XLOP v 0.21

Manuel de l'utilisateur

Jean-Côme Charpentier
28 avril 2005

Table des matières

Chapitre 1

Présentation

L'extension xlop a été développée pour permettre de réaliser automatiquement des calculs arithmétiques sur des nombres de taille quelconque et d'afficher les résultats sous forme posée ou en ligne. Voici un premier exemple permettant de donner un aperçu de la syntaxe de base :

$$\begin{array}{rr} & {\scriptstyle 1} \\ & 4\,5{.}0\,5 \\ + & 7\,8{.}4 \\ \hline & 1\,2\,3{.}4\,5 \end{array}$$

source
```
\opadd{45,05}{78,4}
```

Ce premier exemple appelle quelques commentaires qui permettront de donner une idée sur la façon de manipuler l'extension.

L'addition est posée « comme à l'école » : il s'agit de la présentation par défaut. On a un alignement sur la virgule des opérandes et du résultat, le symbole opératoire placé à gauche est centrée verticalement entre les deux opérandes et le séparateur décimal est un point malgré le fait d'avoir indiquer les opérandes avec une virgule dans l'appel de la macro. Enfin, on notera la présence d'une retenue au dessus de la première opérande.

Mis à part l'alignement sur la virgule qui est obligatoire pour l'addition posée, toutes les autres caractéristiques décrites ci-dessus sont paramétrables. Certaines macros de l'extension et, en tout cas, toutes les macros affichant les opérations arithmétiques admettent un argument optionnel qui permettra de contrôler la présentation. Pour cela, on utilisera une syntaxe « à la keyval » : on spécifie une suite de modifications de paramètres par une liste d'affectations séparées par des virgules. Une affectation a une des deux syntaxes possibles ci-dessous :

```
<paramètres>=<valeur>
<paramètre>
```

la seconde possibilité étant en fait un raccourci pour :

```
<paramètre>=true
```

Dans cette liste d'affectation, on peut faire suivre les virgules par un ou plusieurs espaces mais il ne faut pas mettre d'espace de part et d'autre

du signe égal ou avant la virgule : un paramètre ou une valeur pouvant (potentiellement) comporter le caractère espace.

Ainsi, si l'on veut un séparateur décimal qui soit une virgule, un symbole opératoire placé en face de la seconde opérande et en supprimant la présence des retenues, il suffit d'indiquer :

```
    4 5,0 5
+   7 8,4
-----------
  1 2 3,4 5
```

source

```
\opadd[decimalsepsymbol={,},
       voperator=bottom,
       carryadd=false]{45.05}{78.4}
```

On notera la petite astuce consistant à mettre la virgule entre accolades dans la définition du symbole du séparateur décimal. En effet, la syntaxe :

source

```
\opadd[decimalsepsymbol=,,voperator=bottom,
       carryadd=false]{45.05}{78.4}
```

est fautive : xlop ne comprenant plus trop bien ce qu'est cette « liste » !

Un autre point important quoique moins visible, est que les chiffres sont disposés à des emplacements très précis. Chaque chiffre est placé dans une boîte de largeur et de hauteur fixes (paramétrables), le séparateur décimal est, par défaut, placé dans une boîte de largeur nulle et toutes les lignes sont régulièrement espacées qu'il y ait ou non un trait horizontal. Cela permet d'obtenir des alignements rigoureux et permet également de placer ce que l'on veut à l'emplacement que l'on veut.

```
    1 <------- retenue
    4 5.0 5
+   7 8.4
-----------
  1 2 3.4 5
```

source

```
\psset{xunit=\opcolumnwidth,
       yunit=\oplineheight}
\opadd{45.05}{78.4}
\oplput(1.5,3){retenue}
\psline{->}(1,3.15)(-3.25,3.15)
```

Cet exemple a été réalisé en utilisant l'extension pstricks

Nous avons dit précédemment que xlop était capable de manipuler des nombres de taille quelconque. Nous reviendrons plus en détail sur cette possibilité et nous nous contenterons ici de ne donner qu'un exemple de ce que cela peut offrir. Ne regardez pas trop le code, les explications seront données plus loin dans ce manuel, pour l'instant, admirez seulement le résultat !

source

```
\opdiv[style=text,period]{1}{49}
```

$$1 \div 49 = 0.\underline{0204081632653061224489795918367346938775 51}\ldots$$

L'extension xlop offre quelques autres fonctionnalités. Il est ainsi possible de manipuler les nombres par l'intermédiaire de variables, ces variables pouvant être créées par une assignation simple ou bien comme résultat d'un calcul. On peut également manipuler les chiffres de façon individuelle :

Le premier chiffre après la virgule de $45.05+78.4$ est un 4.

source
```
\opadd*{45.05}{78.4}{r}%
Le premier chiffre après la virgule de
$45.05+78.4$ est un
\opgetdecimaldigit{r}{1}{d}%
$\opprint{d}$.
```

effectuer des tests :

La somme 45.05+78.4 est strictement supérieure à 100.

source
```
\opadd*{45.05}{78.4}{r}%
La somme $45.05+78.4$ est
\opcmp{r}{100}%
\ifopgt strictement supérieure
\else\ifoplt strictement inférieure
\else égale
\fi\fi
à $100$.
```

avoir accès à quelques opérations ou fonctions :

Le pgcd de 182 et 442 est 26

source
```
Le pgcd de $182$ et $442$ est
\opgcd{182}{442}{d}$\opprint{d}$
```

pouvoir réaliser des calculs complexes sous forme infixe :

$$\frac{2+3^2}{\gcd(22,33)} = 1$$

source
```
\opexpr{(2+3^2)/(gcd(22,33))}{r}%
$$\frac{2+3^2}{\gcd(22,33)} =
  \opprint{r}$$
```

Chapitre 2

Instruction xlop

À quelques exceptions près qui seront étudiées en temps voulu, les macros de xlop peuvent éventuellement avoir un argument optionnel entre crochets pour modifier localement la valeur des paramètres de fonctionnement, les autres arguments (obligatoires) étant des nombres. Les deux sections de ce chapitre décrivent en détail ce qu'est un nombre pour xlop et comment se servir de ses paramètres.

2.1 Au début était le nombre

2.1.1 Taille

Avant de voir la syntaxe générale d'un nombre, nous allons nous pencher sur la particularité de xlop qui est de pouvoir manipuler des nombres de taille quelconque.

Pour être tout à fait précis, la taille théorique maximum d'un nombre est de $2^{31}-1$ chiffres. En pratique, cette limite ne pourra pas être atteinte pour deux raisons essentielles. La première est qu'une multiplication avec deux opérandes ayant 2^{25} chiffres demanderait plus de 7 000 années de calcul sur l'ordinateur de l'auteur ! La seconde est beaucoup plus restrictive car elle est liée aux limites de taille des piles de TEX. Voici un tableau indiquant une compilation sous TEX avec une multiplication de deux opérandes de même taille sur une machine Linux, pentium II 600 et 256 Mo de RAM :

nombre de chiffres	100	200	300	400	425	450
temps de compilation (s)	2	8	18	32	36	crash

Le « crash » indiqué dans le tableau est dû au débordement de la table de hachage (hash table). Sous LATEX, la limite avant crash sera plus réduite. D'autre part, ces tests ont été effectués sur un fichier minimum : avec un document source classique, cette limite sera un peu plus basse. Une autre limite qui risque d'être atteinte relativement rapidement est la taille du spouleur (spool size). Pour composer ce document qui contient un grand nombre d'appel aux macros de xlop, l'auteur a augmenté la taille du

spouleur à 250 000 (125 000 s'étant révélé insuffisante) en éditant la ligne `pool_size` du fichier `texmf.cnf`. De même, la table de hachage a dû être augmentée en stipulant la valeur `1000` au niveau de la ligne `hash_extra` du fichier `texmf.cnf`.

2.1.2 Syntaxe

Nous allons présenter cette syntaxe avec la grammaire BNF mais des explications plus humaines suivront :

```
⟨nombre⟩ := {⟨signe⟩}⟨nombre_positif⟩ | ⟨nom⟩
⟨signe⟩ := + | -
⟨nombre_positif⟩ := ⟨entier⟩ | ⟨sep⟩⟨entier⟩ |
    ⟨entier⟩⟨sep⟩ | ⟨entier⟩⟨sep⟩⟨entier⟩
⟨sep⟩ := . | ,
⟨entier⟩ := ⟨chiffre⟩{⟨chiffre⟩}
⟨nom⟩ := ⟨début⟩{caractère}
⟨début⟩ := caractère autre que ⟨signe⟩, ⟨sep⟩
    et ⟨chiffre⟩
```

Le symbole `caractère` désigne presque n'importe quel caractère accepté par TEX. Les seules exceptions sont les caractères `%`, et `#` qui sont totalement interdits. En fait, les caractères actifs risquent de poser des problèmes. Par exemple, la définition de `~` sous LATEX empêche ce caractère de pouvoir faire partie d'un nom. D'autre part, le caractère `\` conserve sont rôle de caractère d'échappement, c'est-à-dire que le nom sera celui obtenu après développement de la macro introduite. Il n'y a aucune autre contrainte comme le montre le code suivant :

source

```
\newcommand\prefix{a/b}
\opadd*{2}{2}{a/b_{^c}!&$}
\opprint{\prefix_{^c}!&$}
```

4

On notera en particulier que `a/b_{^c}!&$` et `\prefix_{^c}!&$` représentent exactement le même nom, si `\prefix` a la définition adéquate évidemment. Cette possibilité d'obtenir un nom en utilisant des macros peut sembler inutile mais il n'en est rien. On peut ainsi réaliser des boucles avec des noms tels que `r1`, `r2`, ..., `r<n>` en utilisant le code `r\the\cpt` comme nom où `cpt` est un compteur (au sens TEX, le mécanisme des compteurs avec LATEX empêche d'être aussi flexible). Nous verrons un exemple d'utilisation de cette forme à la section B.2 page 42.

En pratique, que signifient toutes ces règles ? Elles indiquent d'abord qu'un nombre écrit sous forme décimale peut être précédé par n'importe quelle séquence de signes plus et moins. Évidemment, le nombre sera négatif lorsqu'il y aura un nombre impair de signes moins. Ensuite, un nombre ne peut comporter qu'un seul séparateur décimal qui peut être le point ou la virgule, celui-ci pouvant être placé n'importe où dans le nombre. Enfin, l'écriture d'un nombre s'effectue obligatoirement en

base 10. Attention : ces règles signifient également que `-a` n'est pas valide.

L'extension utilise quelques noms de façon interne et il est plus prudent de ne pas commencer un nom de variable par le caractère `@`.

2.2 Paramètres de **xlop**

Les affectations de paramètres restent locales à la macro lorsqu'elles sont indiquées au niveau de son argument optionnel. Pour rendre de telles affectations globales, il faut utiliser la macro `\opset`. Par exemple

source
```
\opset{decimalsepsymbol={,}}
```

fera que le symbole du séparateur décimal sera la virgule pour tout le reste du document, ou, du moins, jusqu'à une prochaine redéfinition par la macro `\opset`. Dans ce manuel, ce sera le cas à partir de maintenant.

2.2.1 Symboles

Le paramètre `afterperiodsymbol` indique le symbole qui suit l'écriture d'un quotient en ligne lors d'une division avec recherche de période. Sa valeur par défaut est `$\ldots$`

Le paramètre `equalsymbol` indique le symbole utilisé pour l'égalité. Sa valeur par défaut est `$=$`. En réalité, le paramètre est défini avec :

source
```
\opset{equalsymbol={$=$}}
```

c'est-à-dire avec des accolades pour protéger le signe égal. Sans les accolades, il y aurait une erreur à la compilation. On doit procéder de cette façon lorsque la valeur comporte le signe égal ou une virgule

Le paramètre `approxsymbol` indique le symbole utilisé pour les approximations. Sa valeur par défaut est `$\approx$`.

Le paramètre `decimalsepsymbol` indique le symbole utilisé pour le séparateur décimal. Sa valeur par défaut est le point.

Les paramètres `addsymbol`, `subsymbol`, `mulsymbol` et `divsymbol` indiquent les symboles utilisés pour les quatre opérations arithmétiques. Les valeurs par défaut sont respectivement `$+$`, `$-$`, `$\times$` et `$\div$`.

