Nombres, machines et calculs

Roland Denis

CNRS / Institut Camille Jordan

21 Mars 2019

Représentation des nombres

Nombres

Avec quels nombres veut-on calculer ?

- $\mathbb{N}$, entiers naturels : $\{0, 1, 2, \ldots\}$.
- $\mathbb{Z}$, entiers relatifs : $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
- $\mathbb{Q}$, rationnels : a/b avec $a \in \mathbb{Z}$, $b \in \mathbb{Z}^*$.
- $\mathbb{R}$, nombres réels.
- $\mathbb{C}$, nombres complexes.

Binaire

Représentation machine

Il faut traduire les calculs en langage binaire !

- **bit** chiffre en base 2 (0 ou 1)
- **octet (byte)** unité utilisée pour les adresses mémoires (8 bits)
- **mot (word)** unité de base manipulée par un microprocesseur (64 bits)

Possibilité d'encodage

Nombre de bits	Motifs	Nombre de motifs
1 bits	0 1	2
2 bits	00 01 10 11	4
3 bits	000 001 010 011 100 101 110 111	8
8 bits	...	256
64 bits	...	18 446 744 073 709 551 616
n bits	...	2^n

Codage des entiers naturels

En base 10

Avec p décimales d_k, on peut coder 10^p nombres naturels différents sous la forme

$$d_{p-1} \times 10^{p-1} + d_{p-2} \times 10^{p-2} + \cdots + d_2 \times 100 + d_1 \times 10 + d_0 \times 1.$$

Par exemple : $2019 = 2 \times 1000 + 0 \times 100 + 1 \times 10 + 9 \times 1$.

En base 2

Avec p bits b_k, on peut coder 2^p nombres naturels différents sous la forme

$$b_{p-1} \times 2^{p-1} + b_{p-2} \times 2^{p-2} + \cdots + b_2 \times 4 + b_1 \times 2 + b_0 \times 1.$$

Par exemple :

$$\begin{aligned} 11111100011_{(2)} &= 1 \times 2^{10} + 1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 \\ &\quad 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 1024 + 512 + 256 + 128 + 64 + 32 + 2 + 1 = 2019_{(10)} \end{aligned}$$

Codage des entiers relatifs

Représentation naïve

- Le bit de poids fort (puissance la plus élevée) pour représenter le signe,
- Les bits restants pour représenter la valeur absolue.

Exemple

$$
\begin{aligned}
4 &= 0\,0000100_{(2)} \\
-4 &= 1\,0000100_{(2)}
\end{aligned}
$$

Problème : la somme (usuelle) de 4 et -4 ne fait pas 0 dans cette représentation :

$$
\begin{aligned}
4 &= 0\,0000100_{(2)} \\
-4 &= 1\,0000100_{(2)} \\
-8 &= 1\,0001000_{(2)}
\end{aligned}
$$

Codage des entiers relatifs

Complément à 2

Pour éviter ce problème, on utilise le *complément à 2* pour représenter les nombres entiers négatif.

Exemple

Par exemple, pour coder -4 avec 8 bits :

- on prend la représentation de 4 : $00000100_{(2)}$,
- on inverse tous ses bits (NOT) : $11111011_{(2)}$ $(= 2^8 - 1 - 4)$,
- on ajoute 1 : $11111011_{(2)} + 00000001_{(2)} = 11111100_{(2)}$ $(= 2^8 - 4)$.

Un entier négatif k est donc représenter, avec n bits, comme le nombre $2^n - |k|$.
Avec cette méthode, il n'y a plus de problème avec l'addition :

$$\begin{aligned} 4 &= 00000100 \\ -4 &= 11111100 \\ 0 &= 00000000 \end{aligned}$$

Représentation des nombres entiers

En précision finie

Les nombres naturels et relatifs sont en général codés sur $p = 32$ ou $p = 64$ bits.

On peut donc représenter les nombres :

- naturels entre 0 et $2^p - 1$
- relatifs entre $-2^{p-1} + 1$ et $2^{p-1} - 1$

Comment faire pour représenter de grands entiers ?

En précision "infinie"

Représenter les entiers sur plusieurs mots machines, grâce à des bibliothèques comme GMP, beaucoup utilisée en cryptographie et théorie des nombres.

Problème : c'est plus lent !

Représentation des nombres réels

En base 10

En s'inspirant du codage des entiers :

$$d_{p-1} \times 10^{p-1} + \cdots + d_1 \times 10^1 + d_0 \times 10^0 + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + \dots$$

Par exemple :

$$5837.25 = 5 \times 10^3 + 8 \times 10^2 + 3 \times 10^1 + 7 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}.$$

En base 2

De même :

$$b_{p-1} \times 2^{p-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \dots$$

Par exemple :

$$\begin{aligned} 1011011001101.01_{(2)} &= 4096 + 1024 + 512 + 128 + 64 + 8 + 4 + 1 + 2^{-2} \\ &= 5837.25_{(10)} \end{aligned}$$

Représentation des nombres réels

Il est impossible de représenter exactement tous les nombres réels avec une quantité d'information finie !

