Cours Info - 12

Représentation des nombres en machine

D.Malka

MPSI 2015-2016

Sommaire

Sommaire

Principe de la numération par position

Représentation d'un nombre dans un système de numération par position :

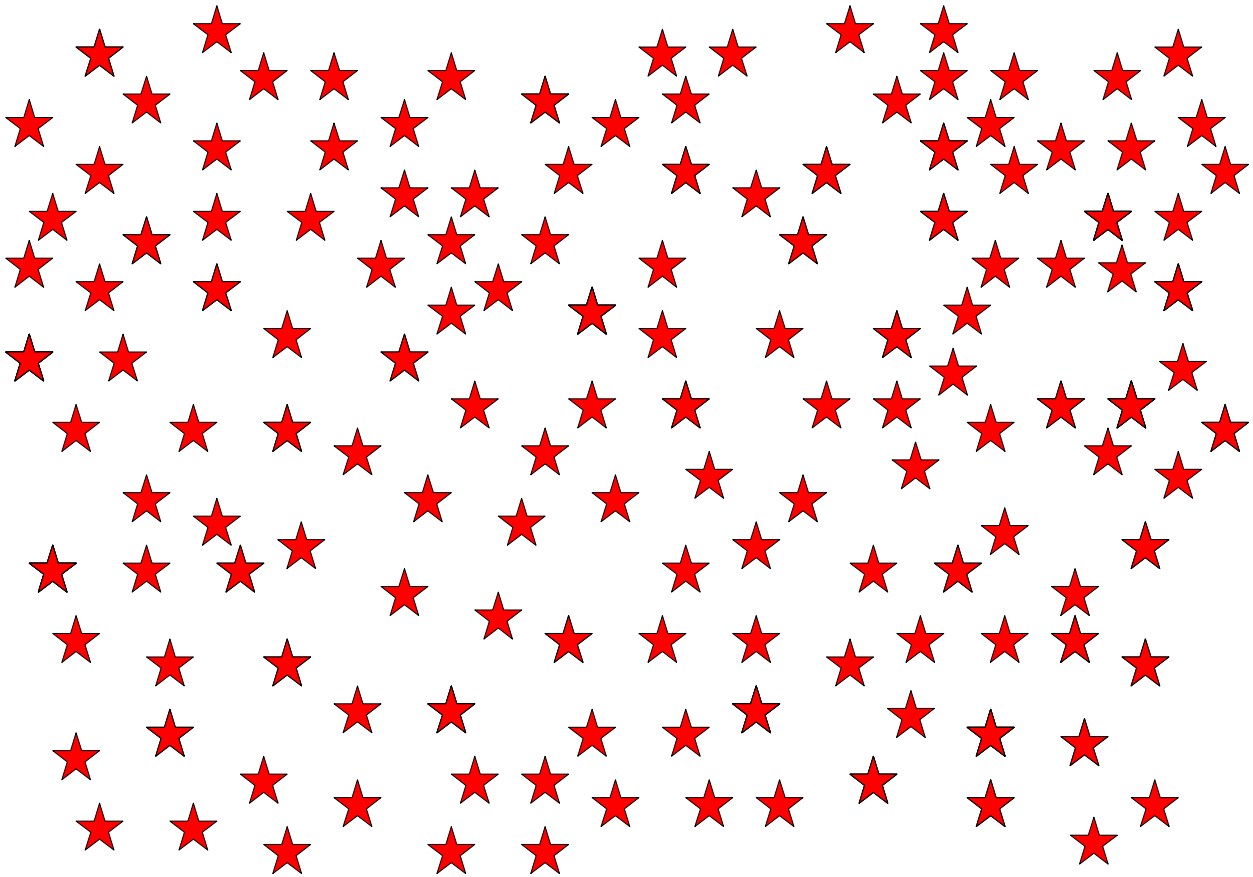
- **On dispose d'un jeu fini (souvent court) de chiffres pour écrire les nombres.**
- **La signification de chaque chiffre dépend de sa position dans le nombre.**

