Les nombres réels et l'ordinateur

Représentation des nombres flottants

Soit b en entier supérieur ou égal à 2.

Les puissances de b positives ou négatives permettent une partition de $\mathbb{R}^*$.

En d'autres termes,

$$\forall x \in \mathbb{R}^*, \exists! e \in \mathbb{Z} \text{ tel que } b^e \leq x < b^{e+1}$$

Ce qui revient à écrire $x = b^e \times m$ avec $1 \leq m < b$.

Représentation des nombres réels normalisée

Soit b une base, tout réel x non nul peut s'écrire de manière unique

$$x = s \times b^e \times m$$

avec $s = \pm 1, e \in \mathbb{Z}, m \in [1, b[$

s : le signe
e : l'exposant
m : la mantisse

Si on écrit le développement de m en base b, on obtient :

$$m = a_0, a_1 a_2 a_3 ... = \sum_{i=0}^{+\infty} a_i \times b^{-i}$$

avec $a_i \in \{0, ..., b-1\}$ et $a_0 \neq 0$.

La norme IEEE 754

$b = 2$

La simple précision :

- sur 32 bits :

  1 bit de signe, 8 bits pour l'exposant, 23 bits pour la mantisse.

- $x = s \times b^e \times m$ est représenté par

  $X =$

s	e $+127$	a_1 a_2 ... a_{23}

- $-127 \leq e \leq 127$ et $2^{127} \approx 1,7 \times 10^{38}$

Les réels représentables en machine sont appelés les **nombres flottants** (tous les bits sont nuls à partir de a_{24}).

La norme IEEE 754 suite

$b = 2$

La double précision :

- sur 64 bits :

  1 bit de signe, 11 bits pour l'exposant, 52 bits pour la mantisse.

- $x = s \times b^e \times m$ est représenté par

  $X =$ | s | e $+1023$ | a_1 a_2 ... a_{52} |

- $-1023 \leq e \leq 1023$ et $2^{1023} \approx 9 \times 10^{307}$

Pour la simple comme pour la double précision, a_0 est toujours égal à 1 et n'est pas représenté en machine. On parle de bit *caché*.

Les nombres flottants spéciaux

La norme IEEE-754 impose que tout calcul ait un résultat !!

Nécessité de 3 flottants particuliers : $+\infty$, $-\infty$ et le ... NaN (Not An Number).

En double précision

- $10^{200} \times 10^{200} = +\infty$
- $10^{200} \times -10^{200} = -\infty$
- $\sqrt{-1} = \text{NaN}$

$+\infty$:

0	1...1	0.....0

$-\infty$:

1	1...1	0.....0

NaN :

0	1...1	*xx...xxxx*

Exemple

Codage de $13,1$ en simple précision.

On distingue la partie entière et la partie décimale.

$13 = 1101\,[2]$

Pour avoir la décomposition de $0,1$ en base 2 :

$0,1 \times 2 = \boxed{0}\,,2$
$0,2 \times 2 = \boxed{0}\,,4$
$0,4 \times 2 = \boxed{0}\,,8$
$0,8 \times 2 = \boxed{1}\,,6$
$0,6 \times 2 = \boxed{1}\,,2$

Donc $0,1 = 0,0\ 0011\ 0011\ \ldots$

Donc $13,1 = 1,1010\ 0011\ 0011\ldots \times 2^3$

Au final,

$13,1$:

0	10000010	10100011001100110011001

$1001100110011\ldots$

Fin

Les modes d'arrondi

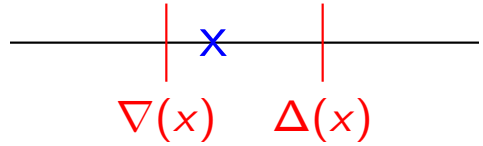
Nécessité d'arrondir

- Si x et y sont deux nombres exactement représentables en machine,
  alors le résultat d'une opération $res = x \odot y$ n'est pas forcément représentable en machine.

- Par exemple, en base $b = 10$, le nombre $1/3$ n'est pas représentable avec un nombre fini de chiffres.
  En base 2, $1/10$ n'est pas représentable exactement par un flottant.

- Il faut *arrondir* le résultat, autrement dit, trouver un nombre *flottant* voisin et savoir comment il a été choisi.

Norme IEEE-754: modes d'arrondis



Chaque réel x qui n'est pas un flottant est compris entre deux flottants consécutifs.

La norme propose 4 modes d'arrondi :

- arrondi *vers* $+\infty$: $\Delta(x)$
- arrondi *vers* $-\infty$: $\nabla(x)$
- arrondi *vers* 0 : $\Delta(x)$ si $x < 0$ et $\nabla(x)$ si $x > 0$
- arrondi *au plus près* (par défaut) : le nombre machine le plus proche de x (pour le milieu de deux nombres machine consécutifs on choisit celui dont la mantisse se termine par un 0, on parle d'*arrondi pair*)

En pratique

$x =$ | s | e $+k$ | $a_1\ a_2 \quad \ldots \quad a_p$ | $a_{p+1} a_{p+2} \ldots$

Pour avoir $\Delta(x)$ ou $\nabla(x)$, dans le désordre, soit on garde les p premiers bits, soit on ajoute 1 à a_p et on propage l'éventuelle retenue.

