L'extension tdsfrmath*

Le TEXnicien de surface†

8 juillet 2008

Résumé

Cette extension veut fournir des macros à « l'utilisateur final » pour créer des documents mathématiques ayant un aspect français.

Abstract

This package provides a bunch of macros to help the "final user" to produce maths texts with a definite french look. For there is a marked aspect of localisation, I don't provide an english documentation.

Table des matières

*Ce document correspond au fichier tdsfrmath v1, du 2008/07/08. Édition du cinquantenaire.

†le.texnicien.de.surface@wanadoo.fr

1 Introduction

Le but de cette extension est de fournir des macros prètes à l'usage à des professeurs de mathématiques des collèges, lycées — et plus si affinités — qui voudraient bien utiliser LaTeX sans trop mettre le nez dans la programmation ni devoir retenir des choses aussi barbares que `\overrightarrow` pour faire un vecteur `;-)`

De plus elle tente de donner aux mathématiques un aspect vraiment plus français. On aura par exemple « dx » au lieu de « dx » dans les intégrales et les dérivées.

`tdsfrmath.sty` s'appuie lourdement sur `amsmath` et `amssymb` qu'il requiert. On n'aura donc pas besoin de les charger avec un `\usepackage` si l'on utilise `tdsfrmath`.

À l'origine de cette extension, je trouve un vieux fichier `.sty` que je m'étais concocté, par petits bouts, lorsque je sévissais encore dans le secondaire. Ayant appris un peu de LaTeX depuis, j'ai pensé à en améliorer les macros. J'ai aussi consacré quelques heures à la francisation de l'aspect, chose à laquelle je n'avais accordé que peu d'attention jusqu'ici car, je dois l'avouer, je m'étais beaucoup servi de LaTeX pour produire des textes mathématiques en anglais.

En tout cas, pour en rassurer certains qui pourraient considérer qu'ils ne pourraient jamais arriver à un tel niveau (*sic*) d'écriture de macros : ceci est le résultat de nombreuses heures étalées sur plus de 15 ans. Je dois par ailleurs remercier publiquement tous ceux qui sur `fr.comp.text.tex` ont répondu à mes questions, pas toujours très profondes d'ailleurs, et m'ont apporté une aide précieuse jusqu'aujourd'hui même dans l'utilisation de notre outil préféré de création de document.

2 Utilisation

Pour la première fois, plutôt que des options, j'utilise le système de clés et valeurs que permet `xkeyval.sty`. De même, j'utilise `xargs.sty` qui permet la définition de commandes admettant plusieurs arguments par défaut.

Dans le cours du texte une clé est écrite **`clé`** et une valeur *`val`*. Les clés dont les noms comportent des majuscules sont booléennes c.-à-d. que leur valeur est soit *`true`* — vrai — soit *`false`* — faux. Les clés marquées « choix » dans la table 3, page 16, permettent de choisir entre quelques valeurs prédéfinies. D'autres enfin attendent un texte avec plus ou moins de restrictions suivant ce à quoi servira le texte.

J'ai amplement (?) commenté la partie contenant le code, et on s'y reportera pour les détails d'impémentation, mais je commence ici par présenter toutes les options et toutes les macros de cette extension.

2.1 Chargement optionnel

`tdsfrmath.sty` permet de charger du code de manière optionnelle. Ce code est placé dans des fichiers d'extension `.sto`, à savoir : `taupe.sto` contenant des macros destinée plutôt à une utilisation en classe prépa ; `suite.sto` dont les macros ne traitent que des suites.

`taupe`

À chacun de ces fichiers correspond une clé booléenne, de même nom, dont la valeur par défaut est *`false`* ce qui entraine que ces fichiers ne sont pas chargés.

`suite`

Si on veut utiliser les macros définies dans `taupe.sto`, on appelera `tdsfrmath` par : `\usepackage[taupe=true]{tdsfrmath}`

`ArgArcMaj`

Le fichier `taupe.sto` contient des définitions qui dépendent de la clé **`ArgArcMaj`** — **arg** et **arc** avec **maj**uscule — qui vaut *`false`* par défaut ce qui entraine que les noms des fonctions circulaires et hyperboliques réciproques — comme argch x — sont écrites en minuscule. En donnant la valeur *`true`* à la clé **`ArgArcMaj`**, ils prennent une majuscule — on a alors Argch x.

2.2 Réglage de la police calligraphique

`CharPoCal`

Trois clés règlent le choix de la police calligraphique en mode mathématique. Par défaut, **`CharPoCal`** — pour **`Char`**ger une **`Po`**lice **`Cal`**ligraphique — est *`true`* ce qui permet de définir la police calligraphique pour remplacer `\mathcal` qui serait celle que l'on obtiendrait si **`CharPoCal`** avait la valeur *`false`*.

Lorsque **`CharPoCal`** vaut *`true`*, il faut définir les clés **`calpack`** et **`calcomd`**.

`calpack` `mathrsfs`

La clé **`calpack`**, qui contient *`mathrsfs`* par défaut, prend pour valeur le nom, sans l'extension `sty`, d'un module donnant accès à une police calligraphique, p. ex., `mathrsfs` ou `euscal`.

`calcomd` `mathscr`

La clé **`calcomd`**, qui contient *`mathscr`* par défaut, prend pour valeur le nom d'une macro **sans** la barre oblique initiale. C'est la macro permettant de *passer* en police calligraphique. L'extension `mathrsfs.sty` contient bien la macro `\mathscr`.

En résumé, si l'on veut utiliser le `\mathcal` tel que proposé par LaTeX plus `amsmath.sty`, on chargera :
`\usepackage[CharPoCal=false]{tdsfrmath}`
— ce que l'on fera également pour utiliser `fourier.sty` si on veut bénéficier de la redéfinition de `\mathcal` qu'opère cette extension — et si l'on veut utiliser `euscal.sty` et sa commande `\mathcal` — eh oui ! cette extension redéfinit `\mathcal` — il faudra
`\usepackage[calpack=eucal, calcomd=mathcal]{tdsfrmath}`
on remarquera que `CharPoCal=true` n'est pas nécessaire puisque c'est la valeur par défaut.

`caloptn`

Si nécessaire, on peut passer une option à l'extension *passée* à **`calpack`**, en renseignant la clé **`caloptn`** comme dans, p. ex.

```
\usepackage[calpack=euscript,
            caloptn=mathcal,
            calcomd=mathcal]{tdsfrmath}
```

dans lequel on remarquera qu'il faut bien donner une valeur à **`calcomd`** comme je l'ai déjà écrit ci-dessus.

`\manus`

Dans tous les cas, on accède à la police calligraphique avec la macro `\manus`, à un seul argument obligatoire, qui est définie de telle sorte que l'on puisse saisir « `et dans \manus{c} on trouve` » pour obtenir « et dans $\mathscr{C}$ on trouve ».

2.3 Réglage de la police « gras de tableau »

Par défaut, le « gras de tableau » (*blackboard bold*) est celui de LaTeX plus `amsmath.sty` c.-à-d. $\mathbb{G}$. Il en existe bien d'autres versions que l'on trouvera dans

le fameux `symbols-a4.pdf` disponible généralement dans votre distribution, et donc sur votre disque, dans `texmf-doc/doc/english/comprehensive/`.

