

Représentation des réels sur ordinateur

Skander Zannad et Judicaël Courant

2013-11-26

Plan

1 Réels	1
2 Virgule fixe	2
3 Virgule flottante	2
4 Virgule flottante en binaire	3
5 Norme IEEE 754	3
6 Problèmes de précision	4
6.1 Problèmes liés aux arrondis	4
6.2 Problèmes liés à la représentation binaire	4
6.3 Origine du problème	5
7 Erreurs d'arrondis : conséquences	6

1 Réels

Mathématiquement : plein de façon de voir les réels

Une façon particulière : partie entière (un entier relatif) et une suite (infinie) de chiffres donnant la partie fractionnaire.

Peut-on représenter une suite de chiffres infinies ?

Oui, par un algorithme.

Peut-on représenter toutes les suites de chiffres infinies ?

Non [Turing].

Pour des besoins de calcul scientifique :

— Pas besoin de représenter tous les réels.

— On travaille avec des approximations.

— Une approximation possible : les nombres décimaux.

NB : qui dit approximation dit *erreur*.

Soit $x \in \mathbb{R}$ et a une approximation de x .

Erreur absolue $|x - a|$

Erreur relative $\frac{|x-a|}{|x|}$ (non définie si $x = 0$)

Besoin de nombres dont la représentation a une taille réduite :

1. Pour prendre une place réduite (en mémoire, sur disque, sur le réseau).
2. Pour calculer vite.

2 Virgule fixe

Nombres décimaux avec nombre n fixé de chiffres après la virgule.

Avantage : on comprend bien comment ça marche.

Inconvénient :

1. besoin de beaucoup de chiffres après la virgule (masse de l'électron : 9×10^{-31} kg, $h \approx 6 \times 10^{-34}$ J.s).
2. garder 30 chiffres après la virgule : inutile pour manipuler un grand nombre (durée de vie moyenne de l'électron : 10^{34} s).

Erreur absolue : au plus 10^{-n} .

Mais : l'important est l'erreur *relative*.

3 Virgule flottante

Idée : notation scientifique des nombres :

1. Un nombre est représenté sous la forme $s \times m \times 10^e$.
2. s : signe (+1 ou -1).
3. $m \in [1, 10[$: décimal, avec n chiffres après la virgule (n fixé).
4. e : entier (relatif) appartenant à une plage de valeurs fixée.

Vocabulaire :

1. s : signe
2. m : mantisse
3. e : exposant

Exemple : calculatrice HP48SX (d'après tests personnels) :

— Mantisse avec 11 chiffres après la virgule.

— Exposant : $e \in \llbracket -499, 500 \rrbracket$.

Permet de représenter :

1. de très grands nombres : jusqu'à $9,999\,999\,999\,99 \times 10^{499}$;
2. de très petits (en valeur absolue) : jusqu'à 10^{-499}
3. et leurs opposés : $-9,999\,999\,999\,99 \times 10^{499}$ et -10^{-499} ;
4. avec une erreur relative inférieure à 10^{-11} .

avec seulement 12 chiffres décimaux, un signe et trois chiffres pour l'exposant.

4 Virgule flottante en binaire

On peut faire de la virgule flottante en base 2. Pour cela :

1. Un nombre est représenté sous la forme $s \times m \times 2^e$.
2. s : signe (+1 ou -1).
3. $m \in [1, 2[$: nombre *dyadique*, avec n chiffres après la virgule (n fixé).
4. e : entier (relatif) appartenant à une plage de valeurs fixée.

Erreur relative au plus : 2^{-n} pour les réels représentables.

Nombre décimal : de la forme $n/10^k$ où $n \in \mathbb{Z}$, $k \in \mathbb{N}$.

Nombre dyadique : de la forme $n/2^k$ où $n \in \mathbb{Z}$, $k \in \mathbb{N}$.

En décimal le nombre $12345/10^3$ s'écrit 12,345.

En binaire, le nombre $10101011/10^{101}$ s'écrit 101,01011. Il vaut $171/2^5 = 5,34375$.

Autre façon de calculer :

$$\underline{101,01011} = 5 + \frac{1}{2^2} + \frac{1}{2^4} + \frac{1}{2^5}$$

5 Norme IEEE 754

1. Norme utilisée dans tous les ordinateurs pour les nombres à virgule flottante.
2. Plusieurs versions (simple précision, double précision, double précision étendue).
3. On ne parlera ici que de la double précision (la plus répandue).

Flottants double précisions : représentés sur 64 bits :

1. 1 bit pour le signe (0 pour +, 1 pour -).
2. 11 bits pour l'exposant décalé e (exposant plus 1023).
3. 52 bits pour les 52 chiffres après la virgule de la mantisse (inutile de garder le premier bit de m : c'est 1).

Interprétation du flottant $se_{10} \dots e_0 m_1 \dots m_{52}$:

Si $e_{10} \dots e_0 \in \llbracket 1, 2047 \rrbracket$, nombre *normalisé* :

$$s \times \underline{1, m_1 \dots m_{52}} \times 2^{(-1023 + e_{10} \dots e_0)} = s \times \left(1 + \sum_{k=1}^{52} \frac{m_k}{2^k} \right) \times 2^{(-1023 + \sum_{k=0}^{10} e_k 2^k)}$$

Si $e = 0$ et $m_1, \dots, m_{52}$ tous nuls : 0 (deux versions : +0 et -0).