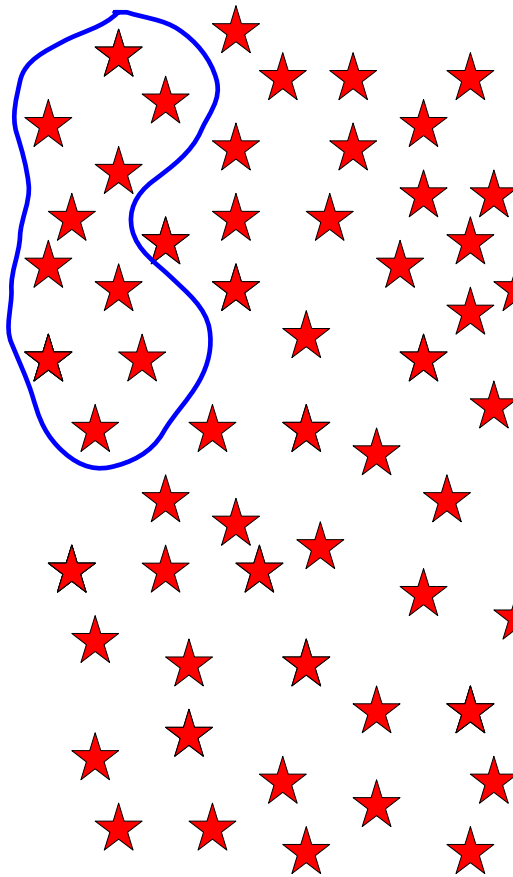
En base 10

- **On dispose de 10 chiffres : 0,1,2,3,4,5,6,7,8,9**
- **Dans le nombre 132 :**
  - **2 représente 2 unités**
  - **3 représente 3 dizaines**
  - **1 représente 1 centaine**

Détermination d'un nombre en base 10

Comptons les étoiles





Résultats :142

Détermination d'un nombre en base 10

Autrement dit :

$$142 = 2 + 4 \times 10 + 1 \times 100$$

ou encore :

142 = 2 unités + 4 paquets de 10 (4 dizaines) + 1 paquet de 10 paquets 10 (1 centaine)

Ecriture d'un nombre en base 10

Ecriture d'un nombre entier en base 10

$$N = \sum_{k=0}^{n} a_k \times 10^k$$

avec $k \in \mathbb{N}$ et $\forall k$, a_k un entier naturel tel que $0 \leq a_k \leq 9$.
On écrit alors le nombre :

$$a_n a_{n-1} .. a_k .... a_1 a_0$$

Ecriture d'un nombre en base 10

Exemple

Ecrire 1458 comme une combinaison linéaire de puissance de 10.

$$1458 = 1 \times 10^3 + 4 \times 10^2 + 5 \times 10 + 8 \times 1$$

La base 2

La base 2 est très utilisée en informatique du fait du codage binaire de l'information.

- **Base à deux chiffres : symboles** 0 **et** 1
- **Même règle d'écriture que la base** 10 **:** 0000**,** 0001**,** 0010**,** 00011**,** 0100 **...**

Ecriture d'un nombre en base 2

Ecriture d'un nombre entier en base 2

$$N = \sum_{k=0}^{n} a_k \times 2^k$$

avec $k \in \mathbb{N}$ et $\forall k$ a_k un entier naturel tel que $0 \leq a_k \leq 1$
On écrit alors le nombre :

$$a_n a_{n-1} .. a_k .... a_1 a_0$$

Conséquence : le plus grand entier codable sur n bits est $2^n - 1$.

