representation_flottante_des_nombres_2H

June 13, 2019

1 Représentation des nombres dans un ordinateur

1.1 Numérisation

```
In [ ]: 0.1 + 0.2
```

- Conversion d'un monde analogique (dans lequel l'information est continue) vers un monde de 0 et 1 appelé **système binaire**
- La machine ne contient que des éléments électriques pour lesquels il n'y a que deux états possibles, soit il y a un signal soit il n'y en a pas. Elle ne comprend donc que des informations binaires: 0 et 1. Le codage consiste à combiner plusieurs bits.
- Le **bit** est la plus petite information manipulable par un ordinateur
- **L'octet** est une unité d'information (appelé mot) composée de 8 bits. Il permet par exemple de stocker un caractère comme une lettre ou un chiffre.

1.2 Représentation binaire des nombres entiers

1.2.1 Numération de position des entiers en base 10

Dans un tel système la valeur que l'on attribue à chaque chiffre d'un nombre dépend * de la valeur qu'il représente quand il est seul * de sa position

Par exemple le nombre 2017 s'écrit:

$$2017 = 2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 7 \times 10^0$$

1.2.2 Généralisation

À construire

1.2.3 Conversion binaire $\longrightarrow$ décimal

Conversion du nombre binaire: $p = 10011010_2$

Indice i	7	6	5	4	3	2	1	0
d_i	1	0	0	1	1	0	1	0
β^i	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

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

La conversion en base 10 donne: $p = 2^7 + 2^4 + 2^3 + 2^1 = 154_{10}$

- Dans l'écriture positionnelle, le bit de poids faible se trouve à droite et le bit de poids fort à gauche
- Le bit de poids faible représente l'unité des entiers

1.2.4 Conversion décimal $\longrightarrow$ binaire

Pour exprimer un nombre décimal en binaire, il suffit d'effectuer une succession de divisions par 2. Le résultat binaire étant la juxtaposition des restes.

Voici la façon de procéder avec le nombre 154

1.2.5 Problème : Impossibilité machine de représenter un ensemble infini

- Un ordinateur ne permet pas de représenter l'ensemble complet des réels. L'électronique impose une limite intrinsèque à la taille de la mantisse et de l'exposant. Cet ensemble est noté $\mathbb{F}(\beta, p, e_{min}, e_{max})$
- Les calculs réalisés sur les nombres réels sont de ce fait entachés d'erreurs en raison de la précision limitée de la représentation des réels dans la machine.
- Nécessité de réaliser une phase d'arrondi (arrondi par *défaut*, par *excès* ou *au plus près*). Ce dernier introduit par la norme IEEE 754 représente un progrès considérable qui a permis d'obtenir une compatibilité totale d'un même algorithme d'une machine à l'autre.

Par exemple en base 10, le résultat de la division de $\frac{10.0}{3.0}$ ne pas pas être écrit sur un nombre fini de chiffres.

1.2.6 Exemple d'un système fini normalisé positif

Considérons l'ensemble de nombres suivant: $\mathbb{F}(2, 2, -2, 1)$

L'ensemble des nombres que l'on va pouvoir écrire est:

$$\begin{aligned}
1,0 \times 2^{-2} &= 0,25_{10} \\
1,1 \times 2^{-2} &= 0,375_{10} \\
1,0 \times 2^{-1} &= 0,5_{10} \\
1,1 \times 2^{-1} &= 0,75_{10} \\
1,0 \times 2^{0} &= 1_{10} \\
1,1 \times 2^{0} &= 1,5_{10} \\
1,0 \times 2^{1} &= 2_{10} \\
1,1 \times 2^{1} &= 3_{10}
\end{aligned}$$

Comment représenter les valeurs 5; 0,1; 1,2 et 1,25 dans l'ensemble $\mathbb{F}$?

1.2.7 Quels conséquences pour les calculs flottants avec un ordinateur?

Attention aux résultats de vos additions et multiplications

```
In [ ]: ex1 = (1.1 + 2.2) - 3.3
        ex2 = (3.1 * 2.3) * 1.1
        ex3 = 3.1 * (2.3 * 1.1)

In [ ]: print(ex1)
```

```
In [ ]: print(ex2, ex3)
```

Connaître les limites de la représentation flottante des nombres

1.2.8 Flottant le plus proche d'un nombre réel

Si $x \in \mathbb{R}$, alors x n'appartient pas forcément à $\mathbb{F}(\beta, p, e_{min}, e_{max})$. Dans ce cas on associe à x une approximation notée x_f de telle sorte que $x_f \in \mathbb{F}(\beta, p, e_{min}, e_{max})$

- x_{fmax} et x_{fmin} respectivement le plus grand et le plus petit nombre positif de $\mathbb{F}(\beta, p, e_{min}, e_{max})$
- Si $|x| > x_{fmax}$ alors on est dans le cas d'un **overflow**
- Si $|x| < x_{fmin}$ alors on est dans le cas d'un **undeflow**
- Si $x_{fmin} \leq |x| \leq x_{fmax}$ alors x_f est le nombre le plus proche de x.

$$\text{Soit} \qquad x_f = \pm x_0, x_1 x_2 \dots x_{p-1} x_p ... x_j ... \times \beta^e$$

- Si $x_p < \frac{\beta}{2}$, x_f est arrondi au flottant le plus proche en dessous

$$x_f = \pm x_0, x_1 x_2 \dots x_{p-1} \times \beta^e$$

- Si $x_p \geqslant \frac{\beta}{2}$ (avec $x_j \neq 0$), x_f est arrondi au flottant le plus proche au dessus

$$x_f = \pm x_0, x_1 x_2 \dots x_{p-2} (x_{p-1} + 1) \times \beta^e$$

- Si $x_p = \dfrac{\beta}{2}$ (avec $x_j = 0, \forall j > p$), on distingue deux façons d'arrondir

  1. x_f est donné par $x_p \geqslant \dfrac{\beta}{2}$
  2. on arrondit pour que le dernier chiffre de x_f soit **pair**, méthode utilisée par la norme IEEE-754.
     - Si x_{p-1} est pair x_f est donné par $x_p < \dfrac{\beta}{2}$
     - Si x_{p-1} est impair x_f est donné par $x_p \geqslant \dfrac{\beta}{2}$

1.2.9 Codage en virgule flottante (IEEE 754)}

Une autre façon de représenter un sous ensemble plus large des réels. Le codage normalisé (IEEE 754) sur 32-bits d'un flottants doit respecter les règles suivantes :

- 1-bit (le poids fort) pour le signe
- 8-bits pour l'exposant
- 23-bits pour la mantisse

1.2.10 En conclusion

La conséquence de tout ceci est que * les calculs se font avec une certaine précision intrinsèque (qui peut être a priori très suffisante entre **7 et 15 chiffres significatifs**), * il faut rester très prudent lorsque l'on traite des nombres qui prennent des **valeurs très différentes** (plusieurs ordres de grandeur de différence) * ou lors de l'utilisation d'une instruction du genre: **if (b-a)==0:** qui est très aléatoire