# CHAPITRE 2 : REPRÉSENTATION DES RÉELS

## I PRINCIPE DE LA REPRÉSENTATION DES NOMBRES RÉELS EN MÉMOIRE

### I.1 L'arithmétique flottante

La notation binaire permet de représenter des nombres à virgules. En notation décimale, les chiffres à gauche de la virgule représentent les unités, dizaines, centaines etc... et ceux à droite de la virgule, les dixièmes, centièmes etc...
Le principe est le même en binaire : les chiffres à droite de la virgule représentent les demis, quarts, huitièmes, seixièmes etc...
Le nombre un et un quart (1,25 en décimale) s'écrit en binaire $1,01$
Cette manière de faire ne permet pas cependant de représenter des nombres très grands ou très petits. On utilise donc une autre représentation similaire à la "notation scientifique" des calculatrices, sauf que celle-ci est en base deux plutôt qu'en base dix.
Un nombre est représenté sous la forme $sm2^n$ où $s$ est le **signe** du nombre, $n$ son **exposant** et $m$ sa **mantisse**. Le signe est + ou -, l'exposant un entier relatif et la mantisse un nombre à virgule, compris entre 1 inclus et 2 exclu.

Quand on utilise 64 bits pour représenter un nombre à virgule, on utilise 1 bit pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse.

- Le signe + est représenté par 0 et le signe - par 1.
- L'exposant $n$ est un entier relatif compris entre -1022 et 1023 ; on le représente comme l'entier naturel $n+1023$, qui est compris entre 1 et 2046. Les deux entiers naturels 0 et 2047 sont réservés pour des situations exceptionnelles ($+\infty$, $-\infty$, $NaN$, etc...).
- La mantisse $m$ est un nombre binaire à virgule compris entre 1 inclus et 2 exclu, comprenant 52 chiffres après la virgule. En effet, le seul chiffre avant la virgule est 1 ; il est donc inutile de le représenter et on utilise les 52 bits pour représenter les 52 chiffres après la virgule. Ceci donne, pour la mantisse, une précision réelle de 53 bits.

Les nombres représentés sous cette forme sont appelés **nombres à virgule flottante** : la virgule de la mantisse pouvant être "déplacée" par le biais de l'exposant.

Ainsi, si l'on note $s$ le bit de signe, $e_1...e_{11}$ les bits d'exposant et $m_1...m_{52}$ les bits de la mantisse, le nombre codé vaut :

$$\boxed{(-1)^s \times 2^{\underline{e_1...e_{11}}-1023} \times \left(1+\sum_{i=1}^{52} m_i \frac{1}{2^i}\right).}$$

### I.2 Exercices

Exercice 1 :
Trouver le nombre à virgule flottante représenté par le mot :
1100010001101001011110000111000000000000000000000000000000000000

*Réponse :* $-\dfrac{206727}{131072} \times 2^{71} \simeq -3,724... \times 10^{21}$

Exercice 2 :
Trouver le nombre à virgule flottante représenté par le mot binaire :
0001000000111101001110010101100000000000000000000000000000000000

Exercice 3 :

a. Quel est le plus grand nombre que l'on peut représenter en virgule flottante sur 64 bits ?
b. Quel est le plus petit nombre (donc négatif) que l'on peut représenter en virgule flottante sur 64 bits ?

### I.3 Cas particulier

Si la mantisse est censée toujours commencer par un 1 implicite, il n'est en principe pas possible de représenter le zéro. Par convention, on décide qu'un nombre vaut zéro si et seulement si tous les bits de son exposant et de sa mantisse valent zéro. Il reste alors le choix pour le bit du signe, il y a donc un zéro positif et un zéro négatif dans les nombres à virgule flottante.

## II CONSÉQUENCES DE LA REPRÉSENTATION LIMITÉE DES NOMBRES RÉELS EN MACHINE

### II.1 Dépassements de capacité et problèmes de précision

De même que les entiers, les nombres à virgule flottante possèdent certaines limites inévitables.
Étant représentés sur un nombre donné de bits, il existe forcément un nombre maximal représentable dans ce format. Plus précisément, 64 bits ne suffisent plus si la représentation du nombre demande :

- un exposant supérieur à 1023, qui est le plus grand exposant représentable sur 11 bits ;
- ou un exposant égal à 1023 et une mantisse supérieure à la plus grande mantisse représentable sur 52 bits c'est à dire 1 (implicite) suivi de 52 fois le chiffre 1 après la virgule.

Tout calcul dont le résultat dépasse cette limite produit une situation de **dépassement arithmétique** (ou **overflow** en anglais).

Une situation similaire se produit lorsque l'on veut représenter un nombre trop proche de 0 ;

ayant un exposant inférieur à -1022, qui est le plus petit exposant représentable sur 11 bits.

On parle alors de **dépassement par valeurs inférieures** (ou **underflow** en anglais).

### II.2 Les arrondis

Il est rare que le résultat d'un calcul faisant intervenir des nombres à virgule flottante donne un résultat représentable exactement sur 64 bits.
Par exemple, le nombre 0,4 admet pour développement en base 2 :

$$\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^6}+\frac{1}{2^7}+\frac{1}{2^{10}}+\frac{1}{2^{11}}+\frac{1}{2^{14}}+\frac{1}{2^{15}}+...=\underline{0,011001100110011...}$$

qui est un développement infini périodique.
On ne peut donc pas le représenter de façon exacte.
La représentation en virgule flottante sera donc forcément une valeur approchée de ce nombre.
Sur 64 bits, cela donnera :

**signe** : 0
**exposant** : 01111111101
**mantisse** : 1001100110011001100110011001100110011001100110011001

arrondie à $1001100110011001100110011001100110011001100110011010$
car le premier bit non représenté est un 1.

Ainsi, la valeur approchée choisie pour 0,4 est la suivante :

$$0,0110011001100110011001100110011001100110011001100110011010$$

$$= \frac{7205759403792794}{18014398509481984}$$

$$\simeq 0,4000000000000000022204460492503$$

D'autres erreurs d'arrondis se présentent lorsque l'on effectue des calculs entre nombres dont les ordres de grandeur sont très différents.

$1 + 2^{-54}$ demanderait 54 bits de mantisse pour être représenté exactement ; il est donc arrondi à 1.
Ainsi, en Python :

```
>>> 1 + 2**-54 -1
0.0
```

Alors que :

```
>>> 1-1 +2**-54
5.551115123125783e-17
```

L'addition sur les nombres à virgule flottante n'est donc pas **associative**.

On retiendra également qu'il n'est pas possible de savoir de façon certaine si le resultat d'un calcul est égal à sa valeur théorique. Un test du type $a == b$ n'a donc en général pas de sens si $a$ et $b$ sont deux nombres à virgule flottante, puisque ceux-ci ont pu subir des erreurs d'arrondis.
Ainsi, en Python :

```
>>> from math import *
>>> cos(pi/2)==0
False
```

### II.3 Exercices

Exercice 4 :
Par quel mot binaire est représenté en machine le nombre entier 13 ?
Et le nombre à virgule flottante 13,0 ?

Exercice 5 :
Déterminer l'écriture binaire et une valeur décimale approchée du plus grand nombre à virgule flottante sur 64 bits strictement inférieur à 1.
Même question avec le plus petit nombre strictement supérieur à 1.