Passage de la base 2 à la base 10

Exemple 1

Que vaut $N = (1101)_2$ en base 10 ?

$$N = 1 \times 1 + 0 \times 2 + 1 \times 2^2 + 1 \times 2^3$$

$$N = 1 + 0 + 4 + 8$$

$$N = 13$$

Passage de la base 2 à la base 10

Exemple 2

Que vaut $N = (01001110)_2$ en base 10 ?

$$N = 0 \times 1 + 1 \times 2 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 0 \times 2^5 + 1 \times 2^6 + 0 \times 2^7$$

$$N = 0 + 2 + 4 + 8 + 64$$

$$N = 78$$

Passage de la base 2 à la base 10

Exemple 3

Quel est le plus grand entier naturel N_{max} codable sur $32bits$:

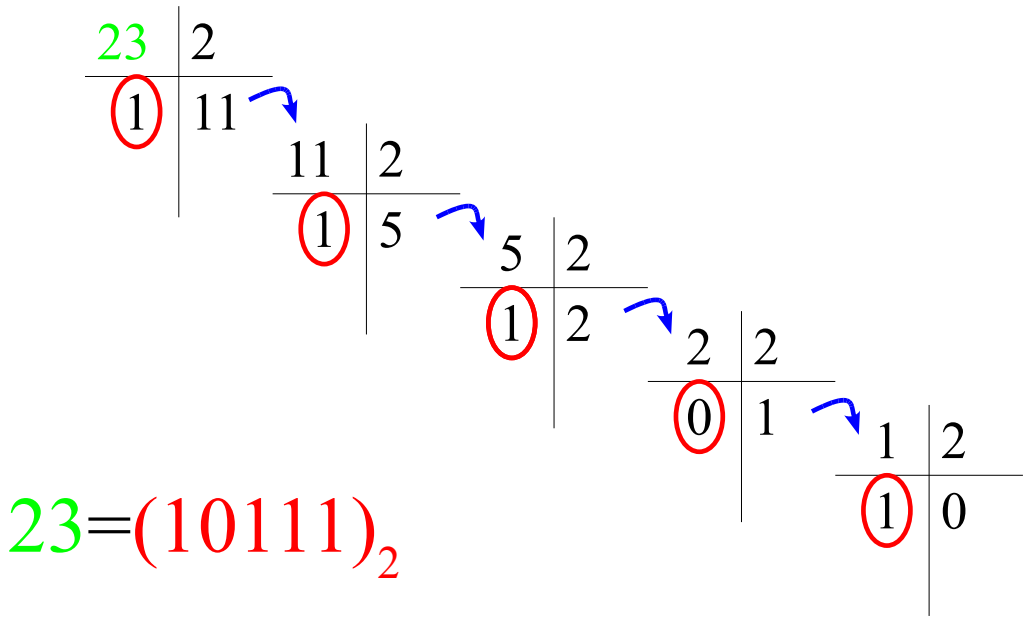
- écrit en base 2 ?
- écrit en base 10 ?
- $$N_{max} = (11111111111111111111111111111111)_2$$

  ou encore

  $$N_{max} = (10000000000000000000000000000000)_2 - 1$$
- $$N_{max} = 2^{32} - 1 = 4294967295$$

Passage de la base 10 à la base 2

Algorithme : diviser par 2 le nombre N puis diviser par 2 les quotients successifs jusqu'à ce que le quotient soit nul. La suite des restes est alors l'écriture binaire de N.



$23 = (10111)_2$

Ecriture d'un nombre en base b

Ecriture d'un nombre entier en base b

$$N = \sum_{k=0}^{n} a_k \times b^k$$

avec $k \in \mathbb{N}$ et $\forall k$ a_k un entier naturel tel que $0 \leq a_k \leq b-1$
On écrit alors le nombre :

$$a_n a_{n-1} .. a_k .... a_1 a_0$$

Conséquence : le plus grand entier codable sur n chiffres est $b^n - 1$.

Ecriture d'un nombre en base b

Exemple

Donner l'écriture décimale du nombre $N = (3221)_4$.