Pour permettre de redéfinir la police du gras de tableau, je fournis un mécanisme similaire à celui qui précède. On utilisera alors la macro `\grastab` pour obtenir le « gras de tableau » choisi.

CharPoGdT La clé booléenne **CharPoGdT** — pour **Char**ger une **Po**lice **G**ras **d**e **T**ableau — vaut *false* par défaut.

gdtpack En fixant **CharPoGdT** à *true*, on peut définir la clé **gdtpack** en lui donnant le nom de l'extension qui fournira le gras désiré, on peut éventuellement lui passer

gdtoptn une option avec **gdtoptn** et, toujours éventuellement, fixer **gdtcomd** avec le nom de la macro désirée — toujours sans barre oblique inverse — seulement, cette fois,

gdtcomd du fait du choix de la valeur par défaut de **gdtcomd**, on n'aura pas besoin de fixer la valeur de **gdtcomd** si la macro est encore `\mathbb`.

Voici ce que l'on écrira pour obtenir le gras de tableau tel que fournit par `dsfont.sty` avec l'option `sans` et la commande `\mathds` — tous les goûts sont dans la nature —

```
\usepackage[gdtpack=dsfont, gdtoptn=sans,
            gdtcomd=mathds]{tdsfrmath}
```

et `\grastab` donnera ce que l'on voit dans `symbols-a4.pdf`.

`\grastab` La macro `\grastab` prend également un seul argument obligatoire. Elle ne passe pas son argument en majuscule car certaines extensions fournissent aussi des minuscules en gras de tableau. Cependant elle assure le mode mathématique. On peut donc saisir « `et comme \grastab{M} est unifère` » pour obtenir « et comme $\mathbb{M}$ est unifère ».

2.4 Des n-uplets, de leur saisie et de leur présentation

nupletsep La clé **nupletsep**. peut prendre la valeur *virgule*, qui est le réglage par défaut, ou *pointvirgule*. Toute autre valeur provoque un avertissement et on se retrouve avec le réglage par défaut.

virgule Le réglage par défaut, *virgule* compose les n-uplets comme ceci : (a, b). Avec l'autre réglage possible *pointvirgule*, on aurait $(a\,;\,b)$. J'ai longtemps utilisé cette dernière[1] pour écrire des textes à destination des élèves du secondaire car on est souvent ammené à utiliser des nombres décimaux et, dans ce cas, le mélange de virgule ne m'a jamais paru très heureux.

`\TdSMnuplet` La macro `\TdSMnuplet` prend un argument obligatoire qui est une liste dont les éléments sont séparés par des espaces. Avec « `\TdSMnuplet{a b c d}` » on obtient « a, b, c, d ». C'est une macro auxiliaire aussi lui ai-je donné un nom qui commence par `TdSM` mais elle peut reservir, directement ou dans la définition d'une commande dont je n'ai pas vu l'utilité, aussi je n'ai pas mis de `@` dans son nom.

`\nuplet` La macro *ordinaire* est `\nuplet`.Avec elle on obtient la présentation *classique* des n-uplets : `\nuplet{a b 3 c 8}` compose $(a, b, 3, c, 8)$.

La définition de `\TdSMnuplet` permet de coder `\nuplet{␣a␣b}` pour (a, b).

Bien entendu, comme d'habitude, on ne peut avoir le beurre et l'argent d'icelui. On peut coder `\nuplet{␣a\times␣b+c␣a\cap␣b}` pour obtenir $(a \times b + c,\, a \cap b)$

1. Pour tout dire, à l'époque, mon fichier de macros ne ressemblait à celui-ci que de très loin mais on apprend avec l'age — au moins pendant un moment.

mais avec `\nuplet{␣a␣+␣b␣+␣c␣}` on aura $(a, +, b, +, c)$. Vous êtes prévenus ! `;-)`

En fait, la présentation obtenue avec `\nuplet` repose sur `\EncloreExtensible`

`\EncloreExtensible` dont la syntaxe est :

`\EncloreExtensible[`⟨*md*⟩`]{`⟨*délim-gauche*⟩`}{`⟨*délim-droite*⟩`}{`⟨*texte*⟩`}`

⟨*md*⟩ vaut `1` par défaut, s'il vaut `0` les mathématiques sont composées en mode hors-texte — `\displaystyle`. Je sais bien que c'est une mauvaise pratique, que ça bousille l'interligne, que ça fiche en l'air l'uniformité du gris typo, &c. MAIS, hélas, parfois, c'est bien utile. Alors je le permets mais avec 0 qui rappelle ce qu'il faut penser d'une telle pratique `;-)`.

⟨*délim-gauche*⟩ est quelque chose qui peut être précédé de `\left` comme `(` ou `\Vert`, ⟨*délim-droite*⟩ est quelque chose qui peut être précédé de `\right` comme `)` ou `\Vert`, Si on ne veut rien de visible à gauche ou à droite, il faut que le 1er ou le 2e argument obligatoire soit un point. ⟨*texte*⟩ est ce qui sera placé entre les délimiteurs, en mode mathématique.

La macro `\EncloreExtensible` nous place en mode mathématique.

En voici un exemple un rien bête : `\EncloreExtensible{(}{\rangle}{x^{2}}` produit $(x^2\rangle$.

Le comportement des délimiteurs varient suivant qu'on est — de manière forcée par l'argument optionnel ou de manière *naturelle* parce que l'on est dans une formule composée hors-texte — en mode mathématique hors-texte ou en mode mathématique en ligne. Dans le 1er cas, les délimiteurs sont extensibles, dans le second ils ne le sont pas. On verra plus bas, page 9, le rendu des parenthèses dans la macro `\repere`.

Cette macro `\EncloreExtensible` me sert à en définir plusieurs autres que voici. Toutes ont la même syntaxe :

`\Macro[`⟨*md*⟩`]{`⟨*texte*⟩`}`

`\parent` où ⟨*md*⟩ et ⟨*texte*⟩ ont le même sens que ci-dessus. Ce sont `\parent` pour obtenir

`\accol` des parenthèses, `\accol` pour des accolades et `\crochet` pour des ... oui, des crochets ! Voici, p. ex., `\parent{a}` qui produit (a) ; `\accol{\vide}` qui produit

`\crochet` $\{\varnothing\}$; `\crochet{8\cdot9}` qui produit $[8 \cdot 9]$.

Dans le même genre, on a `\varabs` pour obtenir la valeur absolue comme ici :

`\varabs` $|-12|$ codé `\varabs{-12}`.

`\norme` Dans la même veine, toujours, `\norme` pour écrire la norme comme suit : $\|\overrightarrow{\imath}\|$ codé `\norme{\vecti}`.

Revenons aux n-uplets. Les macros qui les produisent acceptent, elles aussi, toutes un argument optionnel qui force le mode hors-texte quand il vaut `0`.

`\nuplet` On obtient, comme déjà vu ci-dessus, (c, d) avec `\nuplet{c␣d}` — j'insiste

`\anuplet` sur l'espace, non ? — et avec `\anuplet{c␣d}` on a $\{c, d\}$. Cette dernière doit son nom à ce qu'elle utilise des **a**ccolades.