2.2.2 Présentation générale

Le paramètre `voperation` indique la façon dont une opération posée sera placée par rapport à la ligne de base. Les valeurs possibles sont `top`, `center` et `bottom` (valeur par défaut).

top
$$\begin{array}{r} {\scriptstyle 1} \\ 4\,5 \\ +\;1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

center
$$\begin{array}{r} {\scriptstyle 1} \\ 4\,5 \\ +\;1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

bottom
$$\begin{array}{r} {\scriptstyle 1} \\ 4\,5 \\ +\;1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

source
```
top\quad
\opadd[voperation=top]{45}{172}\par
center\quad
\opadd[voperation=center]{45}{172}\par
bottom\quad
\opadd[voperation=bottom]{45}{172}
```

Le paramètre `voperator` indique comment sera placé le symbole opératoire par rapport aux opérandes. Les valeurs possibles sont `top`, `center` (valeur par défaut) et `bottom`.

top
$$\begin{array}{r} {\scriptstyle 1} \\ +\;4\,5 \\ 1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

center
$$\begin{array}{r} {\scriptstyle 1} \\ 4\,5 \\ +\;1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

bottom
$$\begin{array}{r} {\scriptstyle 1} \\ 4\,5 \\ +\,1\,7\,2 \\ \hline 2\,1\,7 \end{array}$$

source
```
top\quad
\opadd[voperator=top]{45}{172}\par
center\quad
\opadd[voperator=center]{45}{172}\par
bottom\quad
\opadd[voperator=bottom]{45}{172}
```

Le paramètre `deletezero` indique si certains nombres d'une opération doivent être affichés avec ou sans les zéros non significatifs. Le rôle exact de ce paramètre varie en fonction de l'opération et nous y reviendrons lors de la présentation des différentes opérations.

Le paramètre `style` indique si l'opération doit être posée (valeur `display` qui est la valeur par défaut) ou bien être affichée en ligne (valeur `text`). On reviendra sur ce paramètre lors de la présentation de la division car les possibilités sont alors un peu plus nombreuses.

$45 + 172 = 217$

source
```
\opadd[style=text]{45}{172}
```

Dans les opérations en ligne, xlop fait attention à ne pas composer la formule en mode mathématique de façon directe. Cela permet de spécifier ce que l'on veut comme dans l'exemple qui suit et c'est également pour cela qu'il faut indiquer les valeurs classiques des symboles entre délimiteurs mathématiques.

42 plus 172 égal 214

source
```
\opadd[addsymbol=plus,
       equalsymbol=\'egal,
       style=text]{42}{172}
```

Cependant, xlop introduit exactement les mêmes pénalités de coupures et exactement les mêmes espacements que pour une formule mathématique.

Le paramètre `parenthesisnegative` indique comment composer les nombres négatifs dans les opérations en ligne. Les valeurs possibles sont :

- `none` qui compose les nombres négatifs sans parenthèse ;
- `all` qui compose les nombres négatifs en les plaçant entre parenthèses ;
- `last` qui compose les nombres négatifs en les plaçant entre parenthèses s'il ne s'agit pas de la première opérande.

$-12 + -23 = -35$
$(-12) + (-23) = (-35)$
$-12 + (-23) = -35$

source
```
\opadd[style=text,
       parenthesisnegative=none]
       {-12}{-23}\par
\opadd[style=text,
       parenthesisnegative=all]
       {-12}{-23}\par
\opadd[style=text,
       parenthesisnegative=last]
       {-12}{-23}
```

2.2.3 Dimensions

Dans les opérations posées, les chiffres sont placés dans des boîtes de dimensions fixées. La largeur est donnée par le paramètre `columnwidth` et la hauteur par le paramètre `lineheight`. La valeur par défaut de `lineheight` est `\baselineskip` ce qui fait que les lignes des opérations seront espacées, par défaut, comme les lignes d'un paragraphe. La valeur par défaut de `columnwidth` est de `2ex` car la largeur « normale » des chiffres aurait donné des résultats peu lisibles.

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 45{,}89 \\ +\,127{,}5 \\ \hline 173{,}39 \end{array}$$

source
```
\opadd[columnwidth=0.5em]
      {45.89}{127.5}
```

Ce piètre résultat est dû en partie au fait que la virgule est placé dans une boîte dont la largeur est contrôlée par le paramètre `decimalsepwidth` dont la valeur par défaut est nulle. Un essai d'amélioration peut être effectué en donnant à ce paramètre la largeur » normale « d'une virgule.

$$\begin{array}{r} {\scriptstyle 1\,1} \\ 45{,}89 \\ +\,127{,}5 \\ \hline 173{,}39 \end{array}$$

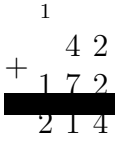
source
```
\opadd[columnwidth=0.5em,
       decimalsepwidth=0.27778em]
      {45.89}{127.5}
```

C'est meilleur mais le fait de donner une largeur non nulle à la boîte contenant le séparateur décimal risque de poser des difficultés si l'on veut placer des éléments externes : cela va à l'encontre de l'idée de placer les chiffres dans une grille fixe. Ceci est donc à éviter en temps normal.

Les deux paramètres `columnwidth` et `lineheight` correspondent aux deux seules dimensions que l'extension rend publiques, à savoir respectivement `\opcolumnwidth` et `\oplineheight`. Il est cependant dangereux de vouloir modifier ces dimensions de façon directe puisque une modification par voie normale n'a pas pour seule conséquence d'obtenir une nouvelle valeur pour ces dimensions. L'extension xlop a rendu ces dimensions publiques uniquement pour pouvoir les lire, pas pour les modifier.

Les deux paramètres suivants permettent de spécifier les largeurs des traits horizontaux et verticaux tracés par xlop. Il s'agit des paramètres `hrulewidth` et `vrulewidth` dont la valeur par défaut est `0.4pt`.

Ces traits sont composés sans perturber la grille, c'est-à-dire sans ajouter d'espace vertical. Ainsi, avec des valeurs importantes pour l'épaisseur, les traits risquent de déborder au niveau des opérandes.



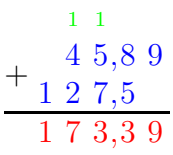
source
```
\opadd[hrulewidth=8pt]{42}{172}
```

Il existe également un paramètre permettant de contrôler le décalage horizontal du séparateur décimal. Il s'agit de `decimalsepoffset` dont la valeur par défaut est égale à `-0.35`. Cette valeur par défaut indique une longueur en prenant `\opcolumnwidth` comme unité. Un exemple d'utilisation de ce paramètre sera donné à la section 3.4 page 17.

2.2.4 Styles des chiffres

L'extension xlop distingue cinq types de nombres et y associe cinq paramètres de style :

- les opérandes avec `operandstyle` ;
- le résultat avec `resultstyle` ;
- les restes avec `remainderstyle` ;
- les résultats intermédiaires avec `intermediarystyle` ;
- les retenues avec `carrystyle`.



source
```
\opadd[operandstyle=\blue,
       resultstyle=\red,
       carrystyle=\scriptsize\green]
       {45.89}{127.5}
```

Rappelons que dans ce manuel, nous utilisons l'extension pstricks.

En réalité, la gestion de ces styles est encore plus puissante car on peut distinguer les différents nombres d'une même classe. Dans une même

opération, on a plusieurs opérandes et, éventuellement plusieurs restes et plusieurs nombres intermédiaires. On accède alors au style de ces nombres individuels en indexant le paramètre de style correspondant.

```
    1 1
    4 5,8 9
+ 1 2 7,5
-----------
  1 7 3,3 9
```

source
```
\opadd[operandstyle=\blue,
       operandstyle.1=\lightgray,
       resultstyle=\red,
       carrystyle=\scriptsize\green]
       {45.89}{127.5}
```

Sur cet exemple, on a indiqué que la première opérande devait être composée avec le style `\lightgray` et comme rien n'était indiqué pour la seconde opérande, celle-ci a repris le style de base de sa classe (donc a été composée avec le style `\blue`).

Ce mécanisme va encore plus loin car on peut indicer à deux niveaux les styles des classes opérande, reste et nombre intermédiaire et à un niveau les styles des classes résultat et retenue pour accéder au style de chacun des chiffres de ces nombres. Afin de rendre les indexations plus facile à gérer, un index positif indiquera le rang d'un chiffre de la partie entière (numérotée de droite à gauche, l'index 1 correspondant au chiffre des unités) et un index négatif indiquera le rang d'un chiffre de la partie décimale (numérotée de gauche à droite, l'index -1 correspondant au chiffre des dixièmes).

```
    1 1
  0 4  ,8
+   2 7,5 0
-----------
  1   3,3 9
```

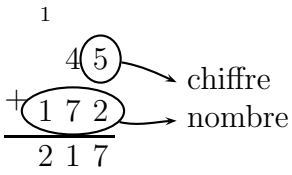
source
```
\opadd[operandstyle.1.1=\white,
       operandstyle.1.-2=\white,
       operandstyle.2.3=\white,
       resultstyle.2=\white,
       deletezero=false]
       {045.89}{127.50}
```

On peut également utiliser une macro à un paramètre comme style.

```
    1 1
    4 •,8 •
+ • 2 7,5
-----------
  1 • 3,3 9
```

source
```
\newcommand\hole[1]{$\bullet$}
\opadd[operandstyle.1.1=\hole,
       operandstyle.1.-2=\hole,
       operandstyle.2.3=\hole,
       resultstyle.2=\hole]
       {45.89}{127.5}
```

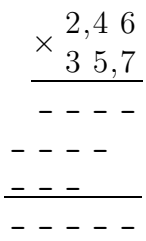
Lorsque le style est une macro, le chiffre constitue le dernier argument de cette macro. Voici un exemple plus compliqué et utilisant l'extension pst-node de la suite pstricks :



source

```
\newcommand\OPoval[3]{%
  \dimen1=#2\opcolumnwidth
  \ovalnode{#1}
    {\kern\dimen1 #3\kern\dimen1}}
\opadd[voperation=top,
  operandstyle.1.1=\OPoval{A}{0},
  operandstyle.2.2=\OPoval{C}{0.8}]
  {45}{172}\qquad
\begin{minipage}[t]{2cm}
  \pnode(0,0.2em){B}\ chiffre
  \ncarc{->}{A}{B}\par
  \pnode(0,0.2em){D}\ nombre
  \ncarc{<-}{D}{C}
\end{minipage}
```

Comme les chiffres, le séparateur décimal tient compte du style d'un nombre. Pour accéder au style du séparateur décimal de façon individuelle, il faut employer l'indice `d` au lieu des indices numériques des chiffres.



source

```
\newcommand\hole[1]{\texttt{\_}}
\opmul[intermediarystyle=\hole,
  resultstyle=\hole,
  resultstyle.d=\white]{2.46}{35.7}
```

Chapitre 3

Opérations arithmétiques

3.1 Addition

L'addition est gérée par la macro \opadd. L'addition, lorsqu'elle est posée, n'affiche que des nombres positifs. Cela va avoir comme conséquence d'afficher une soustraction lorsqu'une des opérandes est négative.

$$\begin{array}{r} -\ 2\ 4\ 5 \\ \quad 7\ 2 \\ \hline -\ 1\ 7\ 3 \end{array}$$

source
```
\opadd{-245}{72}
```

De façon générale, le principe est de poser l'opération qui permet de retrouver le résultat comme on le ferait à la main. En revanche, l'affichage en ligne donnera toujours une addition puisqu'on peut maintenant écrire des nombres négatifs.

$-245 + 72 = -173$

source
```
\opadd[style=text]{-245}{72}
```

Outre les paramètres généraux décrits à la section 2.2, la macro \opadd est sensible aux paramètres carryadd, lastcarry et deletezero.

Le paramètre carryadd est un paramètre booléen c'est-à-dire n'acceptant que les valeurs true et false. Comme il d'usage, l'omission du signe égal et de la partie droite de l'affectation équivaut à =true. Ce paramètre indique si les retenues doivent être ou non affichées. Sa valeur par défaut est true.

