

— `Argument 3:` l'entrée à la position $(2, 1)$

— `Argument 4:` l'entrée à la position $(2, 2)$

IX. Polynômes et séries formelles

1. Polynômes

```
\setpoly    {#1..#2}
\setpolyfrac{#1..#2}
```

— `Argument 1:` l'ensemble auquel les coefficients appartiennent.

— `Argument 2:` cet argument est une suite de "morceaux" séparés par des barres `|`, chaque morceau étant une variable formelle.

2. Séries formelles classiques

```
\setserie    {#1..#2}
\setseriefrac{#1..#2}
```

— `Argument 1:` l'ensemble auquel les coefficients appartiennent.

— `Argument 2:` cet argument est une suite de "morceaux" séparés par des barres `|`, chaque morceau étant une variable formelle.

3. Polynômes et séries formelles de Laurent

```
\setpolylaurent {#1..#2}
\setserielaurent{#1..#2}
```

— `Argument 1:` l'ensemble auquel les coefficients appartiennent.

— `Argument 2:` cet argument est une suite de "morceaux" séparés par des barres `|`, chaque morceau étant une variable formelle.