Pour l'arrondi vers 0, on garde les 23 premiers bits de la mantisse sans se soucier du signe.

Pour l'arrondi vers $+\infty$, on ajoute 1 si x est positif, on ne fait rien sinon.

Pour l'arrondi vers $-\infty$, c'est l'inverse.

Pour l'arrondi au plus près, on ajoute a_{p+1} à a_p (ou presque).

Norme IEEE-754: propriété de l'arrondi correct

Soient x et y sont deux nombres exactement représentables en machine, $\odot$ une des opérations rationnelles $+$, $-$, $\times$, $/$ et $\diamond$ le mode d'arrondi choisi parmi les 4 modes IEEE.

La norme IEEE exige que le résultat d'une opération $x \odot y$ soit égal à $\diamond(x \odot_{exact} y)$. Le résultat doit être le même que si on effectuait le calcul en précision infinie puis on arrondissait ce résultat.

Idem pour la racine carrée.

C'est la propriété de *l'arrondi correct*.

La norme IEEE décrit un algorithme pour l'addition, la soustraction, la multiplication, la division et la racine carrée et exige que ses implémentations produisent le même résultat que cet algorithme.

Gestion des modes d'arrondi en C

Il faut utiliser

```
#include <fenv.h>
```

ce fichier définit 4 entiers

```
FE_DOWNWARD
FE_TONEAREST
FE_TOWARDZERO
FE_UPWARD
```

ainsi que les fonctions

```
int fegetround(void);
int fesetround(int);
```

Ne pas oublier de faire l'édition de liens avec l'option $-lm$.

Fin

La cancellation catastrophique

La propagation d'erreur d'arrondi

Chaque donnée qui n'est pas un flottant est arrondie, i.e. remplacée, par un flottant.

Idem pour chaque résultat intermédiaire.

$\Rightarrow$ la propagation d'erreurs d'arrondi

Présenté le 8 juin 2018, le Summit américain (IBM) atteint 122 millions de milliards d'opérations par seconde (122×10^{15})

Un problème ou pas ?

Un petit exemple

$P(x, y) = 9x^4 - y^4 + 2y^2$

avec

$x = 10864, y = 18817$

et

$x = \frac{1}{3}, y = \frac{2}{3}$

Un petit exemple : le code source

```
#include <stdio.h>
#include <fenv.h>
void main()
{ double x, y, res1, res2, res;
  int k;
  printf("entrer le mode d'arrondi (0 zero, 1 arr, 2 moinf, 3 plinf) :");
  scanf("%d",&k);
  switch(k)
    {case 0 : fesetround(FE_TOWARDZERO); break;
     case 1 : fesetround(FE_TONEAREST); break;
     case 2 : fesetround(FE_DOWNWARD); break;
     case 3 : fesetround(FE_UPWARD); break;
    };
  x = 10864.0; y = 18817.0;
  res = 9*x*x*x*x - y*y*y*y + 2*y*y;
  printf(" Res = %24.15le \n",res);
  x = 1.0/3.0; y = 2.0/3.0;
  res = 9*x*x*x*x - y*y*y*y + 2*y*y;
  printf(" Res = %24.15e \n",res);
}
```

```
gcc -O0 -o poly poly.c -lm
```

```
Arrondi vers 0
=============
 Res =    2.000000000000000e+00
 Res =    8.024691358024689e-01
==============
Arrondi au plus près
=============
 Res =    2.000000000000000e+00
 Res =    8.024691358024691e-01
==============
Arrondi vers moins l'infini
=============
 Res =    2.000000000000000e+00
 Res =    8.024691358024689e-01
==============
Arrondi vers moins l'infini
=============
 Res =   -1.400000000000000e+01
 Res =    8.024691358024692e-01
```

En fait, $P(10864, 18817) = 1$.

Aucun problème pour $P(1/3, 2/3)$.

$$9 \times 10864^4 = 125372283822342144$$

$$18817^4 = 125372284530501121$$

$$9 \times 10864^4 - 18817^4 = -708158977$$

$$2 \times 18817^2 = 708158978$$

d'où $P(10864, 18817) = 1$

La soustraction de 2 flottants voisins entachés d'erreur d'arrondi provoque une remontée de l'erreur d'arrondi dans la mantisse.

$X_1 =$ | s | e | $a_0 a_1 \ldots a_k b_{k+1} b_{k+2} \ldots b_q xxxxx$ |

$X_2 =$ | s | e | $a_0 a_1 \ldots a_k c_{k+1} c_{k+2} \ldots c_q yyyy$ |

$X_1 - X_2 =$ | s | $e-k$ | $d_{k+1} \ldots d_{q-k} zzzzz 0_1 \ldots 0_k$ |

On a perdu k bits significatifs en une opération.