Le paramètre lastcarry est également un paramètre booléen. Il indique si une retenue sans chiffre correspondant au niveau des deux opérandes doit être ou non affichée. Sa valeur par défaut est false. On fera attention au rôle exact de ce paramètre. Ainsi, si la seconde opérande dans l'exemple qui suit avait été 15307, la dernière retenue aurait été affichée quelle que soit la valeur du paramètre lastcarry puisqu'il y aurait eu un chiffre correspondant au niveau de la seconde opérande.

$$\begin{array}{cr} & {\scriptstyle 1\quad 1\quad\ } \\ + & 4825 \\ & 5307 \\ \hline & 10132 \end{array}$$

source
```
\opadd{4825}{5307}
```

$$\begin{array}{cr} + & 4825 \\ & 5307 \\ \hline & 10132 \end{array}$$

source
```
\opadd[carryadd=false]{4825}{5307}
```

$$\begin{array}{cr} & {\scriptstyle 1\ 1\quad\ 1\ } \\ + & 4825 \\ & 5307 \\ \hline & 10132 \end{array}$$

source
```
\opadd[lastcarry]{4825}{5307}
```

Le paramètre `deletezero` est également un paramètre booléen et son rôle et d'indiquer si les zéros non significatifs doivent être supprimés ou non. Sa valeur par défaut est `true`. Si ce paramètre vaut `false`, les opérandes et le résultats auront le même nombre de chiffres, xlop ajoutant des zéros non significatifs pour y parvenir. Les zéros non significatifs des opérandes ne sont pas supprimés également.

$$\begin{array}{cr} & {\scriptstyle \quad 1\ 1\ 1\ } \\ + & 12{,}3427 \\ & 5{,}2773 \\ \hline & 17{,}62 \end{array}$$

$$\begin{array}{cr} & {\scriptstyle \quad\ 1\ 1\ 1\ } \\ + & 012{,}3427 \\ & 005{,}2773 \\ \hline & 017{,}6200 \end{array}$$

source
```
\opadd{012.3427}{5.2773}\par
\opadd[deletezero=false]
  {012.3427}{5.2773}
```

Ce paramètre a exactement le même rôle dans l'affichage en ligne que dans l'affichage posé.

$2{,}8 + 1{,}2 = 4$
$02{,}8 + 1{,}2 = 04{,}0$

source
```
\opadd[style=text]{02.8}{1.2}\par
\opadd[style=text,
       deletezero=false]{02.8}{1.2}\par
```

3.2 Soustraction

La soustraction est gérée par la macro `\opsub`. La soustraction, lorsqu'elle est posée n'affiche que des nombres positifs. Cela va avoir pour conséquence d'affficher une addition lorsqu'une des opérandes est négative.

$$\begin{array}{r} {\scriptstyle 1} \\ 2\ 4\ 5 \\ +\quad 7\ 2 \\ \hline 3\ 1\ 7 \end{array}$$

source
```
\opsub{-245}{72}
```

De façon générale, le principe est de poser l'opération qui permet de retrouver le résultat comme on le ferait à la main. En revanche, l'affichage en ligne donnera toujours une soustraction puisqu'on peut maintenant écrire des nombres négatifs.

$-245 - 72 = -317$

source
```
\opsub[style=text]{-245}{72}
```

Ce principe s'applique également lorsque la première opérande est inférieure à la seconde (cas positif) où on aura une inversion des opérandes.

$$\begin{array}{r} 2{,}4\ 5 \\ -\ 1{,}2 \\ \hline 1{,}2\ 5 \end{array}$$

source
```
\opsub{1.2}{2.45}
```

Bien entendu, l'opération en ligne donnera le résultat exact.

$1{,}2 - 2{,}45 = -1{,}25$

source
```
\opsub[style=text]{1.2}{2.45}
```

Outre les paramètres généraux vus à la section 2.2, `\opsub` est sensible aux paramètres `carrysub`, `lastcarry`, `offsetcarry`, `deletezero` et `behaviorsub`.

Le paramètre `carrysub` est un paramètre booléen qui indique si les retenues doivent être ou non présentes. Sa valeur par défaut est `false` (rappelons que le paramètre `carryadd` avait une valeur par défaut égale à `true`).

$$\begin{array}{r} 1\ 2{\scriptstyle 1}3{\scriptstyle 1}4 \\ -\quad {\scriptstyle 1}5{\scriptstyle 1}6\ 7 \\ \hline 6\ 6\ 7 \end{array}$$

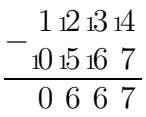
source
```
\opsub[carrysub]{1234}{567}
```

Dans l'exemple précédent, nous pouvons voir qu'il manque en fait une retenue au niveau du dernier chiffre de 1234, cette façon de faire étant assez courante. Néanmoins, on peut contrôler l'affichage de cette dernière retenue avec le paramètre `lastcarry`. Ce paramètre n'a pas tout à fait le même rôle que pour l'addition puisque la dernière retenue ne sera pas affichée dans le cas où la seconde opérande n'a pas de chiffre correspondant (alors que pour l'addition, il fallait que les deux opérandes n'aient pas de chiffres correspondant).

$$\begin{array}{r} 1{\scriptstyle 1}2{\scriptstyle 1}3{\scriptstyle 1}4 \\ -\ 1\quad {\scriptstyle 1}5{\scriptstyle 1}6\ 7 \\ \hline 6\ 6\ 7 \end{array}$$

source
```
\opsub[carrysub,lastcarry]{1234}{567}
```

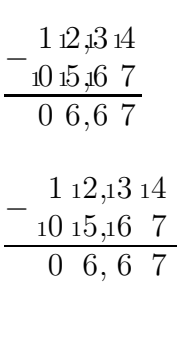
On peut noter dans ce dernier cas qu'il est sans doute préférable de mettre le paramètre `deletezero` à `false` pour obtenir une présentation plus correcte.



source

```
\opsub[carrysub,
       lastcarry,
       deletezero=false]{1234}{567}
```

L'affichage des retenues au niveau des soustractions peut sembler un peu trop compact. On peut élargir la boîte des chiffres avec le paramètre `opcolumnwidth` mais également indiquer le décalage des retenues avec le paramètre `offsetcarry`. La valeur par défaut de ce paramètre est `-0.35`.



source

```
\opsub[carrysub,
       lastcarry,
       deletezero=false]{12.34}{5.67}

\bigskip
\opsub[carrysub,
       lastcarry,
       columnwidth=2.5ex,
       offsetcarry=-0.4,
       decimalsepoffset=-3pt,
       deletezero=false]{12.34}{5.67}
```

Il peut arriver qu'une soustraction de deux nombres positifs, le premier étant inférieur au second, soit le signe d'une erreur de l'utilisateur. Dans ce cas, et uniquement dans ce cas, le paramètre `behaviorsub` permet d'obtenir un rappel à l'ordre. Les trois valeurs possibles de ce paramètre sont `silent` qui est la valeur par défaut et qui donne le résultat, `warning` qui donne également le résultat mais affiche le message d'avertissement :

```
xlop warning.  Substraction with first operand less than second one
    See documentation for further information.
```

et enfin `error` qui affichera le message d'erreur :

```
xlop error.  See documentation for further information.
                Type  H <return>  for immediate help.
! Substraction with first operand less than second one.
```

et l'opération ne sera pas effectuée.

3.3 Multiplication

La multiplication est gérée par la macro `\opmul`.

Nous présenterons les paramètres `hfactor`, `displayintermediary`, `shiftintermediarysymbol`, `displayshiftintermediary` et finalement `deletezero`, les autres paramètres ayant été vus à la section 2.2.

Le paramètre `shiftintermediarysymbol` indique quel symbole sera utilisé (sa valeur par défaut est `$\cdot$`) pour visualiser les décalages des

nombres intermédiaires. Le paramètre `displayshiftintermediary` peut prendre les valeurs `shift` (valeur par défaut) qui ne montre ce symbole que lorsque le décalage est supérieur à un rang, `all` qui indique que ce symbole de décalage sera systématiquement affiché et `none` qui indique que ce symbole ne sera jamais affiché.

source

```
\opmul[displayshiftintermediary=shift]{453}{1001205}\qquad
\opmul[displayshiftintermediary=all]{453}{1001205}\qquad
\opmul[displayshiftintermediary=none]{453}{1001205}
```

$$\begin{array}{r} 4\,5\,3 \\ \times\ 1\,0\,0\,1\,2\,0\,5 \\ \hline 2\,2\,6\,5 \\ 9\,0\,6\,\cdot \\ 4\,5\,3 \\ \hline 4\,5\,3\,\cdot\,\cdot \\ \hline 4\,5\,3\,5\,4\,5\,8\,6\,5 \end{array} \qquad \begin{array}{r} 4\,5\,3 \\ \times\ 1\,0\,0\,1\,2\,0\,5 \\ \hline 2\,2\,6\,5 \\ 9\,0\,6\,\cdot\,\cdot \\ 4\,5\,3\,\cdot\,\cdot\,\cdot \\ \hline 4\,5\,3\,\cdot\,\cdot\,\cdot\,\cdot\,\cdot\,\cdot \\ \hline 4\,5\,3\,5\,4\,5\,8\,6\,5 \end{array} \qquad \begin{array}{r} 4\,5\,3 \\ \times\ 1\,0\,0\,1\,2\,0\,5 \\ \hline 2\,2\,6\,5 \\ 9\,0\,6 \\ 4\,5\,3 \\ \hline 4\,5\,3 \\ \hline 4\,5\,3\,5\,4\,5\,8\,6\,5 \end{array}$$

En réalité, le non affichage des nombres intermédiaires nuls est dû à la valeur par défaut `none` du paramètre `displayintermediary`. La valeur `all` va afficher tous les nombres intermédiaires.

$$\begin{array}{r} 4\,5\,3 \\ \times\ 1\,0\,0\,1\,2\,0\,5 \\ \hline 2\,2\,6\,5 \\ 0\,0\,0 \\ 9\,0\,6 \\ 4\,5\,3 \\ 0\,0\,0 \\ 0\,0\,0 \\ 4\,5\,3 \\ \hline 4\,5\,3\,5\,4\,5\,8\,6\,5 \end{array}$$

source

```
\opmul[displayintermediary=all]
      {453}{1001205}
```

On notera que les nombres intermédiaires nuls sont affichés avec autant de chiffres que le premier facteur.

Le paramètre `displayintermediary` admet la valeur `nonzero` qui a le même rôle que la valeur `none` sauf dans le cas où le second facteur ne comporte qu'un seul chiffre.

source

```
\opmul{3.14159}{4}\qquad
\opmul[displayintermediary=nonzero]{3.14159}{4}
```

$$\begin{array}{r} 3{,}1\,4\,1\,5\,9 \\ \times\ 4 \\ \hline 1\,2{,}5\,6\,6\,3\,6 \end{array} \qquad \begin{array}{r} 3{,}1\,4\,1\,5\,9 \\ \times\ 4 \\ \hline 1\,2\,5\,6\,6\,3\,6 \\ \hline 1\,2{,}5\,6\,6\,3\,6 \end{array}$$

Le paramètre `hfactor` permet d'indiquer comment doit se faire l'alignement des opérandes. La valeur par défaut `right` donne une composition au fer à droite tandis que la valeur `decimal` donne un alignement au niveau de la virgule.

source
```
\opmul{3.1416}{12.8}\qquad\opmul[hfactor=decimal]{3.1416}{12.8}
```

$$\begin{array}{r} 3{,}1\,4\,1\,6 \\ \times\quad 1\,2{,}8 \\ \hline 2\,5\,1\,3\,2\,8 \\ 6\,2\,8\,3\,2 \\ 3\,1\,4\,1\,6 \\ \hline 4\,0{,}2\,1\,2\,4\,8 \end{array} \qquad \begin{array}{r} 3{,}1\,4\,1\,6 \\ \times\ 1\,2{,}8 \\ \hline 2\,5\,1\,3\,2\,8 \\ 6\,2\,8\,3\,2 \\ 3\,1\,4\,1\,6 \\ \hline 4\,0{,}2\,1\,2\,4\,8 \end{array}$$

Pour la multiplication posée, le paramètre `deletezero` ne concerne que les opérandes, le résultat gardant ses éventuels zéros non significatifs puisque ceux-ci sont nécessaires pour effectuer correctement le décalage de la virgule lorsqu'on travaille « à la main ».

source
```
\opmul[deletezero=false]{01.44}{25}\qquad
\opmul{01.44}{25}
```

$$\begin{array}{r} 0\,1{,}4\,4 \\ \times\quad 2\,5 \\ \hline 0\,7\,2\,0 \\ 0\,2\,8\,8 \\ \hline 0\,3\,6{,}0\,0 \end{array} \qquad \begin{array}{r} 1{,}4\,4 \\ \times\quad 2\,5 \\ \hline 7\,2\,0 \\ 2\,8\,8 \\ \hline 3\,6 \end{array}$$

En revanche, dans la multiplication en ligne, ce paramètre retrouve son comportement normal.

source
```
\opmul[deletezero=false,style=text]{01.44}{25}\qquad
\opmul[style=text]{01.44}{25}
```

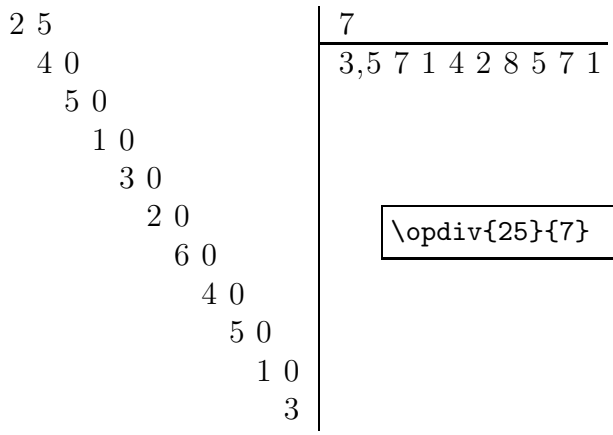
$$01{,}44 \times 25 = 036{,}00 \qquad 1{,}44 \times 25 = 36$$

3.4 Division

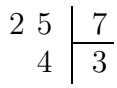
L'extension gère la division « classique » avec la macro `\opdiv` et la division euclidienne avec la macro `\opidiv`. En raison de sa complexité, la division est l'opération qui prend en compte le plus de paramètres.