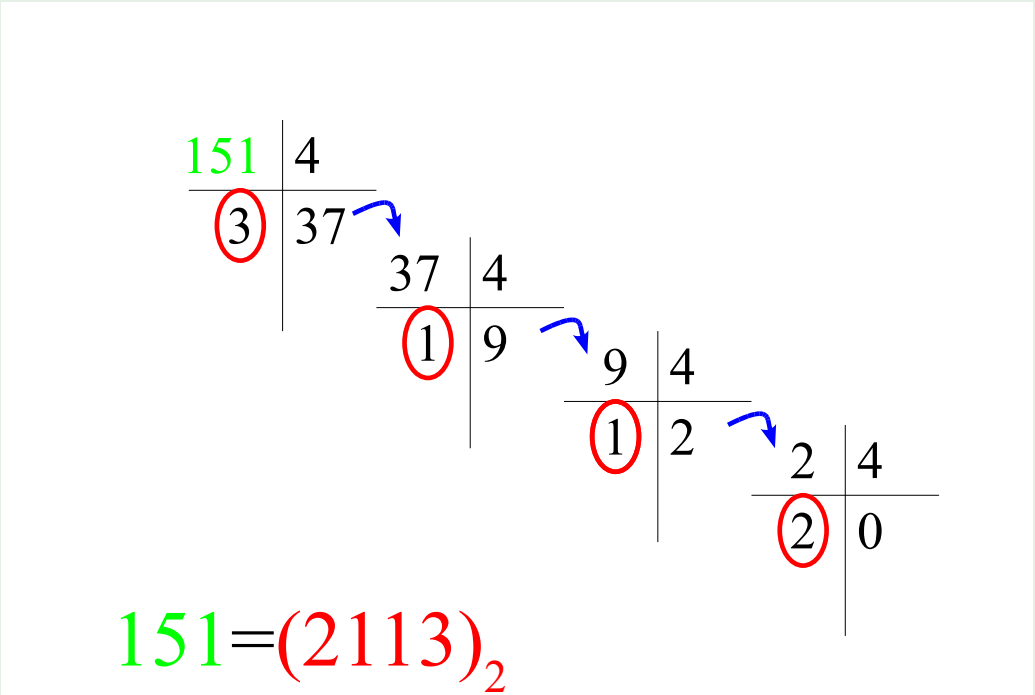
$$N = 1 \times 1 + 2 \times 4 + 2 \times 4^2 + 3 \times 4^3 = 233$$

Passage de la base décimale à la base b

Algorithme : diviser le nombre N puis les quotients successifs par b jusqu'à ce que le quotient soit nul. Le suite des restes successifs est la représentation de N en base b.

Exemple

Donner la représentation en base 4 du nombre 151.



Sommaire

Représentation des entiers naturels

Entiers naturels : type *int*. Entier non signés (i.e. positif).

Codage binaire de l'information $\rightarrow$ mémorisation et calcul avec des nombres représentés en bases 2.

Si processeur 32 bits : entiers codés sur 4 octets (soit 32 bits). Exemple :

00010100001111000010000100110010

Si processeur 64 bits : entiers codés sur 8 octets (soit 64 bits). Exemple :
00000000 00000000 00000000 00000000 00010100 00111100 00100001 00110010

Bit de poids fort / bit de poids fiable

Le *bit de poids fort* est, pour un nombre binaire, le bit ayant dans une représentation donnée la plus grande valeur (celui de gauche dans la représentation positionnelle habituelle).

Le *bit de poids faible* est, pour un nombre binaire, le bit ayant dans une représentation donnée la moindre valeur (celui de droite dans la représentation positionnelle habituelle).

Exemple

Identifier le bit de poids fort et le bit de poids faible dans 10010010.

Bit de poids fort : 1, bit de poids faible : 0.

Dépassement de capacité

Sur 32 bits, le plus grand nombre entier acceptable par le processeur est :

$$N = (11111111\,11111111\,11111111\,11111111)_2$$

$$\Leftrightarrow N = 2^{32} - 1 = 4294967295$$

Sur 64 bits, le plus grand nombre entier acceptable par le processeur est :

$$N = 11111111\,11111111 \ldots 11111111$$

$$\Leftrightarrow N = 2^{64} - 1 = 18446744073709551615$$

Dépassement de capacité

Pour les nombres au delà de ces valeurs : il y a *dépassement de capacité*.

