Karnaj

Zeste de savoir

Outils pour l'écriture des mathématiques en LaTeX

11 août 2019

Table des matières

La composition des mathématiques est l'une des fonctionnalités phares de TeX, donc également de LaTeX. De nombreux outils et d'innombrables *packages* sont disponibles pour composer n'importe quelle mise en forme mathématique, aussi compliquée soit elle, tout en offrant un contrôle très fin sur la composition. Une des difficultés est de connaître les outils existants, et de savoir lesquels utiliser à bon escient.

Le but de ce tutoriel est de présenter une palette d'outils adaptés à des situations de composition courantes, et de donner quelques règles de bases pour bien les utiliser. Tout cela dans le but de composer des maths plus facilement.

Pour cela, nous allons d'abord faire quelques rappels sur le mode mathématique, et voir comment bien l'utiliser. Puis, nous allons nous intéresser à quelques environnements qui permettent de répondre à la majorité des besoins, avant de créer des commandes pour nous faciliter l'écriture. Nous allons finalement nous intéresser aux réglages des paramètres du mode mathématique et voir comment changer le comportement de TeX.

Prérequis
Connaissances de base en LaTeX (un tutoriel est disponible ici).
Connaissances des commandes de base du mode mathématique.
Prérequis optionnel
Connaissances de quelques primitives de TeX.
Objectifs
Présenter les règles basiques de composition des mathématiques.
Présenter des outils pour une composition efficace des mathématiques.

Le résultat fourni par KaTeX ne correspond pas forcément à celui que donnera LaTeX. Les mathématiques affichées par le navigateur pourront donc être différentes de celles affichées par LaTeX et il vaut donc mieux compiler chaque code soi-même pour se faire une idée du résultat. En particulier, KaTeX ne dispose pas de certaines commandes et de certains environnements, et nous devons donc essayer de reproduire leur affichage, mais cela n'est bien sûr pas aussi fidèle que l'affichage que produira LaTeX.

1. Rappels et premières règles

1.1. Le mode mathématique

1.1.1. Les deux modes

TeX et LaTeX offrent une multitude d'outils pour écrire des mathématiques. Tout d'abord, il faut savoir qu'il existe deux modes mathématiques en TeX:

— le mode dit «**en ligne**», où les formules mathématiques sont placées dans le corps du message, dans la continuité du texte. `$ $`, `\( \)` ou encore `\begin{math} \end{math}` permettent d'écrire des maths «en ligne»;
— le mode dit «**hors texte**», où les formules mathématiques sont mises en valeur en dehors du texte (par défaut elles sont centrées). `$$ $$`, `\[ \]` ou encore `\begin{displaymath} \end{displaymath}` permettent d'écrire dans le mode «hors texte».

Notons qu'en mode «en ligne», les caractères mathématiques appartenant à des structures telles que les fractions sont de taille réduite. Voici la différence entre `\(1+\frac{2}{3}\)` et `\[1+\frac{2}{3}\]` qui donnent respectivement $1+\frac{2}{3}$ et

$$1+\frac{2}{3}.$$

Une des particularités du mode mathématique, qu'il soit en ligne ou hors texte, est que les espaces dans le code source sont ignorés. Ainsi `$1+1=2$` ou `$1 + 1 = 2$` sont équivalents et donneront tous les deux $1+1=2$, ce qui laisse une certaine liberté d'écriture. Chacun peut donc profiter de cette liberté pour présenter le code en mode mathématique à sa guise. Par souci de lisibilité, on peut, par exemple, écrire une espace avant et après chaque nombre, parenthèse et opérateur. Ou ne pas le faire!

Il ne peut y avoir la primitive `\par` (ou deux retours à la ligne consécutifs, qui sont équivalents à `\par`) en mode maths, que ce soit en ligne ou hors texte sous peine d'une erreur de compilation.

1.1.2. Les commandes de passage en mode mathématique

Parmi toutes ces possibilités pour passer en mode mathématique, quel choix faut-il faire?

1.1.2.1. Passage en mode « en ligne »

Pour le mode «en ligne», `$ $` est une primitive de TeX tandis que `\( \)` et `\begin{math} \end{math}` sont leurs équivalents en LaTeX. On peut indifféremment choisir `$ $`, qui est plus pratique à écrire, ou préférer d'utiliser `\( \)`, qui est une commande LaTeX. La plupart des gens préfèrent la concision et la facilité de lecture de `$ $`.

1.1.2.2. Passage en mode « hors texte »

Pour le mode «hors texte», nous utiliserons `\[ \]` ou `\begin{displaymath} \end{displaymath}`, mais pas la primitive de TeX `$$ $$`, car cette dernière balise altère, dans certaines circonstances, les espaces verticales placées avant la formule hors texte: c'est pourquoi elle n'est pas recommandée. En LaTeX, il vaut donc mieux utiliser l'une des deux commandes précédentes, qui sont équivalentes. Par souci de concision, nous allons préférer `\[ \]`.

1.1.2.3. La commande `\ensuremath`

La macro `\ensuremath{<code>}` teste si elle se trouve en mode mathématique lorsqu'elle est exécutée. Si c'est le cas, elle compose le `<code>` dans ce mode maths. En revanche, si le mode maths n'est pas en vigueur, elle compose son argument `<code>` en mode mathématique et repasse en mode texte ensuite. La macro `\ensuremath` est surtout utilisée dans des macros que l'utilisateur définit, et en facilite l'usage, puisque ces macros fonctionneront sans erreur que l'on soit en mode maths ou pas. Ainsi, si l'on définit la macro `\PI` de cette façon: `\newcommand*\PI{\ensuremath{\pi}}`, eh bien, la macro `\PI` affichera π quel que soit le mode en cours.

1.1.2.4. Forcer le style d'affichage

Lorsque nous sommes dans le mode «en ligne», nous pourrions souhaiter que l'affichage se fasse comme si nous étions en mode «hors texte». À l'inverse, on pourrait parfois vouloir le contraire, c'est-à-dire vouloir un affichage typique du mode «en ligne» en mode «hors texte».

La primitive `\displaystyle` permet d'afficher l'élément mathématique qui la suit comme il aurait été affiché en mode «hors texte». Elle est à utiliser avec précaution, car si elle est employée dans le mode «en ligne», en affichant des éléments plus grands qu'ils ne devraient l'être, elle peut casser la régularité des interlignes.

La primitive `\textstyle` permet d'afficher l'élément qui la suit comme il aurait été affiché en mode «en ligne». Elle est moins utilisée que `\displaystyle`, mais est quand même utile dans certains cas.

Nous disposons également de deux autres commandes, `\scriptstyle` et `\scriptscriptstyle`.

Notons de plus l'existence de la commande `\everydisplay{<token>}`, qui permet de spécifier une liste de *tokens* que LaTeX insérera au début de toute formule mathématique hors texte, et de la commande `\everymath`, qui est son équivalent en mode «en ligne». Par exemple, avec

`\everymath{\texstyle}`, toutes les formules «en ligne» seront affichées comme si elles étaient en mode «hors texte».

1.1.3. Lecture d'arguments

Faisons à présent un petit rappel quant à la lecture d'arguments par des macros. Un argument d'une macro est:

— un *token* unique, c'est-à-dire un caractère seul ou une commande seule;
— un ensemble de *tokens* entre accolades `{…}`. Lorsque l'argument est lu, les accolades sont retirées et l'argument pris en compte est donc constitué de ce qu'il y avait entre les accolades.

Lorsqu'une macro lit des arguments, elle ignore les espaces. Pour qu'une espace soit prise en compte comme argument d'une macro il **faut** que cette espace soit entourée d'accolades.

Ainsi, `{23}`, `2`, `{2}` et `\pi` sont des arguments. Il est important de comprendre que les arguments `2` et `{2}` sont rigoureusement équivalents donc, plus généralement, lorsqu'un argument est constitué d'un seul *token*, on a le choix de l'entourer d'accolades ou pas. Certains prennent l'habitude de ne jamais le faire, tandis que d'autres prêchent vigoureusement pour que ces accolades soient mises. Il n'y a pas de règle universelle, et ne pas mettre d'accolades lorsqu'on écrit `\frac23` pour écrire «deux tiers» est tout à fait acceptable, tout comme l'est `\frac{2}{3}`. Il faut cependant garder à l'esprit que la présence d'accolades permet de rajouter facilement des caractères à un argument. Cette même manœuvre est plus longue lorsque les accolades sont absentes, puisqu'il faut alors obligatoirement les rajouter.

Par ailleurs, il est important de noter que, lorsqu'il est l'argument d'une macro, `{23}` est un seul argument qui est le nombre 23, alors que `23` représente deux arguments qui sont les nombres 2 et 3. Signalons que `2 3` est également considéré comme deux arguments 2 et 3 puisque l'espace est ignorée. Ainsi, avec la commande `\frac` qui lit deux arguments, ces codes donnent des résultats différents et montrent qu'il est important de bien avoir conscience de la portée d'un argument:

— `$\frac{2\pi}{3}$` donne $\frac{2\pi}{3}$;
— `$\frac2\pi3$` donne $\frac{2}{\pi}3$;
— `$\frac2{\pi3}$` donne $\frac{2}{\pi 3}$.

