

# L'extension `tdsfrmath`\*

Le  $\text{T}_{\text{E}}\text{X}$ nicien de surface<sup>†</sup>

6 août 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

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Utilisation</b>	<b>3</b>
2.1	Chargement optionnel	3
2.2	Réglage de la police calligraphique	4
2.3	Réglage de la police « gras de tableau »	4
2.4	Des n-uplets, de leur saisie et de leur présentation	5
2.5	De la définition des ensembles	7
2.6	Des noms des ensembles classiques	7
2.7	Des vecteurs, des bases et des repères	9
2.8	L'exponentielle	11
2.9	Le nombre $i$	11
2.10	Intégrales	11
2.11	Au bonheur du mathématicien, bazar	12
2.11.1	De l'infini	12
2.11.2	Des intervalles de $\mathbb{R}$	12
2.11.3	La réserve du bazar, miscellanées	13
2.12	Pour les taupes, taupins et taupines	13
2.13	Des suites pour le secondaire	15
<b>3</b>	<b>Récapitulatif</b>	<b>17</b>
3.1	Extensions chargées	17
3.2	Options	17

---

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

<sup>†</sup>`le.texnicien.de.surface@wanadoo.fr`

<b>4</b>	<b>Le code</b>	<b>17</b>
4.1	Options et macros de service	17
4.1.1	Séparateur des n-uplets	18
4.1.2	De l'aspect des noms des ensembles classiques	18
4.1.3	Du choix de la police calligraphique	19
4.1.4	Du choix du gras de tableau	19
4.1.5	Un peu plus pour les taupes	19
4.1.6	Des macros pour les suites	20
4.1.7	Séparateur de définition dans les ensembles	20
4.1.8	Exécutons les options	20
4.2	Les noms des ensembles	23
4.3	Couples, paires, triplets etc.	24
4.4	Vecteurs, bases et repères	25
4.5	L'exponentielle	27
4.6	Le nombre $i$	27
4.7	Intégrales	27
4.8	Au bonheur du mathématicien, bazar	28
4.9	Le fichier <code>taupe.sto</code>	29
4.10	Dérivées partielles	30

# 1 Introduction

Le but de cette extension est de fournir des macros prêtes 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 17, 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.

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

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

**ArgArcMaj** — Le fichier `taupe.sto` contient des définitions qui dépendent de la clé **ArgArcMaj** — `arg` et `arc` avec majuscule — qui vaut *false* par défaut ce qui entraîne 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 **Charger** une **Police Calligraphique** — 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*.

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

**calcomd** — 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 L<sup>A</sup>T<sub>E</sub>X 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 `eucal.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  $\mathcal{C}$  on trouve ».

## 2.3 Réglage de la police « gras de tableau »

Par défaut, le « gras de tableau » (*blackboard bold*) est celui de L<sup>A</sup>T<sub>E</sub>X 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 **Charger** une **Police Gras de Tableau** — 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 une option avec **gdtoptn** et, toujours éventuellement, fixer **gdtcmd** avec le nom de la macro désirée — toujours sans barre oblique inverse — seulement, cette fois, du fait du choix de la valeur par défaut de **gdtcmd**, on n'aura pas besoin de fixer la valeur de **gdtcmd** si la macro est encore `\mathbb`.

**gdtoptn**

**gdtcmd**

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,
             gdtcmd=mathds]{tdsfrmth}
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 unifié » pour obtenir « et comme  $\mathbb{M}$  est unifié ».

## 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<sup>1</sup> pour écrire des textes à destination des élèves du secondaire car on est souvent amené 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 réserver, 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'âge — au moins pendant un moment.

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

`\EncloreExtensible` En fait, la présentation obtenue avec `\nuplet` repose sur `\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 1<sup>er</sup> ou le 2<sup>e</sup> 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  $(x^2)$ .

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 1<sup>er</sup> cas, les délimiteurs sont extensibles, dans le second ils ne le sont pas. On verra plus bas, page 10, 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 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{8\cdot 9}` qui produit  $[8 \cdot 9]$ .

`\varabs` Dans le même genre, on a `\varabs` pour obtenir la valeur absolue comme ici :  $|-12|$  codé `\varabs{-12}`.

`\norme` Dans la même veine, toujours, `\norme` pour écrire la norme comme suit :  $\|\vec{v}\|$  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 sur l'espace, non ? — et avec `\anuplet{c,d}` on a  $\{c, d\}$ . Cette dernière doit son nom à ce qu'elle utilise des accolades.

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  $\TeX$  préférera une de ces habituelles remarques absconces ;-)

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

`\nuplet{a Sa Sb}` pour avoir  $(a, \alpha, \beta)$  lorsque l’extension `paresse.sty`, de votre serveurur, 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’espace 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<sup>2</sup> de nommage des macros.

## 2.5 De la définition des ensembles

`SepDefEnsExt`  
`true`

Je fournis la macro `\ensemble`, cf. page 13, qui permet d’écrire, p. ex., «  $\{x \in \mathbb{R} / x^2 \geq 2\}$  » avec `\ensemble{x\in\mathbb{R}}{x^2\pgq 2}`. Le rendu en est contrôlé par la clé `SepDefEnsExt` — séparateur de la définition d’un ensemble extensible — qui vaut `true` par défaut. Par ailleurs, la macro `\TdSMsepdefens` contient le séparateur et peut-être redéfinie à l’aide d’un `\renewcommand`.

`\TdSMsepdefens`

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

`\TdSM@Decoration`

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

Cette macro *secrète* place les étoiles et signe plus ou moins, ce que j’appelle

<sup>2</sup> 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 est contrôlée 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'étoile en **haut** et le **signe** en **bas**. 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 contrôlée 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 1<sup>er</sup> donne la lettre majuscule symbolisant l'ensemble comme «  $\mathbb{R}$  »  
`\C` pour  $\mathbb{R}$  ; le comportement de la macro varie suivant que le 2<sup>e</sup> est égal à 1, est un entier supérieur à 1, un entier strictement négatif ou l'une de ces sept chaînes de caractères `*`, `+`, `-`, `**`, `++`, `-*` et `*-` — c'est ce qui permet d'obtenir plus tard,  
`\N` p. ex.,  $\mathbb{Q}_+^*$  avec `\Q[+*]` — ; le 3<sup>e</sup> argument est utilisé pour dénoter les ensembles de polynômes comme  $\mathbb{C}_3[X]$  et dans ce cas le 2<sup>e</sup> doit être un nombre négatif ; enfin  
`\Q` le 4<sup>e</sup> doit être un entier qui donne le nombre de `mu` — unité de longueur spécifique  
`\R` au mode mathématique — qui séparent la majuscule du crochet ouvrant.  
`\Z`

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 contrôle car, en dehors de `*`, `+`, `-`, `**`, `++`, `-*` et `*-`, le 2<sup>e</sup> argument doit être un entier relatif. On peut saisir `\R[--4]` — mais pourquoi le ferait-on ? — et 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 2<sup>e</sup> et 3<sup>e</sup> arguments de `\EnsembleDeNombre`. Par défaut, le 1<sup>er</sup> argument vaut 1 et le 2<sup>e</sup> `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 2<sup>e</sup> argument optionnel sans donner d'abord le premier.

Cependant, pour des raisons que l'on peut voir page 15 à propos de `\suite*`, je fournis quelques macros supplémentaires qui, du coup, peuvent abrégé la saisie.

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

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

`\R>`  $\mathbb{R}$  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 23.

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 1<sup>er</sup> argument obligatoire est lui-même une macro comme p. ex. `\imath`.

Cette macro fait appel à `\TdsM@fairevecteur` décrite page 26.

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

`\vectk` On pourra comparer la composition que permet d'obtenir `\TdsM@fairevecteur`, à l'aide des 2<sup>e</sup> et 3<sup>e</sup> arguments qui définissent un nombre de mus, avec ce que donne une composition directe comme ici :  $\vec{i}$  obtenu avec `\vecti` et  $\vec{i}$  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  $\overrightarrow{CD}$  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  $\overrightarrow{AB}$  produit par `\vecteur[10]{AB}`,  $\overrightarrow{AB}$  produit par `\vecteur{AB}[20]` et  $\overrightarrow{AB}$  produit par `\vecteur[10]{AB}[20]`.

`\V` Ce n'est qu'un raccourci de `\vecteur`. Il a donc la même syntaxe :  
`\V*[\langle espace-avant \rangle]{\langle texte \rangle}[\langle espace-après \rangle]`  
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  $(\vec{i})$  avec `\base[1]`,  $(\vec{i}, \vec{j})$  avec `\base` ou `\base[2]`,  $(\vec{i}, \vec{j}, \vec{k})$  avec `\base[3]`.

`\repere` La macro `\repere` fournit un repère à la française. Elle est construite sur `\base` et son 1<sup>er</sup> argument optionnel a le même rôle que celui de `\base`. Le 2<sup>e</sup> argument optionnel de `\repere` définit le centre du repère, c'est 0 par défaut.