Toutes les deux, comme je l'ai déjà signalé, peuvent traiter un nombre quelconque d'arguments séparés par des espaces comme, p. ex., $\{a, b, c, d, e, f\}$ obtenu avec `\anuplet{a b c d e f}`. Il faut toutefois remarquer que si l'on veut utiliser un macro à l'intérieur, p. ex. `\alpha`, il faudra la faire suivre ou l'entourer d'une paire d'accolades pour préserver l'espace, sinon c'est l'erreur assurée et TEX proférera une de ces habituelles remarques absconces `;-)`

On codera donc `\nuplet{a \alpha{} \beta}` pour obtenir (a, α, β). Mais, coquèterie d'auteur, je me suis arrangé pour que l'on puisse coder directement

`\nuplet{a §a §b}` pour avoir (a, α, β) lorsque l'extension `paresse.sty`, de votre serviteur, est chargée.

Avec `\nuplet{\frac{1}{2} \frac{3}{4}}` on produit $(\frac{1}{2}, \frac{3}{4})$.

Je ne suis pas allé plus loin car je pense que je couvre largement les besoins du secondaire avec tout ça. Qui voudrait obtenir une macro du même genre, pourra toujours la définir à l'aide de `\EncloreExtensible` et `\TdMnuplet` qui font le travail principal.

Cependant, je fournis la macro `\rnuplet`, prévue pour être utilisée dans le cas de l'écriture d'une fonction, p. ex. En effet, elle précède la composition du n-uplet d'une espace négative ce qui a pour effet de rapprocher la première parenthèse de ce qui précède. Comparer $f(x, y)$, obtenu avec `\(f\rnuplet{x y}\)`, à $f(x, y)$, `\(f\nuplet{x y}\)`, et à $f\ (x, y)$, `\(f\rnuplet{x y}[5]\)`.

`\rnuplet` Cette macro a pour syntaxe
`\rnuplet[`⟨*md*⟩`]{`⟨*texte*⟩`}[`⟨*ecart*⟩`]`
le seul argument nouveau est ⟨*ecart*⟩ qui règle l'espacement entre ce qui précède la macro et la parenthèse. Par défaut cet argument vaut `\TdSMReculParenthese` dont la valeur est -2, ⟨*ecart*⟩ doit être un nombre.

Le `r` est là pour faire penser (?) à *recul*.

On pourrait donc écrire mais, bien sûr, on **ne le fera pas**, `\(f\rnuplet[0]{\frac{1}{2} 3}[10]\)` pour obtenir l'horreur : $f\ \left(\frac{1}{2}, 3\right)$.

`\TdSMReculParenthese` C'est la macro qui fixe, de manière générale, l'espace entre le texte qui précède et la parenthèse — ou délimiteur équivalent — ouvrante. On peut la redéfinir avec `\renewcommand`.

Remarque : Elle n'est pas *secrète* donc son nom ne comporte pas de `@` mais on n'est pas sensé l'utiliser toutes les trois secondes d'où les capitales. C'est la convention générale [2] de nommage des macros.

2.5 De la définition des ensembles

`SepDefEnsExt` `true` Je fournis la macro `\ensemble`, cf. page 12, qui permet d'écrire, p. ex., « $\{x \in \mathbb{R} \,/\, x^2 \geqslant 2\}$ » avec `\ensemble{x\in\R}{x^{2}\pgq 2}`. Le rendu en est controllé par la clé **`SepDefEnsExt`** — **sép**arateur de la **déf**inition d'un **ens**emble **ext**ensible — qui vaut ***`true`*** par défaut. Par ailleurs, la macro `\TdSMsepdefens` `\TdSMsepdefens` contient le séparateur et peut-être redéfinie à l'aide d'un `\renewcommand`.

Si, comme c'est le cas par défaut, la clé **`SepDefEnsExt`** vaut ***`true`***, la définition de `\TdSMsepdefens` doit être *quelque chose* supportant l'action de `\middle` — qui est à un délimiteur central ce que `\left` et `\right` sont à ceux de gauche et droite — comme, p. ex., `\vert`. Ce qui fait que si l'on veut un séparateur qui ne supporte pas cela, comme `:`, les deux-points, il faut passer la valeur ***`false`*** à la clé **`SepDefEnsExt`**.

2.6 Des noms des ensembles classiques

Il s'agit ici des macros qui permettent d'obtenir $\mathbb{R}$ et $\mathbb{Q}^*$ ou encore $\mathbb{C}_3[X]$.

`\TdSM@Decoration` Cette macro *secrète* place les étoiles et signe plus ou moins, ce que j'appelle

2. Il faut prendre ces conventions pour ce qu'elles sont et on n'aurait pas trop de peine à trouver des exceptions à cette *règle*, exceptions qui ne survivent que par la force de l'habitude.

ici la décoration du nom de l'ensemble. Par défaut on a $\mathbb{R}_+^*$ mais cette disposition

ensdeco est controlée par la clé **ensdeco** qui peut prendre les valeurs ***ehsb***, ***ehsh***, ***sheh***,

ebsh ***ebsb***, ***sbeb*** et ***ebsh***.

Par défaut, on a `[ensdeco=ebsh]`. Toute autre valeur provoque un avertissement et on se retrouve avec le réglage par défaut.

La valeur par défaut ***ehsb*** place l'**é**toile en **h**aut et le **s**igne en **b**as. On pourra retenir que, quand l'ordre importe peu, on commence par l'étoile d'où ***ehsb*** et ***ebsh*** et que, sinon, l'ordre d'apparition de ***e*** et ***s*** règle la place de l'étoile $*$ et du signe.

Grace au mécanisme de `\define@choice*+`, on pourra passer les valeurs en capitales. Donc `[ensdeco=EHSB]` est une écriture valide.

`\TdSM@PlaceSigne` Cette macro tout aussi *secrète* place le signe plus ou moins quand il est seul.

placesigne Par défaut on a $\mathbb{R}^+$ mais cette disposition est controlée par la clé **placesigne**

haut qui peut prendre les valeurs ***haut*** et ***bas***.

Par défaut, on a `[placesigne=haut]`. Toute autre valeur provoque un avertissement et on se retrouve avec le réglage par défaut.

`\EnsembleDeNombre` Cette macro fait le gros boulot de composition. Elle prend 4 arguments obligatoires : le 1er donne la lettre majuscule symbolisant l'ensemble comme « R »

`\C` pour $\mathbb{R}$; le comportement de la macro varie suivant que le 2e est égal à 1, est un entier supérieur à 1, un entier strictement négatif ou l'une de ces sept chaines de

`\N` caractères `*`, `+`, `-`, `*+`, `+*`, `-*` et `-*` — c'est ce qui permet d'obtenir plus tard,

`\Q` p. ex., $\mathbb{Q}_+^*$ avec `\Q[+*]` — ; le 3e argument est utilisé pour dénoter les ensembles de polynômes comme $\mathbb{C}_3[X]$ et dans ce cas le 2e doit être un nombre négatif ; enfin

`\R` le 4e doit être un entier qui donne le nombre de `mu` — unité de longueur spécifique

`\Z` au mode mathématique — qui séparent la majuscule du crochet ouvrant.

Je rappelle au passage que `mu` — pour *maths unit* — est une unité de longueur définie uniquement en mode mathématique. Elle vaut 1/18 d'un `em` qui est la largeur d'un M dans la fonte courante.