3.4.1 Contrôle de l'arrêt

Dans ce qui suivra, le terme d'*étape* indique l'ensemble des calculs permettant d'obtenir un chiffre au niveau du quotient. Ce nombre d'étapes est contrôlé en partie par les paramètres `maxdivstep`, `safedivstep` et `period`. En partie seulement car une division classique s'arrêtera automatiquement lors de l'obtention d'un reste nul, quelles que que soient les valeurs de ces trois paramètres et une division euclidienne s'arrêtera sur un quotient entier, sans tenir compte de ces trois paramètres.

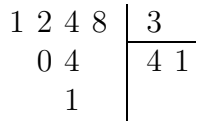


source
```
\opdiv{25}{7}
```



source
```
\opidiv{25}{7}
```

Le premier exemple s'arrête en raison de la valeur de `maxdivstep` qui est égale, par défaut, à 10. On prendra garde à ce que le nombre maximum d'étapes peut entraîner des résultats aberrants lorsqu'il est trop faible.



source
```
\opdiv[maxdivstep=2]{1248}{3}
```

Le résultat précédent est clairement faux mais xlop a fait ce qu'on lui a demandé, en l'occurrence avoir deux chiffres (maximum) au quotient.

L'affichage en ligne va également différer selon que la division s'est arrêtée avec un reste final nul ou non ou selon qu'il s'agit d'une division classique ou euclidienne.

$3{,}14 \div 2 = 1{,}57$
$3{,}14 \div 3 \approx 1{,}046666666$
$314 = 2 \times 157$
$314 = 3 \times 104 + 2$

source
```
\opdiv[style=text]{3.14}{2}\par
\opdiv[style=text]{3.14}{3}\par
\opidiv[style=text]{314}{2}\par
\opidiv[style=text]{314}{3}
```

On notera l'emploi de `equalsymbol` ou `approxsymbol` selon le cas ainsi que l'affichage avec une troncature et non un arrondi. Nous verrons comment obtenir un arrondi à la section 4.5

L'affichage en ligne de `\opdiv` tient compte de `maxdivstep`. Cela signifie que l'on peut obtenir des résultats vraiment faux avec des valeurs trop faible de ce paramètre et, contrairement à l'affichage posé, l'affichage en ligne ne permettra pas de comprendre ce qu'il s'est passé.

$1248 \div 3 \approx 41$

source
```
\opdiv[maxdivstep=2,style=text]
      {1248}{3}
```

Si en plus, le dernier reste calculé est nul, on atteint un summum :

$1208 \div 3 = 4$

source
```
\opdiv[maxdivstep=1,style=text]
      {1208}{3}
```

puisqu'il n'y a même plus d'approximation !

Une division non euclidienne peut également s'arrêter sur la détection de la survenue d'une période. Pour cela, il suffit de donner la valeur `true` au paramètre `period`.

$$\begin{array}{r|l} 1\,0\,0 & 3 \\ \cline{2-2} 1\,0 & 3\,3{,}3 \\ 1\,0 & \\ 1 & \end{array}$$

source
```
\opdiv[period]{100}{3}
```

Pour ne pas avoir à effectuer des comparaisons de chaque reste avec tous les restes précédents, xlop calcule dès le départ la longueur de la période ce qui permet de n'effectuer qu'une seule comparaison à chaque étape et donc d'accélérer notablement les calculs[1]. Malheureusement, ces calculs se font avec des nombres directement accessibles à TEX ce qui a pour conséquence de ne pas pouvoir utiliser d'opérandes dont la valeur absolue excède $\left\lfloor \frac{2^{31}-1}{10} \right\rfloor = 214748364$.

Pour ne pas entraîner des calculs trop long, xlop ne dépassera pas la valeur du paramètre `safedivstep` dans les divisions avec détection de période. Sa valeur par défaut est égale à 50. Cependant, xlop signalera le problème. Par exemple si on demande un tel calcul avec le code :

```
\opdiv[period]{1}{289}
```

on obtiendra le message d'avertissement :

```
xlop warning.  Period of division is too big (272 > safedivstep).
    Division will stop before reach it.
    See documentation for further information.
```

qui indique que la période de cette division est de 272 et qu'elle ne sera donc pas atteinte à cause de la valeur de `safedivstep`.

L'affichage en ligne d'une telle division présente quelque particularités.

[1] Je remercie à cette occasion Olivier Viennet pour ses précisions mathématiques qui ont permis d'implanter correctement ces calculs.

$150 \div 7 = 21{,}\underline{428571}\ldots$

source
```
\opdiv[period,style=text]{150}{7}
```

On obtient donc une égalité au lieu d'une approximation, la présence d'un trait sous la période et des points de suspension à la suite de la période. Tous ces éléments peuvent être configurés. Le symbole d'égalité est donné par le paramètre `equalsymbol` (valeur par défaut `{$=$}`), la largeur du trait par le paramètre `hrulewidth` (valeur par défaut `0.4pt`), sa position verticale par le paramètre `vruleperiod` (valeur par défaut `-0.2`) qui indique un décalage vertical en prenant `\oplineheight` comme unité et les points de suspension sont donnés par le paramètre `afterperiodsymbol` (valeur par défaut `$\ldots$`).

$150 \div 7 \approx 21{,}\overline{428571}$

source
```
\opdiv[period,style=text,
       equalsymbol=$\approx$,
       hrulewidth=0.2pt,
       vruleperiod=0.7,
       afterperiodsymbol=]
      {150}{7}
```

3.4.2 Éléments supplémentaires

Les divisions posées peuvent comporter les soustractions successives permettant le calcul des restes. Pour xlop, les nombres qui sont soustraits sont des nombres intermédiaires donc les différentes façons de représenter les soustractions utiliserons le paramètre `displayintermediary` déjà vu pour la multiplication. La valeur `none` (défaut) n'affichera aucune soustraction, la valeur `all` affichera toutes les soustractions et la valeur `nonzero` affichera les soustractions avec un nombre non nul.

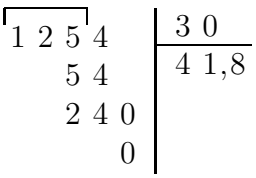
source
```
\opdiv[displayintermediary=none,voperation=top]
      {251}{25}\quad
\opdiv[displayintermediary=nonzero,voperation=top]
      {251}{25}\quad
\opdiv[displayintermediary=all,voperation=top]
      {251}{25}
```

```
2 5 1     | 2 5
  0 1 0 0 | 1 0,0 4
        0 |
```

```
  2 5 1     | 2 5
− 2 5       | 1 0,0 4
  0 1 0 0   |
− 1 0 0     |
      0     |
```

```
  2 5 1     | 2 5
− 2 5       | 1 0,0 4
  0 1       |
− 0         |
  1 0       |
− 0         |
  1 0 0     |
− 1 0 0     |
      0     |
```

Lorsqu'on pose une division, on peut dessiner un « pont » au-dessus de la partie du dividende qui sera prise en compte pour la première étape du calcul. L'extension xlop permet d'afficher ce symbole grâce au paramètre booléen `dividendbridge` (valeur par défaut `false`).

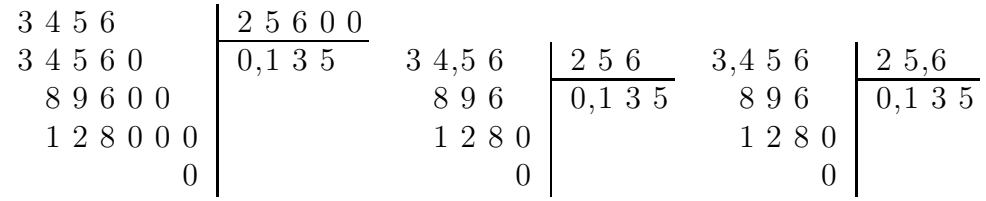


source
```
\opdiv[dividendbridge]{1254}{30}
```

3.4.3 Nombres non entiers et négatifs

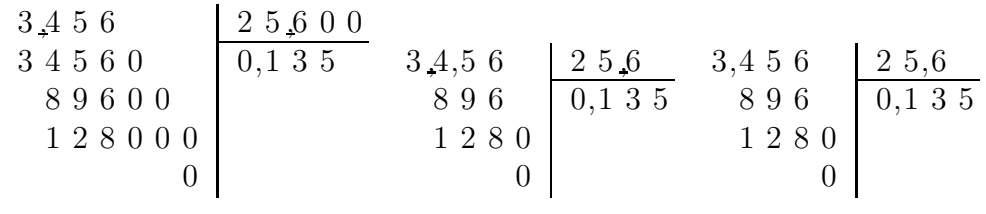
La présentation d'opérandes non entières est gérée par le paramètre `shiftdecimalsep`. Sa valeur par défaut est `both` et indique que le séparateur décimal sera décalé pour obtenir un diviseur et un dividende entiers. La valeur `divisor` indique qu'il y aura le décalage nécessaire pour obtenir un diviseur entier et la valeur `none` indique qu'il n'y aura aucun décalage.

source
```
\opdiv[shiftdecimalsep=both]{3.456}{25.6}\quad
\opdiv[shiftdecimalsep=divisor]{3.456}{25.6}\quad
\opdiv[shiftdecimalsep=none]{3.456}{25.6}
```



Un symbole indiqué par le paramètre `strikedecimalsepsymbol` est réservé pour montrer l'ancien emplacement de la virgule lorsqu'on effectue un décalage. La valeur par défaut de ce paramètre est vide ce qui fait que l'on ne voyait rien sur les exemples précédents.

source
```
\opset{strikedecimalsepsymbol={\rlap{,}\rule[-1pt]{3pt}{0.4pt}}}
\opdiv[shiftdecimalsep=both]{3.456}{25.6}\quad
\opdiv[shiftdecimalsep=divisor]{3.456}{25.6}\quad
\opdiv[shiftdecimalsep=none]{3.456}{25.6}
```



La présence d'un symbole non vide pour le séparateur décimal barré peut laisser les zéros non significatifs au niveau des opérandes.

$$\begin{array}{rl|l} 0{\times}0\ 3{,}4\ 5\ 6 & & 2{\times}5\ 6 \\ \cline{3-3} \quad 8\ 9\ 6 & & 0{,}0\ 1\ 3\ 5 \\ \quad 1\ 2\ 8\ 0 & & \\ \qquad\quad 0 & & \end{array}$$

source

```
\opdiv[shiftdecimalsep=divisor,
       strikedecimalsepsymbol=%
         \hspace{-3pt}\tiny$\times$]
      {0.03456}{2.56}
```

Comme nous l'avons déjà vu, la macro `\opidiv` donne un quotient entier, cela même si les opérandes sont non entières. Il est un peu bizarre de vouloir réaliser une division euclidienne sur des nombres non entiers et la macro `\opidiv` sera assez stricte sur sa présentation. Les paramètres `maxdivstep`, `safedivstep` et `period` seront sans effet ainsi que le paramètre `shiftdecimalsep`, les deux opérandes étant rendues entières.

$$\begin{array}{rl|l} 3\ 4{\times}5\ 7 & & 7{\times}0\ 0 \\ \cline{3-3} \quad 6\ 5\ 7 & & 4 \end{array}$$

source

```
\opidiv[strikedecimalsepsymbol=%
        \hspace{-3pt}\tiny$\times$]
        {34.57}{7}
```

Lorsque les opérandes sont négatives, l'affichage en ligne de `\opidiv` différera des données obtenues avec l'affichage posée. Le reste sera nécessairement un nombre compris entre zéro (inclus) et la valeur absolue du diviseur (exclu).