1.1.3.1. Arguments optionnels

Certaines macros admettent un argument optionnel, c'est-à-dire que lorsqu'il est omis, une valeur par défaut définie par le concepteur de la macro est donnée à cet argument. LaTeX adopte la convention de mettre l'argument optionnel d'une macro immédiatement après cette macro, et entre crochets. Ainsi, la macro `\sqrt` admet un argument optionnel, vide par défaut, qui est le nombre placé au-dessus du signe radical pour exprimer la racine n-ième. Cet argument est vide par défaut: `\sqrt{2}` donne $\sqrt{2}$ alors que `\sqrt[3]{2}` donne $\sqrt[3]{2}$.

1.2. Les délimiteurs

Lorsque nous écrivons quelque chose entre parenthèses, par exemple, nous aimerions bien que la taille de ces parenthèses soit adaptée à la taille de cet élément. Par exemple, `\[ (\frac{23}{12}) \]` donne

$$(\frac{2}{3}).$$

1.2.1. Des délimiteurs de taille adaptée

Heureusement, il existe un moyen simple pour que les parenthèses (et les autres délimiteurs) s'adaptent à leur contenu. Il suffit d'utiliser les deux primitives `\left` et `\right`, qui agissent sur le délimiteur de gauche et de droite. Ces deux commandes vont obligatoirement de pair; elles prennent chacune en paramètre un délimiteur, et adaptent leur hauteur à ce qui se trouve entre elles. Les délimiteurs suivants sont utilisables: `(`, `)`, `\{`, `\}`, `[`, `]`, `\langle`, `\rangle` et `|`. Voici quelques exemples d'utilisation:

— $\displaystyle\left(\frac{2}{3}\right)$ avec `$\displaystyle\left(\frac{2}{3}\right)$`;
— $\displaystyle\left]\frac{2}{3}\right[$ avec `$\displaystyle\left]\frac{2}{3}\right[$` (remarquons que le sens des délimiteurs n'a aucune importance);
— $\displaystyle\left]\frac{2}{3}\right)$ avec `$\displaystyle\left]\frac{2}{3}\right[$` (remarquons qu'on peut associer des délimiteurs différents).

Il existe un autre délimiteur, `.` qui est un délimiteur «vide» au sens qu'il n'affiche rien. Il existe parce que, dans le cas où, par exemple, nous ne voulons rien afficher à droite, le `\right` reste obligatoire, et il faudra alors mettre un `\right.`. Ainsi, on obtient $\displaystyle\left|\frac{2}{3}\right.$ avec `$\displaystyle\left|\frac{2}{3}\right.$`.

Les commandes `\left` et `\right` sont donc des alliées précieuses. Depuis que le moteur e-TeX existe (c'est-à-dire très longtemps maintenant), la primitive `\middle` permet d'insérer un délimiteur à taille ajustable entre ceux spécifiées par `\right` et `\left`. Ainsi, `\[ \left\{ \frac{1}{x} \middle| x \neq 0 \right\} \]` produit:

$$\left\{\frac{1}{x}\middle| x\neq 0\right\}.$$

1.2.2. Des délimiteurs plus avancés

En fait, on peut même contrôler explicitement la taille de nos délimiteurs. En général, il vaut mieux utiliser `\left`, `\right` et `\middle`, mais dans certains cas, ils ne donnent pas une taille optimale aux délimiteurs, ou ne donnent pas la taille qui nous convient. Pour régler ce problème, d'autres commandes sont disponibles, telles que `\big`, `\Big`, `\bigg` et `\Bigg`, qui permettent

de choisir une taille exacte. Ces commandes ont des versions ouvrantes `\bigl`, `\Bigl`, `\biggl` et `\Biggl` et des versions fermantes `\bigr`, `\Bigr`, `\biggr` et `\Biggr`. Il faut respecter leur rôle, et toujours utiliser la version avec `l` pour un délimiteur ouvrant, la version avec `r` pour un délimiteur fermant, et la version initiale pour un délimiteur ni ouvrant ni fermant. Par exemple, `\[\biggl\{ \frac{1}{x} \bigg| x \neq 0 \biggr\} \]` produit:

$$\left\{\frac{1}{x}\middle| x \neq 0\right\}.$$

Ces commandes sont présentes, mais il est rare que leur emploi soit nécessaire. Retenons donc que, dans la grande majorité des cas, utiliser `\left`, `\right` et `\middle` donne un résultat satisfaisant.

1.3. Le texte en mode mathématique

La règle précédemment établie sur l'entrée en mode mathématique et la sortie du mode mathématique mène à cette question.

? Si dans notre formule on veut écrire un mot en mode texte, que faire?

Notre règle permet de répondre assez facilement à cette question. Pour avoir un code source clair, il faut sortir du mode mathématique pour écrire du texte. Ainsi, nous écrirons `$\frac{1}{x}$ existe si $x \neq 0$` pour obtenir $\frac{1}{x}$ existe si $x \neq 0$. De toutes les façons, `$\frac{1}{x}` `existe si x \neq 0$` ne donne clairement pas le résultat escompté, puisqu'on obtient $\frac{1}{x}existesix \neq 0$. Dans ce dernier exemple, le texte reste en italique, mais en plus, les espaces ne sont pas prises en compte.

Pourtant, il y a des moments où il est impossible de sortir du mode mathématique pour écrire du texte. Voici un exemple.

```
\[
   \frac{x}{2}
\]
est un entier s'il existe un entier $q$ tel que
\[
   x = 2q.
\]
```

… donne…

$$\frac{x}{2}$$

est un entier s'il existe un entier q tel que

$$x = 2q.$$

Alors que l'on aimerait obtenir:

$$\frac{x}{2} \text{ est un entier s'il existe un entier } q \text{ tel que } x = 2q.$$

Pour obtenir ce résultat ici, la commande `\text` du *package* **amsmath** de l'American Mathematical Society est utilisée. Plus que ça, c'est la **seule** commande à utiliser pour faire ce travail. Pour se convaincre que d'autres commandes ne conviennent pas, examinons ces exemples:

— la commande `\mathrm` permet d'obtenir $x_{\mathrm{initialestici}}$ et $x^{\mathrm{finalestlà}}$;
— la commande `\mbox` permet d'obtenir $x_{\mbox{initial est ici}}$ et $x^{\mbox{final est là}}$;
— la commande `\text` permet d'obtenir $x_{\text{initial est ici}}$ et $x^{\text{final est là}}$.

Les résultats obtenus avec les commandes `\mathrm` et `\mbox` ne sont pas corrects.

Avec `\mathrm`, les espaces ne sont pas prises en compte. En fait, `\mathrm` permet de ne pas écrire en italique en mode mathématique; les lettres sont donc droites (on dit qu'on écrit en romain). Cette commande est donc à utiliser pour des mathématiques à mettre en romain. On peut par exemple l'utiliser pour les constantes ou le «d» de la différentielle dx, qui doivent être écrits en romain.

La commande `\mbox` ne met pas les textes en indice ou en exposant à la bonne taille. Ce n'est pas une commande qui écrit du texte, elle sert à créer une boîte autour de son argument (dans notre exemple, du texte).

i

Le *package* amsmath est un incontournable lors de l'écriture de mathématiques. Nous allons donc toujours le charger lorsque nous écrivons des mathématiques. Mieux que ça, nous allons charger le *package* **mathtools** à sa place. Ce dernier charge amsmath et règle certains de ses problèmes en plus de fournir des fonctionnalités supplémentaires.

1.4. Les polices mathématiques et les changements de fontes

Nous venons de voir la commande `\mathrm`, qui permet d'écrire des maths en romain plutôt qu'en italique. C'est le moment parfait pour s'intéresser aux différentes polices et fontes mathématiques, et à leur fonctionnement.

1.4.1. Les changements de fontes

Il en existe plusieurs, donc nous n'allons pas toutes les voir. Nous allons voir les plus utilisées, et celles qui peuvent parfois poser problème. Commençons par voir les commandes de changement de fonte:

— la commande `\mathbf` permet de mettre le texte en **gras (*bold font*)**. Notons que le texte n'est pas seulement en gras, mais est aussi droit;
— la commande `\mathrm` permet d'écrire en romain;
— la commande `\mathit` permet de mettre le texte en *italique*.

On pourrait croire que l'écriture en mode mathématique est déjà en italique, mais ces commandes modifient également l'espacement.

Tout ce que nous avons dit précédemment à propos de la commande `\mathrm` s'applique à ces commandes.

Il existe également des polices différentes:

— la commande `\mathbb` permet d'obtenir les lettres dites «ajourées» comme $\mathbb{R}$;
— la commande `\mathcal` permet d'obtenir des lettres «calligraphiques» comme $\mathcal{C}$;
— la commande `\mathfrak` permet d'obtenir des lettres au style gothique comme $\mathfrak{D}$.

Mis à part la commande `\mathfrak`, ces commandes ne marchent que pour les lettres majuscules, et donnent des symboles pour les autres lettres et les chiffres. Notons que `\mathbb` fonctionne quand même avec le k minuscule.

Pour utiliser ces trois dernières commandes, il nous faudra charger le *package* **amsfont**. En fait, nous allons plutôt charger les *packages* mathtools et **amssymb**, amsfont étant automatiquement chargé par amssymb. Le *package* amsmath nous offre de nombreuses commandes, dont `\text`, pour faciliter l'écriture des mathématiques, et le *package* amssymb nous donne des symboles mathématiques supplémentaires.