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

Je rappelle qu'il faut le 1<sup>er</sup> argument optionnel si l'on veut préciser le 2<sup>e</sup>.  
Voyons maintenant le rendu des repères dans une formule hors-texte :

$$(O, \vec{i}) \quad (O, \vec{i}, \vec{j}) \quad (O, \vec{i}, \vec{j}, \vec{k})$$

Viennent maintenant des macros qui servent essentiellement d'abréviations.

`\rog` Tout d'abord ce qu'il faut pour écrire « repère orthogonal  $(O, \vec{i}, \vec{j})$  » avec  
`\ron` `\rog` puis « repère orthonormal  $(O, \vec{i}, \vec{j})$  » avec `\ron` et enfin « repère ortho-  
`\rond` normal direct  $(O, \vec{i}, \vec{j})$  » avec `\rond`.

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

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

Je fournis de quoi écrire les repères à la mode du collègue<sup>3</sup> mais je ne traite que le cas d'un repère du plan.

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

`\Rog` Viennent ensuite des abréviations, construites sur le même modèle que les pré-  
`\Ron` cédentes : `\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  
`\Rond` 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 9. 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 \(\np{2,7}\)}` grâce 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, grâce 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  $i^2 = -1$  » 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

$$\int_a^b f(x) dx \quad \text{plutôt que ça :} \quad \int_a^b f(x)dx$$

- `\FixeReculIntegrande` 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 2<sup>e</sup> argument optionnel ne peut être donné que si le 1<sup>er</sup> 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

$$\int_2^5 \frac{dx}{\ln x}$$

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  $\langle var \rangle$ , qui est le symbole de la variable, et un argument optionnel final  $\langle avance \rangle$  qui règle la distance entre l'intégrande et le  $\D$ ;  $\langle avance \rangle$  est soumis aux mêmes règles que le  $\langle recul \rangle$ . Par défaut  $\langle avance \rangle$  vaut  $4\mu$ .

On code `\(\integrer{0}{\pi}{\cos 2x}{x}\)` pour avoir

$$\int_0^{\pi} \cos 2x \, dx$$

`\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 3<sup>e</sup> 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  $\int_a^b f(x) \, dx$  dans le cours du texte et

$$\int_a^b f(x) \, dx$$

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` J'ai mis très longtemps à retenir le nom de `\infty`, aussi je me suis fait  
`\moinsinf` `\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` On peut écrire les différents intervalles avec les macros `\interff`, `\interoo`,  
`\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*.

`\interfo` Le premier argument  $\langle md \rangle$  est optionnel est règle le mode mathématique, il vaut 1 par défaut. Le 2<sup>e</sup>  $\langle avant \rangle$ , optionnel, vaut 0 par défaut et donne le nombre de  $\mu$ s qui sépare le délimiteur ouvrant du texte. Le 4<sup>e</sup> et dernier  $\langle après \rangle$ , qui vaut  
`\interof` 0 par défaut, est également optionnel. Il définit, en nombre de  $\mu$ s, la distance qui sépare le texte intérieur du délimiteur fermant.

Le 3<sup>e</sup> argument  $\langle a b \rangle$  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`      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 `\(f\): \(\mathbb{R} \text{ dans } \mathbb{R}\)`; `\(x \text{ donne } 2x\)` compose : «  $f : \mathbb{R} \longrightarrow \mathbb{R}; x \longmapsto 2x$  »<sup>4</sup>.
- `\vide`        Je préfère  $\emptyset$  à  $\emptyset$ <sup>5</sup> et par paresse encore, je me suis fait une `\vide` qui permet de saisir `\vide et autre` pour obtenir «  $\emptyset$  et autre ». Merci `\ensuremath et \xspace`.
- `\ppq`  
`\pgq`        Je veux ceci  $0 \leq 1$  et  $2 \geq 1$ . Comme `\leqslant` et `\geqslant`, c'est pas beau, je me suis fait `\ppq — plus petit que —` et `\pgq — grand`.
- `\ensemble`    La macro `\ensemble` a deux arguments obligatoires, elle sert à écrire la définition d'un ensemble comme «  $\{x \in \mathbb{R} / f(x) \geq \frac{1}{2}\}$  » obtenu avec `\ensemble{x \in \mathbb{R} \pgq \frac{1}{2}}` et dont l'aspect est

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

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[\langle avant \rangle]{\langle 1er texte \rangle}{\langle 2e texte \rangle}[\langle après \rangle]`

où  $\langle avant \rangle$  et  $\langle après \rangle$  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 mus.

## 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  $\mathbb{C}$ .

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

Je redéfinit 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 : `argch x`, `argsh y` et `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 redéfinit également les macros `\arccos`, `\arcsin` et `\arctan` pour qu'elles aient le même aspect.

TABLE 1 – Macros redéfinies dans `taupe.sto`

<code>\sinh</code>	<code>sh x</code>	<code>\cosh</code>	<code>ch x</code>
<code>\tanh</code>	<code>th x</code>	<code>\cot</code>	<code>cotan y</code>

TABLE 2 – Macros dont l'aspect dépend de la clé `ArgArcMaj` — aspect par défaut

<code>\arccos</code>	<code>arccos x</code>	<code>\arcsin</code>	<code>arcsin x</code>
<code>\arctan</code>	<code>arctan x</code>	<code>\argsh</code>	<code>argsh x</code>
<code>\argch</code>	<code>argch x</code>	<code>\argth</code>	<code>argth x</code>

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

`\tendversen` Pour écrire  $f(x) \xrightarrow{0} +\infty$ , 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  $DL_4(0)$  avec `\devlim{4}` et  $DL_7(1)$  avec `\devlim[1]{7}`. On remarque donc que `<en>` vaut 0 par défaut. N'est-ce pas étrange ?

`\parties` Pour écrire  $\mathcal{P}(E)$ , je fournis `\parties` utilisable en mode texte. Sa syntaxe est `\parties[<n>]{<ensemble>}` où `n` est un nombre de mus qui permet de régler la distance entre  $\mathcal{P}$  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 «  $\frac{df(x)}{dx}$  », 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  $\frac{df(x)}{dx}$  vaut », en bousillant l’interligne comme prévu !

`\interent` Avec `\interent{3 12}` on obtient  $\llbracket 3, 12 \rrbracket$ . 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 12. Sa définition utilise `\llbracket` et `\rrbracket` fournis par `stmaryrd.sty` chargé quand `taupe` vaut `true`.

`\interzn` Cette macro permet d’obtenir  $\llbracket 0, n \rrbracket$  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 `\[\derpart{f\rnuplet{x y z}}{xyyyz}\]`.

`\TdSMDerPartSepar` Cette macro contient ce qui sépare, p. ex., un  $\partial x^2$  du  $\partial y$  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  $\partial$  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 suivre 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 2<sup>e</sup> 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]`.

L<sup>A</sup>T<sub>E</sub>X 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 9.

`\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 15.

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 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 2<sup>e</sup> 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` entraîne 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 17, je note « texte  $\TeX$  » pour dire que la valeur passée à la clé doit être une chaîne 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
<code>taupe</code>	booléen	<code>false</code>	3
<code>ArgArcMaj</code>	booléen	<code>false</code>	4
<code>suite</code>	booléen	<code>false</code>	3
<code>suitedeco</code>	booléen	<code>false</code>	16
<code>SepDefEnsExt</code>	booléen	<code>true</code>	7
<code>CharPoCal</code>	booléen	<code>true</code>	4
<code>calpack</code>	texte	<code>mathrsfs</code>	4
<code>calcomd</code>	texte $\TeX$	<code>mathscr</code>	4
<code>caloptn</code>	texte $\TeX$	<code>***</code>	4
<code>CharPoGdT</code>	booléen	<code>false</code>	5
<code>gdtpack</code>	texte	<code>***</code>	5
<code>gdtcomd</code>	texte $\TeX$	<code>***</code>	5
<code>gdtoptn</code>	texte $\TeX$	<code>***</code>	5
<code>placesigne</code>	choix	<code>haut</code>	8
<code>ensdeco</code>	choix	<code>ebsh</code>	8

## 4 Le code

### 4.1 Options et macros de service

Je commence par charger une extension bien utile tout de suite.

```
1 \RequirePackage{ifthen}
```

et une autre qui ne l'est pas moins.

```
2 \RequirePackage{xkeyval}
```

`tdsfrmath.sty` est la première extension où je fais usage de `xkeyval.sty` et de sa gestion de clés pour les options de l’extension.

#### 4.1.1 Séparateur des n-uplets

`\TdSM@separateur` Cette macro, déterminée par la clé `nupletsep`, fournit le séparateur utilisé dans l’écriture des  $n$ -uplets comme dans  $(a, b)$ .

J’en profite pour signaler que toutes les macros « secrètes » de cette extension commence par `\TdSM@`.

Pour que, plus bas, `\ExecuteOptionsX` ait le bon gout de considérer les options de l’extension, il faut utiliser le 2<sup>e</sup> argument de `\define@choicekey` en lui passant le nom complet du fichier contenant l’extension.