Dans certains langage, au cours d'une opération donnant lieu à un dépassement de capacité, les bits surnuméraire sont perdus ! Et le résultat et faux !

Le stockage d'entier plus long est possible mais plus complexe : on parle de *long integer*.
Conversion `int->long` int automatique en Python.

Comment stocker les entiers relatifs ?

1ère idée : réserver un *bit de signe* (ici le bit de poids fort).

Exemple sur 1 octet (8 bits)

$$(00110110)_2 = 54$$

$$(10110110)_2 = -54$$

Le bit de poids fort vaut 1 si le nombre est négatif et 0 s'il est positif.

Inconvénients

- Zéro peut être représenté comme $(00000000)_2$ ou $(10000000)_2$
- L'addition classique ne fonctionne pas pour les nombres négatifs

Comment stocker les entiers relatifs ?

Solution retenue : *le complément à deux*.

Représentation de x en complément à deux sur n bits

- $(x)_2$, si x est positif
- $(2^n - |x|)_2$, si x est négatif

19 et -19 sur 8 bits

Représentation en complément à 2 ? $(19)_{10} = (00010011)_2, (-19)_{10} = (11101101)_2$

Comment stocker les entiers relatifs ?

Astuce pour calculer x négatif

On peut calculer x négatif en complément deux en suivant l'algorithme suivant (fondé sur $2^n - |x| = (2^n - 1) - |x| + 1$) :

- **exprimer** $|x|$ **en base 2,**
- **changer les** 0 **en** 1**, les** 1 **en** 0 **dans** $(|x|)_2$**,**
- **ajouter** 1 **au résultat.**

Exemple

-6 sur 4 bits ? valeur absolue de -6 : $6 = (0110)_2$ $0 \rightarrow 1$ et $1 \rightarrow 0$: 1001 ajout de 1 : 1010

Comment stocker les entiers relatifs ?

- Le bit de poids fort s indique aussi le signe.
- Soit v la valeur des $n-1$ bits de poids les plus faibles.
  - si $s = 0$, le nombre vaut v.
  - si $s = 1$, le nombre vaut $-2^{n-1} + v$.

Avantage

- Valeurs représentables de -2^{n-1} à $2^{n-1} - 1$.
- L'addition usuelle fonctionne.

Sommaire

Problématique

Erreur de représentation

CODE PYTHON

```
In [1]: 1-1/3==2/3
Out[1]: False
```

Tester le programme suivant :

```
1 s=0
2 while s!=1:
3     s=s+1/10
4     print(s)
```

. . . le résultat est surprenant !

Problématique

Absorption

CODE PYTHON

```
In [2]: a=1/11223344556677; a
Out[2]: 8.910000000000706e-14

In [3]: 1/(a-(a+a**3))
Out[3]: ZeroDivisionError Traceback (most recent call last)
<ipython-input-86-5859010bc9f2> in <module>() ---> 1 1/(a-(a+a**3))
ZeroDivisionError: float division by zero

In [4]: a**3
Out[4]: 7.073479710001682e-40

In [5]: a-(a+a**3)
Out[5]: 0.0

In [6]: (a-a)-a**3
Out[6]: -7.073479710001682e-40
```

L'addition n'est plus associative ?

Problématique

Cancellation

CODE PYTHON

```
In [7]: a=123456789;b=123456788
In [8]: 1/a,1/b
Out[7]: (8.100000073710001e-09, 8.100000139320002e-09)

In [9]: 1/a-1/b
Out[8]: -6.561000080657073e-17

In [10]: (b-a)/(a*b)
Out[9]: -6.561000172554304e-17
```

Problématique

Dépassement de capacité

CODE PYTHON

```
In [11]: 1e-200
Out[10]: 1e-200

In [12]: 1e-400
Out[11]: 0

In [13]: 1e308
Out[12]: 1e308

In [14]: 1e500
Out[13]: inf
```

Problématique

Quelle est l'origine des phénomènes observés ?

