Arithmétique des ordinateurs

Sujet proposé par Olivier Devillers

Olivier.Devillers@sophia.inria.fr

1 Préambule : nombres en virgule flottante

C'est l'approche informatique la plus classique. Plutôt que d'utiliser des nombres réels on va se contenter d'une approximation. Cette approche est conforme au calcul numérique que l'on peut faire à la main dans lequel un nombre est représenté sous la forme d'un nombre décimal. Ici on utilise une représentation avec des chiffres binaires. La norme IEEE 754 [IEE85] spécifie exactement les formats possibles pour représenter de tels nombres ainsi que les règles d'arrondis à appliquer pour obtenir le résultat d'un calcul.

1.1 Représentation

Un nombre flottant double précision (`double` en java) est représenté par 64 bits. Sur ces 64 bits se répartissent comme suit : 1 bit s, 52 bits m et 11 bits e représentent le nombre $(-1)^s \cdot 0.1m \cdot 2^{e-1024}$ en notation normalisée. La notation normalisée signifie que l'on choisit l'exposant pour que la mantisse soit comprise entre $\frac{1}{2}$ et 1, son développement binaire commence donc toujours par 0.1 et il n'est pas nécessaire de représenter ces deux chiffres.

Les nombres représentables en machines sont donc régulièrement espacé avec des intervalles de 2^{p-52} entre 2^p et 2^{p+1}.

Lorsque l'on se rapproche de zéro cette notation permet donc, lorsque $e = 0$ de représenter les nombres de $[2^{-1025}, 2^{-1024}]$ avec des écarts de 2^{-1077} mais pas du tout les nombres entre 0 et 2^{-1025} pour pallier à ce trou dans les nombres représentables on utilise lorsque $e = 0$ la représentation dénormalisée. Si $e = 0$, la valeur du nombre représentée est $(-1)^s \cdot 0.m \cdot 2^{-1024}$ on peut ainsi représenter les nombres de l'intervalle $[0, 2^{-1024}]$ avec des écarts réguliers de 2^{-1076}.

1.2 Règles d'arrondis

Si on a deux nombres flottants `a` et `b` le résultat d'une opération arithmétique sur ces nombres, la multiplication pour fixer les idées, n'est en général pas un nombre représentable. Si on appelle s et s' les deux nombres représentables qui encadrent la vrai valeur de `a`$\cdot$`b`.

La valeur calculée par la machine comme le résultat de `a*b` est alors s ou s' ($s < s'$) selon le mode d'arrondi choisi. La norme IEEE 754 propose quatre modes d'arrondi : au plus près, vers $+\infty$, vers $-\infty$ et vers 0.

En mode «au plus près» `a*b` est arrondi au plus proche flottant représentable, il faut donc comparer la valeur exacte `a`$\cdot$`b` avec $\frac{s+s'}{2}$.

En mode vers $+\infty$ on utilise s'.

En mode vers $-\infty$ on utilise s.

Et en mode vers 0 on utilise s si `a`$\cdot$`b`< 0 et s' sinon.

Il important de noter que la notion d'arrondi est définie très précisément. La valeur utilisée pour `a*b` est parfaitement définie par la norme afin d'assurer la portabilité sur différents processeurs.

Cette norme définissant l'arrondi avec précision s'applique au quatre opérations `+,-,*,/` et à la racine carré. Il faut parfois utiliser certaines options de compilation pour que le processeur se conforme à la norme.

Des travaux pour étendre ce type de norme définissant l'arrondi avec précision à d'autres fonctions telles que sinus, exponentielle. . .existent. La difficulté résidant dans le fait qu'il ne suffit pas de calculer une approximation précise de la valeur recherchée, mais qu'il faut comparer *exactement* cette valeur avec les bornes des intervalles ce qui peut s'avérer très délicat quand on est très près de ces bornes.

1.3 Propriétés

Les propriétés de l'arithmétique machine ne sont pas celle des réels, la propriété la plus cruciale étant l'associativité.

Commutativité L'addition et la multiplication sont commutatives.
$\texttt{a+b}=\texttt{b+a}$ et $\texttt{a*b}=\texttt{b*a}$.

Monotonicité L'arrondi préserve l'ordre respectif de deux nombres, toutefois les inégalités strictes deviennent large. En effet il est clair que deux nombres différents peuvent avoir le même arrondi.
$\texttt{a}\cdot\texttt{b} < \texttt{c}\cdot\texttt{d} \Longrightarrow \texttt{a*b} \leq \texttt{c*d}$.

Compatibilité soustraction et comparaison La soustraction et la comparaison sont bien compatible on a $\texttt{a==b} \Longleftrightarrow \texttt{a-b == 0}$

Cette propriété est liée à l'usage des nombres dénormalisés qui garanti que l'écart entre deux nombres représentables est également un nombre représentable.

Exactitude Comme on l'a vu, le résultat d'un calcul est toujours arrondi à un nombre représentable, mais si on peut garantir que le résultat exact est représentable, alors les nombres flottants font en fait du calcul exact. Les flottants double précision font ainsi d'excellents entiers sur 53 bits avec les avantages suivants :
— gestion plus facile des débordements (hors de l'intervalle $[-2^{53}, 2^{53}]$). En effet le calcul sur les entiers machines est fait modulo le plus grand entier, il n'est donc pas immédiat de savoir si une multiplication a débordé.
— possibilité de travailler dans les multiples de 2^k $(k \in \mathbb{Z})$ directement, sans faire explicitement les multiplications par 2^{-k} pour repasser aux entiers.
— sur certaines architectures, le calcul sur les flottants double précision est plus rapide que sur les entiers 64 bits voire même plus rapide que les entiers 32 bits.