Pour des raisons de tranquillité je regroupe toutes les clés définies au niveau de l’extension dans le *trousseau* — la documentation de `xkeyval` parle simplement d’un préfixe — `TdSM`, c’est la raison du 1<sup>er</sup> argument.

Le 3<sup>e</sup> argument contient le nom de la clé. Le 4<sup>e</sup> contient deux macros qui prendront pour la première la valeur passée à la clef si elle est valide et pour la deuxième un nombre valant  $-1$  si la valeur est invalide sinon le numéro de la valeur dans la liste — aux valeurs séparées par des virgules — constituant le 5<sup>e</sup> argument. Les valeurs sont numérotées en partant de 0.

Le 6<sup>e</sup> argument contient du code qui est exécuté quand la valeur passée à la clé est valide. Ici, je me sers de `\nr` dans le `\ifcase` pour définir `\TdSM@separateur` congruement au choix de l’utilisateur : le 0<sup>e</sup> cas est celui de la *virgule*, &c.

Le 7<sup>e</sup> argument contient le code exécuté quand la valeur est invalide. Ici ça consiste en un avertissement — erreur non fatale — inscrit à la console et dans le `.log`.

Grace au `+` on peut utiliser un 7<sup>e</sup> argument. Grace au `*`, la vérification de la validité de la valeur passée à la clé est faite après que l’entrée et la liste ont été passée en minuscules. L’utilisateur peut donc saisir `[nupletsep=Virgule]` sans troubler l’extension.

À la sortie du traitement, on met `\val` à `\relax` pour éviter de le rendre inutilisable alors que l’on n’en a plus besoin. On fera la même chose avec `\nr` après l’exécution des options.

```

3 \define@choicekey*+[TdSM]{tdsfrmath.sty}{nupletsep}[\val\nr]%
4 {virgule,pointvirgule}{%
5   \ifcase\nr\relax
6   \newcommand{\TdSM@separateur}{\string,\,}\or
7   \newcommand{\TdSM@separateur}{\,\string;\,}\fi
8 }{%
9   \PackageWarningNoLine{tdsfrmath}{la clef <<nupletsep>> ne connaît pas
10     <<\val>>\MessageBreak <<nupletsep=virgule>> en vigueur}
11 }
12 \let\val\relax

```

#### 4.1.2 De l’aspect des noms des ensembles classiques

`\TdSM@Decoration` Cette macro place la décoration. Elle est réglée par `ensdeco`. C’est par la déclaration de cette clé que je commence. La macro `\TdSM@DecoChoix` sert à conserver la trace de la valeur passée à la clé. On s’en servira au moment de définir la macro `\TdSM@Decoration`

```

13 \define@choicekey*+[TdSM]{tdsfrmath.sty}{ensdeco}[\TdSM@DecoChoix\nr]%
14 {ehsb,ehsb,ehsh,ebsh,sbeb,shh}{%
15   \PackageInfo{tdsfrmath}{Vous avez choisi \TdSM@DecoChoix@gobble}
16 }{%
17   \PackageWarningNoLine{tdsfrmath}{la clef <<ensdeco>> ne connaît pas
18     <<\TdSM@DecoChoix>>\MessageBreak <<ensdeco=ehsb>> en vigueur}%
19   \def\TdSM@DecoChoix{ehsb}}

```

`\TdSM@PlaceSigne` Cette macro place le signe seul. Elle est réglée par `placesigne`. La macro `\TdSM@PlaSiChoix` sert à conserver la trace de la valeur passée à la clé. On s'en servira au moment de définir la macro `\TdSM@PlaceSigne`

```

20 \define@choicekey*+[TdSM]{tdsfrmath.sty}{placesigne}[\TdSM@PlaSiChoix\nr]%
21 {haut,bas}{%
22   \PackageInfo{tdsfrmath}{Vous avez choisi \TdSM@PlaSiChoix@gobble}
23 }{%
24   \PackageWarningNoLine{tdsfrmath}{la clef <<placesigne>> ne connaît pas
25     <<\TdSM@PlaSiChoix>>\MessageBreak <<placesigne=haut>> en vigueur}%
26   \def\TdSM@PlaSiChoix{haut}}

```

### 4.1.3 Du choix de la police calligraphique

Le mécanisme de `xkeyval` crée la macro `\ifTdSM@CharPoCal` et ses sœurs `\TdSM@CharPoCaltrue` et `\TdSM@CharPoCalfalse`. Le 5<sup>e</sup> argument devrait contenir la valeur par défaut mais, dans le cadre de la définition des options d'une extension, ce mécanisme ne fonctionne pas comme il semble le devoir d'après le manuel. Je dois avouer qu'ici quelque chose m'échappe.

```

27 \define@boolkey+[TdSM]{tdsfrmath.sty}[TdSM@]{CharPoCal}[]{}%
28 {\PackageWarningNoLine{tdsfrmath}{CharPoCal attend la valeur
29   << true >> ou << false >>}}

```

Avec `\define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{calpack}[]{}` je définis la clé `calpack` et conjointement une macro `\TdSM@calpack` qui contiendra la valeur passée à la clé. Le préfixe de la commande est le contenu du 3<sup>e</sup> argument. La clé est *attachée au trousseau* `TdSM` grâce au 1<sup>er</sup> argument et, une fois encore, la famille est obligatoire pour le fonctionnement ultérieur du `\ExecuteOptionsX` et `\ProcessOptionsX`, c'est l'objet du 2<sup>e</sup> argument.

Le dernier pourrait contenir du code mais je m'occupe de la valeur passée à la clé plus bas, *à la main*!

```

30 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{calpack}[]{}
31 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{calcmd}[]{}
32 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{caloptn}[]{}

```

### 4.1.4 Du choix du gras de tableau

```

33 \define@boolkey[TdSM]{tdsfrmath.sty}[TdSM@]{CharPoGdT}[]{}
34 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{gdtpack}[]{}
35 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{gdtcmd}[]{}
36 \define@cmdkey[TdSM]{tdsfrmath.sty}[TdSM@]{gdtoptn}[]{}

```

### 4.1.5 Un peu plus pour les taupes

Avec la clé booléenne `taupe`, on charge le fichier `taupe.sto`. Cela permet de garder l'extension assez mince pour une utilisation dans le secondaire.

Si la valeur passée à la clé est `true` alors `\ifTdsM@taupe` est définie comme valant `true` ce qui entrainera plus bas le chargement de l'extension `stmaryrd.sty` qui fournit des symboles mathématiques dont celui dont je me sers pour écrire les intervalles de  $\mathbb{N}$ .

```
37 \define@boolkey[TdsM]{tdsfrmath.sty}[TdsM@]{taupe}[]{}

```

Clé fixant l'aspect des fonctions circulaires et hyperboliques réciproques.

```
38 \define@boolkey[TdsM]{tdsfrmath.sty}[TdsM@]{ArgArcMaj}[]{}

```

#### 4.1.6 Des macros pour les suites

Avec la clé booléenne `suite`, on charge le fichier `suite.sto`.

```
39 \define@boolkey[TdsM]{tdsfrmath.sty}[TdsM@]{suite}[]{}

```

```
40 \define@boolkey[TdsM]{tdsfrmath.sty}[TdsM@]{suitedeco}[]{}

```

#### 4.1.7 Séparateur de définition dans les ensembles

La clé booléenne `SepDefEnsExt`.

```
41 \define@boolkey[TdsM]{tdsfrmath.sty}[TdsM@]{SepDefEnsExt}[]{}

```

#### 4.1.8 Exécutons les options

Il est temps d'exécuter les options par défaut. Puis de s'occuper de celles fournies par l'utilisateur. Pour finir, on relache une macro qui n'a plus d'usage.

```
42 \ExecuteOptionsX[TdsM]{%
43   taupe=false,%
44   ArgArcMaj=false,%
45   suite=false,%
46   suitedeco=false,%
47   nupletsep=virgule,%
48   SepDefEnsExt=true,%
49   placesigne=haut,%
50   ensdeco=ehsb,%
51   CharPoCal=true,calpack=mathrsfs,calcmd=mathscr,caloptn=***,%
52   CharPoGdT=false,gdtpack=***,gdtdcmd=***,gdtoptn=***}
53 \ProcessOptionsX[TdsM]\relax
54 \let\nr\relax

```

On charge maintenant les extensions nécessaires à cette extension. La moins connue, car la plus récente, est peut-être `xargs.sty` que l'on verra à l'œuvre plusieurs fois pour définir des macros acceptant plusieurs arguments optionnels.

```
55 \RequirePackage{amsmath, amssymb}
56 \RequirePackage{xspace}
57 \RequirePackage{xargs}
58 \RequirePackage{suffix}

```

Si on a passé `[taupe=true]`, on charge `stmaryrd.sty`

```
59 \ifTdsM@taupe
60 \RequirePackage{stmaryrd}

```

puis, à la fin de l'extension, on inclut le fichier `taupe.sto` si on le trouve, sinon on grommèle.

```
61 \AtEndOfPackage{%