$\Longrightarrow$ Mode de représentation des réels.

L'ensemble des réels $\mathbb{R}$ est approché en machine par l'ensemble des flottants $\mathbb{F}$.
Soit $a \in \mathbb{R}$, on notera sa représentation en machine $fl(a)$ $(\in \mathbb{F})$.

Nombres décimaux

Ecriture d'un nombre décimal en base 10

$$N = \sum_{k=-m}^{n} a_k \times 10^k$$

avec $k \in \mathbb{Z}$ et $\forall k$ a_k un entier naturel tel que $0 \leq a_k \leq 9$.
On écrit alors le nombre :

$$a_n a_{n-1} .. a_k .... a_1 a_0 , a_{-1} a_{-2} ... a_{-m}$$

Ex : $1,12 = 1 \times 10^0 + 1 \times 10^{-1} + 2 \times 10^{-2}$

Une première solution : nombre binaire à virgule

Sur le modèle des nombres décimaux, on réserve des bits au stockage de la partie non entière du nombre.

Ecriture d'un nombre binaire à virgule

$$N = \sum_{k=-m}^{n} a_k \times 2^k$$

avec $k \in \mathbb{Z}$ et $\forall k$ a_k un entier naturel tel que $0 \leq a_k \leq 1$.
On écrit alors le nombre :

$$(a_n a_{n-1}..a_k....a_1 a_0, a_{-1} a_{-2}...a_{-m})_b$$

Ex : $1,25 = 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (1,01)_2$

Une première solution : nombre binaire à virgule

Exemple

- Décomposer 103,456 en puissance de 10 :
  $1 \times 10^2 + 0 \times 10 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$
- Décomposer 11,52 en puissance de 2 puis l'écrire en base 2 :
  $1011,1000100\ldots$

Rappel : écriture scientifique en base 10

Ecriture scientifique des nombre en base 10

Soit un réel r. L'écriture scientifique de r est :

$$r = a.10^z$$

où $1 \leq a < 10 \in \mathbb{R}$ et $z \in \mathbb{Z}$.

Exemple

Ecriture scientifique de $3562,2145$:

$$3,56221465.10^3$$

Nombres à virgule flottante

On représente le nombre sur le modèle de l'écriture scientifique mais en base 2.

Nombre à virgules flottantes (*floats*)

Soit N un nombre à virgule. N est représenté en mémoire sous la forme d'un nombre à virgule flottante

$$N = (-1)^s.1,m.2^e \quad e \in \mathbb{Z}$$

Nombres à virgule flottante

Nombre à virgules flottantes (*floats*

Le codage se fait sur 64 *bits* (processeur 64 *bits*). On l'écrit sous la forme :

$$se'm$$

Pour les nombres normalisés :

- s **est le bit de signe :** 0 **pour positif,** 1 **pour négatif.**
- m **est la *mantisse* : nombre binaire à virgule codée sur** 52 *bits*, **avec** $m < 1$**.**
- e' **est l'exposant** e **décalé de** 1023 **: nombre entier naturel codé sur** 11 *bits* **:** $1 \leq e \leq 2046$**.**

Nombres à virgule flottante (*floats*)

Exemples

Que vaut 1110000001011110.....0000 en base 10 ?

- **bit de signe** 1 **donc nombre négatif** $-$
- **exposant** $e' = 11000000101 = 1 \times 2^{10} + 2^9 + 2^2 + 2^0 = 1541$ **donc** $e = e' - 1023 = 518$
- **mantisse** $m = 1110.....0000 = 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} ....0 \times 2^{-52} = 0,875$
- $N = -1,875.2^{518} = -2,185...10^{156}$

Valeurs spéciales

Valeurs	s	m	e'	e
$+0$	0	000...00	000...00 (0)	-1023
-0	1	000...00	000...00 (0)	-1023
$+\infty$	0	000...00	111...11 (2047)	1024
$-\infty$	1	000...00	111...11 (2047)	1024
NAN	1 ou 0	$\neq 0$	111...11 (2047)	1024
Nombres dénormalisés	1 ou 0	$\neq 0$	000...00 (0)	-1023

Nombre dénormalisés : $N = 0, m.2^{-1022}$.