Nombres irrationnels

π, e, $\sqrt{2}$, ... et tous les nombres irrationnels ne sont pas représentables avec un nombre fini de chiffres (sinon ils seraient rationnels).

Nombres normaux

Un nombre normal est un nombre réel tel que la fréquence d'apparition de tout n-uplet dans la suite de ses « décimales » dans toute base est équirépartie.

Nombre rationnels

Dépend de la base :

- $\frac{1}{3} = 0.3333\ldots_{(10)} = 0.1_{(3)}$
- $\frac{1}{10} = 0.1_{(10)} = 0.00011001100110011\ldots_{(2)}$

Et les nombres rationnels ?

On pourrait représenter les nombres rationnels sous forme de fractions irréductibles $\frac{a}{b}$ avec a et b des entiers signés.

Avantage

Tout nombre réel peut être approché d'aussi près qu'on veut par des nombres rationnels (densité).

Inconvénients

On ne peut pas borner la taille (nombre de chiffres) des numérateurs et dénominateurs.

Conséquence : le coût des opérations élémentaires n'est pas constant (i.e. le temps de calcul pour une addition peut varier en fonction des opérandes).

Nombres à virgule flottante

Nombres à virgule flottante

Pour un nombre fixé p de chiffres, on peut approcher un plus grand intervalle de valeurs en utilisant la notation à virgule flottante : on place la virgule juste à gauche du premier chiffre non nul et on multiplie par une puissance de la base.

La représentation se décompose alors en un signe, une mantisse et un exposant.

Flottants en base 10

nombre	flottant	signe	mantisse	exposant
0.0121	$+0.121 \times 10^{-1}$	$+$	0.121	-1
-5837.25	-0.583725×10^{4}	$-$	0.583725	4

Flottants en base 2

nombre	flottant	signe	mantisse	exp.
0.0121	$+0.1100011000 \times 2^{-6}$	$+$	0.1100011000	-6
-5837.25	$-0.101101100110101 \times 2^{13}$	$-$	0.101101100110101	13

Nombres à virgule flottante

En virgule flottante, en base b, un nombre réel x est représenté par :

- un signe $s \in \{0, 1\}$
  - 0 : positif
  - 1 : négatif
- une mantisse m, écrite en virgule fixe en base b sur p chiffres appelés digit, sous la forme $0.d_1 d_2 d_3 d_4 \ldots d_p$.
- un exposant $e \in \{e_{min}, ..., e_{max}\}$,

$$x = (-1)^s \times m \times b^e$$

On dit que le nombre flottant x est de précision p (avec $p \geq 1$).

Cas particulier de la base 2 : d_1 vaut toujours 1 et n'a pas besoin d'être stocké (bit implicite).

Standardisation - IEEE 754

- Avant les années 1980 : chaque fabricant utilisait sa propre représentation des nombres flottants et de son arithmétique.
  - Quelle base b était utilisée ? Avec quelle amplitude $[e_{min}, e_{max}]$ de l'exposant ?
  - Un même code donnait des résultats différents sur différentes architectures de processeur.
- Besoin de standardisation en base 2.
  - Fixer précisément le format des données et leur représentation.
  - Définir le comportement et la précision des opérations de bases.
  - Définir les valeurs spéciales, les modes d'arrondis et la gestion des exceptions.
- En 1985 : publication du standard IEEE 754-1985.
- En 2008 : révision du standard : IEEE 754-2008.

Représentation standard des nombres flottants

IEEE 754

	Simple précision	Double précision
Précision p	24	53
Longueur de l'exposant e	8	11
Longueur de la représentation $(p+e)$	32	64
e_{min}, e_{max}	-126, 127	-1022, 1023
Type C/C++	*float*	*double*

Autres contributions du standard IEEE 754

- Valeurs spéciales : $-\infty$ (1/0), $+\infty$ ($-1/0$), *NaN* (0/0), ...
- Modes d'arrondi : vers $+\infty$, $-\infty$, 0, ou vers le flottant le plus proche.
- Exceptions : opération invalide, dépassement (*overflow*), "soupassement" (*underflow*), division par zéro, ...

Nombres à virgule flottante : normaux et sous-normaux

Nombres normaux

Les nombres normaux sont ceux dont la représentation est celle vu précédemment :

- En base 10, la mantisse est de la forme $m = 0.n_1 n_2 \dots n_m$ où n_i est compris entre 0 et 9 avec n_1 non nul.
- En base 2, c'est un peu différent, la mantisse est de la forme $m = 1.n_1 n_2 \dots n_m$, où les n_i sont dans $\{0, 1\}$. Le 1 avant la virgule n'est pas codé, il est implicite.