```

```

62 \InputIfFileExists{taupe.sto}{%
63   \PackageInfo{tdsfrmath}{fichier taupe.sto inclus \@gobble}}{%
64   \PackageWarningNoLine{tdsfrmath}{fichier taupe.sto introuvable}}
65 \fi

  On traite la clé suite de même :
66 \ifTdsM@suite
67 \AtEndOfPackage{%
68   \InputIfFileExists{suite.sto}{%
69     \PackageInfo{tdsfrmath}{fichier suite.sto inclus \@gobble}}{%
70     \PackageWarningNoLine{tdsfrmath}{fichier suite.sto introuvable}}
71 \fi

  Si on a CharPoCal=true, on s'occupe de la police calligraphique
72 \ifTdsM@CharPoCal
73   \PackageInfo{tdsfrmath}{La clef CharPoCal est vraie \@gobble}
  on s'inquiète de l'existence de l'extension réclamée, on la charge, avec l'éventuelle
  option, si on la trouve
74   \IfFileExists{\TdsM@calpack.sty}{%
75     \ifthenelse{\equal{\TdsM@caloptn}{***}}{%
76       {\RequirePackage{\TdsM@calpack}}
77       {\RequirePackage[\TdsM@caloptn]{\TdsM@calpack}}
78       \def\TdsM@MathCalPol{\csname\TdsM@calcomd\endcsname}
  et, une fois chargée l'extension, on teste l'existence de la commande demandée
79       \@ifundefined{TdsM@MathCalPol}}%
  on grommèle si la macro est inconnue
80       {\PackageWarningNoLine{tdsfrmath}{La macro
81         << \TdsM@calcomd\space >> n'est pas connue\MessageBreak
82         par l'extension \TdsM@calpack\space!\MessageBreak Revoyez
83         la valeur de la clef << calcomd >> SVP.}}%
  ou on signale que tout s'est passé correctement.
84       {\PackageInfo{tdsfrmath}{La police calligraphique est obtenue via
85         << \TdsM@calcomd\space >> \@gobble}}}%
  Vient le cas où l'extension n'est pas présente : on grommèle.
86   {\PackageWarningNoLine{tdsfrmath}{Extension \TdsM@calpack\space
87     pas vue sur la machine !}}
  Cela dit, il faut bien définir \TdsM@MathCalPol dans le cas où on la veut sortie de
  boîte c.-à-d. égale à \mathcal.
88 \else
89   \PackageInfo{tdsfrmath}{La clef CharPoCal est fausse.\MessageBreak
90     La police calligraphique est << mathcal >> \@gobble}
91   \def\TdsM@MathCalPol{\mathcal}
92 \fi

  Si on a CharPoGdT=true, on s'occupe de la police calligraphique en employant la
  même méthode — et le même code ;- ) — avec la différence qu'il faut tenir compte
  du fait que, par défaut, \TdsM@gdtcomd contient *** :
93 \ifTdsM@CharPoGdT
  TdsM@CharPoGdT est true.
94   \IfFileExists{\TdsM@gdtpack.sty}

```

L'extension passée à `gdtpack` existe. Première branche du `\IfFileExists`.

```
95  {%
On traite l'option éventuelle pour \RequirePackage.
96  \ifthenelse{\equal{\TdSM@gdtoptn}{***}}{%
97    {\RequirePackage{\TdSM@gdtpack}}
98    {\RequirePackage[\TdSM@gdtoptn]{\TdSM@gdtpack}}
```

On traite la commande éventuelle.

```
99  \ifthenelse{\equal{\TdSM@gdtcomd}{***}}{%
100   {\def\TdSM@MathGdTPol{\mathbb}}
```

On n'est pas dans le cas banal, on vérifie que la commande demandée est bien définie.

```
101   {\def\TdSM@MathGdTPol{\csname\TdSM@gdtcomd\endcsname}
102   \ifundefined{\TdSM@MathGdTPol}%
103   {\PackageWarningNoLine{tdsfrmath}{La macro
104     << \TdSM@gdtcomd\space >> n'est pas connue\MessageBreak
105     par l'extension \TdSM@gdtpack\space!\MessageBreak Revoyez
106     la valeur de la clef << gdtcomd >> SVP.}}%
107   {\PackageInfo{tdsfrmath}{Le gras de tableau est obtenu via
108     << \TdSM@gdtcomd\space >>\@gobble}}}%
```

L'extension passée à `gdtpack` n'est pas trouvée. Seconde branche du `\IfFileExists`.

```
109   {\PackageWarningNoLine{tdsfrmath}{Extension \TdSM@gdtpack\space
110     pas vue sur la machine !}}
111 \else
TdSM@CharPoGdT est false.
112   \def\TdSM@MathGdTPol{\mathbb}
113 \fi
```

Les options étant traitées, `\TdSM@DecoChoix` contient la valeur passée à la clé `ensdeco`. On peut définir `\TdSM@Decoration` congruement.

```
114 \ifthenelse{\equal{\TdSM@DecoChoix}{ehsb}}{%
115   {\newcommand\TdSM@Decoration[2]{\^{#1}_{#2}}}{%
116   \ifthenelse{\equal{\TdSM@DecoChoix}{sheh}}{%
117     {\newcommand\TdSM@Decoration[2]{\^{#2#1}}}{%
118     \ifthenelse{\equal{\TdSM@DecoChoix}{ehsb}}{%
119       {\newcommand\TdSM@Decoration[2]{\^{#1}_{#2}}}{%
120       \ifthenelse{\equal{\TdSM@DecoChoix}{ehsb}}{%
121         {\newcommand\TdSM@Decoration[2]{_{#1#2}}}{%
122         \ifthenelse{\equal{\TdSM@DecoChoix}{sbeb}}{%
123           {\newcommand\TdSM@Decoration[2]{_{#2#1}}}{%
124           \ifthenelse{\equal{\TdSM@DecoChoix}{ebsh}}{%
125             {\newcommand\TdSM@Decoration[2]{\^{#2}_{#1}}}{%
126             }}}}}}}
127 \let\TdSM@DecoChoix=\relax
```

De même pour `\TdSM@PlaceSigne` :

```
128 \ifthenelse{\equal{\TdSM@PlaSiChoix}{haut}}{%
129   {\newcommand\TdSM@PlaceSigne[1]{\^{#1}}}{%
130   {\newcommand\TdSM@PlaceSigne[1]{_{#1}}}{%
131   \let\TdSM@PlaSiChoix=\relax
```

## 4.2 Les noms des ensembles

Pour pouvoir écrire  $\mathbb{R}$  afin d'obtenir  $\mathbb{R}$ , il faut quelques macros auxiliaires par lesquelles je commence.

```

\EnsembleDeNombre La suivante fait le boulot de composition.
132 \newcommand\EnsembleDeNombre[4]{%
133   \ensuremath{%
134     \grastab{#1}%
135     \ifthenelse{\equal{#2}{*}}{\ast}{%
136       \ifthenelse{\equal{#2}{+}}{\TdsM@PlaceSigne{+}}{%
137         \ifthenelse{\equal{#2}{-}}{\TdsM@PlaceSigne{-}}{%
138           \ifthenelse{\equal{#2}{+*}\or\equal{#2}{*+}}{%
139             \TdsM@Decoration{\ast}{+}}{%
140             \ifthenelse{\equal{#2}{-*}\or\equal{#2}{*-}}{%
141               \TdsM@Decoration{\ast}{-}}{%
142               \ifthenelse{#2=1}{\number{#2}}{%
143                 \ifthenelse{#2>1}{\number{#2}}{%
144                   \ifthenelse{#2<0}{\number{-#2}\mspace{#4mu}\left[ #3\right]}{
145                     [\string?\string?\string?]}%
146                   }}}}}}\xspace}

147 \newcommand\N[2][1=1,2=X]%
148   {\EnsembleDeNombre{N}{#1}{#2}{\TdsMReculParentese}}
149 \newcommand\Z[2][1=1,2=X]%
150   {\EnsembleDeNombre{Z}{#1}{#2}{\TdsMReculParentese}}
151 \newcommand\Q[2][1=1,2=X]%
152   {\EnsembleDeNombre{Q}{#1}{#2}{\TdsMReculParentese}}
153 \newcommand\R[2][1=1,2=X]%
154   {\EnsembleDeNombre{R}{#1}{#2}{\TdsMReculParentese}}
155 \newcommand\C[2][1=1,2=X]%
156   {\EnsembleDeNombre{C}{#1}{#2}{\TdsMReculParentese}}

```

David KASTRUP ayant écrit récemment que la commande  $\text{\LaTeX \@ifstar}$  qui teste la présence d'une étoile  $*$  derrière une commande n'était pas des plus recommandables, j'utilise son extension `suffix.sty` pour définir les commandes étoilées et, tant que j'y suis, les commandes plussées, moinsées et autres sur le même modèle.

```

157 \WithSuffix\newcommand\N*\{N[*]}
158 \WithSuffix\newcommand\Z*\{Z[*]}
159 \WithSuffix\newcommand\Q*\{Q[*]}
160 \WithSuffix\newcommand\R*\{R[*]}
161 \WithSuffix\newcommand\C*\{C[*]}
162 \WithSuffix\newcommand\Q+{\Q[+]}
163 \WithSuffix\newcommand\R+{\R[+]}
164 \WithSuffix\newcommand\Q-{\Q[-]}
165 \WithSuffix\newcommand\R-{\R[-]}
166 \WithSuffix\newcommand\R>{\R[+*]}
167 \WithSuffix\newcommand\R<{\R[-*]}

```

Commençons par nous présenter puisque c'est la première fois que l'on rencontre ce fichier auxiliaire :

```

168 \ProvidesFile{taupe.sto}%
169 [\filedate\space\fileversion\space Pour tdsfrmath -- option taupe]

```

```

170 \newcommandx\K[2][1=1,2=X]%
171     {\EnsembleDeNombre{K}{#1}{#2}{\TdSMReculParentese}}
172 \WithSuffix\newcommand\K*{\K[*]}

```

### 4.3 Couples, paires, triplets etc.

Il s'agit maintenant de définir les macros qui permettent d'obtenir, p. ex.,  $(a, b, c)$  avec `\nuplet{a_b_c}`. On aura noté que la délimitation des arguments est obtenue avec des espaces.