La macro opère un certain controle car, en dehors de `*`, `+`, `-`, `*+`, `+*`, `-*` et `-*`, le 2e argument doit être un entier relatif. Attention, on peut saisir `\R[--4]` mais ça donne $\mathbb{R}^{--4}$!

Comme il serait fastidieux d'avoir à taper `\EnsembleDeNombre{N}{1}{}{}` pour obtenir simplement $\mathbb{N}$, je fournis maintenant des commandes courtes auxquelles j'ai déjà fait allusion ci-dessus. Ce sont `\N` pour $\mathbb{N}$, `\Z` pour $\mathbb{Z}$, `\Q` pour $\mathbb{Q}$, `\R` pour $\mathbb{R}$, `\C` pour $\mathbb{C}$ et, enfin, si on a passé la valeur ***true*** à la clé **taupe**, `\K` pour $\mathbb{K}$.

Je ne fournis pas `\D` pour les décimaux car, d'une part, je doute finalement de l'utilité de cet ensemble et, d'autre part, je réserve cette macro pour plus tard.

J'utilise ici, avec beaucoup de satisfactions, l'extension `xargs.sty` afin que ces macros prennent deux arguments optionnels qui fourniront, dans l'ordre, les 2e et 3e arguments de `\EnsembleDeNombre`. Par défaut, le 1er argument vaut `1` et le 2e `X`.

Voici toutes les façons d'utiliser `\R`, p. ex., et ce qu'elles produisent :

– `\R[*]` donne $\mathbb{R}^*$;

– `\R[+]` donne $\mathbb{R}^+$;

– `\R[-]` donne $\mathbb{R}^-$;

– `\R[+*]` ou `\R[*+]` donne $\mathbb{R}_+^*$;
– `\R[-*]` ou `\R[*-]` donne $\mathbb{R}_-^*$;
– `\R[5]` donne $\mathbb{R}^5$;
– `\R[-6]` donne $\mathbb{R}_6[X]$;
– `\R[-6][Y]` donne $\mathbb{R}_6[Y]$.

On notera que l'on ne peut pas donner le 2e argument optionnel sans donner d'abord le premier.

Cependant, pour des raisons que l'on peut voir page 14 à propos de `\suite*`, je fournis quelques macros supplémentaires qui, du coup, peuvent abréger la saisie.

`\R*` `\N`, `\Z`, `\Q`, `\R` et `\C` ont une forme étoilée qui fait la même chose que la macro avec une `*` pour 1er argument. On aura donc, p. ex., $\mathbb{N}^*$ avec `\N*` comme avec `\N[*]`. Il en est de même avec `\K` si `taupe.sto` est chargé.

`\R+` `\R-` `\Q` et `\R` ont une forme *plussée* et une forme *moinsée* qui font, respectivement, la même chose que la macro avec `+` et `-` pour 1er argument optionnel. On aura donc, p. ex., $\mathbb{Q}^+$ avec `\Q+` comme avec `\Q[+]` et $\mathbb{R}^-$ avec `\R-` comme avec `\R[-]`.

`\R>` `\R<` `\R` bénéficie de deux autres raccourcis, à savoir `\R>` qui produit $\mathbb{R}_+^*$ c.-à-d. la même chose que `\R[+*]` et `\R<` qui produit $\mathbb{R}_-^*$ c.-à-d. comme `\R[-*]`. Si l'on veut définir d'autres raccourcis du même genre on pourra regarder le code page **??**.

Toutefois, si pour une raison quelconque, on voulait « $\mathbb{Q}+\mathbb{Q}$ » on devra coder `\(\Q{} + \Q\)`. Il arrive que certaines épines aient des roses...

2.7 Des vecteurs, des bases et des repères

`\definirvecteur` Cette macro permet, comme son nom l'indique presque, de définir des macros qui produisent des vecteurs. Sa syntaxe est :
`\definirvecteur[`⟨*bb*⟩`]{`⟨*a*⟩`}{`⟨*n*⟩`}{`⟨*m*⟩`}`

Avec `\definirvecteur{`⟨*a*⟩`}{`⟨*n*⟩`}{`⟨*m*⟩`}` on obtient une macro qui s'appelle `\vecta` et qui produit, en se plaçant dans le mode mathématique, la lettre `a` surmontée de la flèche que donne `\overrightarrow` avec un décalage de `n` mus devant et `m` mus derrière le texte.

L'argument optionnel ⟨*bb*⟩ permet d'obtenir le nom `\vectbb` ce qui est indispensable quand le 1er argument obligatoire est lui-même une macro comme p. ex. `\imath`.

Cette macro fait appel à `\TdSM@fairevecteur` décrite page **??**.

`\redefinirvecteur` Elle teste l'existence d'une macro du même nom et produit une erreur s'il en existe déjà une. Si l'on veut redéfinir une commande comme `\vecti`, on utilisera `\redefinirvecteur` qui a la même syntaxe que sa grande sœur et qui, elle, produira une erreur si on tente de redéfinir un vecteur qui ne l'est pas encore.

`\vecti` `\vectj` `\vectk` `\vectu` `\vectv` Grâce à `\definirvecteur`, je définis quelques vecteurs courants et utiles : `\vecti` $\overrightarrow{i}$; `\vectj` $\overrightarrow{j}$; `\vectk` $\overrightarrow{k}$; `\vectu` $\overrightarrow{u}$ et enfin `\vectv` $\overrightarrow{v}$.

On pourra comparer la composition que permet d'obtenir `\TdSM@fairevecteur`, à l'aide des 2e et 3e arguments qui définissent un nombre de `mus`, avec ce que donne une composition directe comme ici : $\overrightarrow{i}$ obtenu avec `\vecti` et $\overrightarrow{i}$ produit par `\(\overrightarrow\imath\)`.

`\vecteur` Cette macro peut être suivie par une étoile. Elle prend un argument optionnel, valant `1` par défaut, qui détermine l'espace placé devant le texte sous la flèche.

Elle prend un argument obligatoire qui donne le ⟨*texte*⟩ qui sera placé sous la flèche du vecteur. Avec la version sans étoile, le texte est composé *normalement* en mode mathématique comme dans $\overrightarrow{AB}$ produit par `\vecteur{AB}`. Avec la version étoilée le texte est en caractères romains, ou, plus exactement, est composé dans la police en vigueur pour l'argument de `\text` de l'extension `amstext.sty`, chargée ici par l'intermédiaire de `amsmath.sty`. On a donc $\overrightarrow{\text{CD}}$ avec `\vecteur*{CD}`.

Enfin, le troisième argument, optionnel, règle l'espace supplémentaire, toujours en nombre de `mus`, qui suit le texte. Comparez $\overrightarrow{AB}$ produit par `\vecteur{AB}` avec $\overrightarrow{\vphantom{AB}AB}$ produit par `\vecteur[10]{AB}`, $\overrightarrow{AB\vphantom{AB}}$ produit par `\vecteur{AB}[20]` et $\overrightarrow{\vphantom{AB}AB\vphantom{AB}}$ produit par `\vecteur[10]{AB}[20]`.

`\V` Ce n'est qu'un raccourci de `\vecteur`. Il a donc la même syntaxe :
`\V*[`⟨*espace-avant*⟩`]{`⟨*texte*⟩`}[`⟨*espace-après*⟩`]`
où ⟨*espace-avant*⟩ et ⟨*espace-après*⟩ sont des nombres.