Nombres sous-normaux

Afin d'avoir une meilleure répartition des nombres proches de 0, on définit les nombres sous-normaux. Lorsque l'exposant est minimal ($e = e_{\min}$), on autorise la mantisse à ne plus suivre le modèle des nombres normaux :

- en base 10, n_1 peut-être égal à 0 (par exemple, 0.0125×10^{-12}),
- en base 2, la mantisse est de la forme $m = 0.n_1 n_2 \dots n_m$.

Attention aux performances dégradées avec les nombres sous-normaux (DEMO)

Nombres à virgule flottant : limites

Propriétés et définitions

- Plus petit nombre normal représentable : $\epsilon = \pm 0.1 b^{e_{min}}$
- Plus petit nombre sous-normal représentable : $\epsilon = \pm 0.0 \dots 1 b^{e_{min}}$
- Plus grand nombre représentable : $M = \pm 0.(b-1)\dots(b-1) b^{e_{max}}$
- Dépassement de capacité : nombre plus petit que $\epsilon \implies$ débordement par valeur inférieure (underflow) ; Nombre plus grand que $M \longrightarrow$ débordement par valeur supérieure (overflow).

Limites

- Certains réels sont par définition impossibles à représenter en numération classique : $1/3$, π ...
- La représentation en un nombre fixe d'octets oblige le processeur à faire appel à des approximations afin de représenter les réels (DEMO).
- Le degré de précision de la représentation par virgule flottante des réels est directement proportionnel au nombre de bits alloués à la mantisse, alors que le nombre de bits alloués à l'exposant conditionnera l'amplitude de l'intervalle des nombres représentables.

Nombres à virgule flottante : plus loin ?

Et si on veut plus de précision ?

- type `long double`, `binary128`, `binary256`, pas très normalisés ...
- Bibliothèques :
  `GNU MPFR` (Gnu Multiple Precision)
  http://www.mpfr.org/ (Paul Zimmermann, Inria Nancy).
  Mais c'est forcément *très* lent.

Quelques propriétés des nombres à virgule flottante

Les ensembles $F(b, p, e_{min}, e_{max})$ des flottants décrivent seulement un sous-ensemble fini des nombres réels. DeclareMathOperator

Arrondi

- Si $x \in F(b, p, e_{min}, e_{max})$, $\text{Arrondi}(x) = x$.
- Si $x \notin F(b, p, e_{min}, e_{max})$:
  - $\text{Arrondi}(x) =$ nombre de $F(b, p, e_{min}, e_{max})$ le plus proche de x.
  - $\text{Arrondi}(x) =$ nombre de $F(b, p, e_{min}, e_{max})$ immédiatement supérieur.
  - $\text{Arrondi}(x) =$ nombre de $F(b, p, e_{min}, e_{max})$ immédiatement inférieur.
  - $\text{Arrondi}(x) =$ nombre de $F(b, p, e_{min}, e_{max})$ le plus proche en direction de zéro.

DEMO

Unit in the Last Place (ULP)

Taille de l'intervalle séparant chaque nombre du nombre représentable le plus proche (dans la direction opposée de celle de zéro.
DEMO

Arithmétique flottante

Opérations sur les flottants

Opérations élémentaire

- $x + y \longrightarrow \text{Arrondi}(\text{Arrondi}(x) + \text{Arrondi}(y))$
- $x - y \longrightarrow \text{Arrondi}(\text{Arrondi}(x) - \text{Arrondi}(y))$ ...

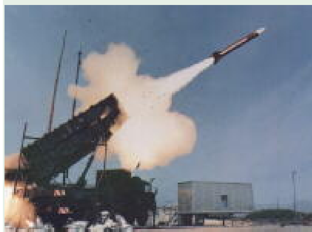
Attention

Plusieurs propriétés de l'arithmétique (associativité, distributivité, ...) ne sont plus valides en arithmétique flottante (DEMO) !

Conséquence : le même programme, compilé par deux compilateurs (optimiseurs) différents ne donne pas toujours *exactement* le même résultat.

Arithmétique flottante : conséquences

Arrondis - Anti-missile Patriot, 1991



- Tics de l'horloge interne en $1/10s$.
- Temps écoulé mesuré en nombre de tics (entier).
- Calcul de la position d'interception dépend du $\text{temps} = \text{nombre de tic} \times 1/10$ (erreur de 0.000000095 sur 1/10 en 24bits).
- Après 100h de fonctionnement, vitesse de $1676m/s$: $0.000000095 \times 100 \times 3600 \times 10 \times 1676 = 573m$

Dépassement - Ariane 5, 1996

- Vitesse horizontale en double convertie en entier signé sur 16 bits.
- Valeur supérieure à 32 767 $\Longrightarrow$ échec de la conversion.