Là encore, je commence par une macro auxiliaire. Elle a pour tâche de récupérer une liste d'arguments délimités par des espaces et de fournir une liste de ces mêmes arguments séparés par le séparateur fixé par `nupletsep`. Elle est basée sur une macro que Manuel PÉGOURIÉ-GONNARD, l'auteur de `xargs.sty`, m'a, très aimablement, fourni en réponse à une mienne question sur `fr.comp.text.tex`.

La seule « astuce » est l'utilisation de `#1#2#3`, ou l'équivalent ensuite, pour récupérer ce que je veux. Comme `#1` est immédiatement suivi de `#2`, T<sub>E</sub>X le considère comme non-délimité et, quand on utilise la macro, il prendra le premier lexème *différent d'un espace*. Le `#2` se chargera de récupérer alors tout ce qui suit jusqu'à l'espace suivant. Et s'il n'y a rien à récupérer il restera vide. Avec cette manœuvre, je permets de coder `\nuplet{_a_b}` pour obtenir  $(a, b)$ .

```

173 \newcommand\TdSMnuplet[1]{\TdSM@nuplet #1 \@nil}
174 \def\TdSM@nuplet #1#2 #3{%
175     \ifx\@nil #3%
176     #1#2%
177     \else
178     #1#2\TdSM@separateur%
179     \TdSM@nupletaux #3\fi}
180 \def\TdSM@nupletaux#1\fi{
181     \fi\TdSM@nuplet#1}

```

`\EncloreExtensible` Pour *passer* le mode à l'intérieur de la macro, en vue de prendre la bonne définition des délimiteurs, j'ai besoin d'un booléen.

```

182 \newboolean{TdSM@horstexterequis}
183 \setboolean{TdSM@horstexterequis}{false}
184 \newcommandx\EncloreExtensible}[4][1=1]{%
185     \ifthenelse{#1=0}
186     {\setboolean{TdSM@horstexterequis}{true}}
187     {\setboolean{TdSM@horstexterequis}{false}}

```

J'ai réglé le cas de l'argument optionnel. Il faut voir si on ne serait pas en mode mathématique (`\ifmmode`) interne (`\ifinner`) c.-à-d. mathématique en ligne, ou, au contraire en mode mathématique hors-texte où il faut faire quelque chose :

```

188     \ifmmode\ifinner\else
189         \setboolean{TdSM@horstexterequis}{true}
190     \fi\else\fi

```

On ouvre un groupe et on s'assure d'être en mode mathématique et on agit en accord avec la valeur du booléen `TdSM@horstexterequis` qui contient le renseignement nécessaire :

```

191     {\ensuremath{%
192         \ifthenelse{\boolean{TdSM@horstexterequis}}
193         {\displaystyle\def\@v@nt{\left #2}\def\@pr@s{\right #3}}

```

En hors-texte — naturel ou forcé — on a des délimiteurs extensibles,

```
194     {\ifthenelse{\equal{#2}{.}}{\def\@v@nt{\relax}}{\def\@v@nt{#2}}}%
195     \ifthenelse{\equal{#3}{.}}{\def\@pr@s{\relax}}{\def\@pr@s{#3}}}
```

il n'en est rien en mode en ligne mais il faut tenir compte du délimiteur fantôme donné par le point.

Et, pour finir, on compose le texte attendu.

```
196     \@v@nt #4 \@pr@s}}
```

Avec `\EncloreExtensible`, je définis maintenant plusieurs macros usuelles.

```
197 \newcommandx{\parent}[2][1=1]{\EncloreExtensible[#1]{()}{#2}}
198 \newcommandx{\accol}[2][1=1]{\EncloreExtensible[#1]{\}{\}{#2}}
199 \newcommandx{\crochet}[2][1=1]{\EncloreExtensible[#1]{[]}{#2}}
200 \newcommandx{\varabs}[2][1=1]{%
201   \EncloreExtensible[#1]{\lvert}{\rvert}{#2}}
202 \newcommandx{\norme}[2][1=1]{\EncloreExtensible[#1]{\lVert}{\rVert}{#2}}
203 \newcommand{\nuplet}[2][1]{\parent[#1]{\TdSMnuplet{#2}}}
204 \newcommand{\anuplet}[2][1]{\accol[#1]{\TdSMnuplet{#2}}}
```

`\TdSMReculParenthese`

```
205 \newcommand\TdSMReculParenthese{-2}
```

`\rnuplet`

```
206 \newcommandx\rnuplet[3][1=1,3=\TdSMReculParenthese]{%
207   \mspace{#3mu}\nuplet[#1]{#2}}
```

## 4.4 Vecteurs, bases et repères

`\definirvecteur`

```
208 \newcommand{\definirvecteur}[4][***]{%
209   \ifthenelse{\equal{#1}{***}}{%
210     {\@ifundefined{vect#2}%
211       {\expandafter\def\csname vect#2\endcsname{%
212         \TdSM@fairevecteur{#2}{#3}{#4}\xspace}}%
213       {\PackageError{tdsfrmath}%
214         {Erreur il y a un vecteur de nom << #2 >>}%
215         {Vous pouvez utiliser la macro << redefinivecteur >>
216           \MessageBreak
217           si c'est bien le nom que vous vouliez}}}%
218     {\@ifundefined{vect#1}%
219       {\expandafter\def\csname vect#1\endcsname{%
220         \TdSM@fairevecteur{#2}{#3}{#4}\xspace}}%
221       {\PackageError{tdsfrmath}%
222         {Erreur il y a un vecteur de nom << #1 >>}%
223         {Vous pouvez utiliser la macro << redefinivecteur >>
224           \MessageBreak
225           si c'est bien le nom que vous vouliez}}}}
226 \newcommand{\redefinirvecteur}[4][***]{%
227   \ifthenelse{\equal{#1}{***}}{%
228     {\@ifundefined{vect#2}%
229       {\PackageError{tdsfrmath}%
230         {Erreur il n'y a pas de vecteur de nom << #2 >>}%
231         {Vous pouvez utiliser la macro << definivecteur >>
```

```

232     \MessageBreak
233     si c'est bien le nom que vous vouliez}}
234     {\expandafter\let\csname vect#1\endcsname=\relax%
235     \definirvecteur[#1]{#2}{#3}{#4}}}%
236     {\@ifundefined{vect#1}%
237     {\PackageError{tdsfrmath}%
238     {Erreur il n'y a pas de vecteur de nom << #1 >>}}%
239     {Vous pouvez utiliser la macro << definirvecteur >>
240     \MessageBreak
241     si c'est bien le nom que vous vouliez}}
242     {\expandafter\let\csname vect#1\endcsname=\relax%
243     \definirvecteur[#1]{#2}{#3}{#4}}}}

244 \definirvecteur[i]{\imath}{-1}{3}
245 \definirvecteur[j]{\jmath}{0}{5}
246 \definirvecteur{k}{-1}{1}
247 \definirvecteur{u}{0}{3}
248 \definirvecteur{v}{0}{3}

```

La macro suivante fait le travail de composition.

```

249 \newcommand\TdSM@fairevecteur[3]{%
250   \ensuremath{\overrightarrow{\mspace{#2mu}{#1}\mspace{#3mu}}}}

```

`\vecteur` J'utilise `suffix.sty` pour définir la commande `\vecteur*` après avoir défini `\vecteur`.