Problèmes posés les calculs en flottants

- Dépassements de capacité
- Erreur d'affectation : $0,1 = 0,0001100110011001100110011\ldots$
- Absorption : $1 + 10^{-98} = 1$
- Cancellation : $0.01002566478 - 0.01002566593 = -0.0000000115$

Dépassement de capacité

overflow

- $e > 1023$ **non représentable**
- $e = 1023$ **et mantisse>**11111.....11111 **non représentable**
- **les calculs donnant de tels nombres donnent lieu à un dépassement de capacité (*overflow*)**
- **renvoie d'un infini ou** *NaN* **(Not a Number) par la machine**

Dépassement de capacité

underflow

- **exposant<-1023 non représentable**
- **nombre inférieur au plus petit nombre dénormalisé représentable**
- **les calculs donnant de tels nombres donnent lieu à un soupassement de capacité (*underflow*)**
- **renvoie d'un 0 ou d'une erreur**

Erreur d'affectation

Exemple

$$\frac{1}{10} = 0001100110011\ldots$$

Précision machine ε

Si la mantisse est trop longue (taille supérieure à t) alors, on suppose qu'elle est tronquée alors, la plus grande erreur relative ε qu'on peut commettre vaut :

$$\varepsilon \leq 2^{-52} \sim 10^{-16}$$

<Erreur d'affectation maximale

Soit $a \in \mathbb{R}$ et $fl(a)$ sa représentation dans $\mathbb{F}$. Alors l'erreur relative sur a est au plus de ε

$$\frac{|fl(a) - a|}{|a|} \leq \varepsilon$$

Absorption

Exemple

$$1 + 10^{-98} = 1$$

Absorption

Si $\alpha < \varepsilon$ alors $fl(1+\alpha) = 1$. Plus généralement, si $\frac{A}{B} < \varepsilon$ alors $fl(A+B) = B$.

Exemple2

$2 - (2 + 10^{-20}) = 0$ tandis que $(2-2) + 10^{-20} = 10^{-20}$: l'addition n'est plus associative !

Cancellation

Pour simplifier, nous raisonnons en base 10 plutôt qu'en base 2.

Exemple

$$1,36584.10^8 - 1,36583.10^8 = 0,00001.10^8 = 1,00000.10^3$$

Supposons une erreur d'affectation de 10^3 sur $1,36584.10^8$ et $1,36583.10^8$: l'erreur relative est donc de l'ordre de $\frac{10^3}{10^8} = 10^{-5}$.

Après soustraction, l'erreur absolue sur $1,00000.10^3$ est encore de l'ordre de 10^3 : l'erreur relative est donc de l'ordre $\frac{10^3}{10^8} = 1 = 100\%$. Suite à la soustraction, l'erreur relative a explosé.

Cancellation

Lorsqu'on calcule la différence de deux flottants très proches, l'erreur relative augmente considérablement : c'est la cancellation.

Eviter de soustraire des nombres très proches ! Ex : calculer $1/x - 1/(x+1)$ comme $1/(x(x+1))$.

Quelques conseils

Quelques conseils

1. Eviter la soustraction de deux nombres très proches (Risque de Cancellation)
2. Eviter l'addition de deux nombres très différents (Risque d'Absorption)
3. L'arithmétique des réels ne s'appliquent pas aux flottants :
   1. pas de comparaison avec les flottants
   2. en général, $(x+y)+z \neq x+(y+z)$: l'addition n'est pas associative.

Peut-on calculer avec les flottants ?

Calcul avec les flottants

Les calculs avec les flottants sont toujours approchés !

On peut calculer avec les flottants . . . à condition :

- de réaliser judicieusement les calculs,
- d'évaluer l'erreur commise, la convergence,
- d'interpréter de façon critique le résultat obtenu.