$124 \div 7 \approx 17{,}71428571$
$124 = 7 \times 17 + 5$
$124 = -7 \times -17 + 5$
$-124 = 7 \times -18 + 2$
$-124 = -7 \times 18 + 2$

source

```
\opdiv[style=text]{124}{7}\par
\opidiv[style=text]{124}{7}\par
\opidiv[style=text]{124}{-7}\par
\opidiv[style=text]{-124}{7}\par
\opidiv[style=text]{-124}{-7}
```

Cette condition sur le reste reste valable même avec un diviseur non entier.

$1{,}24 = 0{,}7 \times 1 + 0{,}54$
$1{,}24 = -0{,}7 \times -1 + 0{,}54$
$-1{,}24 = 0{,}7 \times -2 + 0{,}16$
$-1{,}24 = -0{,}7 \times 2 + 0{,}16$

source

```
\opidiv[style=text]{1.24}{0.7}\par
\opidiv[style=text]{1.24}{-0.7}\par
\opidiv[style=text]{-1.24}{0.7}\par
\opidiv[style=text]{-1.24}{-0.7}
```

Chapitre 4

Autres commandes

4.1 Macros étoilées

Les cinq macros vues au chapitre précédent ont une version étoilée. Ces macros étoilées réalisent le calcul mais ne procèdent à aucun affichage, le résultat étant stocké dans une variable donnée en dernier argument.

Comme ces commandes n'affichent rien, les paramètres ne seront pas acceptés pour les macros \opadd*, \opsub*, \opmul* et \opidiv*. En revanche, les paramètres maxdivestep, safedivstep et period influencent les calculs et la macro \opdiv* acceptera donc un argument optionnel pour pouvoir en tenir compte.

source
```
\opmul*{2}{2}{a}%
\opmul*{a}{a}{a}\opmul*{a}{a}{a}%
\opadd[style=text]{a}{1}
```

$256 + 1 = 257$

Pour les macros \opdiv et \opidiv, il y aura deux arguments supplémentaires pour pouvoir recevoir le quotient et le reste final.

source
```
\opdiv*[maxdivstep=1]{-88}{16}{q}{r}%
\opmul*{q}{16}{bq}%
\opmul[style=text]{16}{q}\par
\opadd[style=text]{bq}{r}
```

$16 \times -5 = -80$
$-80 + -8 = -88$

4.2 Entrées-sorties

La macro \opcopy recopie son premier argument dans son deuxième argument. Le premier argument est donc un nombre écrit sous forme décimale ou via une variable alors que le second sera considérée comme un nom de variable.

La macro \opprint affiche son argument. L'exemple qui suit utilise le compteur \time qui indique le nombre de minutes écoulés depuis minuit.

Il est 2 heures et 55 minutes

source
```
\opidiv*{\the\time}{60}{h}{m}%
Il est \opprint{h}~heures et
\opprint{m}~minutes
```

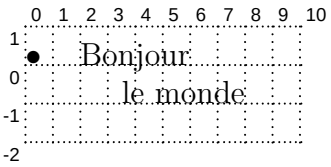
On verra à la section 4.4 comment améliorer cet affichage avec des tests.

La macro \opdisplay affiche également un nombre mais en écrivant chaque chiffre dans une boîte de largeur donnée par `columnwidth` et de hauteur donnée par `lineheight`. Le style est spécifié par le premier argument et cette macro accepte un argument optionnel pour permettre de donner un style particulier aux chiffres individuels.

1 2 **9**,1 **9** 2

source
```
\opdisplay[resultstyle.1=\bfseries,
           resultstyle.-2=\bfseries]
          {resultstyle}{129.192}
```

Les macros \oplput et \oprput permettent de placer un objet à un emplacement déterminé. La syntaxe de ces deux commandes ne suit pas celle des autres macros de xlop puisque l'emplacement est indiqué sous forme de coordonnées entre parenthèses. Les coordonnées utilisent \opcolumnwidth et \oplineheight comme unités ce qui permet à l'utilisateur de construire lui-même ses propres « opérations ».



source
```
\psset{xunit=\opcolumnwidth,
  yunit=\oplineheight}%
\psgrid[subgriddiv=1,gridlabels=7pt,
        griddots=5](0,1)(10,-2)
\oplput(2,0){Bonjour}
\oprput(8,-1){le monde}
$\bullet$
```

Sur l'exemple ci-dessus, on peut voir que ces deux macros ne modifient pas le point de référence. Elles prennent même la précaution d'inhiber l'espace automatique qui les suit pour qu'il ne soit pas nécessaire de mettre un % en fin de ligne.

Les macros \ophline et \opvline complète les deux précédentes pour donner à l'utilisateur les outils lui permettant de construire ses propres opérations. La macro \ophline permet de placer un trait horizontal dont la longueur est donné par le paramètre qui suit les coordonnées. La macro \opvline fait de même avec les traits verticaux. On rappelle que les paramètres `hrulewidth` et `vrulewidth` indiquent les épaisseurs respectives de ces types de traits.

source
```
\par\vspace{2\oplineheight}
\oplput(1,2){O}\oplput(2,2){N}\oplput(3,2){E}
\oplput(0,1.5){$+$}
\oplput(1,1){O}\oplput(2,1){N}\oplput(3,1){E}
\ophline(0,0.8){4}
\oplput(1,0){T}\oplput(2,0){W}\oplput(3,0){O}
```

$$\begin{array}{r} \text{ONE} \\ +\,\text{ONE} \\ \hline \text{TWO} \end{array}$$

4.3 Chiffres d'un nombre

Les macros \opwidth, \opintegerwidth et \opdecimalwidth indiquent respectivement le nombre de chiffres du nombre dans sa totalité, de sa partie entière et de sa partie décimale. Le premier argument est le nombre sur lequel s'effectue le comptage et le second argument indique la variable où sera stocké le résultat.

source

```
\opcopy{123456.1234}{a}%
\opwidth{a}{na}%
\opintegerwidth{a}{ia}%
\opdecimalwidth{a}{da}%
\opprint{a} s'\'ecrit avec \opprint{na}
chiffres (\opprint{ia} en partie
enti\`ere et \opprint{da} en partie
d\'ecimale).
```

123456,1234 s'écrit avec 10 chiffres (6 en partie entière et 4 en partie décimale).

La macro \opunzero permet de supprimer les zéros non significatifs du nombre passé en argument.

source

```
\opcopy{00150.00250}{a}%
Avant : \opprint{a}\par
\opunzero{a}%
Apr\`es : \opprint{a}
```

Avant : 00150,00250
Après : 150,0025

Les macros \integer et \opdecimal donnent respectivement la partie entière et la partie décimale d'un nombre. Le premier argument est le nombre à traiter et le second est la variable qui contiendra le résultat.

source

```
\opcopy{-37.69911}{a}%
\opinteger{a}{ia}%
\opdecimal{a}{da}%
Partie enti\`ere : \opprint{ia}\par
Partie d\'ecimale : \opprint{da}
```

Partie entière : 37
Partie décimale : 69911

Six macros servent à écrire ou lire un chiffre d'un nombre. On peut lire ou écrire un chiffre selon son rang dans le nombre, dans sa partie entière ou dans sa partie décimale. Les chiffres pour le nombre dans sa totalité ou pour sa partie entière sont numérotés de droite à gauche et, pour sa partie décimale de gauche à droite. Ainsi, avec le nombre 1234,56789, le deuxième chiffre est 8, le deuxième chiffre de sa partie entière est 3 et le deuxième chiffre de sa partie décimale est 6. Il est alors facile de deviner le rôle respectif des six macros :

- opgetdigit ;
- opsetdigit ;

– opgetintegerdigit ;
– opsetintegerdigit ;
– opgetdecimaldigit ;
– opsetdecimaldigit ;

La syntaxe est la même pour ces six macros. Le premier argument est le nombre sur lequel doit porter la lecture ou l'écriture, le deuxième argument est l'index donnant la position du chiffre et le troisième argument est le nom de la variable qui contiendra le chiffre lu ou bien ce qui devra être écrit. Si l'index est en dehors du nombre, les macros de lecture donneront 0 comme résultat et les macros d'écriture étendront le nombre pour pouvoir atteindre cet index en créant des zéros dans les nouvelles positions.

4.4 Comparaisons

Lorsqu'on désire concevoir des macros évoluées, il est très souvent utile de pouvoir réaliser des tests. Pour cela, xlop met à disposition la macro \opcmp dont les deux arguments sont des nombres et qui mettra à jour les tests \ifopgt, \ifopge, \ifople, \ifoplt, \ifopeq et \ifopneq pour respectivement indiquer que la première opérande est strictement supérieure, supérieure, inférieure, strictement inférieure, égale ou différente de la seconde opérande.

Pour des raisons techniques, xlop donne des définitions globales aux six tests précédents. Ceux-ci ne seront donc pas protégés par les groupes. Comme ces tests sont utilisés par un grand nombre de macros de xlop, une conséquence pratique est qu'il faut **toujours** réaliser les tests \ifop... immédiatement après le \opcmp, ou, du moins, avant toute autre utilisation de macros de xlop sous peine de bogues éventuels difficiles à comprendre !

On va reprendre la macro d'affichage de l'heure donnée à la section 4.2 mais en vérifiant si l'argument est compris entre 0 (inclus) et 1440 (exclu) puis en réalisant les tests nécessaires pour voir si « heure » doit être ou non au pluriel ainsi que « minute ».

source

```
\newcommand\heure[1]{%
  \opcmp{#1}{0}\ifopge
  \opcmp{#1}{1440}\ifoplt
    \opidiv*{#1}{60}{h}{m}%
    \opprint{h} heure%
    \opcmp{h}{1}\ifopgt
      s%
    \fi
    \opcmp{m}{0}\ifopneq
      \space\opprint{m} minute%
      \opcmp{m}{1}\ifopgt
        s%
```

```
      \fi
    \fi
  \fi\fi
}
\heure{60} -- \heure{1080} -- \heure{1081} -- \heure{1082}
```

1 heure – 18 heures – 18 heures 1 minute – 18 heures 2 minutes

4.5 Opérations évoluées

Les macros qui nous restent à voir proviennent soit de commandes utilisées de façon interne et qu'il aurait été dommage de ne pas rendre publiques, soit de demandes d'utilisateurs.

Les macros utilisées de façon interne sont `\opgcd` qui donne le pgcd de deux nombres et `\opdivperiod` qui donne la longueur de la période d'un quotient de deux nombres. Pour des raisons d'efficacité, ces macros n'utilisent pas les nombres de xlop mais des nombres directement accessibles à TEX. Cela a pour conséquence que les nombres passés en paramètres dans les deux premiers arguments ne devront pas excéder `2147483647` pour `\opgcd` et `214748364` pour `\opdivperiod`. Un message d'avertissement rappellera à l'ordre en cas de dépassement. Le résultat sera stocké dans la variable indiqué en troisième paramètre.

Il y aura également quelques vérifications sur les deux premiers paramètres. Un pgcd ne peut pas avoir d'argument nul et le calcul de la longueur d'une période ne pourra pas se faire avec un quotient nul. D'autre part, si un nombre non entier est passé en paramètre, seule la partie entière sera prise en compte.