`\vecteur` et `\vecteur*` bénéficient eux aussi de `\newcommandx` qui permet l'utilisation de plusieurs arguments optionnels.

```

251 \newcommandx\vecteur[3][1=1,3=5]{\TdSM@fairevecteur{#2}{#1}{#3}}

```

Nous voyons que le code suivant est placé sous l'égide de PARMENTIER : réutilisation des restes ;-)

```

252 \WithSuffix\newcommandx\vecteur*[3][1=1,3=5]{\vecteur{\text{#2}}}

```

`\V`

```

253 \newcommand{\V}{\vecteur}

```

La macro suivante gère les erreurs dans la macro d'après. C'est en français puisque cette extension est au bon goût de chez nous ;-). La seule difficulté c'est de ne pas mettre de lettre accentuée dans le message sinon ça donne des hyéroglyphes — enfin, presque — lors de l'affichage tant dans le fichier `.log` qu'à l'écran. Petit exercice de style!

```

254 \newcommand\TdSM@ErreurArgBase{%
255   \PackageError{tdsfrmath}{Argument optionnel hors limites}{%
256     L'argument optionnel vaut 2 par défaut\MessageBreak
257     mais ne prend que les valeurs 1, 2 ou 3.}}

```

`\base` La macro `\base` admet un seul argument, s'il est nul ou s'il est plus grand que 3 la macro se plaint à l'aide de la macro précédente.

```

258 \newcommand{\base}[1][2]{%
259   \ifcase #1\TdSM@ErreurArgBase\or
260   \nuplet{\vecti}\or
261   \nuplet{\vecti}{ \vectj}\or
262   \nuplet{\vecti}{ \vectj}{ \vectk}\or
263   \TdSM@ErreurArgBase\fi\xspace}

```

`\repere` Reprend en le modifiant le code de `\base`.

```
264 \newcommandx{\repere}[2][1=2,2=0]{%
265   \ifcase #1\TdSM@ErreurArgBase\or
266   \nuplet{#2 \vecti}\or
267   \nuplet{#2 \vecti{} \vectj}\or
268   \nuplet{#2 \vecti{} \vectj{} \vectk}\or
269   \TdSM@ErreurArgBase\fi\xspace}
```

Viennent maintenant des macros qui servent essentiellement d'abréviations. Comme je ne sais quel sera le codage choisi pour le document final, je place les accents à la  $\text{\TeX}$ , ce qui assure la *portabilité*.

```
270 \newcommand{\rog}{rep\`ere orthogonal \repere}
271 \newcommand{\ron}{rep\`ere orthonormal \repere}
272 \newcommand{\rond}{rep\`ere orthonormal direct \repere}
273 \newcommand{\repcom}{\nuplet{0 \vectu{} \vectv}\xspace}
274 \newcommand{\roncom}{rep\`ere orthonormal \repcom}
275 \newcommand{\rondcom}{rep\`ere orthonormal direct \repcom}
```

`\Repere` Pour la forme étoilée je reprends `\WithSuffix`.

```
276 \newcommand{\Repere}{\nuplet{0 I J}}
277 \WithSuffix\newcommand\Repere*{\nuplet{\text{0} \text{I} \text{J}}}

278 \newcommand{\Rog}{rep\`ere orthogonal \Repere}
279 \newcommand{\Ron}{rep\`ere orthonormal \Repere}
280 \newcommand{\Rond}{rep\`ere orthonormal direct \Repere}
```

## 4.5 L'exponentielle

`\E` On peut penser que un `\DeclareMathOperator` pourrait suffire ici mais avec la définition choisie on peut utiliser la même macro dans le texte autant qu'en mode mathématique, sans passer explicitement en mode math, cela me semble suffisant pour justifier l'effort de codage — si tant est que ce soit un effort ;-)

```
281 \newcommand*\E{\ensuremath{\textup{e}}\xspace}
```

`\eu` et la fonction

```
282 \newcommand*\eu[1]{\ensuremath{E^{#1}}}
```

## 4.6 Le nombre i

`\I` Même opération qu'avec `\E`.

```
283 \newcommand\I{\ensuremath{\textup{i}}\xspace}
```

## 4.7 Intégrales

Je commence par définir des registres de dimensions mathématiques — dont on ne peut pas dire qu'on en abuse dans les différentes extensions de  $\text{\LaTeX}$  — pour régler des distances à l'intérieur des intégrales. Leurs noms me semblent assez parlants.

```
284 \newmuskip\TdSM@reculintegrande
285 \newmuskip\TdSM@avancedx
286 \TdSM@reculintegrande=6mu
287 \TdSM@avancedx=4mu
```

Je donne à l'utilisateur de quoi les modifier globalement.

```
288 \newcommand\FixeReculIntegrande[1]{\TdSM@reculintegrande=#1mu}
289 \newcommand\FixeAvanceDx[1]{\TdSM@advancedx=#1mu}
```

Je fournis la macro `\D` avec un `\providecommand` cf. page 11 pour la raison.

```
290 \providecommand*\D{\textup{d}}
```

`\intgen` Vient ensuite la macro la plus générale.

```
291 \newcommandx{\intgen}[5][1=1,2=\the\TdSM@reculintegrande]{%
292   \ensuremath{%
293     \ifnum #1=0\displaystyle\fi
294     \int_{#3}^{#4}\mspace{-#2}{#5}}
```

`\integrer`

```
295 \newcommandx{\integrer}[7]%
296 [1=1,2=\the\TdSM@reculintegrande,7=\the\TdSM@advancedx]{%
297   \ensuremath{%
298     \ifnum #1=0\displaystyle\fi
299     \int_{#3}^{#4}\mspace{-#2}{#5}\mspace{#7}\D{#6}}}
```

```
300 \newcommandx{\integrale}[7]%
301 [1=1,2=\the\TdSM@reculintegrande,7=\the\TdSM@advancedx]{%
302   \integrer[#1][#2]{#3}{#4}{#5(#6)}{#6}[#7]}
303 \newcommandx{\intabfx}[3]%
304 [1=1,2=\the\TdSM@reculintegrande,3=\the\TdSM@advancedx]{%
305   \integrale[#1][#2]{a}{b}{f}{x}[#3]\xspace}
```

## 4.8 Au bonheur du mathématicien, bazar

```
306 \newcommand{\plusinf}{\ensuremath{+\infty}\xspace}
307 \newcommand{\moinsinf}{\ensuremath{-\infty}\xspace}
```

La macro qui suit est une macro de service, elle place le texte à l'intérieur des délimiteurs des intervalles et assimilés.

```
308 \newcommand{\TdSM@dedans}[3]{\mspace{#1mu}\TdSMnuplet{#2}\mspace{#3mu}}
```

Je m'en sers :

```
309 \newcommandx{\interff}[4][1=0,2=1,4=0]%
310   {\crochet[#1]{\TdSM@dedans{#2}{#3}{#4}}}
311 \newcommandx{\interoo}[4][1=0,2=1,4=0]{%
312   \EncloreExtensible[#1]{[]}{[]}{\TdSM@dedans{#2}{#3}{#4}}}
313 \newcommandx{\interof}[4][1=0,2=1,4=0]{%
314   \EncloreExtensible[#1]{[]}{[]}{\TdSM@dedans{#2}{#3}{#4}}}
315 \newcommandx{\interfo}[4][1=0,2=1,4=0]{%
316   \EncloreExtensible[#1]{[]}{[]}{\TdSM@dedans{#2}{#3}{#4}}}
317 \newcommandx{\intferab}[2][1=0,2=0]{\interff[#1]{a b}[#2]}
```

Et maintenant quelque chose de complètement différent :

```
318 \newcommand{\mdfrac}[2]{\ensuremath{\dfrac{#1}{#2}}}
319 \newcommand{\mfrac}[2]{\ensuremath{\frac{#1}{#2}}}
```

Les fameuses polices :

```
320 \newcommand*\manus[1]{%
321   \ensuremath{\TdSM@MathCalPol{\MakeUppercase{#1}}}}
322 \newcommand*\grastab[1]{%
323   \ensuremath{\TdSM@MathGdTPol{#1}}}
```

Des abréviations qu'elles sont utiles :

```

324 \newcommand*\cnp}[2]{\ensuremath{\binom{#1}{#2}}}
325 \newcommand*\dans{\longrightarrow}
326 \newcommand*\donne{\longmapsto}
327 \newcommand*\vide{\ensuremath{\varnothing}\xspace}
328 \newcommand*\ppq{\ensuremath{\leqslant}}
329 \newcommand*\pgq{\ensuremath{\geqslant}}

et pour écrire les définitions des ensembles
330 \newcommand*\TdSMsepdefens{/}
331 \ifTdSM@SepDefEnsExt
332 \newcommandx*\ensemble}[4][1=3,4=3]{%
333   \accol{#2}%
334   \mspace{#1mu}%
335   \ifthenelse{boolean{TdSM@horstexterequis}}{\middle}{}%
336   \TdSMsepdefens
337   \mspace{#4mu}%
338   {#3}}
339 \else
340 \newcommandx*\ensemble}[4]{%
341   \accol{#2}\mspace{#1mu}\TdSMsepdefens\mspace{#4mu}{#3}}
342 \fi

```

## 4.9 Le fichier `taupe.sto`

Ce qui suit n'est chargé, et donc défini, que si l'on a passé la valeur `true` à la clé `taupe`.

```

343 \newcommandx*\prodscal}[4][1=1,2=1,4=1]{%
344   \EncloreExtensible[#1]{\langle}{\rangle}{\TdSM@dedans{#2}{#3}{#4}}

```

On pourra réutiliser l'astuce : pour redéfinir une macro créée avec des moyens  $\text{\LaTeX}$ iens, il faut d'abord la rendre équivalente — par `\let` — à `\relax`. La redéfinition devient alors possible avec les moyens  $\text{\LaTeX}$ iens classiques.