Notons que nous ne pouvons pas utiliser ces commandes de changement de fontes sur les symboles mathématiques. Ainsi, `\mathbf{2 + 3 = \alpha}` produit $\mathbf{2 + 3 = \alpha}$. Pour mettre les symboles en gras, nous pouvons utiliser la commande `bm` du *package* **bm** . Nous écrirons alors `bm{2 + 3 = \alpha}`.

Nous pouvons de plus remarquer que les lettres grecques majuscules ne sont pas en italique (`\Gamma` donne Γ). Le *package* amsmath fournit les commandes `varGamma` et autres, `\varGamma` produisant $\varGamma$.

1.4.2. Les polices mathématiques

Il est important de savoir que les polices mathématiques sont assez rares. Très peu de polices sont à la fois des polices en mode texte et en mode mathématique. Les polices qui sont généralement utilisées (Computer Modern ou Latin Modern , par exemple) sont aussi des polices mathématiques, mais beaucoup de polices ne sont que des polices de texte, et n'ont pas de support pour le mode mathématique. La liste de ces polices pourra être utile si l'on cherche une police différente de celle par défaut, et qui supporte le mode mathématique.

Lorsque l'on utilise une police sans support des mathématiques, la police mathématique par défaut sera utilisée. Par exemple, si nous affichons des formules mathématiques dans un document utilisant la police French Cursive , les formules mathématiques seront affichées à l'aide de la police par défaut (sûrement Computer Modern).

De plus, certains *packages* fournissent des polices mathématiques, comme mentionné ici . Cela permet d'utiliser une police sans support des mathématiques, et d'utiliser une police mathématique différente de la police par défaut. Par exemple, avec ce code, nous utilisons la police French Cursive pour le texte, et la police Euler pour les mathématiques.

```
% Début du préambule %
\usepackage[default]{frcursive}
\usepackage{eulervm}

\begin{document}
Voici un document écrit avec la police French Cursive, et qui
    utilise la police Euler pour
les mathématiques.
\[
    \sum_{k = 0}^{10} \int_0^{k} x \mathrm{d}x = \sum_{k = 0}^{10}
    \frac{k^2}{2} = \frac{385}{2}
\]
\end{document}
```

1.5. Les espaces

Pour laisser de l'espace entre des éléments en mode mathématique, il va falloir le spécifier explicitement avec des commandes adéquates, puisque les espaces sont ignorées en mode maths.

D'autre part, le placement des objets en mode mathématique, même s'il est très souvent excellent, peut parfois nécessiter des ajustements manuels.

TeX est richement doté en matière d'espaces horizontaux. Ceux spécifiques au mode mathématique ont un nom en gras.

Commande	Nom	Effet
`\`	espace justifiante	$a\ b$
`\>` ou `\:`	**espace moyenne**	$a\,b$
`\;`	**espace épaisse**	$a\;b$
`\,`	espace fine	$a\,b$
`\!`	**espace fine négative**	ab
`\enspace`	demi-cadrat	$a\ \ b$
`\quad`	cadrat	$a\quad b$

`\qquad`	double cadrat	a b
`~`	espace insécable	$a\ b$
`\kern<l>`	espace insécable de longueur `<l>` (qui peut être négative)	a b (on a pris `l`=2 cm)
`\hfil` ou `\hfill`	espace d'élasticité infinie (s'étire autant qu'il le faut)	

Il est également utile de signaler la macro de LaTeX `\hphantom{…}`, dont l'effet est d'insérer une espace horizontale de la même dimension que son argument: `\mathrm C^{15}_9` donne C^{15}_9 tandis que `\mathrm C^{15}_{\hphantom{1}9}` donne $\mathrm{C}^{15}_{\hphantom{1}9}$.

Voici un premier exemple où le modulo est espacé de la relation de congruence d'un cadrat avec `\[x\equiv 10 \quad[7]\]`:

$$x \equiv 10 \quad [7]$$

Dans l'exemple ci-dessous, l'emploi d'une espace négative semble préférable pour retoucher à la main le placement de l'exposant. En effet, le code `\[ \left(\dfrac 12\right)^n \]` donne:

$$\left(\frac{1}{2}\right)^n$$

L'exposant peut être approché de la parenthèse avec `\!`, voire avec `\!\!`, et `\[ \left(\dfrac 12\right)^{\!n} \qquad \left(\dfrac 12\right)^{\!\!n} \]` produit:

$$\left(\frac{1}{2}\right)^n \qquad \left(\frac{1}{2}\right)^n$$

Il ne faut pas abuser de ces retouches manuelles, la plupart des objets sont généralement bien placés.

1.5.1. Dans les nombres

Si nous voulons formater l'affichage des nombres (regrouper les nombres avant et après le séparateur décimal en groupes de trois séparés par une espace, et supprimer l'espace indésirable après la virgule), le *package* **siunitx** nous sera d'une grande aide. Il s'agit d'un *package* très complet, qui offre plusieurs types de formatages pour les nombres, et permet également de taper des unités très facilement.

Pour l'utiliser, nous allons tout d'abord faire quelques petites configurations. En effet, siunitx utilise le point comme séparateur décimal par défaut. Nous allons donc charger la configuration française.

```
\sisetup{locale = FR}
```

Voyons un exemple d'utilisation du *package*. La commande que nous allons voir ici est la commande `\num`. Elle peut s'utiliser en mode texte et en mode mathématique. Voici un code commenté où cette commande est utilisée.

```
\num{299792458}  et          % Il y a des espaces.
\num{299792,458} et          % On obtient une virgule comme
    séparateur décimal.
\num{.34}        et          % Même en utilisant le point, on
    obtient une virgule et un zéro est placé.
\num{299792,458 +- 0.09} et  % On peut obtenir le signe « plus ou
    moins » avec +-.
\num{3e8}        et          % « e » permet d'obtenir la puissance
    de 10 (« E » aussi).
\num{3d8}.                   % « d » et « D » permettent aussi
    d'obtenir la puissance de 10.
```

Après compilation, on obtient bien le formatage attendu.

Notons que la commande `\numprint` du *package* **numprint** peut également faire ce travail de formatage. Elle peut être utilisée en dehors du mode mathématique, et est vraiment très simple: `\numprint{299792,458}`.

2. De bons environnements

2.1. Équations seules

Pour obtenir une équation (et bien une seule), nous pouvons utiliser l'environnement `equation`. Il s'utilise en dehors du mode mathématique, et donne comme résultat des mathématiques «hors texte». Les équations obtenues sont automatiquement numérotées. On obtient alors:

$$y = 2x + 5 \quad (1)$$

Et ce avec ce code.

```
\begin{equation}
   y = 2x + 5
\end{equation}
```

Notons que le *package* amsmath nous procure une version étoilée de l'environnement `equation`, la seule différence avec l'environnement `equation` étant que la numérotation n'est pas faite. Finalement, avec l'environnement `equation*`, on obtient la même chose qu'avec `\[ \]`.

2.1.1. Les sous-équations

Le *package* amsmath nous procure également l'environnement `subequations`, qui agit sur la numérotation des équations qu'il contient. Les équations placées à l'intérieur de cet environnement seront considérées comme des sous-équations, et auront donc une «sous-numérotation». Voyons un exemple.

```
On a l'équation d'une droite $\Delta$
\begin{equation}
  y = ax + b.
\end{equation}
Et on connaît les coordonnées de deux points de la droite. Elles
    nous permettent d'écrire :
\begin{subequations}
        \begin{equation}
          10 = 5a + b
        \end{equation}
        \begin{equation}
```

```
11              0 = a + b
12            \end{equation}
13  \end{subequations}
```

Ici, on aura la première équation numérotée (1) et les sous-équations numérotées (1a) et (1b). Il est bien sûr possible de changer le style de numérotation.

2.1.2. Des équations sur plusieurs lignes avec `multline`

L'environnement `multline` est un environnement qui permet de présenter des équations sur plusieurs lignes. La première ligne est alignée à gauche, les suivantes sont centrées, et la dernière est alignée à droite. `multline` s'utilise en dehors du mode mathématique, et est assez intuitif. Il suffit de séparer chaque équation par le signe `\\`, qui permet de passer à la ligne suivante. L'environnement `multline` numérote la dernière ligne. Sa variante étoilée permet de ne pas numéroter. On obtient alors:

$$\begin{aligned} 2 \times 20 = 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \\ +2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \\ +2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \\ +2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \end{aligned}$$

… avec…

```
1  \begin{multline*}
2     2 \times 20 = 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2\\
3                 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2\\
4                 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2\\
5                 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2
6  \end{multline*}
```

Notons que nous n'avons pas placé de `\\` après la dernière ligne. Si nous l'avions fait, la dernière ligne aurait été centrée, et non pas alignée à droite. En fait, il vaut mieux ne pas placer de `\\` après la dernière ligne. Dans tous les autres environnements que nous verrons, nous obtiendrons une erreur en le faisant.

2.1.3. Des équations sur plusieurs lignes avec `split`

L'environnement `split` permet également d'écrire des équations sur plusieurs lignes, mais nous propose en plus un alignement vertical. Au contraire de `multline`, il doit être utilisé dans un environnement mathématique. Il ne numérote pas l'équation. Si nous voulons la numéroter, il nous faudra utiliser `split` dans un environnement qui numérote (`equation` par exemple).