Recherche de formulations stables

Des formules mathématiquement équivalentes ont des comportements très différents sur ordinateur.

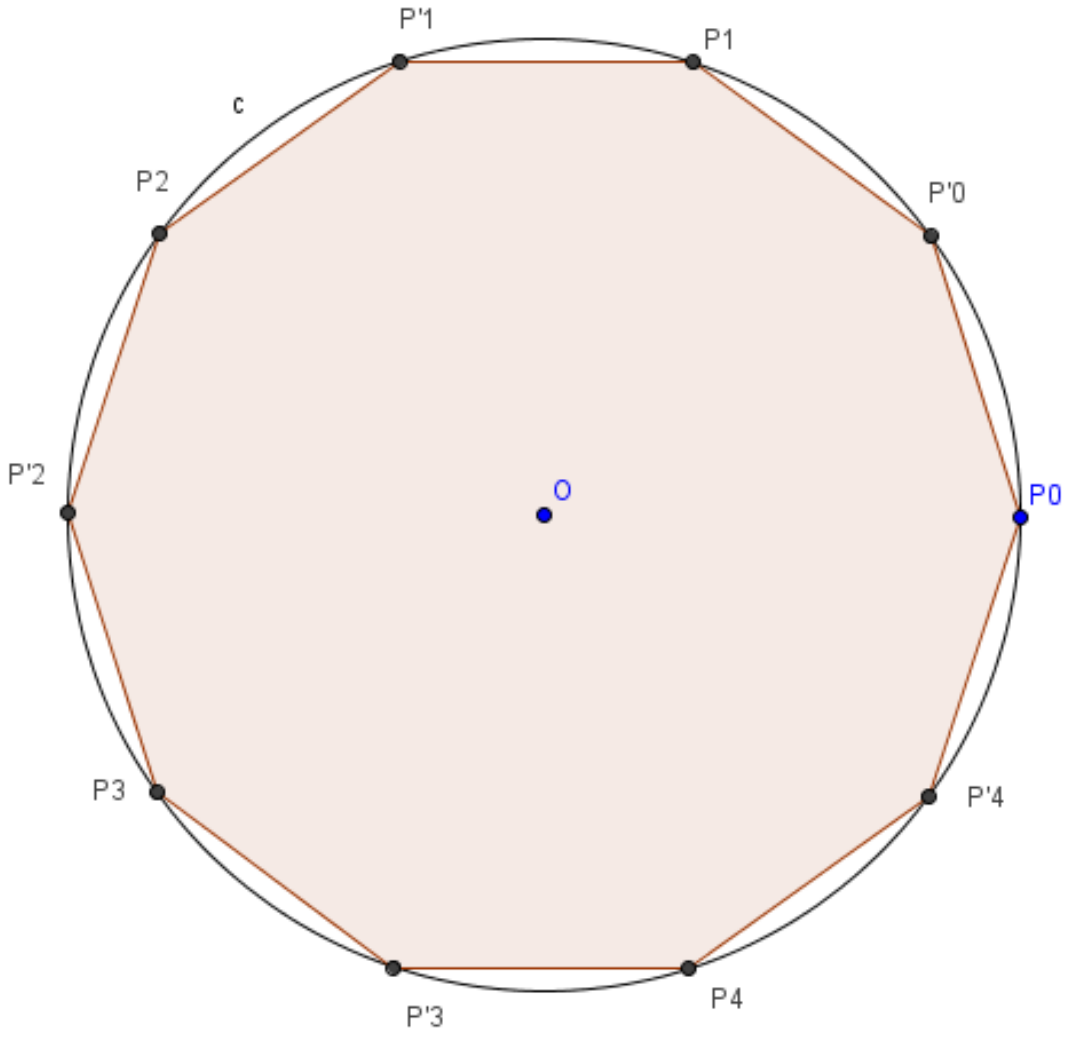
Exemple : la formule d'Archimède (-250 avant JC).

Soit le polygone régulier à N côtés inscrit dans le cercle unité.

Soit L_N la longueur d'un côté.

La circonférence du polygone NL_N approxime la circonférence du cercle 2π, donc $\pi \approx NL_N/2$ pour N suffisamment grand.

Le décagone



On admet que

$$L_{2N}^2 = 2(1 - \sqrt{1 - L_N^2/4}).$$

Cette récurrence est équivalente à:

$$L_{2N}^2/4 = \frac{L_N^2/4}{2\,(1 + \sqrt{1 - L_N^2/4})}$$

La deuxième formulation est beaucoup plus stable.

En effet, $L_N \to 0$ donc $1 - L_N^2/4 \to 1$

$\Rightarrow$ la soustraction $1 - \sqrt{1 - L_N^2/4}$ est catastrophique.

Fin