Ici, cependant, pour me faciliter la tâche, j'écris une macro auxiliaire `\TdSM@ReDeclareMathOperator` qui fait bien ce que je demande avec toutefois un léger désavantage : elle ne vérifie pas que la macro « redéfinie » existe bien au préalable. Du fait de ce fonctionnement pas très canonique, elle restera cachée ;-)

```

345 \newcommand{\TdSM@ReDeclareMathOperator}{%
346   \@ifstar{\@redeclmathop m}{\@redeclmathop o}}
347 \long\def\@redeclmathop#1#2#3{%
348   \let#2=\relax%
349   \DeclareRobustCommand{#2}{\qopname\newmcodes@#1{#3}}

```

et je m'en sers

```

350 \TdSM@ReDeclareMathOperator{\cosh}{ch}
351 \TdSM@ReDeclareMathOperator{\sinh}{sh}
352 \TdSM@ReDeclareMathOperator{\tanh}{th}
353 \TdSM@ReDeclareMathOperator{\cot}{cotan}

et ensuite uniquement si ArgArcMaj vaut true
354 \ifTdSM@ArgArcMaj
355 \TdSM@ReDeclareMathOperator{\arccos}{Arccos}
356 \TdSM@ReDeclareMathOperator{\arcsin}{Arcsin}
357 \TdSM@ReDeclareMathOperator{\arctan}{Arctan}
358 \fi

```

et j'ajoute de nouvelles définitions.

```

359 \ifTdsM@ArgArcMaj
360 \DeclareMathOperator{\argch}{Argch}
361 \DeclareMathOperator{\argsh}{Argsh}
362 \DeclareMathOperator{\argth}{Argth}
363 \else
364 \DeclareMathOperator{\argch}{argch}
365 \DeclareMathOperator{\argsh}{argsh}
366 \DeclareMathOperator{\argth}{argth}
367 \fi

368 \DeclareMathOperator{\Ker}{Ker}
369 \DeclareMathOperator{\Img}{Im}

370 \newcommand*\tendversen[1]{\xrightarrow[#1]\relax}
371 \newcommand*\devlim[2][0]{\ensuremath{\text{DL}_\{#2\}\parent{#1}}}
372 \newcommand*\parties[2][-2]{%
373   \ensuremath{\manus{p}\mspace{#1mu}\parent{#2}}}
374 \newcommand{\drv}[2]{\ensuremath{\frac{D#1}{D#2}}}
375 \newcommand{\ddrv}[2]{\ensuremath{\dfrac{D#1}{D#2}}}
376 \newcommandx\interent[4][1=1,2=2,4=2]{%
377   \EncloreExtensible[#1]{\llbracket}{\rrbracket}%
378     {\TdsM@dedans{#2}{#3}{#4}}}
379 \newcommandx\interzn[3][1=1,2=2,3=2]{\interent{0 n}\xspace}

```

## 4.10 Dérivées partielles

`\TdsMDerPartSepar` Cette macro contient ce qui sépare, p. ex., un  $\partial x^2$  du  $\partial y$  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`.

```

380 \newcommand\TdsMDerPartSepar{\,}

```

`\derpart` Je vais maintenant tenter d'expliquer clairement la construction de la macro `\derpart`.

Attention : je n'ai aucune prétention à fournir ici un code optimisé. Je l'ai testé, il fait ce que je veux qu'il fasse et « sam suffi ».

Remarque : ce code est basé une fois encore sur celui fourni par Manuel PÉGOURIÉ-GONNARD pour traiter les noms de fichiers.

Pour commencer, on notera que la définition de `\derpart` est entourée d'une double paire de parenthèses `{\dots}`. Cela assure que les macros définies à l'intérieur resteront inconnues au niveau du document et que les compteurs utilisés retrouveront leur état antérieur au sortir de la macro. Bref on utilise la capacité de  $\TeX$  à « localiser » définitions et modifications de compteurs et macros.

J'utilise `\count@` et `\count0` parce qu'ils existent déjà et que ça m'évite d'en créer deux exprès. On pourra se reporter à la documentation sur le noyau de  $\LaTeX 2_\epsilon$ , [source2e.pdf](#) disponible avec toutes les bonnes distributions de  $\TeX$ .

`\count0` contient le nombre de lettres identiques successives trouvées à l'itération considérée. `\count@` vaut à la fin le nombre total de caractères contenus dans la chaîne constituant le 2<sup>e</sup> argument de la macro `\derpart`.

La macro `\TdsM@sentinelle` a la valeur `@@` et la garde tout le temps. La macro `\TdsM@precedent` commence l'aventure avec la même valeur. Cela permet de tester le début de la lecture du 2<sup>e</sup> argument que j'appellerai désormais *la chaîne*.

Au début du jeu, les macros `\TdSM@DenomAux` et `\TdSM@Denom` sont vides. À la fin `\TdSM@Denom` contient le dénominateur de la dérivée partielle, passé en 2<sup>e</sup> argument de `\frac`. La macro `\TdSM@DenomAux`, quant à elle, permet de construire `\TdSM@Denom` en me facilitant la tâche à la fin de la chaîne.

```
381 \newcommand\derpart[2]{%
382   \count@=1
383   \def\TdSM@sentinelle{@@}%
384   \def\TdSM@precedent{@@}%
385   \def\TdSM@DenomAux{}%
386   \def\TdSM@Denom{}}%
```

`\TdSM@Puissance` place un exposant — il viendra derrière le caractère — si cet exposant est supérieur à 1.

```
387   \def\TdSM@Puissance{%
388     \ifnum\count@>1 ~{\the\count@}\fi}%
```

`\TdSM@FaireDenom` sert à créer le dénominateur. Comme sa définition est donnée à l'intérieur d'une définition — celle de `\derpart` — il faut doubler les *dièses* # dénotant les arguments. J'utilise `\edef` pour forcer le développement immédiat de `\TdSM@Denom` dans la définition même de `\TdSM@Denom`. Supposons, pour faire simple, que `\TdSM@Denom` contienne `tralala` au moment où `TEX` arrive sur `\TdSM@FaireDenom{truc}`. Après l'exécution de cette commande, `\TdSM@Denom` se développe en `tralala \mathchar"140 truc`.

En utilisant un simple `\def` à la place du `\edef`, on partait dans une de ces boucles infinies dont `TEX` se sort en invoquant la finitude de sa mémoire.

```
389   \def\TdSM@FaireDenom##1{%
390     \edef\TdSM@Denom{\TdSM@Denom \partial ##1}}
```

Voici maintenant la macro `\TdSM@derpartaux` qui fait la plus grosse partie du boulot.

```
391   \def\TdSM@derpartaux##1##2{%
392     \ifx\@nil##2%
```

Voici ce qui arrive lorsque le 2<sup>e</sup> argument vaut `\@nil` c.-à-d. lorsque l'on arrive à la fin de la chaîne : on place la valeur du 1<sup>er</sup> argument `##1` dans `\TdSM@actuel`.

```
393     \def\TdSM@actuel{##1}%
```

On regarde s'il a la même valeur que le précédent, placé dans `\TdSM@precedent`.

```
394     \ifx\TdSM@actuel\TdSM@precedent
```

Si oui, on incrémente le compteur 0 et on doit tenir compte de la puissance

```
395         \advance\count0 by 1
396         \TdSM@FaireDenom{\TdSM@precedent\TdSM@Puissance}%
397     \else
```

sinon on regarde si la valeur précédente est égale à la sentinelle

```
398     \ifx\TdSM@precedent\TdSM@sentinelle
```

auquel cas `\TdSM@actuel` contient l'unique lettre de la chaîne et on peut écrire le dénominateur.

```
399         \TdSM@FaireDenom{\TdSM@actuel}%
400     \else
```