Son utilisation est similaire à celle de `multline`, avec toujours `\\` pour passer à la ligne. L'alignement vertical se fait au niveau du symbole `&`. Il nous faudra donc le placer sur chaque ligne. On obtient par exemple:

$$\begin{aligned} 2 \times 20 = 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \\ + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 \end{aligned}$$

en utilisant ce code.

```
\[
\begin{split}
   2 \times 20 = 2 &+ 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2\\
                   &+ 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2
\end{split}
\]
```

Notons que l'équation est centrée.

Nous pouvons également utiliser `split` pour écrire une suite d'opérations avec par exemple ceci.

```
\[
\begin{split}
   (2(a + b) - 3a) \times (2 + a) &= (2a + 2b - 3a) \times (2 +
    a)\\
                                  &= (2b - a) \times (2 + a)\\
                                  &= 4b + 2ab - 2a + a^2
\end{split}
\]
```

D'après la documentation de amsmath, l'environnement `split` est conçu pour être le corps complet d'une équation (ou d'autres environnements mathématiques). En particulier, il ne peut rien y avoir avant ou après le `split` dans le corps de cette équation.

En pratique, cela fonctionne parfois, mais lorsque nous avons besoin d'autres choses que le contenu du `split`, nous utiliserons un autre environnement.

2.2. Des groupes d'équations

Nous allons maintenant voir comment faire pour écrire plusieurs équations à la suite. Nous pourrions pour cela utiliser plusieurs environnements d'équation seule à la suite, mais nous allons plutôt profiter des environnements que nous propose notamment le *package* amsmath. Ils nous permettront, par exemple, d'aligner les équations.

2.2.1. Des groupes d'équations centrées avec `gather`

L'environnement `gather` de amsmath permet d'écrire une équation par ligne, chacune de ces équations étant centrée et numérotée. Là encore, il s'utilise avec `\\` pour passer à la ligne suivante. Il a, lui aussi, une version étoilée, qui a pour seule différence de ne pas numéroter les équations (en fait, la majorité des environnements d'équations numérotées fonctionnent de cette manière, et ont une version étoilée qui ne numérote pas). Par exemple on obtient:

$$2x + 3 = 23$$
$$a + b + c + d = e$$
$$x = 10$$

... avec...

```
\begin{gather*}
   2x + 3 = 23\\
   a + b + c + d = e\\
   x = 10
\end{gather*}
```

2.2.1.1. La variante `gathered`

L'environnement `gathered` est une variante de l'environnement `gather`. On peut le considérer comme sa version «sous-bloc», dans le sens où il faut déjà être dans un environnement mathématique pour l'utiliser. On peut l'utiliser en mode «hors texte», mais aussi en mode «en ligne». De plus, les lignes ne sont pas numérotées. Ainsi, on peut par exemple l'utiliser dans un environnement `equation` pour numéroter tout un bloc d'équations et non chacune des équations, avec ce code.

```
\begin{equation}
   \begin{gathered}
      a &= 34 + x & b &= 2 + a & c &= a + b \\
      d &= 22 + a & d &= 22x   & d - c &= 0
   \begin{gathered}
\end{equation}
```

Nous remarquons que la numérotation obtenue est centrée.

La plupart des environnements que nous verrons possèdent une variante finissant en `ed` qui s'utilise en mode mathématique, et peut être utilisée autant en mode «hors texte» qu'en mode «en ligne». Le *package* mathtools en fournit même une pour l'environnement `multline`.

2.2.2. Des équations alignées avec `align`

L'environnement `align` nous permet de faire plusieurs colonnes d'équations. Il s'utilise comme un environnement de type tableau, `&` permettant de passer à la colonne suivante et `\\` permettant de passer à la ligne suivante. Le groupe d'équations obtenu est centré, et chaque ligne est numérotée. Pour ne pas avoir de numérotation, il nous faudra utiliser sa version étoilée.

L'environnement `align` fonctionne de cette manière: il n'y a pas d'espace entre une colonne de rang impair et la colonne suivante, de sorte qu'un signe `&` sur deux a un comportement d'alignement. L'espace entre une colonne paire et la commande impaire qui suit est calculé de sorte qu'ils soient tous égaux à l'espace entre la première colonne et la marge de gauche, et à l'espace entre la dernière colonne et la marge de droite.

Cela peut sembler flou, mais un exemple nous permettra de mieux comprendre son fonctionnement:

$$\begin{aligned} a &= 34 + x & b &= 2 + a & c &= a + b \\ d &= 22 + a & d &= 22x & d - c &= 0 \end{aligned}$$

Lequel est obtenu avec ce code.

```
\begin{align*}
   a &= 34 + x & b &= 2 + a & c &= a + b \\
   d &= 22 + a & d &= 22x   & d - c &= 0
\end{align*}
```

Nous obtenons six colonnes, l'espace entre la première et la deuxième colonne est nul, de même que celui entre la troisième et la quatrième, ou que celui entre la cinquième et la sixième.

i

La documentation de amsmath parle de « colonnes d'équations » pour désigner deux colonnes qui ne sont pas séparés par un espace (donc une colonne de rang impair et la colonne qui suit). Cela semble naturel, on voit dans notre exemple précédent qu'on a bien trois colonnes d'équations. Il nous faut bien distinguer le nombre de colonnes, et le nombre de colonnes d'équations.

2.2.2.1. La variante `aligned`

`align` a également une version «sous-bloc», qu'on peut utiliser pour numéroter tout le bloc d'équations plutôt que chaque équation. En outre, le fait de pouvoir l'utiliser en mode «en ligne» permet d'écrire facilement des systèmes d'équations, tels que $\begin{cases} 2x + 5y = 24 \\ 23x + y = 25 \end{cases}$ obtenu avec ce code.

```
$
\left\{
   \begin{aligned}
       2x + 5y &= 24 \\
       23x + y &= 25
   \end{aligned}
\right.
$
```

2.2.3. Des équations alignées avec `alignat`

L'environnement `alignat` est une variante de l'environnement `align`, qui ne place pas d'espaces entre les différentes colonnes d'équations, ce qui fait qu'ils devront être si besoin spécifiés par l'utilisateur. Cet environnement prend un argument qui correspond au nombre de colonnes d'équations (donc au nombre de colonnes divisé par 2). Sa variante étoilée permet, là encore, de ne pas avoir de numérotation.

$$\begin{aligned} x &= a + 2b && \text{selon la première relation} \\ x &= 2a + d && \text{selon la relation fondamentale} \end{aligned}$$

Cet exemple s'obtient avec le code suivant.

```
\begin{alignat*}{2}
    x &= a + 2b & \quad & \text{selon la première relation}\\
    x &= 2a + d &       & \text{selon la relation fondamentale}
\end{alignat*}
```

2.2.3.1. La variante `alignedat`

L'environnement `alignat` a aussi une version «sous-bloc». Notons qu'elle s'appelle `alignedat` et non `alignated`. On peut, par exemple, l'utiliser pour définir une fonction de cette manière:

$$\begin{aligned} f\colon E &\to F \\ x &\mapsto f(x) \end{aligned}$$

… avec…

```
\[
\begin{alignedat}{2} f \colon & E &\: &\to  F\\
                              & x & &\mapsto f(x)
\end{alignedat}
```

```
\]
```

2.3. Commandes pour les équations

2.3.1. Des références aux équations

Nous pouvons profiter du système de référence de LaTeX pour renvoyer à une équation particulière. On pourra alors renvoyer vers «l'équation (1) de la page 6». Pour ce faire, nous allons utiliser la commande `\label` habituelle pour créer nos étiquettes. Cependant, la commande `\ref` sera remplacée par la commande `\eqref` du *package* amsmath. Essayons, par exemple, ce code.

```
On a l'équation d'une droite $\Delta$
\begin{equation}
   y = ax + 5. \label{eq:droite}
\end{equation}
Le point $A(5, 10)$ appartient à la droite $\Delta$, donc, d'après
    l'équation
\eqref{eq:droite}, on a que $10 = 5a + 5$ c'est-à-dire $a = 1$.
```

Comme pour les références normales, il sera parfois nécessaire de compiler plusieurs fois pour obtenir un résultat.

2.3.2. Supprimer la numérotation d'une équation

Il peut arriver que l'on veuille supprimer la numérotation de certaines équations dans un groupe d'équations numérotées. Pour ce faire, nous allons utiliser la commande `\nonumber`, à placer après le signe `\\` qui marque la fin de la ligne en question. Avec le *package* amsmath, nous pouvons aussi utiliser la commande `\notag`, qui est équivalente à `\nonumber`. Par exemple, avec le code qui suit, la deuxième ligne ne sera pas numérotée.

```
\begin{align}
   a &= 2\\
   b &= 3\\ \notag
   c &= 5
\end{align}
```

2.3.3. La commande `intertext`

Toujours dans un environnement de regroupement d'équations, il peut arriver que nous voulions quitter temporairement cet environnement pour écrire du texte «normalement» entre deux équations. Nous pourrions bien sûr quitter l'environnement, écrire notre texte, et ensuite rouvrir un nouvel environnement, mais ce choix ne permettrait pas, par exemple, de garder un alignement entre les deux environnements. Le *package* amsmath nous fournit alors la commande `\intertext`, qui interrompt une série d'équations pour placer du texte sans quitter l'environnement en question. Par exemple, avec ce code.

```
\begin{align*}
   a + b &= e\\
   c + b &= z\\
   \intertext{et on a encore}
   a + c &= y + z - d
\end{align*}
```

L'espacement vertical laissé par `\intertext` peut ne pas nous plaire. Le *package* mathtools fournit alors la commande `\shortintertext` qui produit de moins grands espaces verticaux.