Associativité Le caractère associatif est perdu, les bits les moins significatifs du résultat d'une opération disparaissent.

On peut parfois utiliser cette non associativité, par exemple l'arrondi entier d'un nombre $\texttt{a} \in [-2^{51}, 2^{51}]$ peut être obtenu par `Ea = (a + CST) - CST` où `CST` vaut $3 \cdot 2^{51}$.

La démonstration que cette opération fonctionne bien peut être effectuée comme suit :

La valeur exacte de `a + CST` appartient à l'intervalle $[2^{52}, 2^{53}]$, dans cet intervalle les nombres représentables exactement (`double`) coïncident avec les entiers puisqu'un tel nombre s'écrit en binaire $2^{53} \cdot 0.1 \ldots$ et que le 53ème chiffre derrière la virgule correspond bien à une unité. La valeur calculée de `a + CST` est donc l'arrondi entier au plus proche de la valeur réelle.

La valeur calculée de `(a + CST) - CST` est alors l'arrondi entier au plus proche de `a` puisque le résultat exact de la soustraction est nécessairement un nombre représentable.

On peut ensuite récupérer la partie fractionnaire par `da = a - Ea`, le résultat réel de cette opération est un nombre exactement représentable, cette soustraction sera donc calculée exactement.

La seule opération a être non exacte est donc la première addition `a + CST`, mais grâce aux spécifications exactes des règles d'arrondi, on contrôle parfaitement le résultat.

2 Détail du sujet

On peut représenter un grand nombre entier par une liste de chiffres en base 2^{52}, chacun de ces chiffres étant représenté par un `double`. On s'inspire ici d'un travail de Priest [Pri91] qui représente lui des grands flottants.

Écrire une telle classe `MyBigInt`.

Écrire pour cette classe les fonctions d'addition, soustraction, multiplication, comparaison et affichage

```
static MyBigInt add(MyBigInt x, MyBigInt y)
static MyBigInt sub(MyBigInt x, MyBigInt y)
static MyBigInt mult(MyBigInt x, MyBigInt y)
static boolean isGreater(MyBigInt x, MyBigInt y)
static void ecrit(MyBigInt x)
```

Pour l'écriture de ces fonctions, on utilisera les propriétés de l'arithmétique IEEE 754 (tel que l'arrondi expliqué plus haut) pour programmer les opérations intermédiaires.

Les algorithmes utilisés seront les algorithmes *de l'école primaire*. Par exemple pour l'addition, à une étape donnée on disposera de deux chiffres x et y ainsi que d'une retenue r. On commencera par calculer $x + y + r$ et à l'arrondir correctement pour trouver la retenue R à passer pour le chiffre plus à gauche, ensuite on calculera $(-R) + x + y + r$ pour déterminer le chiffre du résultat avant de passer au chiffre suivant. On contrôlera (sur le papier, pas dans le programme) que le calcul flottant ne fait pas d'arrondi non voulu.

Pour la fonction d'affichage (en base 10), on pourra programmer une division par les puissances de 10, ou plus simplement précalculer les puissances de 10 dans un tableau et procéder par comparaison.

Calculer factorielle, jusqu'au débordement de capacité des nombres flottants. Comparer avec Maple pour être sur de l'exactitude des calculs.

3 Pour aller plus loin

Le test de cocyclicité consiste à évaluer le signe du déterminant

$$\begin{vmatrix} x_p & x_q & x_r & x_t \\ y_p & y_q & y_r & y_t \\ x_p^2 + y_p^2 & x_q^2 + y_q^2 & x_r^2 + y_r^2 & x_t^2 + y_t^2 \\ 1 & 1 & 1 & 1 \end{vmatrix}.$$

Le signe de ce déterminant décrit si le point t est à l'intérieur ou à l'extérieur du cercle circonscrit au triangle pqr (supposé direct).

Comment programmer ce test pour des grands entier en effectuant le moins de calcul possible, c'est à dire que si il est possible de conclure sur le signe sans calculer tous les chiffres du résultat, on souhaite le faire.

Donner des exemples de telles situation, et de situation où il faut pousser les calculs dans le cas où les coordonnées des points ont un seul chiffre significatif en base 2^{52} (ces nombres s'écrivent $x(2^{52})^p$ où $x \in [-2^{52}, 2^{52}]$ et p est une (petit) entier).

Références

[IEE85] *IEEE Standard for binary floating point arithmetic, ANSI/IEEE Std* 754 − 1985. New York, NY, 1985. Reprinted in SIGPLAN Notices, 22(2) :9–25, 1987.

[Pri91] D. Priest. Algorithms for arbitrary precision floating point arithmetic. In *Proc. 10th Symp. on computer arithmetic*, pages 132–143, 1991. Disponible : `ftp://ftp-sop.inria.fr/geometrica/devillers/tmp/Priest.pdf`