Arithmétique flottante : annulation catastrophique

Catastrophic cancellation

Perte de précision qui résulte de la soustraction de deux nombres voisins.

Annulation catastrophique - Différences finies

Exemple pour l'approximation de la dérivée première par la méthode des différences finies avec un pas d'espace h trop petit :

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

DEMO

Racines d'un polynôme du second degré

Résolution de $ax^3 + bx^2 + c = 0$ par la méthode classique :

$$\Delta = b^2 - 4ac \text{ puis } x = \frac{\pm b + \sqrt{\Delta}}{2a}.$$

DEMO

Calcul de récurrences

Calcul des termes de la suite :

$$u_{n+1} = 4u_n - 1$$

avec $u_0 = 1/3$.
DEMO

Calcul de récurrences

Calcul des termes de la suite :

$$u_{n+1} = 3u_n - 1$$

avec $u_0 = 1/2$.
DEMO

Arithmétique flottante : erreurs d'arrondi

Mais qu'est ce que cette histoire ? ? ?

Mais qu'est ce que cette histoire ? ? ?

Explication :

1. $u_{n+1} = 4u_n - 1$ avec $u_0 = 1/3$:

$$\frac{1}{3} = \frac{1}{4}\sum_{i=0}^{\infty}\frac{1}{4^i} = \frac{1}{4}\sum_{i=0}^{\infty}\frac{1}{2^{2i}},$$

   et donc $u_0 = 1/3$ ne peut pas être représenté exactement en flottant. L'erreur initiale est amplifiée...

Mais qu'est ce que cette histoire ? ? ?

Explication :

1. $u_{n+1} = 4u_n - 1$ avec $u_0 = 1/3$:

$$\frac{1}{3} = \frac{1}{4}\sum_{i=0}^{\infty} \frac{1}{4^i} = \frac{1}{4}\sum_{i=0}^{\infty} \frac{1}{2^{2i}},$$

   et donc $u_0 = 1/3$ ne peut pas être représenté exactement en flottant. L'erreur initiale est amplifiée...

2. $u_{n+1} = 3u_n - 1$ avec $u_0 = 1/2$.

Arithmétique flottante : erreurs d'arrondi

Mais qu'est ce que cette histoire ? ? ?

Explication :

1. $u_{n+1} = 4u_n - 1$ avec $u_0 = 1/3$:

$$\frac{1}{3} = \frac{1}{4}\sum_{i=0}^{\infty}\frac{1}{4^i} = \frac{1}{4}\sum_{i=0}^{\infty}\frac{1}{2^{2i}},$$

   et donc $u_0 = 1/3$ ne peut pas être représenté exactement en flottant. L'erreur initiale est amplifiée...

2. $u_{n+1} = 3u_n - 1$ avec $u_0 = 1/2$.
   En base 2 :
   - $1/2$ s'écrit 0.1,
   - $3/2$ s'écrit 1.1

   donc le calcul est exact .

Arithmétique flottante : que peut-on calculer ?

Stabilité aux perturbations

On ne peut effectuer que des calculs pour lesquels la solution dépend *gentiment* des données (*problèmes bien posés*).

- Les nombres à virgule flottante doivent être regardés avec méfiance, mais ils n'ont pas empêché le développement du calcul et de ses applications : ce ne sont pas les erreurs d'arrondi qui limitent la validité de la prévision météorologique, pour ne citer que cet exemple.
- La stabilité des algorithmes vis-à-vis des petites perturbations doit être étudiée.

Exemple de problème mal posé (conditionnement élevé)

Résoudre de système linéaire $Ax = b$ en prenant pour A la matrice de Hilbert de taille n, avec $A_{i,j} = 1/(i + j - 1)$ (DEMO).

- La solution est dans $\mathbb{Q}$: calcul exact si on a les moyens de calculer dans $\mathbb{Q}$ (Exemple : bibliothèque LINBOX)
- On peut alors comparer la solution dans $\mathbb{Q}$ à la solution « flottante ».

Bibliographie

Nombres à virgule flottante : références bibliographiques

- *What every scientist should know about floating-point arithmetic*. David Goldberg.
  Texte disponible à de nombreux endroits, entre autres à :
  http://perso.ens-lyon.fr/jean-michel.muller/goldberg.pdf.
- *Handbook of Floating-Point Arithmetic*, Muller et collaborateurs (ENS Lyon).
- *Calcul mathématique avec Sage*. Casamayou, Alexandre et Connan, Guillaume et Dumont, Thierry et Fousse, Laurent et Maltey, François et Meulien, Matthias et Mezzarobba, Marc et Pernet, Clément et Thiéry, Nicolas et Zimmermann, Paul.