Si ce n'est pas le cas il faut placer la lettre précédente avec la bonne puissance puis placer l'actuelle :

```
401         \TdSM@FaireDenom{%
```

```

402         \TdSM@precedent\TdSM@Puissance
403         \TdSM@DerPartSepar\partial\TdSM@actuel}%
404     \fi
405 \fi

Dans le cas où le 2e argument ne vaut pas \@nil c'est que l'on n'est pas encore
au bout.
406 \else
407 \def\TdSM@actuel{##1}%

On tient compte du cas où le \TdSM@precedent est égal à la sentinelle, auquel cas
on met le compteur à 0.
408 \ifx\TdSM@precedent\TdSM@sentinelle
409 \count0=1
410 \else

Sinon, soit la lettre actuelle est égale à la précédente et on incrémente le compteur 0
411 \ifx\TdSM@actuel\TdSM@precedent
412 \advance\count0 by 1
413 \else

soit elle ne l'est pas et on place dans le dénominateur la lettre précédente à la
bonne puissance et on remet le compteur 0 à zéro.
414 \TdSM@FaireDenom{%
415 \TdSM@precedent\TdSM@Puissance\TdSM@DerPartSepar}%
416 \count0=1
417 \fi
418 \fi

Enfin, on incrémente le compteur @ général, on place la lettre actuelle dans
\TdSM@precedent et on appelle la macro \TdSM@derpart@continue sur le 2e argu-
ment. L'astuce ici est que le \fi ferme le premier \if et que, dans le même
temps, il délimite l'argument de la macro \TdSM@derpart@continue
419 \advance\count@ by 1
420 \let\TdSM@precedent\TdSM@actuel%
421 \TdSM@derpart@continue##2%
422 \fi}%

comme on le voit dans sa définition qui consiste à replacer un \fi et rappeler
\TdSM@derpartaux sur ce qui suit :
423 \def\TdSM@derpart@continue##1\fi{\fi\TdSM@derpartaux##1}%

Maintenant que \TdSM@derpartaux est définie, on s'en sert sur le 2e argument de
\derpart en plaçant le \@nil qui signale la fin de la chaîne puis on compose la
fraction en assurant le mode mathématique.
424 \TdSM@derpartaux#2\@nil%
425 \ensuremath{\frac%
426 {\partial\ifnum\count@>1^{\the\count@}\fi #1}%
427 {\TdSM@Denom}}%
428 }}

429 \ProvidesFile{suite.sto}%
430 [\filedate\space\fileversion\space Pour tdsfrmath -- option suite]

431 \ifTdSM@suitedeco
432 \newcommandx\suite[2][1=\N,2=u]{\ensuremath{\parent{{#2}_n}_{#1}}}
433 \newcommandx\suitar[6][1=\N,2=u,4=0,6={}]{}%

```

```

434 \suite[#1][#2]#6 la suite arithm\’etique de raison %
435 $r = #3$ et de premier terme $#2_{#4}=#5$}
436 \newcommandx\suitgeo[6][1=\N,2=u,4=0,6={}]{%
437 \suite*[#1][#2]#6 la suite g\’eom\’etrique de raison %
438 $q = #3$ et de premier terme $#2_{#4}=#5$}
439 \WithSuffix\newcommand\suite*[1][u]{\parent{#1}_{n}}
440 \WithSuffix\newcommandx\suitar*[5][1=u,3=0,5={}]{%
441 \suite*[#1]#5 la suite arithm\’etique de raison %
442 $r = #2$ et de premier terme $#1_{#3}=#4$}
443 \WithSuffix\newcommandx\suitgeo*[5][1=u,3=0,5={}]{%
444 \suite*[#1]#5 la suite g\’eom\’etrique de raison %
445 $q = #2$ et de premier terme $#1_{#3}=#4$}
446 \else
447 \newcommand\suite[1][u]{\parent{#1}_{n}}
448 \newcommandx\suitar[5][1=u,3=0,5={}]{%
449 \suite[#1]#5 la suite arithm\’etique de raison %
450 $r = #2$ et de premier terme $#1_{#3}=#4$}
451 \newcommandx\suitgeo[5][1=u,3=0,5={}]{%
452 \suite[#1]#5 la suite g\’eom\’etrique de raison %
453 $q = #2$ et de premier terme $#1_{#3}=#4$}
454 \WithSuffix\newcommandx\suite*[2][1=\N,2=u]{%
455 \ensuremath{\suite[#2]_{#1}}}
456 \WithSuffix\newcommandx\suitar*[6][1=\N,2=u,4=0,6={}]{%
457 \suite*[#1][#2]#6 la suite arithm\’etique de raison %
458 $r = #3$ et de premier terme $#2_{#4}=#5$}
459 \WithSuffix\newcommandx\suitgeo*[6][1=\N,2=u,4=0,6={}]{%
460 \suite*[#1][#2]#6 la suite g\’eom\’etrique de raison %
461 $q = #3$ et de premier terme $#2_{#4}=#5$}
462 \fi

```

Et ici se termine l’extension tdsfrmath.  
 Scripsit TdS.

## Liste des tableaux

1	<a href="#">Macros redéfinies dans <code>taupe.sto</code></a>	14
2	<a href="#">Macros dont l’aspect dépend de la clé <code>ArgArcMaj</code> — aspect par défaut</a>	14
3	<a href="#">les clés de <code>tdsfrmath.sty</code></a>	17

## Changements

v1  
 Général : 1<sup>re</sup> version publique. . . . 1

## Index

Les nombres en italique ou en bleu renvoient à la page où l’entrée est décrite; les nombres soulignés renvoient à la ligne de code de la définition; les nombres en caractères romains renvoient à la ligne de code où l’entrée est utilisée.

**Symbols**            `\@redeclmathop` 346, 347    `\@v@nt` . . . . 193, 194, 196  
`\@pr@s` . . . . 193, 195, 196



<code>\suitar</code>	433, 440, 448, 456	<code>\TdsM@derpartaux</code>	.. 350–353, 355–357
<code>\suite</code>	432, 434, 437, 439, 441, 444, 447, 449, 452, 454, 455, 457, 460	....	391, 423, 424
suite (CLÉ)	.....	<code>\TdsM@ErreurArgBase</code>	..... 254, 259, 263, 265, 269
suitedeco (CLÉ)	.....	<code>\TdsM@FaireDenom</code>	389, 396, 399, 401, 414
<code>\suitgeo</code>	436, 443, 451, 459	<code>\TdsM@fairevecteur</code>	.. 212, 220, 249, 251
<b>T</b>			
<code>\tanh</code>	.....	<code>\TdsM@gdtcmd</code>	..... .. 99, 101, 104, 108
taupe (CLÉ)	.....	<code>\TdsM@gdtoptn</code>	... 96, 98
<code>\TdsM@actuel</code>	..... .. 393, 394, 399, 403, 407, 411, 420	<code>\TdsM@gdtpack</code>	..... .. 94, 97, 98, 105, 109
<code>\TdsM@advancedx</code>	..... ..... 285, 287, 289, 296, 301, 304	<code>\TdsM@MathCalPol</code>	.. 78, 91, 321
<code>\TdsM@calcmd</code>	78, 81, 85	<code>\TdsM@MathGdTPol</code>	.. 100, 101, 112, 323
<code>\TdsM@caloptn</code>	... 75, 77	<code>\TdsM@nuplet</code>	..... ..... 173, 174, 181
<code>\TdsM@calpack</code>	..... .. 74, 76, 77, 82, 86	<code>\TdsM@nupletaux</code>	179, 180
<code>\TdsM@DecoChoix</code>	... .. 13, 15, 18, 19, 114, 116, 118, 120, 122, 124, 127	<code>\TdsM@PlaceSigne</code>	20, 129, 130, 136, 137
<code>\TdsM@Decoration</code>	.. ..... 13, 115, 117, 119, 121, 123, 125, 139, 141	<code>\TdsM@PlaSiChoix</code>	20, 22, 25, 26, 128, 131
<code>\TdsM@dedans</code>	..... .. 308, 310, 312, 314, 316, 344, 378	<code>\TdsM@precedent</code>	... ..... 384, 394, 396, 398, 402, 408, 411, 415, 420
<code>\TdsM@Denom</code>	386, 390, 427	<code>\TdsM@Puissance</code>	... .. 387, 396, 402, 415
<code>\TdsM@DenomAux</code>	... 385	<code>\TdsM@reculintegrande</code>	.. 284, 286, 288, 291, 296, 301, 304
<code>\TdsM@derpart@continue</code>	..... 421, 423	<code>\TdsM@ReDeclareMathOperator</code>	..... 345,
		<code>\TdsM@sentinelle</code>	.. 383, 398, 408
		<code>\TdsM@separateur</code>	3, 178
		<code>\TdsM@DerPartSepar</code>	.. 380, 403, 415
		<code>\TdsMnuplet</code>	..... .. 173, 203, 204, 308
		<code>\TdsM@ReculParenthese</code>	..... 148, 150, 152, 154, 156, 171, 205, 206
		<code>\TdsMsepdefens</code>	..... 330, 336, 341
		<code>\tendversen</code>	..... 370
		true [ArgArcMaj]	4, 14, 29
		true [SepDefEnsExt]	.. 7
		true [suite]	..... 15
		true [sitedeco]	..... 16
		true [taupe]	..... .. 8, 13, 15, 17, 29
<b>V</b>			
		<code>\V</code>	..... 253
		<code>\val</code>	..... 3, 10, 12
		<code>\varabs</code>	..... 200
		<code>\vecteur</code>	..... 251, 253
		<code>\vecti</code>	260–262, 266–268
		<code>\vectj</code>	261, 262, 267, 268
		<code>\vectk</code>	..... 262, 268
		<code>\vectu</code>	..... 273
		<code>\vectv</code>	..... 273
		<code>\vide</code>	..... 327
		virgule [nupletsep]	..... 5
<b>Z</b>			
		<code>\Z</code>	..... 149, 158