$\pgcd(5376,2304) = 768$

source

```
\opcopy{5376}{a}%
\opcopy{2304}{b}%
\opgcd{a}{b}{gcd(ab)}%
\newcommand\pgcd{%
  \mathop{\mathrm{pgcd}}}%
$\pgcd(\opprint{a},\opprint{b}) =
  \opprint{gcd(ab)}$
```

Si vous voulez vous amuser à trouver de grandes périodes de divisions, sans entrer dans les détails mathématiques, les carrés de nombres premiers sont de bons candidats. Par exemple, avec $257^2 = 66049$ on trouve :

$\frac{1}{66049}$ a une période de longueur 65792.

source

```
\opdivperiod{1}{66049}{p}%
$\frac{1}{66049}$ a une p\'eriode
de longueur $\opprint{p}$.
```

Les macros `\opcastingoutnines` et `\opcastingoutelevens` vont permettre de composer des preuves par neuf et par onze. L'extension xlop

ne propose pas directement ces compositions puisqu'elles nécessitent des traits en diagonal et donc le recours à d'autres extensions. En réalité, la macro \opcastingoutnines va faire la somme modulo 9 des chiffres du premier argument et stockera le résultat dans le second argument tandis que la macro \opcastingoutelevens fera la somme des chiffres de rangs impairs, la somme des chiffres de rangs pairs puis la différence modulo 11 de ces deux sommes.

source

```
\newcommand\castingoutnines[3]{%
  \opcastingoutnines{#1}{cna}%
  \opcastingoutnines{#2}{cnb}%
  \opmul*{cna}{cnb}{cna*cnb}
  \opcastingoutnines{cna*cnb}{cna*cnb}%
  \opcastingoutnines{#3}{cn(a*b)}%
  \begin{pspicture}(-3.5ex,-3.5ex)%
                   (3.5ex,3.5ex)
    \psline(-3.5ex,-3.5ex)(3.5ex,3.5ex)
    \psline(-3.5ex,3.5ex)(3.5ex,-3.5ex)
    \rput(-2.75ex,0){\opprint{cna}}
    \rput(2.75ex,0){\opprint{cnb}}
    \rput(0,2.75ex){\opprint{cna*cnb}}
    \rput(0,-2.75ex){\opprint{cn(a*b)}}
  \end{pspicture}
}
\castingoutnines{157}{317}{49669}
```

Incidemment, cet exemple montre que $157 \times 317 \neq 49669$! La réponse correcte est $157 \times 317 = 49769$.

Les deux macros suivantes sont très simples. Il s'agit de \opneg qui calcul l'opposé de son premier argument et le sauvegarde dans la variable indiquée par le second argument et de \opabs qui réalise la même chose mais avec la valeur absolue.

La macro \oppower permet de calculer des puissances entières de nombres. Cette macro demande trois paramètres, le troisième paramètre étant la variable recevant le résultat du premier paramètre à la puissance le deuxième paramètre. Le deuxième paramètre doit être un nombre entier. Lorsque le premier argument est nul, si le deuxième paramètre est nul, le résultat sera 1, s'il est strictement positif, le résultat sera nul et s'il est strictement négatif il y aura une erreur et aucun résultat ne sera fourni. Il n'y a aucune limitation sur le premier paramètre ce qui peut entraîner quelques problèmes. Par exemple :

source

```
\opcopy{0.8}{a}\opcopy{-17}{n}%
\oppower{a}{n}{r}%
$\opprint{a}^{\opprint{n}} = \opprint{r}$
```

$$0{,}8^{-17} = 44{,}408920985006261616945266723632812 5$$

Avec 0,7 au lieu de 0,8, le problème aurait été encore pire :

source
```
\opcopy{0.7}{a}\opcopy{-8}{n}%
\oppower{a}{n}{r}%
\opdecimalwidth{r}{dr}
$\opprint{a}^{\opprint{n}}$ a \opprint{dr} chiffres apr\‘es la
virgule.
```

$0{,}7^{-8}$ a 72 chiffres après la virgule.

Tout cela est dû au fait que lorsque l'exposant est négatif, xlop calcule *d'abord* l'inverse du nombre pour *ensuite* calculer la puissance avec l'opposé de l'exposant. Si on avait laissé -17 au lieu de -8 dans l'exemple précédent, les capacités de TEX auraient été dépassées.

Les trois macros qui suivent permettent de contrôler la précision des nombres manipulés. Elles permettent de construire un nombre en donnant une valeur approchée par défaut, par excès ou un arrondi d'un nombre donné en précisant le rang où devra se faire l'arrondi. Ces macros sont respectivement `\opfloor`, `\opceil` et `\oproun`d. Elles demandent trois paramètres qui seront dans l'ordre le nombre de départ, le rang de l'arrondi et le nom de la variable qui contiendra le résultat.

Le rang est indiqué par une valeur entière donnant le nombre de chiffres après la virgule qui doivent être présents. Si le rang est négatif, l'arrondi se fera avant la virgule. Si le rang positif indique plus de chiffres que la partie décimale n'en a, des zéros seront ajoutés. Si le rang négatif indique plus de chiffres que la partie entière n'en a, l'arrondi restera bloqué pour donner au moins le premier chiffre du nombre.

Voici un tableau récapitulatif pour mieux comprendre le fonctionnement de ces trois macros.

`\op...{3838.3838}{n}{r}`			
n	floor	ceil	round
6	3838,383800	3838,383800	3838,383800
4	3838,3838	3838,3838	3838,3838
3	3838,383	3838,384	3838,384
0	3838	3839	3838
−1	3830	3840	3840
−2	3800	3900	3800
−6	3000	4000	4000

`\op...{-3838.3838}{n}{r}`			
n	floor	ceil	round
6	−3838,383800	−3838,383800	−3838,383800
4	−3838,3838	−3838,3838	−3838,3838
3	−3838,384	−3838,383	−3838,384
0	−3839	−3838	−3838
−1	−3840	−3830	−3840
−2	−3900	−3800	−3800
−6	−4000	−3000	−4000

La dernière macro qui nous reste à voir est \opexpr qui permet de réaliser le calcul d'une expression complexe. Cette macro demande deux paramètres : le premier est l'expression à calculer donnée sous forme infixe (la forme habituelle pour un humain) et le second est le nom de la variable qui contiendra le résultat.

Initialement, la formule devait être donnée sous forme polonaise inverse (la notation des calculatrices HP ou du langage PostScript par exemple) mais un travail commun avec Christophe Jorssen a finalement abouti à la possibilité de donner l'expression sous une forme plus agréable pour l'utilisateur.

Les formules acceptent les opérateurs arithmétiques habituels `+`, `-`, `*` et `/` ainsi que l'opérateur `:` pour la division euclidienne et `^` pour l'exponentiation. L'opérateur `-` a les deux rôles d'opérateur binaire de la soustraction et d'opérateur unaire pour l'opposé. L'opérateur `+` a également les deux rôles d'opérateur binaire de l'addition et d'opérateur unaire ne faisant ... rien ! Les opérandes sont écrites sous forme décimale ou par l'intermédiaire de nom de variable cependant, la macro \opexpr va introduire une petite restriction sur les noms de variables puisque ceux-ci devront être différents des noms de fonctions reconnus par cette macro. Les fonctions accessibles sont :

- `abs(a)` ;
- `ceil(a,i)` ;
- `decimal(a)` ;
- `floor(a,i)` ;
- `gcd(a,b)` ;
- `integer(a)` ;
- `mod(a,b)` qui donne le résultat de `a` modulo `b` ;
- `rest(a,b)` qui donne le reste de la division de `a` par `b` (la différence entre reste et modulo est la différence qui existe entre division non euclidienne et division euclidienne) ;
- `round(a,i)`.

où les fonctions non décrites ci-dessus font appel aux macros correspondantes (la fonction `xxx` faisant appel à la macro `\opxxx`). Pour les fonctions `ceil`, `floor` et `round`, le nombre `i` indique le rang sur lequel doit se faire l'arrondi.

La macro \opexpr accepte un argument optionnel puisqu'elle peut réaliser des divisions et que ces divisions doivent pouvoir être contrôlées via les paramètres `maxdivstep`, `safedivstep` et `period`. Notre premier exemple est assez basique :

source
```
\opexpr{3--gcd(15*17,25*27)*2}{r}%
$3--\gcd(15\times17,25\times27)\times2 = \opprint{r}$
```

$$3--\gcd(15\times17,25\times27)\times2 = 33$$

Voici un autre exemple montrant que des données peuvent provenir d'une macro :

source

```
\newcommand\try{2}%
\opexpr{\try+1/
  (\try+1/
    (\try+1/
      (\try+1/
        (\try+1/
          (\try)))))}{r}
La fraction continue de base $u_n=2$ vaut \opprint{r} au rang~5.
```

La fraction continue de base $u_n = 2$ vaut 2,414285714 au rang 5.

Annexe A

Aide-mémoire

A.1 Temps de compilation

Les temps de compilation ont été mesurés sur une machine à processeur Pentium II 600 MHz ayant 256 Mo de RAM et tournant sous linux (Debian woody). Le principe a été de faire un fichier TeX minimum dont le schéma général est donné par :

```
\input xlop
\count255=0
\loop
\ifnum\count255<1000
  <opération à tester>
 \advance\count255 by1
\repeat
\bye
```

Le temps de compilation avec `<opération à tester>` vide a été soustrait des autres tests et seul le temps utilisateur a été pris en compte. Les résultats sont donnés en millisecondes et sont évidemment à prendre avec beaucoup de précautions.

Le tableau suivant donne le temps de compilation des opérations en millisecondes. Les opérandes utilisées l'ont été avec une écriture sous forme de chiffres mais des essais avec des opérandes sous forme de noms de variable ont montré que les différences étaient vraiment minimes.

La première ligne indique le nombre de chiffres des deux opérandes. Ces nombres ont été construits de la façons suivantes :

- A = 1 et B = 9 pour un chiffre ;
- A = 12 et B = 98 pour deux chiffres ;
- A = 123 et B = 987 pour trois chiffres ;
- A = 12345 et B = 98765 pour cinq chiffres ;
- A = 1234567890 et B = 9876543210 pour dix chiffres ;
- A = 12345678901234567890 et B = 98765432109876543210 pour vingt chiffres ;

Voici les résultats, quelques commentaires suivront :

	1	2	3	5	10	20
`\opadd*{A}{B}{r}`	1.1	1.4	1.6	2.1	3.3	5.8
`\opadd*{B}{A}{r}`	1.1	1.4	1.6	2.1	3.3	5.8
`\opsub*{A}{B}{r}`	1.7	2.1	2.4	3.0	4.8	8.3
`\opsub*{B}{A}{r}`	1.5	1.7	2.0	2.6	4.0	7.0
`\opmul*{A}{B}{r}`	4.6	6.3	8.2	12.8	29.9	87.0
`\opmul*{B}{A}{r}`	5.0	6.6	8.5	13.2	30.3	87.8
`\opdiv*{A}{B}{q}{r}`	46.4	53.8	53.8	64.3	85.8	124.7
`\opdiv*{B}{A}{q}{r}`	12.4	48.9	55.7	58.6	72.8	111.0
`\opdiv*[maxdivstep=5]{A}{B}{q}{r}`	26.8	30.0	32.6	37.6	49.5	73.5
`\opdiv*[maxdivstep=5]{B}{A}{q}{r}`	12.4	29.1	32.6	35.2	43.3	67.9
`\opidiv*{A}{B}{q}{r}`	10.8	12.2	13.5	16.0	22.3	35.5
`\opidiv*{B}{A}{q}{r}`	11.6	13.0	14.2	16.6	23.0	36.7
`\opidiv*{A}{2}{q}{r}`	10.7	12.0	15.3	22.3	42.9	83.0

Il est normal que l'inversion des opérandes n'ait aucune influence pour l'addition. Il peut alors sembler anormal qu'elle en ait une pour la soustraction mais en fait, avoir une seconde opérande supérieure à la première va entraîner quelques instructions supplémentaires (double inversion, gestion plus longue du signe du résultat).

Il est normal que le temps de la division soit supérieur à celui de la multiplication. Il peut alors sembler anormal que la division ait l'air de « rattraper » son retard. En fait, une multiplication voit sa complexité fortement augmenter avec la taille des opérandes mais la division normale est bloquée par le paramètre `maxdivstep`. On le voit bien sur l'exemple où on limite ce maximum à 5 étapes.

Quelques résultats semblent bizarres. Ainsi, `\opdiv*{9}{1}{q}{r}` est anormalement rapide : cela est dû au fait d'avoir un quotient à un seul chiffre. Plus bizarre encore, `\opdiv*{123}{987}{q}{r}` est plutôt rapide. Ici, l'explication est plus subtile : cela est dû à la présence de nombreux zéros au quotient.

Avec des opérandes de taille comparable, la division euclidienne est très rapide par rapport à la division non euclidienne. Cela est dû au fait que le quotient n'aura que peu de chiffres (un seul avec tous les nombres `A` et `B`). La dernière ligne du tableau est plus représentative des temps de compilation que l'on peut obtenir avec cette opération.