2.4. Les disjonctions de cas

Les disjonctions de cas sont assez fréquentes en mathématiques. Par exemple, pour définir une fonction par morceau, on pourrait écrire:

$$f(x) = \begin{cases} 0 & \text{si } x = 0 \\ \frac{\sin x}{x} & \text{sinon} \end{cases}$$

Pour obtenir ce résultat, le *package* amsmath propose l'environnement `cases`. Il doit être utilisé en mode mathématique, et fonctionne autant en mode «hors texte» qu'en mode «en ligne». Le résultat précédent s'obtient grâce à ce code.

```
\[
f(x) =
\begin{cases}
   0                 &\text{si $x = 0$}\\
   \frac{\sin x}{x} &\text{sinon}
\end{cases}
\]
```

Notons l'usage du signe `&`, qui nous permet de spécifier l'endroit où doit se faire l'alignement. On ne peut utiliser qu'un seul signe `&` par ligne.

2.4.1. Les commandes similaires à `cases` de mathtools

Le *package* mathtools nous donne plusieurs variantes de l'environnement `cases`. Tout d'abord, l'environnement `dcases` (pour *display cases*) compose les mathématiques en mode «hors texte» (dans notre exemple, $\frac{\sin x}{x}$ aurait alors changé de taille).

On peut également placer l'accolade à droite de nos différents cas avec l'environnement `rcases` pour *right cases*.

Finalement, le *package* mathtools fournit une version étoilée de `cases`, dans laquelle les éléments de la seconde colonne (donc ce qui est placé après `&`) sont composés en mode texte.

Et toutes ces fonctionnalités peuvent être combinés, ce qui donne lieu à ces environnements:

— `dcases*` qui combine les fonctionnalités de `cases*` et de `dcases`;
— `rcases*` qui combine les fonctionnalités de `cases*` et de `rcases`;
— `drcases` qui combine les fonctionnalités de `rcases` et de `dcases`;
— `drcases*` qui combine les fonctionnalités de `cases*`, de `rcases` et de `dcases`.

2.5. Les matrices

Les matrices sont également des structures communes en mathématiques. Nous pourrions utiliser un environnement d'alignement, et utiliser des délimiteurs; mais là encore, le *package* amsmath propose des environnements pour obtenir le résultat souhaité. En fait, il offre une demi-douzaine d'environnements, chacun étant associé à une paire de délimiteur. Ainsi:

— `matrix` ne place aucun délimiteur;
— `pmatrix` permet d'obtenir $\begin{pmatrix};$
— `bmatrix` permet d'obtenir $\begin{bmatrix};$
— `Bmatrix` permet d'obtenir $\begin{Bmatrix};$
— `vmatrix` permet d'obtenir $\begin{vmatrix};$
— `Vmatrix` permet d'obtenir $\begin{Vmatrix};$

Ces environnements s'utilisent dans le mode mathématique, et sont assez intuitifs à utiliser. On utilise `\\` pour passer à la ligne suivante, et `&` pour passer à l'élément suivant. Ainsi, on obtient:

$$\begin{pmatrix} 2 & 1 & 32 & 1 \\ 0 & \pi & 1 & 1 \\ 0 & 0 & -\sqrt{2} & 1 \\ 0 & 0 & 0 & 245 \end{pmatrix}$$

… avec ce code…

```
\[
\begin{pmatrix}
```

```
   2 & 1 & 32 & 1\\
   0 & \pi& 1 & 1\\
   0 & 0 & -\sqrt{2} & 1\\
   0 & 0 & 0 & 245
\end{pmatrix}
\]
```

Si nous souhaitons placer des points à la place d'éléments d'une matrice, il existe la commande `hdotsfor[s]{n}` qui remplit n colonnes de points espacés de s.

Notons que le *package* mathtools fournit une version étoilée de ces environnements. Ces environnements étoilés prennent un argument optionnel qui permet de spécifier l'alignement des colonnes de la matrice. Ainsi, avec `l`, elles seront alignées à gauche, avec `c` elles seront centrées, et avec `r` elles seront alignées à droite. Par défaut, les colonnes sont centrées (et cela revient alors à utiliser les environnements non étoilés).

2.5.1. Fractions en éléments de matrices

En essayant de placer une fraction avec `\dfrac` dans une matrice, nous obtenons un résultat qui n'est pas très esthétique: les espaces sont vraiment mauvais. Pour régler ce problème, nous allons utiliser la commande `\strut` avec le numérateur et avec le dénominateur de chaque fraction. Nous allons alors écrire ceci.

```
\[
\begin{pmatrix}
   \dfrac{\strut 3}{\strut 2}\\
   \dfrac{\strut 1}{\strut 2}
\end{pmatrix}
\]
```

2.5.2. Matrices en mode « en ligne »

Les commandes précédentes peuvent également s'utiliser en mode «en ligne», mais ont le désavantage de donner une matrice de taille conséquente. Pour écrire une matrice plus petite, le *package* amsmath nous fournit l'environnement `smallmatrix`, qui s'utilise exactement de la même manière que l'environnement `matrix`. Il n'en fournit pas de variante pour les différents délimiteurs, mais là encore le *package* mathtools vient le compléter et fournit toutes les variantes de `smallmatrix`. Ainsi, nous pourrons utiliser les environnements `vsmallmatrix` ou encore `Bsmallmatrix`.

Ces environnements disposent eux aussi de versions étoilées permettant de spécifier l'alignement des colonnes.

2.6. Un environnement souple, array

L'environnement `array` permet de faire des tableaux en mode mathématique. Il a un paramètre obligatoire qui correspond à l'alignement de chacune des colonnes du tableau. Cet argument se compose des lettres `l`, `c`, `r` et du symbole `|`, et peut par exemple être `|c|lr|l|`. Il se comprend de cette manière:

— `l` signifie `left`, donc que les éléments de la colonne correspondante seront alignés à gauche;
— `c` signifie `center`, donc que les éléments de la colonne correspondante seront centrés;
— `r` signifie `right`, donc que les éléments de la colonne correspondante seront alignés à droite;
— `|` symbolise la barre verticale qui sépare deux colonnes.

Ainsi, avec `|c|lr|l|`, on demande un tableau de quatre colonnes. Les éléments de la première colonne sont centrés, ceux de la deuxième et de la quatrième sont alignés à gauche et ceux de la troisième sont alignés à droite. De plus, il y a un trait vertical avant la première colonne, entre la première et la deuxième colonne, entre la troisième et la quatrième colonne et après la quatrième colonne.

Voyons maintenant que mette à l'intérieur de l'environnement. Pour cela, analysons un exemple.

```
\[
\begin{array}{c|l r|}
   c   & l   & r \\ \hline
   c_1 & l_1 & r_1
   c_2 & l_2 & r_2 \\ \hline
   c_3 & l_3 & r_3
\end{array}
\]
```

On obtient:

$$\begin{array}{c|l r|}
   c   & l   & r \\ \hline
   c_1 & l_1 & r_1
   c_2 & l_2 & r_2 \\ \hline
   c_3 & l_3 & r_3
\end{array}$$

Et on voit que:

— le symbole `&` permet de séparer les différents éléments d'une même ligne d'un tableau;
— le symbole `\\` permet de passer à la ligne suivante;
— la commande `\hline` insère une ligne horizontale sur la longueur du tableau.

Son utilisation n'est pas compliquée.

Il nous faire attention à différents points. Déjà, le nombre de colonnes du tableau est fixé par le paramètre de l'environnement `\array`. On ne peut pas indiquer en paramètre que le tableau a n colonnes pour ensuite faire une ligne qui n'a pas n colonnes. De plus, notons qu'après la dernière ligne du tableau, il n'y a pas de symbole `\\`.

L'environnement `array` a pour particularité d'être extrêmement souple: nous pouvons obtenir les alignements que nous voulons, nous pouvons placer des lignes verticales et horizontales là où nous le voulons... Nous pourrions, par exemple, l'utiliser pour faire des matrices avec `\left( \begin{array} \end{array}\left)`.

3. Créer ses propres commandes

Nous avons vu quelques outils. Mais c'est un peu fatigant de réécrire à chaque fois la même chose. Et puis, on peut ne pas écrire la même chose, et ça pour la normalisation, on en reparlera. Donc pour se simplifier la vie, nous allons créer de jolies commandes.

3.1. Ensembles et intervalles

3.1.1. $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$

Nous allons commencer par des commandes pour les ensembles et les intervalles. La commande `\mathbb` permet d'avoir les notations pour les ensembles de nombres $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$. Ce serait bien d'avoir des commandes pour eux. Bien sûr, il est logique de penser à écrire une commande de ce genre pour chaque ensemble: `\newcommand*\N[1]{\mathbb{#1}}`.