`\base` La macro `\base` admet un seul argument, optionnel, qui ne doit prendre que les valeurs `1`, `2` — valeur par défaut — ou `3`. On obtient alors $(\overrightarrow{\imath})$ avec `\base[1]`, $(\overrightarrow{\imath}, \overrightarrow{\jmath})$ avec `\base` ou `\base[2]`, $(\overrightarrow{\imath}, \overrightarrow{\jmath}, \overrightarrow{k})$ avec `\base[3]`.

`\repere` La macro `\repere` fournit un repère à la française. Elle est construite sur `\base` et son 1er argument optionnel a le même rôle que celui de `\base`. Le 2e argument optionnel de `\repere` définit le centre du repère, c'est `O` par défaut.

On a donc $(O, \overrightarrow{\imath}, \overrightarrow{\jmath})$ ou $(O, \overrightarrow{\imath}, \overrightarrow{\jmath}, \overrightarrow{k})$ ou $(O', \overrightarrow{\imath}, \overrightarrow{\jmath})$ avec `\repere` ou `\repere[3]` ou `\repere[2][O']`. On a même $(O, \overrightarrow{\imath})$ avec `\repere[1]`.

Je rappelle qu'il faut le 1er argument optionnel si l'on veut préciser le 2e.

Voyons maintenant le rendu des repères dans une formule hors-texte :

$$(O, \overrightarrow{\imath}) \quad (O, \overrightarrow{\imath}, \overrightarrow{\jmath}) \quad \left(O, \overrightarrow{\imath}, \overrightarrow{\jmath}, \overrightarrow{k}\right)$$

Viennent maintenant des macros qui servent essentiellement d'abréviations.

`\rog` `\ron` `\rond` Tout d'abord ce qu'il faut pour écrire « repère orthogonal $(O, \overrightarrow{\imath}, \overrightarrow{\jmath})$ » avec `\rog` puis « repère orthonormal $(O, \overrightarrow{\imath}, \overrightarrow{\jmath})$ » avec `\ron` et enfin « repère orthonormal direct $(O, \overrightarrow{\imath}, \overrightarrow{\jmath})$ » avec `\rond`.

Ces trois commandes acceptent les mêmes arguments que `\repere` ce qui fait que l'on peut obtenir « repère orthonormal direct $(A, \overrightarrow{\imath}, \overrightarrow{\jmath}, \overrightarrow{k})$ » avec `\rond[3][A]`. On **ne** doit **pas** les utiliser en mode mathématiques.

`\repcom` `\roncom` `\rondcom` Je fais de même avec les repères pour le plan complexe, où, en général, on utilise $\overrightarrow{u}$ et $\overrightarrow{v}$ pour la base. On a donc « $(O, \overrightarrow{u}, \overrightarrow{v})$ » avec `\repcom`, « repère orthonormal $(O, \overrightarrow{u}, \overrightarrow{v})$ » avec `\roncom` et enfin « repère orthonormal direct $(O, \overrightarrow{u}, \overrightarrow{v})$ » avec `\rondcom`.

Je fournis de quoi écrire les repères à la mode du collège[3] mais je ne traite que le cas d'un repère du plan.

`\Repere` `\Repere*` Cette macro a une forme étoilée. Sans étoile, on obtient « (O, I, J) » et avec l'étoile — c.-à-d. avec `\Repere*` — c'est « (O, I, J) ».

`\Rog` `\Ron` `\Rond` Viennent ensuite des abréviations, construites sur le même modèle que les précédentes : `\Rog` pour « repère orthogonal (O, I, J) », `\Ron` pour « repère orthonormal (O, I, J) » et enfin `\Rond` pour « repère orthonormal direct (O, I, J) ». Elles ont toutes une forme étoilée qui permet d'obtenir les lettres « droites » — avec

3. Enfin, c'est comme cela que j'y pensais du temps où j'enseignais en lycée. Est-ce bien encore le cas ?

les mêmes remarques qu'à propos de `\vecteur*`, cf. page 8. On a donc, p. ex., « repère orthonormal direct (O, I, J) » avec `\Rond*`.

2.8 L'exponentielle

`\E` La macro `\E` permet d'obtenir un « e » droit quelque soit l'environnement : « *Le nombre* e *vaut approximativement* 2,7 » codé `\emph{Le nombre \E vaut approximativement \(\np{2,7}\).}` grace à `\textup` mais il n'est en *romain* que si l'environnement est en romain : « *Le nombre* e *vaut approximativement* 2,7 » où j'ai utilisé `\textsl` pour obtenir des caractères sans empattements.

`\eu` La macro `\eu` prend un argument obligatoire qui sera placé en exposant. On saisit `\eu{2x+3}` pour obtenir e^{2x+3}. Une fois encore, grace à `\ensuremath`, on n'a pas besoin de passer explicitement en mode mathématique.

2.9 Le nombre i

`\I` Je définis `\I` pour qu'elle donne un « i » droit qui est ce que l'on devrait utiliser en français pour noter « la racine carrée de -1 » — pour parler comme les anciens.

On a donc « le nombre i qui vérifie $\mathrm{i}^2 = -1$ » avec le code « `le nombre \I qui vérifie \(\I^{2}=-1\)` ».

Les remarques faites ci-dessus à propos de `\E` s'appliquent également à `\I`.

2.10 Intégrales

$$\int_a^b f(x)\,\mathrm{d}x \qquad \text{plutôt que ça :} \qquad \int_a^b f(x)dx$$

`\FixeReculIntegrande` `\FixeAvanceDx` Ces deux macros prennent un **nombre** pour unique argument obligatoire. Elles permettent de *fixer* le nombre de `mus` dont l'intégrande sera rapproché du signe somme et celui dont l'intégrande et le dx seront séparés.

`\D` Je fournis la macro `\D` avec un `\providecommand` car elle est déjà définie par `kpfonts.sty` de Christophe Caignaert, avec le même effet mais par un autre tour. Cela permet d'utiliser `kpfonts.sty` et `tdsfrmath.sty` sans craindre un conflit de nom.

`\intgen` C'est la macro la plus générale pour écrire une intégrale. Sa syntaxe est :
`\intgen[`⟨*md*⟩`][`⟨*recul*⟩`]{`⟨*inf*⟩`} {`⟨*sup*⟩`}{`⟨*intégrande*⟩`}`
où ⟨*md*⟩ est le mode dans lequel sera composé la formule, par défaut le mode mathématique courant, valeur `1`, avec `0` on est en mode hors-texte — je ne fais pas de rappel sur ce qu'il faut penser de cette manœuvre `;-)`

⟨*recul*⟩ vaut par défaut `6mu` ou la valeur fixée par `\FixeReculIntegrande`, sinon ce doit être un nombre de `mus` — explicitement on écrira `[1][-8mu]`, et je rappelle que le 2e argument optionnel ne peut être donné que si le 1er est donné également. ⟨*inf*⟩ et ⟨*sup*⟩ sont les bornes inférieure et supérieure de l'intégrale, ⟨*intégrande*⟩ est — surprise ! — l'intégrande.