Toutes ces remarques sont faites pour bien insister sur la difficulté à évaluer un temps de compilation a priori : il dépend de trop de paramètres. Cela dit, le tableau permet d'avoir quand même une idée de ce à quoi il faut s'attendre.

A.2 Liste des macros

Macro	Description
\opabs{n}{N}	N reçoit la valeur absolue de n.
\opadd[P]{n1}{n2}	Affiche le résultat de l'opération n1+n2.
\opadd*{n1}{n2}{N}	Calcule n1+n2 et place le résultat dans N.
\opcastingoutelevens{n}{N}	Calcule la différence (modulo 11) de la somme des chiffres de rang impair et de la somme des chiffres de rang pair de n et place le résultat dans N.
\opcastingoutnines{n}{N}.	Calcule la somme modulo 9 des chiffres de n et place le résultat dans N.
\opceil{n}{T}{N}	Place dans N la valeur approchée par excès de n au rang T.
\opcmp{n1}{n2}	Compare les nombres n1 et n2 et réalise la mise à jour correspondante des tests \ifopeq, \ifopneq, \ifopgt, \ifopge, \ifople et \ifoplt.
\opcopy{n}{N}	Copie le nombre n dans N.
\opdecimal{n}{N}	Copie la partie décimale (nombre entier positif) de n dans N.
\opdecimalwidth{n}{N}	Le nombre N reçoit la largeur de la partie décimale du nombre n.
\opdisplay[P]{S}{n}	Affiche le nombre n avec le style S en plaçant chaque chiffre dans une boîte de largeur \opcolumnwidth et de hauteur \oplineheight.
\opdiv[P]{n1}{n2}	Affiche le résultat de l'opération n1/n2.
\opdiv*[P]{n1}{n2}{N1}{N2}	Calcule n1/n2, place le quotient dans N1 et le reste dans N2.
\opdivperiod{T1}{T2}{N}	Calcule la longueur de la période de la division de T1 par T2 et place le résultat dans N.
\opexpr[P]{F}{N}	Évalue la formule F est place le résultat final dans le nombre N.
\opfloor{n}{T}{N}	Place dans N la valeur approchée par défaut de n au rang T.
\opgcd{T1}{T2}{N}	Calcule le pgcd de T1 et T2 et place le résultat dans N.
. . . à suivre . . .	

Macro	Description
\opgetdecimaldigit{n}{T}{N}	Construit le nombre N avec le seul chiffre situé en Tième position de la partie décimale du nombre n.
\opgetdigit{n}{T}{N}	Construit le nombre N avec le seul chiffre situé en Tième position du nombre n.
\opgetintegerdigit{n}{T}{N}	Construit le nombre N avec le seul chiffre situé en Tième position de la partie entière du nombre n.
\ophline(T1,T2){T3}	Trace un trait horizontal de longueur T3, d'épaisseur hrulewidth et débutant en (T1,T2) par rapport au point de référence.
\opidiv[P]{n1}{n2}	Affiche le résultat de l'opération n1/n2 (division euclidienne, c'est-à-dire avec un quotient entier).
\opidiv*{n1}{n2}{N1}{N2}	Calcule n1/n2 (division euclidienne), place le quotient (entier) dans N1 et le reste (compris entre 0 inclus et |n2| exclu) dans N2.
\opinteger{n}{N}	Copie la partie entière (nombre entier positif) de n dans N.
\opintegerwidth{n}{N}	Le nombre N reçoit la longueur de la partie entière du nombre n.
\oplput(T1,T2){<objet>}	Place <objet> à droite du point situé en (T1,T2) par rapport au point de référence.
\opmul[P]{n1}{n2}	Affiche le résultat de l'opération n1*n2.
\opmul*{n1}{n2}{N}	Calcule n1*n2 et place le résultat dans N.
\opneg{n}{N}	Le nombre N reçoit l'opposé de n.
\oppower{n}{T}{N}	Calcule n à la puissance T et place le résultat dans N.
\oppprint{n}	Affiche le nombre n de façon directe.
\opround{n}{T}{N}	Place dans N la valeur arrondie de n au rang T.
\oprput(T1,T2){<objet>}	Place <objet> à gauche du point situé en (T1,T2) par rapport au point de référence.
. . . à suivre . . .	

Macro	Description
\opset{L}	Effectue une affectation globale des paramètres de xlop désignés dans la liste L.
\opsetdecimaldigit{n}{T}{N}	Modifie le Tième chiffre de la partie décimale de N pour qu'il soit égal à n.
\opsetdigit{n}{T}{N}	Modifie le Tième chiffre de N pour qu'il soit égal à n.
\opsetintegerdigit{n}{T}{N}	Modifie le Tième chiffre de la partie entière de N pour qu'il soit égal à n.
\opsub[P]{n1}{n2}	Affiche le résultat de l'opération n1-n2.
\opsub*{n1}{n2}{N}	Calcule n1-n2 et place le résultat dans N.
\opunzero{N}	Supprime les zéros non significatifs du nombre N.
\opvline(T1,T2){T3}	Trace un trait vertical de longueur T3, d'épaisseur hrulewidth et débutant en (T1,T2) par rapport au point de référence.
\opwidth{n}{N}	Le nombre N reçoit le nombre de chiffres du nombre n.

Dans ce tableau, les paramètres :
- n et ni (où i représente un indice) indiquent que le paramètre doit être un nombre donné sous forme décimale ou sous forme d'un nom de variable ;
- N et Ni (où i représente un indice) indiquent que le paramètre doit être un nombre donné sous forme d'un nom de variable ;
- [P] indique que la macro accepte un paramètre optionnel permettant de modifier les paramètres de xlop ;
- T et Ti (où i représente un indice) indiquent que le paramètre doit être un nombre donné sous forme décimale ou sous forme d'un nom de variable mais ne devant pas excéder la taille des nombres directement acceptables par TEX (en l'occurrence $-2147483648 \leq$ T ≤ 2147483647).

A.3 Liste des paramètres

Paramètre	Défaut	Signification
afterperiodsymbol	$\ldots$	Symbole utilisé à la suite de la période d'une division.
... à suivre ...		

Paramètre	Défaut	Signification
approxsymbol	$\approx$	Symbole utilisé comme relation d'égalité approximative dans les opérations en ligne.
equalsymbol	{$=$}	Symbole utilisé comme relation d'égalité dans les opérations en ligne.
addsymbol	$+$	Symbole utilisé comme opérateur d'addition.
subsymbol	$-$	Symbole utilisé comme opérateur de soustraction.
mulsymbol	$\times$	Symbole utilisé comme opérateur de multiplication.
divsymbol	$\div$	Symbole utilisé comme opérateur de division pour les opérations en ligne.
decimalsepsymbol	.	Symbole utilisé comme séparateur décimal.
strikedecimalsepsymbol		Symbole utilisé pour un séparateur décimal déplacé au niveau du dividende et du diviseur dans les divisions posées.
shiftintermediarysymbol	$\cdot$	Symbole utilisé pour montrer les décalages des nombres intermédiaires dans les multiplications posées.
displayshiftintermediary	shift	Indique que le caractère de décalage dans les multiplications sera affiché uniquement pour le décalage supplémentaire (valeur `shift`), pour tous les décalages (valeur `all`) ou jamais (valeur `none`).
voperation	bottom	Type d'alignement vertical d'une opération posée. La valeur `bottom` indique que le bas de l'opération sera au niveau de la ligne de base. La valeur `top` indique que la première ligne de l'opération sera sur la ligne de base. La valeur `center` indique que l'opération sera centrée verticalement sur la ligne de base.

. . . à suivre . . .

Paramètre	Défaut	Signification
voperator	center	Positionnement vertical de l'opérateur dans les opérations posées. La valeur `top` place l'opérateur au niveau de la première opérande. La valeur `bottom` place l'opérateur au niveau de la seconde opérande. La valeur `center` place l'opérateur entre les deux opérandes.
hfactor	decimal	Type d'alignement des opérandes dans les multiplications posées. La valeur `decimal` indique un alignement sur la virgule. La valeur `right` indique une composition au fer à droite.
vruleperiod	-0.2	Position verticale du trait indiquant la période du quotient dans les divisions en ligne.
dividendbridge	false	Indique si le « pont » au-dessus du dividende est présent ou non.
shiftdecimalsep	both	Indique la façon dont se fait le décalage de virgule au niveau des opérandes d'une division posée. La valeur `both` indique que le décalage doit rendre le diviseur et le dividende entiers. La valeur `divisor` indique que le décalage doit rendre le diviseur entier. La valeur `none` indique qu'il n'y aura aucun décalage.
maxdivstep	10	Nombre d'étapes maximum d'une division.
safedivstep	50	Nombre d'étapes maximum d'une division lors d'un arrêt sur période.
period	false	Indique si la division doit s'arrêter sur une détection de période ou non.
deletezero	true	Indique si on affiche (`false`) ou on supprime (`true`) les zéros non significatifs.
carryadd	true	Indique si les retenues doivent être affichées pour les additions posées.
carrysub	false	Indique si les retenues doivent être affichées pour les soustractions posées.
offsetcarry	-0.35	Décalage horizontal pour les retenues dans les soustractions posées.
style	display	Indique si l'opération doit être en ligne (`text`) ou posée (`display`).

... à suivre ...

Paramètre	Défaut	Signification
displayintermediary	nonzero	Indique si les résultats intermédiaires doivent être affichés (all) ou seulement les non nuls (nonzero) ou aucun (valeur none) dans les multiplications et les divisions.
lastcarry	false	Indique si la retenue au niveau d'une absence de chiffre devra être affichée ou non.
parenthesisnegative	none	Comportement à adopter pour l'affichage des nombres négatifs dans les opérations en lignes. La valeur none les affiche sans parenthèse, all les mettra systématiquement les nombres négatifs entre parenthèses et last les mettra entre parenthèses sauf pour la première valeur d'une expression.
columnwidth	2ex	Largeur des boîtes contenant un chiffre.
lineheight	\baselineskip	Hauteur des boîtes contenant un chiffre.
decimalsepwidth	0pt	Largeur de la boîte contenant le séparateur décimal.
decimalsepoffset	0pt	Décalage horizontal du séparateur décimal.
hrulewidth	0.4pt	Épaisseur des lignes horizontales.
vrulewidth	0.4pt	Épaisseur des lignes verticales.
behaviorsub	silent	Comportement de xlop face à une soustraction « impossible », c'est-à-dire une soustraction ayant ses deux opérandes positives avec une seconde opérande strictement supérieure à la première. La valeur silent effectuera l'opération en permutant les opérandes de façon silencieuse. La valeur warning fera de même mais en émettant un message d'avertissement. Enfin, la valeur error affichera un message d'erreur et l'opération ne sera pas effectuée.
... à suivre ...		

Paramètre	Défaut	Signification
`country`	`french`	Indique le comportement des opérations posées en fonction du pays. L'extension prévoit pour l'instant `french`, `american` et `russian` mais les différentes façons de présenter les opérations ne sont pas implantées dans la version 0.2.
`operandstyle`		Style utilisé pour les opérandes.
`resultstyle`		Style utilisé pour les résultats.
`remainderstyle`		Style utilisé pour les restes.
`intermediarystyle`		Style utilisé pour les résultats intermédiaires (nombres intermédiaires dans une multiplication et nombres à soustraire dans une division où apparaissent les soustractions successives).
`carrystyle`	`\scriptsize`	Style utilisé pour les retenues. La valeur par défaut lorsqu'on ne compile pas sous LaTeX est `\sevenrm`.

Annexe B

Trucs et astuces

B.1 Comparaison avec calc et fp

On pourrait croire que xlop puisse remplacer avantageusement des extensions telles que calc et fp. En réalité, c'est un peu plus compliqué que cela. Bien sûr, xlop permet des calculs complexes et sur des nombres comportant un nombre arbitraire de chiffres mais, contrairement à calc, il ne permet pas de gérer directement les unités. La comparaison avec fp est un peu plus réaliste mais il ne faut pas perdre de vue que xlop peut devenir gourmand en mémoire et en temps de calcul.