Ce n'est pas ce que nous allons faire. Imaginons que plus tard, nous décidons d'utiliser par exemple cette notation $\mathbf{N}$ disponible avec `\mathbf{N}`. Il nous faudrait réécrire toutes les commandes. C'est pourquoi nous allons d'abord définir une commande `\ensembleNombre` qui se chargera d'appliquer à nos lettres le formatage désiré.

```
\let\ensembleNombre\mathbb
\newcommand*\N{\ensembleNombre{N}}
\newcommand*\Z{\ensembleNombre{Z}}
\newcommand*\Q{\ensembleNombre{Q}}
\newcommand*\R{\ensembleNombre{R}}
\newcommand*\C{\ensembleNombre{C}}
```

Pour encore plus de facilité dans la saisie des macros que nous venons de définir, on pourrait souhaiter qu'elles puissent être appelées aussi bien en mode mathématique qu'en mode texte. Nous aurions pu utiliser la macro `\ensuremath{…}` pour définir la macro `\N` de cette manière.

```
\newcommand*\N{\ensuremath{\ensembleNombre{N}}}
```

Nous n'avons désormais plus à nous soucier de savoir si `\N` est appelée en mode mathématique ou pas, et elle affichera $\mathbb{N}$ dans tous les cas.

Nous avons défini des commandes permettant d'écrire très facilement ces ensembles.

3.1.2. Les intervalles

Nous pouvons écrire plusieurs sortes d'intervalles. Donc nous allons également faire d'abord une commande générale, qui se chargera du formatage, puis utiliser cette commande pour définir cinq autres commandes pour les quatre types d'intervalles et les intervalles d'entiers.

```
\newcommand*\intervalle[4]{\left#1 #2 \, ; #3 \right#4}
\newcommand*\intervalleOO[2]{\intervalle{]}{#1}{#2}{[}}
\newcommand*\intervalleFF[2]{\intervalle{[}{#1}{#2}{]}}
\newcommand*\intervalleOF[2]{\intervalle{]}{#1}{#2}{]}}
\newcommand*\intervalleFO[2]{\intervalle{[}{#1}{#2}{[}}
```

`\intervalleOO` permet d'obtenir un intervalle ouvert des deux côtés, `\intervalleOF` un intervalle ouvert à gauche et fermé à droite...

En fait, on peut même créer une commande pour écrire ces intervalles de manière naturelle.

Le code qui suit est difficile à comprendre pour un débutant et nous n'allons pas le commenter. Nous pouvons parfaitement nous contenter de l'utiliser.

```
\newcommand\actimath[1]{\mathcode`#1"8000 \begingroup \lccode`~`#1
    \lowercase{\endgroup\def~}}
\newcommand\interv{%
    \begingroup
    \actimath\;{\mathpunct{}\mathpunct{\mathchar`\;}}%
    \actimath\[{\right\delimiter\delcode`\[ \endgroup}%
    \actimath\]{\right\delimiter\delcode`\] \endgroup}%
    \left\delimiter\delcode`}
```

Elle s'utilise en écrivant `$\interv[1;10]$` et `$\interv]\frac{1}{2};25]$` pour obtenir $[1;10]$ et $\left]\frac{1}{2};25\right]$.

Pour définir `\intervalleEntier`, nous allons utiliser les commandes `\llbracket` et `\rrbracket` du *package* **stmaryrd** pour obtenir les doubles crochets.

```
\newcommand*\intervalleEntier[2]{\intervalle{\llbracket}{#1}{#2}{\rrbracket}}
```

Nous aurions aussi pu utiliser `[\![` et `]\!]`, mais ces crochets ne sont pas exactement les bons.

On obtient ainsi très facilement $[\![1, 10]\!]$ avec `\intervalleEntier{1}{10}`.

3.1.3. Les ensembles du type « tel que »

Il nous reste à définir une commande pour les ensembles définis en compréhension, c'est-à-dire avec «tel que». Nous voulons donc faire une commande qui nous permette d'obtenir $\{x \in \mathbb{R},\ x \neq 0\}$. Cette commande est assez simple à faire. La seule question à se poser est qu'utiliser pour représenter «tel que»; la virgule, le *slash*, la barre verticale ou même «tq» sont des choix possibles. Prenons, par exemple, la virgule. Nous allons ici appeler notre fonction `enstq` pour ensemble tel que.

```
\newcommand*\enstq[2]{\left\{#1,\; #2\right\}}
```

3.2. Symboles en furie

Il y a une autre chose pour laquelle il peut être judicieux de créer ses commandes. C'est pour tout ce qui nécessite l'usage de symboles, surtout si ceux-ci ont une taille variable.

3.2.1. Valeur absolue

Un exemple flagrant est celui de la valeur absolue. Avoir écrit `\left\lvert x \right\rvert` dans un code ne laisse pas du tout penser qu'il s'agit d'un code pour afficher $|x|$. Pour rendre ceci plus lisible, mettons-le dans une commande, et profitons-en pour faire la même chose pour la partie entière et pour la norme.

```
\newcommand*\ent[1]{\left\lfloor #1\right\rfloor}
\newcommand*\norme[1]{\left\lVert #1\right\rVert}
\newcommand*\abs[1]{\left\lvert #1\right\rvert}
```

Nous aurions également pu nous intéresser aux commandes `\DeclarePairedDelimiter` ou `\DeclarePairedDelimiterx` du *package* mathtools pour faire ce travail.

3.2.2. Vecteurs

La commande `\vec{arg}` permet d'afficher un vecteur à l'aide d'une flèche, et d'obtenir $\vec{x}$. Cependant, pour écrire un vecteur avec deux lettres par exemple, elle n'est pas adaptée. Essayons: $\vec{AB}$. Nous préférerions avoir ceci: $\overrightarrow{AB}$. Il nous suffit de créer une commande qui place une flèche au-dessus de son argument.

```
\let\vecteur\overrightarrow
```

Et elle permet également d'obtenir $\vec{x}$ ou $\overrightarrow{ABC}$. On a donc bien une commande adaptée à tous les cas.

? Une petite question: pourquoi ne pas avoir redéfini la commande `\vec` plutôt que d'en définir une nouvelle?

On aurait pu en effet redéfinir la commande `\vec`, mais il vaut mieux choisir de créer de nouvelles commandes. Cela peut prévenir des erreurs, et on ne sait jamais si on aura besoin de la commande originelle. En fait, les commandes à redéfinir sont celles qui, par exemple, obéissent aux notations anglaises, alors qu'on sait que, dans notre document, on n'utilisera que les notations françaises...

En fait, là encore un *package*, le *package* **esvect** nous permet d'obtenir une flèche de la bonne taille sur nos vecteurs grâce à sa commande `\vv`. En fait, il fait même encore mieux, puisqu'il nous permet d'obtenir une flèche plus esthétique, et offre quasiment une dizaine de types de flèches. Il ne faut pas hésiter à l'utiliser.

Cependant, il ne règle pas un problème: certains préfèrent avoir toutes les flèches des vecteurs à la même hauteur ; ils veulent avoir $\overrightarrow{AB}$ et $\overrightarrow{x}$ plutôt que $\overrightarrow{AB}$ et $\vec{x}$. Créons donc une commande qui place la flèche de cette manière. Pour cela, nous allons placer une espace verticale invisible de taille supérieure à celle des caractères. Ainsi, la flèche sera toujours au-dessus de cette espace, donc à la même hauteur.

La commande `\mathstrut` permet de placer l'espace voulue. Elle est strictement équivalente à `\vphantom{(}`, c'est-à-dire qu'elle produit une espace verticale de la taille d'une parenthèse. Ce qui nous permet d'écrire notre commande.

```
1 \newcommand*\vecteur[1]{\vv{\mathstrut #1}}
```

Dans certains documents, nous pourrons préférer écrire les vecteurs en gras par souci de simplicité (en physique notamment). La question qui se pose est quelle commande utiliser pour mettre en gras votre vecteur. Nous allons utiliser `\mathbf` pour obtenir $\mathbf{AB}$. Nous pouvons aussi utiliser la commande `\boldsymbol` qui, elle, donne $\boldsymbol{AB}$. Notre commande `\vecteur` devient doncceci.

```
1 \let\vecteur\boldsymbol
```

3.3. Les fonctions

Dans le mode mathématique, les fonctions les plus courantes disposent d'une commande qui les affiche en romain. Ainsi, `$\sin x$` permet d'obtenir $\sin x$. Mais comment faire pour écrire les noms d'autres fonctions ou opérateurs en romain? Prenons l'exemple d'une fonction «card».

Nous aurions pu utiliser la commande `\text`, mais comme nous l'avons dit, elle est là pour écrire du **texte**, et ce n'est pas du texte que nous voulons écrire. De même, la commande

`\mathrm` ne permet tout simplement pas d'obtenir le comportement d'une fonction, puisque `$x\mathrm{cos} x$` donne $x\mathrm{cos}x$ au lieu de $x \cos x$.

Le *package* amsmath dispose d'une commande `\operatorname` qui nous permet d'obtenir le résultat désiré. Ainsi, `\operatorname{card} x` donne $\operatorname{card} x$.

En fait, on a encore mieux avec la commande `\DeclareMathOperator` (toujours du *package* amsmath), qui permet de définir une commande pour un nouvel opérateur. Elle prend comme premier paramètre la commande à définir, et comme deuxième paramètre ce qui doit être affiché.

```
1 \DeclareMathOperator{\card}{card} % On déclare l'opérateur \card
      qui correspond à « card ».
```

Ainsi, `\card E` donnera $\operatorname{card} E$.

Notons que la variante étoilée de la commande `\DeclareMathOperator` permet d'obtenir un opérateur mathématique dont les exposants et les indices seront placés comme pour `\sum`, c'est-à-dire au-dessus et au-dessous de l'opérateur.