On l'utilise lorsque l'intégrande et le dx sont *mélangés* comme dans

$$\int_2^5 \frac{\mathrm{d}x}{\ln x}$$

codé avec `\[\intgen{2}{5}{\dfrac{\D x}{\ln x}}\]`.

`\integrer` Vient la macro pour le cas où l'intégrande est séparé de dx. Sa syntaxe est : `\integrer[`⟨*md*⟩`][`⟨*recul*⟩`]{`⟨*inf*⟩`} {`⟨*sup*⟩`}{`⟨*integrande*⟩`}{`⟨*var*⟩`} [`⟨*avance*⟩`]`.

On retrouve les arguments de `\intgen` mais on trouve un argument obligatoire supplémentaire ⟨*var*⟩, qui est le symbole de la variable, et un argument optionnel final ⟨*avance*⟩ qui règle la distance entre l'intégrande et le `\D` ; ⟨*avance*⟩ est soumis aux mêmes règles que le ⟨*recul*⟩. Par défaut ⟨*avance*⟩ vaut `4mu`.

On code `\[\integrer{0}{\pi}{\cos 2x}{x}\]` pour avoir

$$\int_0^{\pi} \cos 2x \,\mathrm{d}x$$

`\integrale` La macro suivante est construite sur `\integrer` mais est conçue pour être un raccourci de `\integrer{a}{b}{f(x)}{x}` avec `\integrale{a}{b}{f}{x}`.

À l'exception du 3e argument obligatoire qui est un *symbole* de fonction — comme f, g &c — tous ses arguments sont ceux de `\integrer`.

`\intabfx` Enfin, raccourci du raccourci `\intabfx` remplace `\integrale{a}{b}{f}{x}` et compose $\int_a^b f(x)\,\mathrm{d}x$ dans le cours du texte et

$$\int_a^b f(x)\,\mathrm{d}x$$

en hors-texte.

2.11 Au bonheur du mathématicien, bazar

Je regroupe ici plusieurs macros qui me facilitent la vie dans la saisie des mathématiques. J'y fais une utilisation intense de `\ensuremath` et `\xspace`.

2.11.1 De l'infini

`\plusinf` `\moinsinf` J'ai mis très longtemps à retenir le nom de `\infty`, aussi je me suis fait `\plusinf`, $+\infty$, et `\moinsinf`, $-\infty$. J'espère que leurs seuls noms me dispense d'en dire plus sauf qu'il me faut préciser qu'elles assurent le mode mathématique et s'occupe de l'espace derrière ce qui permet d'écrire `et en \moinsinf on trouve` pour composer « et en $-\infty$ on trouve ».

2.11.2 Des intervalles de $\mathbb{R}$

`\interff` `\interoo` `\interfo` `\interof` On peut écrire les différents intervalles avec les macros `\interff`, `\interoo`, `\interof` et `\interfo`. Leur syntaxe commune est `\int...[`⟨*md*⟩`][`⟨*avant*⟩`]{`⟨*a b*⟩`} [`⟨*après*⟩`]`. On retiendra que `\inter` est mis pour `intervalle` puis que la première lettre donne le *sens* du crochet gauche et la suivante celui du crochet droit avec `f` pour *fermé* et `o` pour *ouvert*.

Le premier argument ⟨*md*⟩ est optionnel est règle le mode mathématique, il vaut `1` par défaut. Le 2e ⟨*avant*⟩, optionnel, vaut `0` par défaut et donne le nombre de `mu`s qui sépare le délimiteur ouvrant du texte. Le 4e et dernier ⟨*après*⟩, qui vaut `0` par défaut, est également optionnel. Il définit, en nombre de `mu`s, la distance qui sépare le texte intérieur du délimiteur fermant.

Le 3e argument ⟨*a b*⟩ est obligatoire, il fournit le texte à placer entre les délimiteurs. L'espace sépare les deux valeurs extrêmes de l'intervalle. On code donc `\interff{12␣37/5}` pour obtenir $[12,\ 37/5]$ mais il faudra coder `\interoo {\moinsinf{}␣\plusinf}` pour avoir $]-\infty,\ +\infty[$.

Le séparateur des valeurs extrêmes de l'intervalle est soumis à la clé **nupletsep**.

2.11.3 La réserve du bazar, miscellanées

`\mdfrac` `\mfrac` Deux macros pour fainéant donc pour moi ;-) : `\mdfrac` et `\mfrac` permettent de saisir les fractions comme si on utilisait `\(\dfrac{...}{...}\)` et `\(\frac{...}{...}\)` respectivement. On pourra donc coder `\mfrac{1}{2}` pour obtenir $\frac{1}{2}$.

`\cnp` Il fut une époque où, en France, on ne notait pas le nombre de combinaisons comme dans le monde anglo-saxon, d'où `\cnp`. La tradition s'est perdue mais la macro est restée pour fournir la notation *nouvelle vague*. Avec `\cnp{n}{p}` on a $\binom{n}{p}$. Là encore il n'est pas nécessaire de passer explicitement en mode mathématique.

`\dans` `\donne` Deux abréviations pour écrire les définitions de fonctions. Je trouve que `\dans` est plus court et plus facile à retenir que `\longrightarrow` et qu'il en est de même de `\donne` vis-à-vis de `\longmapsto`. De fait `\(f\): \(\R \dans \R\); \(x \donne 2x\)` compose : « $f : \mathbb{R} \longrightarrow \mathbb{R}\,;\ x \longmapsto 2x$ »[4].

`\vide` Je préfère ∅ à ∅[5] et par paresse encore, je me suis fait une `\vide` qui permet de saisir `\vide et autre` pour obtenir « ∅ et autre ». Merci `\ensuremath` *et* `\xspace`.

`\ppq` `\pgq` Je veux ceci $0 \leqslant 1$ et $2 \geqslant 1$. Comme `\leqslant` et `\geqslant`, c'est pas beau, je me suis fait `\ppq` — **p**lus **p**etit **q**ue — et `\pgq` — **g**rand.

`\ensemble` La macro `\ensemble` a deux arguments obligatoires, elle sert à écrire la définition d'un ensemble comme « $\{x \in \mathbb{R} \,/\, f(x) \geqslant \frac{1}{2}\}$ » obtenu avec `\ensemble {x\in\R}{f(x)\pgq \frac{1}{2}}` et dont l'aspect est

$$\left\{ x \in \mathbb{R} \,\middle/\, f(x) \geqslant \frac{1}{2} \right\}$$

en hors-texte, du fait de la présence de `\middle`, lorsque la clé booléenne **SepDefEnsExt** vaut ***true*** comme c'est le cas par défaut.

Sa syntaxe complète est :
`\ensemble[`⟨*avant*⟩`]{`⟨*1er texte*⟩`}{`⟨*2e texte*⟩`}[`⟨*après*⟩`]`
où ⟨*avant*⟩ et ⟨*après*⟩ doivent être des nombres. Leur valeur par défaut est `3`. Ces arguments optionnels règlent la distance avant et après le symbole de séparation, en nombre de `mu`s.

2.12 Pour les taupes, taupins et taupines

Les macros de cette section ne sont définies que si l'on a passé la valeur ***true*** à la clé **taupe**. Je ne pense pas que ces macros soient utiles avant le supérieur. Cependant aucun mécanisme n'est prévu pour s'assurer de la destination du document final ;-)

4. Début seconde, peut-être ;-)

5. Celui-là je m'en sers tellement que j'avais oublié son nom. Ce n'est pas `\nothing` mais `\emptyset` pour faciliter le travail de la mémoire.