Si vous voulez vraiment réaliser des calculs sur des longueurs, vous pouvez toujours utiliser le fait qu'une affectation d'un registre de longueur à un compteur donnera un nombre correspondant à cette longueur avec l'unité `sp`.

source

```
\newcommand\getsize[2]{%
  \dimen0=#1\relax
  \count255=\dimen0
  \opcopy{\the\count255}{#2}}
\getsize{1pt}{r}$1\,\mathrm{pt} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1pc}{r}$1\,\mathrm{pc} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1in}{r}$1\,\mathrm{in} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1bp}{r}$1\,\mathrm{bp} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1cm}{r}$1\,\mathrm{cm} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1mm}{r}$1\,\mathrm{mm} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1dd}{r}$1\,\mathrm{dd} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1cc}{r}$1\,\mathrm{cc} = \opprint{r}\,\mathrm{sp}$\quad
\getsize{1sp}{r}$1\,\mathrm{sp} = \opprint{r}\,\mathrm{sp}$\quad
```

$1\,\mathrm{pt} = 65536\,\mathrm{sp}$ $1\,\mathrm{pc} = 786432\,\mathrm{sp}$ $1\,\mathrm{in} = 4736286\,\mathrm{sp}$ $1\,\mathrm{bp} = 65781\,\mathrm{sp}$
$1\,\mathrm{cm} = 1864679\,\mathrm{sp}$ $1\,\mathrm{mm} = 186467\,\mathrm{sp}$ $1\,\mathrm{dd} = 70124\,\mathrm{sp}$
$1\,\mathrm{cc} = 841489\,\mathrm{sp}$ $1\,\mathrm{sp} = 1\,\mathrm{sp}$

N'oubliez cependant pas que le but principal de xlop est d'*afficher* automatiquement les opérations.

Muni de la macro \getsize, il est possible de réaliser des calculs sur des longueurs.

La surface d'empagement est de 308,10 cm^2

source

```
\getsize{1cm}{u}%
\getsize{\textwidth}{w}%
\getsize{\textheight}{h}%
\opexpr{w*h/u^2}{S}%
\opround{S}{2}{S}%
La surface d'empagement est de
\opprint{S}\,$\mathrm{cm}^2$
```

B.2 Création d'opérations complexes

L'utilisation des macros de xlop associées au mécanisme de boucle de TEX permet de créer des opérations à volonté. Nous ne donnerons que deux exemples. Le premier est la décomposition d'un nombre en facteurs premiers, le second est un calcul général de fraction continue.

source

```
\newcount\primeindex
\newcount\tryindex
\newif\ifprime
\newif\ifagain
\newcommand\getprime[1]{%
  \opcopy{2}{P0}%
  \opcopy{3}{P1}%
  \opcopy{5}{try}
  \primeindex=2
  \loop
    \ifnum\primeindex<#1\relax
    \testprimality
    \ifprime
      \opcopy{try}{P\the\primeindex}%
      \advance\primeindex by1
    \fi
    \opadd*{try}{2}{try}%
    \ifnum\primeindex<#1\relax
      \testprimality
      \ifprime
        \opcopy{try}{P\the\primeindex}%
        \advance\primeindex by1
      \fi
      \opadd*{try}{4}{try}%
    \fi
  \repeat
}
\newcommand\testprimality{%
  \begingroup
    \againtrue
```

```
    \global\primetrue
    \tryindex=0
    \loop
      \opidiv*{try}{P\the\tryindex}{q}{r}%
      \opcmp{r}{0}%
      \ifopeq \global\primefalse \againfalse \fi
      \opcmp{q}{P\the\tryindex}%
      \ifoplt \againfalse \fi
      \advance\tryindex by1
    \ifagain
    \repeat
  \endgroup
}
```

Avec ce code, on dispose d'un moyen de créer une liste de nombres premiers (en l'occurrence les 20 premiers).

source

```
\getprime{20}%
\opprint{P0}, \opprint{P1}, \ldots,
\opprint{P9}, \ldots \opprint{P19}.
```

2, 3, …, 29, …71.

On notera toutefois que ce code est très mauvais, il est excessivement lent et n'apporte rien par rapport à une approche directe avec des nombres manipulés par les opérations natives de TEX. Ce n'est qu'un exemple pédagogique. On notera également l'astuce permettant d'emboîter deux boucles avec la macro `\testprimality` entièrement mise dans un groupe : s'il est nécessaire de rendre les affectations `\primetrue` et `\primefalse` globales, les opérations de xlop rendent leurs résultats globaux directement.

Une fois que vous avez votre « table » de nombres premiers, vous pouvez vous en servir pour décomposer un nombre en facteurs premiers.

source

```
\newcommand\primedecomp[2][nil]{%
  \begingroup
    \opset{#1}%
    \opcopy{#2}{NbtoDecompose}%
    \opabs{NbtoDecompose}{NbtoDecompose}%
    \opinteger{NbtoDecompose}{NbtoDecompose}%
    \opcmp{NbtoDecompose}{0}%
    \ifopeq
      Je refuse de d\'ecomposer z\'ero.
    \else
      \setbox1=\hbox{\opdisplay{operandstyle.1}%
          {NbtoDecompose}}%
      {\setbox2=\box2{}}%
      \count255=1
      \primeindex=0
      \loop
```

```
      \opcmp{NbtoDecompose}{1}\ifopneq
        \opidiv*{NbtoDecompose}{P\the\primeindex}{q}{r}%
        \opcmp{0}{r}\ifopeq
          \ifvoid2
            \setbox2=\hbox{%
              \opdisplay{intermediarystyle.\the\count255}%
                {P\the\primeindex}}%
          \else
            \setbox2=\vtop{%
              \hbox{\box2}
              \hbox{%
                \opdisplay{intermediarystyle.\the\count255}%
                  {P\the\primeindex}}}
          \fi
          \opcopy{q}{NbtoDecompose}%
          \advance\count255 by1
          \setbox1=\vtop{%
            \hbox{\box1}
            \hbox{%
              \opdisplay{operandstyle.\the\count255}%
                {NbtoDecompose}}
          }%
        \else
          \advance\primeindex by1
        \fi
      \repeat
      \hbox{\box1
        \kern0.5\opcolumnwidth
        \opvline(0,0.75){\the\count255.25}
        \kern0.5\opcolumnwidth
        \box2}%
    \fi
  \endgroup
}

\getprime{20}%
\primedecomp[operandstyle.2=\red,
             intermediarystyle.2=\red]{252}
```

2 5 2	2
1 2 6	2
6 3	3
2 1	3
7	7
1	

Dans ce code, on notera l'emploi d'un groupe entourant l'ensemble de la macro pour protéger les modifications de paramètres de xlop. À ce propos,

on notera également qu'un paramètre vide n'est pas accepté. C'est tout à fait volontaire, l'auteur de l'extension pensant qu'un utilisateur tapant des crochets sans rien mettre à l'intérieur est sans doute en train de commettre une erreur. Pour palier à cette impossibilité de transmettre un paramètre vide il existe le paramètre particulier `nil` qui a exactement ce rôle.

On notera enfin l'astuce `{\setbox2=\box2}` qui permet d'obtenir un registre de boîte vide et les manipulations finales permettant de représenter la barre verticale de façon lisible.

Le second exemple permet de calculer une fraction continue du type :

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$$

en donnant la suite $a_0, a_1, a_2, a_3, \ldots$ à la macro. L'exemple donne les fractions correspondant au nombre d'or et aux racines carrées de 2 et 3.

source

```
\begingroup
\catcode`\:=12
\long\gdef\continuedfraction#1#2{%
  \let\@mirror\relax
  \@for\op@Nb:=#1\do
  {%
    \ifx\@mirror\relax
      \edef\@mirror{\op@Nb}%
    \else
      \edef\@mirror{\op@Nb,\@mirror}%
    \fi
  }%
  \let\Op@result\relax
  \@for\op@Nb:=\@mirror\do
  {%
    \ifx\Op@result\relax
      \opcopy{\op@Nb}{result}%
    \else
      \opexpr{\op@Nb+1/result}{result}%
    \fi
  }%
  \opcopy{result}{#2}%
}
\endgroup
\continuedfraction{1,1,1,1,1,1,1,1,1,1}{r}\opprint{r}\quad
\continuedfraction{1,2,2,2,2,2,2,2,2,2}{r}\opprint{r}\quad
\continuedfraction{1,1,2,1,2,1,2,1,2,1}{r}\opprint{r}
```

1,618181818 1,414213624 1,732057416

Une fois n'est pas coutume, nous avons fait appel à des commandes LaTeX pour effectuer une boucle. Ce manuel étant composé en français, le caractère « : » est actif ce qui empêche la macro `\@for` de fonctionner correctement d'où le groupe où ce caractère reçoit un code de catégorie adéquat.

B.3 Accès direct aux nombres

Lorsqu'on récupère un nombre dans une variable, on peut le traiter de multiples façons à l'aide des macros de xlop. Toutefois, dans certaines situations, on peut souhaiter construire ses propres macros ou utiliser des macros externes en passant un tel nombre en paramètre.

Passer directement `\opprint{var}` est inopérant car cette macro est suffisamment complexe pour induire des effets de bord dans ce genre de situations. Il devient alors nécessaire d'accéder directement à ce nombre. Lorsqu'un nombre est mémorisé dans une variable `var`, xlop crée une macro `\Op@var` qui contient ce nombre. On notera le « O » majuscule et le « p » minuscule. L'arrobas est là pour rendre cette définition privée c'est-à-dire qu'il faut faire un effort pour y accéder avec l'utilisation des macros `\makeatletter` et `\makeatother` de LaTeX ou bien en indiquant un code de catégorie égal à 11 (lettre) pour ce caractère lorsqu'on travaille sous TeX.

source

```
\opcopy{1234}{a}\opcopy{56}{b}%
\opmul*{a}{b}{r}%
\makeatletter
$\nombre{\Op@a} \times
 \nombre{\Op@b} = \nombre{\Op@r}$
\makeatother
```

$1\,234 \times 56 = 69\,104$

Annexe C

Versions futures

L'extension xlop en est à sa version 0.2 qui est essentiellement une version corrigée de la version 0.1 (première version publique). La prochaine version sera la 0.3 dont la version « stable » sera alors la version 0.4.

L'ensemble des spécificités de la version 0.3 n'est pas totalement arrêté mais il y a déjà plusieurs points prévus :

- gestion internationale des opérations posées ;
- opérations en base 2 à 36 ;
- ajout de fonctions de haut niveau avec les racines (`\oproot` pour les racines quelconques et `\opsqrt` pour la racine carrée), exponentielle, logarithme, fonctions trigonométriques (directes, inverses, hyperboliques) ;
- ajout d'une macro permettant de réaliser une écriture formatée, c'est-à-dire une écriture d'un nombre où les longueurs des parties entière et décimale seront indiquées (si ces longueurs ne sont pas celles du nombre, il y aura un débordement ou un remplissage) ; cette macro existait dans la version 0.1 et permettait essentiellement d'afficher des nombres avec un alignement sur la virgule, au fer à droite ou au fer à gauche ;
- ajout d'une macro permettant les additions à plus de deux opérandes ;
- ajout d'un paramètre permettant l'écriture scientifique ou ingénieur ;
- possibilité d'écrire un nombre sur plusieurs lignes et/ou en utilisant un séparateur des milliers ;
- retenues dans les multiplications ;
- rendre publics les restes successives d'une division ;
- les valeurs négatives de `maxdivstep` et `safedivstep` réaliseront un comptage sur les chiffres décimaux du quotient ;
- manuel en anglais.

Pour le premier point, seules la division anglo-saxonne, la multiplication dite russe et la multiplication dite babylonienne sont en cours d'étude, l'auteur ne connaissant pas les habitudes des autres pays en la matière. Si vous connaissez d'autres façons de poser les opérations à part

celles présentées dans ce manuel, la division anglo-saxonne, la multiplication russe (double colonne avec des divisions par deux sur une colonne et des multiplications par deux sur l'autre) et la multiplication babylonienne (ou vénitienne qui consiste à faire une grande grille de multiplication à un chiffre et à effectuer les sommes finales en diagonale), l'auteur vous sera éternellement reconnaissant de le contacter à l'adresse :

`Jean-Come.Charpentier@wanadoo.fr`

en plaçant le mot « `xlop` » dans le sujet du message.

Il serait souhaitable d'avoir un manuel du hacker qui expliquerait en détail le code source. Cet outil pourrait être tout à fait bénéfique pour que chacun puisse apporter plus facilement des améliorations au code. Malheureusement, le code actuel fait plus de 3 900 lignes et le travail nécessaire risque d'être trop important. Éventuellement, il pourra y avoir un manuel du hacker expliquant les spécifications générales du code sans entrer dans trop de détails techniques.

Annexe D

Index