Si $e = 0$ et $m_1, \dots, m_{52}$ non tous nuls : nombre *dénormalisé* :

$$s \times \underline{0, m_1 \dots m_{52}} \times 2^{-1022} = s \times \left(\sum_{k=1}^{52} \frac{m_k}{2^k} \right) \times 2^{-1022}$$

Si $e = 2047$ et $m_1, \dots, m_{52}$ tous nuls : $s\infty$ (+ ∞ ou - ∞).

Si $e = 2047$ et $m_1, \dots, m_{52}$ non tous nuls : NaN .

Nombres normalisés : permettent de représenter de façon précise les réels de $[-M, -m] \cup [m, M]$ avec

$$m \approx 2^{-1022} \approx 2 \times 10^{-308}$$

$$\text{et } M \approx 2^{1024} \approx 1,8 \times 10^{308}$$

L'essentiel à savoir :

1. Le principe de la représentation des nombres *normalisés* (vu).
2. Les problèmes de précision : origine.
3. Leurs conséquences.

6 Problèmes de précision

Différents types :

1. Problèmes liés aux arrondis des calculs
2. Problèmes liés à la représentation binaire.

6.1 Problèmes liés aux arrondis

En décimal : $1,23 \times 1,56 = 1,9188$

Pour garder deux chiffres après la virgule : arrondir

Pour l'addition : $1,23 \times 10^3 + 4,56 \times 10^0 = 1,23456 \times 10^3$

Pour garder deux chiffres après la virgule : arrondir

6.2 Problèmes liés à la représentation binaire

En fait : problèmes liés au *passage* décimal/binaire.

Exemple en Python :

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.1 + 0.2 - 0.3
5.551115123125783e-17
```

???

6.3 Origine du problème

Théorème :

1. Pour tout n , $1/n$ est un nombre décimal si et seulement si n est de la forme $2^\alpha 5^\beta$ où $(\alpha, \beta) \in \mathbb{N} \times \mathbb{N}$.
2. Pour tout n , $1/n$ est un nombre dyadique si et seulement si n est de la forme 2^α où $\alpha \in \mathbb{N}$.

Conséquence : $\frac{1}{10}$ n'est pas un nombre dyadique.

$$1/10 = \underline{0,00011001100110011001100110011\dots}$$

Le flottant (arrondi par défaut) représentant $\frac{1}{10}$ est donc

$$\frac{1,10011001100110011001100110011001100110011001}{2^4}$$

L'arrondi au plus près x vaut

$$1,10011001100110011001100110011001100110011010 \times 2^{-4}$$

Qui est la représentation exacte de

0,10000000000000000055511151231257827021181583404541015625

De même le flottant y (arrondi au plus proche) représentant $\frac{2}{10}$ est

$$\underline{1,10011001100110011001100110011001100110011010} \times 2^{-3}$$

Somme $x + y$:

$$\begin{aligned} & \underline{0}, \underline{11001100110011001100110011001100110011010}, 2^{-3} \\ & + \underline{1}, \underline{10011001100110011001100110011001100110011010}, 2^{-3} \\ = & \underline{10}, \underline{0110011001100110011001100110011001100110011010}, 2^{-3} \end{aligned}$$

Arrondi au flottant le plus proche :

$$\underline{1, 001100110011001100110011001100110011001100110100} \cdot 2^{-2}$$

Soit

0,300000000000000000444089209850062616169452667236328125

Quand on effectue le calcul $0.1 + 0.2 - 0.3$:

1. nouvelles erreurs d'arrondi
2. «négligeables» devant $0,1$
3. mais pas devant $4 \times 10^{-17} \dots$
4. d'où le résultat final de l'ordre de 6×10^{-17} .

7 Erreurs d'arrondis : conséquences

1. Un test de la forme $x == 0$ ou $x == y$ pour des flottants n'a aucun sens !
2. Seule possibilité parfois raisonnable : test de «petitesse». Par exemple : $\text{abs}(x) < \text{epsilon}$ avec $\text{epsilon} = 1\text{e-}6$.
3. Quelle valeur de ϵ choisir ? Pas de réponse universelle, à étudier en fonction du problème...

De même, se méfier des test $x < y$ ou $x \leq y$.

Exemple : construisons une équation du second degré.

```
>>> r1 = 1 + 1.2e-16
```

```
>>> r1
1.0000000000000002
```

```
>>> r2 = 1
```

```
>>> a, b, c = 1, -(r1+r2), r1 * r2
```

Normalement r_1 et r_2 sont les deux racines réelles de $aX^2 + bX + c$.

Vérifions :

```
>>> Delta = b**2 - 4*a*c
```

```
>>> sqrt(Delta)
```

```
Traceback (most recent call last):
```

File "<stdin>", line 1, in <module>

```
ValueError: math domain error
```

```
>>> Delta
```

-8.881784197001252e-16

Oups...

```
>>> a*r1**2 + b*r1 + c
```

2.220446049250313e-16

```
>>> a*r2**2 + b*r2 + c
```

2.220446049250313e-16

On peut aussi trouver des cas où Delta est nul avec le polynôme qui s'annule au moins sur deux flottants, dont l'un n'est pas supposé être une racine...