`\K` La macro `\K` donne $\mathbb{K}$, le corps bien connu, alias de $\mathbb{R}$ ou $\mathbb{C}$.

`\prodscal` Cette macro permet d'écrire le produit scalaire comme on le trouve assez souvent dans les bouquins pour taupins. Elle ne prend qu'un seul argument obligatoire qui est une liste dont les éléments sont séparés par des espaces. Les remarques formulées à propos de `\nuplet` s'appliquent donc ici.

Avec `\prodscal{u v}` on obtient « $\langle u, v\rangle$ » et avec `\prodscal{\vectu{} \vectv}` ou `\prodscal{{\vectu} \vectv}` on obtient « $\langle \vec{u}, \vec{v}\rangle$ ».

Je redéfinis quelques macros classiques pour leur donner un aspect français comme on le trouve encore souvent.

Ce sont les fonctions taupiques usuelles `\sinh`, `\cosh`, `\tanh` auxquelles j'ajoute `\cot` parce que j'ai dû en avoir besoin un jour.

On aura donc, p. ex., « $\operatorname{ch} x$ » en codant `\(\cosh x\)`.

Je crée les macros `\argsh`, `\argch` et `\argth` pour les fonctions hyperboliques réciproques. Par défaut elles ont l'aspect suivant : $\operatorname{argch} x$, $\operatorname{argsh} y$ et $\operatorname{argth} z$.

Si on a passé la valeur *true* à la clé **ArgArcMaj** alors je définis les macros `\argsh`, `\argch` et `\argth` pour qu'elles soient écrites avec une majuscule comme $\operatorname{Argsh} x$. Dans ce cas, je redéfinis également les macros `\arccos`, `\arcsin` et `\arctan` pour qu'elles aient le même aspect.

Table 1 – Macros redéfinies dans `taupe.sto`

`\sinh`	$\operatorname{sh} x$	`\cosh`	$\operatorname{ch} x$
`\tanh`	$\operatorname{th} x$	`\cot`	$\operatorname{cotan} y$

Table 2 – Macros dont l'aspect dépend de la clé **ArgArcMaj** — aspect par défaut

`\arccos`	$\arccos x$	`\arcsin`	$\arcsin x$
`\arctan`	$\arctan x$	`\argsh`	$\operatorname{argsh} x$
`\argch`	$\operatorname{argch} x$	`\argth`	$\operatorname{argth} x$

Pour noter le noyau et l'image avec une majuscule, je fournis `\Ker` — $\operatorname{Ker} f$ avec `\(\Ker f\)` à comparer à $\ker f$ avec `\(\ker f\)` — et `\Img` qui donne $\operatorname{Im} f$ avec `\(\Img f\)` — `\Im` est déjà prise pour noter la partie imaginaire d'un complexe.

`\tendversen` Pour écrire $f(x) \underset{0}{\to} +\infty$, je fournis `\tendversen{`⟨*en*⟩`}` à utiliser **en mode mathématique**. J'ai codé `\(f(x)\tendversen{0} \plusinf\)` l'exemple ci-avant.

`\devlim` Je fournis `\devlim[`⟨*en*⟩`]{`⟨*ordre*⟩`}` à utiliser en mode mathématique pour obtenir $\mathrm{DL}_4(0)$ avec `\devlim{4}` et $\mathrm{DL}_7(1)$ avec `\devlim[1]{7}`. On remarque donc que ⟨*en*⟩ vaut `0` par défaut. N'est-ce pas étrange ?

`\parties` Pour écrire $\mathscr{P}(E)$, je fournis `\parties` utilisable en mode texte. Sa syntaxe est `\parties[`⟨*n*⟩`]{`⟨*ensemble*⟩`}` où *n* est un nombre de `mu`s qui permet de régler la distance entre $\mathscr{P}$ et la parenthèse ouvrante, *n* vaut `-2` par défaut ; *ensemble* est le nom de l'ensemble dont on considère l'ensemble des parties, étonnant, non ?

`\drv` Pour écrire « $\frac{\mathrm{d}f(x)}{\mathrm{d}x}$ », je fournis `\drv{`⟨*fonction*⟩`}{`⟨*var*⟩`}` utilisable en mode texte. J'ai codé `\drv{f(x)}{x}` l'exemple ci-dessus.

\ddrv `\ddrv` est à `\drv`, ce que `\dfrac` est à `\frac` et donc « `et \ddrv{f(x)}{x} vaut` » compose « et $\dfrac{\mathrm{d}f(x)}{\mathrm{d}x}$ vaut », en bousillant l'interligne comme prévu !

\interent Avec `\interent{3 12}` on obtient $[\![3,\,12]\!]$. Cette macro a pour syntaxe complète : `\interent[`⟨*md*⟩`][`⟨*avant*⟩`]{`⟨*n m*⟩`} [`⟨*après*⟩`]`, les arguments jouant le même rôle que ceux des macros pour intervalles, cf. page 11.

Sa définition utilise `\llbracket` et `\rrbracket` fournis par `stmaryrd.sty` chargé quand `taupe` vaut *true*.

\interzn Cette macro permet d'obtenir $[\![0,\,n]\!]$ avec `\interzn`. Sa syntaxe est dérivée de celle de `\interent` et prend les mêmes arguments *optionnels* avec la même signification. Sa syntaxe est donc `\interzn[`⟨*md*⟩`][`⟨*avant*⟩`][`⟨*après*⟩`]`.

Je consacre quelques lignes à la macro `\derpart` qui permet d'obtenir — et je passe en mode mathématique hors-texte pour l'occasion —

$$\frac{\partial^6 f(x,\,y,\,z)}{\partial x^2\,\partial y^3\,\partial z}$$

avec le code `\[\derpart{f\rnuplet{x y z}}{xxyyyz}\]`.

\TdSMDerPartSepar Cette macro contient ce qui sépare, p. ex., un ∂x^2 du ∂y qui le suit. Par défaut, elle est définie comme étant égale à `\,` ce qui, à mon sens, améliore le rendu. Mais on peut la redéfinir avec un coup de `\renewcommand`.

\derpart Comme on vient de le voir cette macro permet d'obtenir l'écriture de la dérivée partielle. Sa syntaxe est :
`\derpart{`⟨*dessus*⟩`}{`⟨*dessous*⟩`}`
où ⟨*dessus*⟩ est le texte qui sera composé à coté du ∂ au numérateur et ⟨*dessous*⟩ est une liste de lexèmes — à priori des lettres mais on peut y placer une macro comme `\alpha` en la faisant d'un espace — qui formeront le dénominateur.

Cette macro assure le passage en mode mathématique si nécessaire.

Allez, encore un petit exemple, `\[\derpart{f}{xyz␣␣zz\alpha\alpha␣x} \]` compose

$$\frac{\partial^8 f}{\partial x\,\partial y\,\partial z^3\,\partial\alpha^2\,\partial x}$$

et on voit que l'on peut se permettre de placer des espaces inutiles `;-)`

Remarque : pour les utilisateurs de `paresse.sty`. On ne peut pas utiliser `§a` comme raccourci de `\alpha` dans le 2e argument de `\derpart`. Ça serait analysé comme `§` puis `a` ce qui n'est peut-être pas tout à fait ce que l'on veut.