3.4. Environnements particuliers

3.4.1. Les systèmes d'équations

Nous avons vu dans la partie précédente comment écrire des systèmes d'équations en utilisant l'environnement `aligned`. Il paraît judicieux de créer une commande qui mette en place l'environnement. Cette commande n'est pas compliquée à faire, elle ne prend qu'un argument, et ne fait que le placer dans un environnement `aligned`.

```
1 \newcommand*\Sys[1]{\left\{ \begin{aligned} #1 \end{aligned} \right.
      \kern-\nulldelimiterspace}
```

Notons également l'existence du *package* **systeme** , qui permet également d'obtenir des systèmes d'équations.

3.4.2. Définir une fonction

Nous allons juste placer un environnement `alignedat` dans une commande `fonction`.

```
1 \newcommand*\fonction[5]{
2 #1 \colon \left\{\begin{alignedat}{2}  &#2 &\: &\to      #3\\
3                                 &#4 &   &\mapsto  #5
```

```
\end{alignedat} \right. \kern-\nulldelimiterspace}
```

On peut alors obtenir f : $\begin{cases} E & \to F \\ x & \mapsto f(x) \end{cases}$ avec `\fonction{f}{E}{F}{x}{f(x)}`.

Nous aurions aussi pu utiliser l'environnement `aligned`.

3.4.3. Les coordonnées d'un point

On peut écrire les coordonnées d'un point de l'espace de plusieurs manières en mathématique. On peut vouloir les noter entre parenthèses séparées d'une virgule, ou encore l'une au-dessus de l'autre dans le cas des coordonnées d'un vecteur. Dans le premier cas, il n'est pas nécessaire de créer de commandes, puisque `$A(8, 9)$` permet déjà d'obtenir facilement $A(8, 9)$, mais dans les deux autres cas, il est bon de faire une commande pour s'affranchir une bonne fois pour toutes de ce travail. Voici ce que nous aimerions obtenir:

$$\overrightarrow{AB} \begin{pmatrix} 10 \\ 6 \end{pmatrix} \text{ ou } \overrightarrow{AB} \left| \begin{matrix} 10 \\ 6 \end{matrix} \right.$$

Pour ce faire, nous allons tout simplement utiliser les environnements de matrices.

```
\newcommand*\coord[2]
{
   \vecteur{#1} \,
   \begin{pmatrix}
      #2
   \end{pmatrix}
}
```

Avec cette commande il faut écrire `\[\coord{AB}{12\\21} \coord{BC}{21\\12\\32}\]` pour obtenir:

$$\overrightarrow{AB} \begin{pmatrix} 12 \\ 21 \end{pmatrix} \overrightarrow{BC} \begin{pmatrix} 21 \\ 12 \\ 32 \end{pmatrix}$$

Pour obtenir l'écriture avec la barre, il nous suffit d'utiliser `matrix` à la place de `pmatrix`, et de placer les délimiteurs adaptés, à savoir `\left|` et `\right.`.

N'oublions pas, si nous voulons insérer des fractions dans un environnement de type matrice et garder un bon espacement, nous utilisons la commande `\mathstrut`.

Voilà, nous avons vu la création de quelques commandes. Nous pouvons faire des commandes pour d'autres choses. Mais attention, ne tombons pas dans l'excès en faisant une commande pour tout et n'importe quoi!

Nous avons également vu que certains *packages* pouvaient déjà faire ce que nous voulions. Si un *package* existe déjà pour faire ce que nous voulons, il vaut mieux l'utiliser que de chercher à tout prix à faire notre propre commande. Bien sûr, dans certains cas, ce *package* ne fait pas exactement ce que nous voulons, ou n'existe tout simplement pas. Dans ce cas, nous pouvons faire une commande, et même en faire profiter les autres en créant un *package*!

Lorsque nous décidons d'utiliser un *package*, il faut vérifier qu'il n'est pas obsolète, et s'il n'existe pas de *package* le remplaçant. Par exemple, le *package* **theorem** est obsolète et il est conseillé d'utiliser à la place le *package* **ntheorem** ou le *package* **amsthm** (à noter qu'un tutoriel sur le *package* ntheorem est disponible ici).

4. Réglages du mode mathématique

Dans ce chapitre, qui peut être omis en première lecture, nous allons voir comment changer quelques paramètres du mode mathématique, en particulier les paramètres correspondant à des longueurs (taille d'une espace, par exemple). Les valeurs doivent être données avec leur unité, qui peut être en `cm`, en `pt`, en `em` ou toutes les autres unités de longueur acceptées par TeX.

Tous les paramètres que nous allons voir ici sont des paramètres de TeX.

4.1. Gestion des mathématiques hors texte

Commençons par voir comment modifier l'espace entre les formules mathématiques «hors texte» et le texte qui suit.

4.1.1. Espace verticale avant le mode mathématique

Il est en effet possible de changer l'espacement vertical qui précède des mathématiques hors texte. Pour cela, il nous faut changer la longueur `\abovedisplayskip`. Par exemple…

```
1  Voici un petit calcul écrit pour observer l'espace avant la
       formule :
2  \[
3     3x = 32.
4  \]
5  Voici un petit calcul écrit pour observer l'espace avant la
       formule :
6  \[
7     \abovedisplayskip = 1cm
8     3x = 32.
9  \]
10 Voici un petit calcul écrit pour observer l'espace avant la
       formule :
11 \[
12    3x = 32.
13 \]
```

Les espaces verticaux avant l'entrée en mode «hors texte» ont été changés. Notons que le changement de valeur n'est pas permanent. En effet, l'espace avant la troisième formule n'est pas de `1cm`, mais est le même que celui avant la première formule. Cela est dû au fait que, quand on change la valeur dans `\[ \]`, elle n'est changée que pour cette formule. Pour changer la valeur par défaut, il nous faut changer la valeur à l'extérieur du mode mathématique. De cette manière, la valeur par défaut sera changée. Voici un code commenté pour mieux comprendre cela.

```
\abovedisplayskip = 1cm    % On la fixe à 1cm.
\[
                           % Ici, elle vaut 1cm.
\]
\[
   \abovedisplayskip = 2cm % Ici elle vaut 2cm.
\]
\[
                           % Ici, elle vaut 1cm.
\]
\abovedisplayskip = 1.5cm  % On la fixe à 1.5cm.
```

4.1.1.1. Le cas où la ligne précédente est trop courte

Maintenant, observons le résultat produit par ce code.

```
Voici un petit calcul écrit pour observer l'espace avant la
    formule :
\[
   \abovedisplayskip = 1cm
   3x = 32.
\]
Encore :
\[
   \abovedisplayskip = 1cm
   3x = 32.
\]
```

Nous remarquons que la commande n'est pas prise en compte pour la seconde formule. Cela est dû au fait que la ligne précédant cette formule est très courte (ici un seul mot). Dans ce cas, la longueur de l'espace inséré n'est pas `\abovedisplayskip`, mais `\abovedisplayshortskip`. C'est donc cette valeur qu'il faudra modifier.

```
Encore :
\[
   \abovedisplayshortskip = 1cm
```

```
4     3x = 32.
5 \]
```

Nous obtenons bien le résultat voulu.

4.1.2. Espace verticale après le mode mathématique

Nous disposons également de paramètres pour l'espace suivant la sortie du mode mathématique. Ainsi, le code suivant modifie cet espace en changeant la longueur `\belowdisplayskip`.

```
1 Voici un petit calcul écrit pour observer l'espace après la
      formule :
2 \[
3     1 + 1 = 2.
4 \]
5 \belowdisplayskip = 0.5cm
6 Voici un petit calcul écrit pour observer l'espace après la
      formule :
7 \[
8     1 + 1 = 2.
9 \]
```

4.1.2.1. Le cas où la ligne précédente est trop courte

De la même manière que nous avons changé la valeur de `\abovedisplayshortskip`, il nous faudra changer la valeur de `\belowdisplayshortskip` pour gérer l'espacement suivant la sortie du mode mathématique si la ligne qui **précède** le mode mathématique est trop courte. On écrira donc ceci.

```
1 Encore :
2 \[
3     \abovedisplayshortskip = 1cm
4     \belowdisplayshortskip = 0.5cm
5     3x = 32.
6 \]
7 Et on recommence à écrire.
```

i Tout ce que nous avons dit à propos de `abovedisplayskip` est valable pour `abovedisplayshortskip`, `\belowdisplayskip` et `\belowdisplayshortskip`.

4.1.3. Le paramètre `\predisplaysize`

Nous avons dit que la longueur des espaces est `\abovedisplayshortskip` et `\belowdisplayshortskip` dans le cas où la ligne précédant la formule est trop courte.

?

Mais à quoi correspond ce «trop courte»? À partir de quelle longueur peut-on dire que cette ligne est trop courte?

En fait, le comportement par défaut est le suivant: si la ligne précédant la formule et la formule se recouvrent horizontalement (c'est-à-dire si la formule mathématique débute à gauche de la fin de la ligne la précédant), les espaces verticaux avant et après la formule seront respectivement de longueur `\abovedisplayskip` et `belowdisplayskip`, sinon, ils seront de longueur `\abovedisplayshortskip` et `\belowdisplayshortskip`.

Pour savoir s'il y a recouvrement horizontal, TeX calcule la largeur de la ligne précédant un affichage mathématique, et donne à `\predisplaysize` cette valeur. Il compare ensuite `\predisplaysize` à la largeur de la formule mathématique pour savoir s'il y a recouvrement. Ainsi, en modifiant la valeur de `\predispalysize`, nous pouvons modifier la condition nécessaire à l'utilisation de `\abovedisplayshortskip` à la place de `\abovedisplayskip`. Nous pouvons, par exemple, faire en sorte que l'espace soit de longueur `\abovedisplayshortskip` si la formule mathématique et la ligne la précédant sont distantes horizontalement d'au moins 1cm, avec ce code.

```
1 Voici une formule :
2 \[
3    \advance \predisplaysize by 1cm
4    2x = 3
5 \]
```

Ici, nous utilisons la primitive `\advance`, qui permet d'augmenter (ou de diminuer) la valeur d'une variable numérique. Nous aurions aussi pu écrire `\advance \predisplaysize 1cm`. En augmentant la valeur de `\predisplaysize` de `1cm`, la comparaison de la taille de la ligne précédente et de celle de la formule ne conduira à un recouvrement que si l'espace horizontal entre les deux est supérieur à `1cm`.

!

La valeur de `\predisplaysize` est calculée à chaque fois que l'on rentre en mode «hors texte». Changer cette valeur en dehors de ce mode n'a donc aucun effet.

4.1.4. Indentation en mode mathématique

Nous avons également la possibilité de modifier l'espace par laquelle TeX indente l'affichage des mathématiques grâce au paramètre `\displayindent`. Sa valeur initiale est de 0. En la changeant, on modifie donc le placement horizontal. Par exemple, si on lui donne la valeur de

`1cm`, les formules ne seront plus affichées exactement au centre, mais au centre décalé d'un centimètre vers la droite. On peut bien sûr lui donner une valeur négative. Observons par exemple l'affichage obtenu avec ce code.

```
\[
   3x = 32.
\]
\[
   \displayindent = 0.5cm
   3x = 32.
\]
```

Changer la valeur de `\displayindent` en dehors du mode mathématique n'a aucun effet. Si on veut que cette valeur soit changée pour toutes les formules mathématiques, on ne peut donc pas la changer en dehors du mode mathématique au début du document. Pour faire cela, nous pouvons utiliser la commande `\everydisplay`. Elle nous permettra de spécifier une valeur de `\displayindent` pour toutes les formules mathématiques.

4.2. Les pénalités

Il arrive parfois qu'un passage à la ligne se fasse alors qu'on est dans le mode «en ligne». Il faut alors se demander quand se fait la coupure. Ce ne serait pas très apprécié, par exemple, que la coupure se fasse au milieu du nombre `123` avec `12` sur la première ligne et `3` rejeté sur la ligne suivante. Par défaut, Tex ne s'autorise pas à couper les nombres, et préfère dans ce cas rejeter le nombre en entier à la ligne suivante. Il nous suffit d'essayer un code où un nombre (`1000000000299798453` par exemple) se retrouve à la fin d'une ligne.

La gestion des coupures est faite grâce aux pénalités. Une pénalité est une valeur qui indique à TeX si un potentiel point de coupure est bon ou mauvais. Une pénalité positive indique que le point de coupure est mauvais et une pénalité négative indique au contraire qu'il s'agit d'un bon point de coupure. TeX essaie donc de couper les lignes aux endroits où les pénalités sont faibles. On en déduit alors que la pénalité entre les différents chiffres d'un nombre est grande.

Tout ceci nous mène au fait que nous avons la possibilité d'indiquer une pénalité dans notre code source. Ceci se fait grâce à la commande `\penalty` qui prend en paramètre la valeur de pénalité que l'on veut insérer.

Une pénalité supérieure à `10000` empêche une coupure et une pénalité inférieure à `-10000` force une coupure.

On peut alors, grâce à cela, forcer une coupure dans l'écriture du nombre avec `$1000000\penalty-10000 000299798453$`.

Les pénalités après un chiffre ou après `=` sont différentes, et nous avons la possibilité de changer ces différentes pénalités.

Nous allons parler des opérateurs binaires et des opérateurs de relations. Il s'agit simplement de catégories de symboles mathématiques. `+`, par exemple, est un opérateur binaire, alors que `=` est un opérateur de relation. La liste de ces symboles est disponible sur plusieurs sites.

4.2.1. Pénalité après un opérateur binaire

Pour changer la pénalité après un opérateur binaire, il nous faudra changer la valeur du paramètre `\binoppenalty`. Plain TeX fixe sa valeur par défaut à 700. Ainsi, en mode «en ligne», `2 + 2` ne sera probablement pas coupé au niveau de `+`, mais plutôt écrit à la ligne suivante. On peut modifier ce comportement en modifiant la valeur de `binoppenalty`.

Si nous changeons cette valeur à l'intérieur de `$$`, elle ne sera changée que pour cette formule, alors que si on la change à l'extérieur, cette modification sera effective tant qu'on ne modifiera pas cette valeur à nouveau. Par exemple…

```
\binoppenalty = -10000 % Sa valeur est de -10000.
Voici un texte très long avec plusieurs formules mathématiques.
On a que $2 + 2 + 2 + 2 = 2 \times 4 = 4 + 4 = 8$.

De plus, on peut aussi écrire que $\binoppenalty = 1000 2 + 2 + 2 +
    2 + 2 + 2 + 2 + 2 = 2 \times 8
= 8 + 8 = 16$.          % Sa valeur est de 1000 pour cette formule.

                        % Sa valeur est toujours de -10000.
```

4.2.2. Pénalité après une relation

De la même manière que pour les opérateurs binaires, nous pouvons changer la pénalité après les relations en changeant la valeur du paramètre `relpenalty`, sa valeur par défaut étant fixée à 500 par Plain TeX. On peut alors faire en sorte de toujours interdire les coupures après `=` avec `\relpenalty = 10000`. Par exemple…

```
\binoppenalty = -10000
Voici un texte très long avec plusieurs formules mathématiques.
On a que $2 - 2 = 1 - 1 = 0 = 0 + 0 = 0 \times 0 = 0 \times 2 = 0$.

De plus, on peut aussi écrire que $\relpenalty = 1000 0 = 0 = 0 = 0
    = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0$.
```

Notons que si nous voulons changer la pénalité après un symbole en particulier et pas pour les autres symboles de la même famille (relation ou opérateur binaire), il ne nous faut pas changer la valeur de pénalité après ce type de symboles, mais insérer une pénalité avec `\penalty`.

4.2.3. Pénalité pour une coupure de page avant l'entrée en mode « hors texte »

Si une formule mathématique peut être placée en fin de page, elle sera placée en fin de page, alors que l'on aurait peut-être préféré la voir repoussée au début de la page suivante . Cela est dû à la valeur par défaut de la pénalité `\predisplaypenalty`, qui est fixée à 10000 par Plain Tex (rappelons-nous qu'une pénalité de 10000 équivaut à une pénalité infinie).

La pénalité `\predisplaypenalty` correspond à la pénalité pour une coupure de page juste avant un affichage mathématique. Le fait qu'elle soit mise à 10000 signifie que Tex empêche toute coupure. Pour changer ce comportement, il nous suffit de changer la valeur de `\predisplaypenalty`.

Nous pouvons la changer à l'intérieur du mode mathématique (elle ne sera changée que pour la formule correspondante), ou à l'extérieur, auquel cas elle sera changée jusqu'à un nouveau changement.

4.2.4. Pénalité pour une coupure de page après l'entrée en mode « hors texte »

De même que pour la coupure avant une formule mathématique, il existe une pénalité pour les coupures après les formules mathématiques. Il s'agit de la pénalité `\postdisplaypenalty`. Sa valeur par défaut est fixée à 0 par Plain TeX.

Elle permet à TeX de savoir s'il faut ou non passer à une nouvelle page dans le cas où une formule mathématique est écrite en bas de page, mais qu'il reste un peu de place pour écrire le texte qui suit. Cela signifie que, si on donne à `\postdisplaypenalty` la valeur -10 000, la formule mathématique restera toujours seule en bas de page, alors que si on lui donne, par exemple, la valeur 500, TeX sera moins propice à passer à une nouvelle page, et écrira probablement la ligne qui suit en dessous de la formule mathématique.

Tout comme `\predisplaypenalty`, la valeur de `\postdisplaypenalty` peut être modifiée à l'intérieur du mode mathématique et à l'extérieur du mode mathématique.

C'est la fin de ce tutoriel, mais pas la fin de l'apprentissage. Nous n'avons fait qu'aborder le sujet, et il nous reste encore beaucoup de chemin à parcourir. Mais nous avons déjà fait un gros travail. N'oublions pas, écrire un document, c'est bien, mais pouvoir le relire facilement après, c'est mieux. Soyons simples et efficaces.

Nous n'avons pas beaucoup parlé des fonctionnalités offertes par mathtools, donc n'hésitons pas à aller regarder sa documentation. Il permet notamment de personnaliser l'affichage des *tags*, et offre de nouvelles commandes et environnements. Nous avons parlé des environnements de matrice ou du type `cases`, mais il y a également `multlined`, des commandes pour gérer finement l'affichage des textes en indice d'un symbole, de nouveaux symboles, etc.

J'espère que ce tutoriel vous a plu et vous a été utile.

Je remercie tous les membres qui m'ont fait part de leurs retours sur le tutoriel, en particulier @Holosmos pour ses remarques très constructives. Merci également à @Thunderseb pour ses corrections et son travail de validation. Et un grand merci à @ct, dont les conseils et les corrections ont permis une grande amélioration du tutoriel et à @**Dominus Carnufex** pour ses corrections et ses précieux conseils.