2.13 Des suites pour le secondaire

Lorsque l'on passe la valeur *true* à la clé `suite`, on charge le fichier `suite.sto` qui donne accès à quelques macros concernant les suites.

\suite La première macro `\suite` a pour syntaxe `\suite[`⟨*texte*⟩`]` et la valeur par défaut de ⟨*texte*⟩ est `u`. Elle assure le mode mathématique et on peut donc coder `\suite` pour avoir « (u_n) ».

\suite* La version étoilée a pour syntaxe `\suite*[`⟨*deco*⟩`][`⟨*texte*⟩`]` où ⟨*texte*⟩ a la même fonction que dans la version sans étoile et où ⟨*deco*⟩ vaut `\N` c.-à-d. $\mathbb{N}$ par défaut. On a donc « $(u_n)_{\mathbb{N}}$ » avec `\suite*`, « $(u_n)_{\mathbb{N}^*}$ » avec `\suite*[\N*]` et « $(w_n)_{\mathbb{N}}$ » avec `\suite*[\N][w]`.

LaTeX ne supporte pas les arguments optionnels imbriqués, les parenthèses dans `\suite*[{\N[*]}]` sont absolument indispensables, c'est pourquoi j'ai défini les macros étoilées, plussées et moinsées présentées en page 8.

`\suitar` La commande `\suitar` — `ar` pour *arithmétique* — a pour syntaxe
`\suitar[`⟨*texte*⟩`]{`⟨*raison*⟩`}[`⟨*rang*⟩`]{`⟨*prem*⟩`} [`⟨*entre*⟩`]`
où la valeur par défaut de ⟨*texte*⟩ est encore `u`, ⟨*raison*⟩ donne la valeur de la raison de la suite et ⟨*prem*⟩ la valeur du premier terme dont ⟨*rang*⟩ est le rang. Enfin ⟨*entre*⟩, qui vaut `{}` par défaut, compose le texte entre la suite et sa description.

En codant `\suitar{3}{5}`, on compose « (u_n) la suite arithmétique de raison $r = 3$ et de premier terme $u_0 = 5$ » et, avec `\suitar[w]{3}{5}`, on obtient « (w_n) la suite arithmétique de raison $r = 3$ et de premier terme $w_0 = 5$ », enfin, avec `\suitar{3}[1]{5}`, on obtient « (u_n) la suite arithmétique de raison $r = 3$ et de premier terme $u_1 = 5$ ».

Avec `\suitar{3}{5}[␣est]` on compose « (u_n) est la suite arithmétique de raison $r = 3$ et de premier terme $u_0 = 5$ », on fera attention à laisser un blanc devant le texte ici. Avec `\suitar{3}{5}[,]` on compose « (u_n), la suite arithmétique de raison $r = 3$ et de premier terme $u_0 = 5$ ».

`\suitgeo` La commande `\suitgeo` a la même syntaxe que `\suitar` mais cette fois elle concerne les suites géométriques. En codant `\suitgeo{3}{5}`, on compose « (u_n) la suite géométrique de raison $q = 3$ et de premier terme $u_0 = 5$ » et, avec `\suitgeo[w]{3}{5}`, on obtient « (w_n) la suite géométrique de raison $q = 3$ et de premier terme $w_0 = 5$ », enfin, avec `\suitgeo[w]{3}[2]{5}`, on obtient « (w_n) la suite géométrique de raison $q = 3$ et de premier terme $w_2 = 5$ ».

`\suitar*` Les versions étoilées, `\suitar*` et `\suitgeo*` ont la syntaxe suivante :
`\suitar*[`⟨*deco*⟩`][`⟨*texte*⟩`]{`⟨*raison*⟩`}[`⟨*rang*⟩`] {`⟨*prem*⟩`}[`⟨*entre*⟩`]`
où on retrouve l'argument optionnel ⟨*deco*⟩ de `\suite*` avec la même signification, cf. page 14.

On a donc « $(v_n)_{\mathbb{N}^*}$ la suite arithmétique de raison $r = 3$ et de premier terme $v_1 = 9$ » avec le code `\suitar*[\N*][v]{3}[1]{9}`. On prendra garde au fait que la macro ne cherche pas à assurer la cohérence entre l'ensemble des indices et le rang du premier terme `;-)`

`suitedeco` Ce que je viens de décrire est le comportement par défaut de ces macros `\suite`, `\suitar`, `\suitgeo`, `\suite*`, `\suitar*` et `\suitgeo*`, comportement
`false` obtenu lorsque la clé `suitedeco` a la valeur *`false`*. Lorsque l'on passe la valeur *`true`* à la clé `suitedeco` le comportement des macros avec et sans étoile est inversé.

Quitte à être un peu lourd, avec `suitedeco=false` on a « (u_n) » avec `\suite` et « $(u_n)_{\mathbb{N}}$ » avec `\suite*`. Avec `suitedeco=true` on a « $(u_n)_{\mathbb{N}}$ » avec `\suite` et « (u_n) » avec `\suite*`.

On fera attention que, si l'on a donné explicitement les premiers arguments optionnels de `\suitar`, p. ex., dans le cas où `suitedeco=true` on ne pourra pas tout bonnement repasser à `suitedeco=false` sans remplacer les formes sans étoiles par des formes étoilées et vice-versa en faisant, de plus, attention au 2e argument optionnel donnant le « nom » de la suite. Bref, on choisira une fois pour toute la forme de base et on s'y tiendra !

3 Récapitulatif

3.1 Extensions chargées

L'appel de `tdsfrmath` avec `\usepackage` entraine le chargement des extensions suivantes : `ifthen`, `xkeyval`, `amsmath`, `amssymb`, `xspace`, `xargs`, `suffix` et `stmaryrd` si la clé `taupe` a la valeur *`true`*.

Il n'est donc pas nécessaire de les appeler avec `\usepackage` dans le préambule d'un document chargeant `tdsfrmath`.

3.2 Options

Dans la table 3, page 16, je note « texte TEX » pour dire que la valeur passée à la clé doit être une chaine de *lettres* au sens de TEX, c.-à-d. les minuscules et majuscules non accentuées de l'ASCII comme on le trouve pour le nom des macros. Le « texte » tout court est ce qui sert à écrire les noms des extensions, on ne devrait donc pas y trouver de caractères *bizarres* mais on peut y voir des chiffres.

TABLE 3 – les clés de `tdsfrmath.sty`

clé	type	`valeur par défaut`	référence voir page
taupe	booléen	`false`	2
ArgArcMaj	booléen	`false`	3
suite	booléen	`false`	2
suitedeco	booléen	`false`	15
SepDefEnsExt	booléen	`true`	6
CharPoCal	booléen	`true`	3
calpack	texte	`mathrsfs`	3
calcomd	texte TEX	`mathscr`	3
caloptn	texte TEX	`***`	3
CharPoGdT	booléen	`false`	4
gdtpack	texte	`***`	4
gdtcomd	texte TEX	`***`	4
gdtoptn	texte TEX	`***`	4
placesigne	choix	`haut`	7
ensdeco	choix	`ebsh`	7

Liste des tableaux