Chapitre 5

Représentation des données

Représentation approximative des réels

« Tout cela est-il bien réel ? »

I De la calculatrice à l'ordinateur

Dans une machine on *ne peut pas* représenter tous les nombres réels car la majorité a une écriture décimale illimitée et parmi celle-ci la majorité a une écriture décimale illimitée sans qu'aucun motif se répète, comme c'est le cas pour $\pi \approx 3{,}141\,592\,653\,589\,793$.
Ceci dit on peut donner une valeur approchée d'un nombre réel r en écriture décimale en utilisant l'*écriture scientifique* vue au collège :

$$\boxed{r \approx (-1)^s \times d \times 10^e}$$

- s vaut 0 ou 1.
- d est un nombre décimal entre 1 inclus et 10 exclu.
- e est un entier relatif.

Ainsi, avec la calculatrice, on obtient :

$$\boxed{5,4^{-5} \approx (-1)^0 \times 2,177866231 \times 10^{-4}}$$

Dans ce cas, le nombre d comporte 10 chiffres décimaux, appelés **chiffres significatifs**.
Dans un ordinateur ne travaille pas avec des écritures décimales, mais avec des écritures **dyadiques** (l'équivalent des nombres décimaux en base 2).

Exemple : nombre décimal / nombre dyadique

- Considérons le nombre $x = 53,14$ (écrit en base 10). C'est un **nombre décimal** et on peut écrire :

$$\begin{aligned} x &= 53 + 1 \times 0,1 + 4 \times 0,04 \\ &= 5 \times 10^1 + 3 \times 10^0 + 1 \times 10^{-1} + 4 \times 10^{-2} \end{aligned}$$

- Les nombres dyadiques sont l'équivalent des nombres décimaux en base 2 : considérons $y = (101,011)_2$, on peut écrire :

$$\begin{aligned} y &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} \\ &= 4 + 1 + 0,25 + 0,125 \\ &= 5,375 \end{aligned}$$

Exemple : écriture scientifique pour un nombre dyadique

- En reprenant l'exemple précédent on a $53,14 = 5,314 \times 10^1$, c'est une écriture scientifique.
- Pour $(101,011)_2$, en se rappelant que multiplier par 2 décale la virgule d'un cran vers la gauche, on a $(101,011)_2 = (1,01011)_2 \times 2^2$.

Il faut donc décider d'un **format de représentation** des nombres dyadiques dans l'ordinateur :

- Combien de bits significatifs pour le nombre dyadique ?
- Quelle est la plage de valeurs pour l'exposant ?

II Le format IEEE 754

Ce format est une norme pour représenter les nombres dyadiques (notamment par PYTHON, en format 64 bits).
Par simplicité commençons par le format 32 bits (4 octets, donc). Voici comment cela fonctionne :

Définition

On considère un mot de 32 bits.

- Le premier bit s, indique le signe.
- Les 8 bits suivants $e_7 e_6 \dots e_0$ servent à coder l'exposant e de 2, qui vaut $e = (e_7 \dots e_0)_2 - 127$.
- Les 23 bits restants $m_1 \dots m_{23}$ servent à coder la **mantisse**, notons $m = (1, m_1 \dots m_{23})_2$.

- En définitive, ce mot de 32 bits représente

$$\boxed{(-1)^s \times m \times 2^e}$$

Ce qu'on peut noter

$$se_7 \dots e_0 m_1 \dots m_{23} \mapsto (-1)^s \times (1, m_1 \dots m_{23})_2 \times 2^{(e_7 \dots e_0)_2 - 127}$$

$se_7 \dots e_0 m_1 \dots m_{23}$ s'appelle une écriture **normalisée**.

En format 64 bits, les nombres sont représentés sur 8 octets : 1 bit de signe, 11 bits d'exposant (donc une plage de -1023 à 1024) et 52 bits de mantisse. Le principe est le même qu'en 32 bits.

Exemple : des 32 bits au nombre

Que représente 1 $\underbrace{0100\ 0011}$ $\underbrace{1110\ 0000\ 0000\ 0000\ 0000\ 000}$?

- $s = 1$.
- $e = (0100\ 0011)_2 - 127 = 67 - 127 = -60$
- $m = (\boxed{1}, 11100000000000000000000)_2 = 1 + 2^{-1} + 2^{-2} + 2^{-3} = 1,875$
- $1\ 0100\ 0011\ 1110\ 0000\ 0000\ 0000\ 0000\ 000 \rightsquigarrow (-1)^1 \times 1,875 \times 2^{-60}$

Cela fait environ $-1,6263032587282567 \times 10^{-18}$

III Les limitations du format IEEE 754

Lorsqu'un format de représentation en virgule flottante est choisi (32 ou 64 bits), on ne peut pas représenter tous les nombres réels : il y a un plus grand nombre représentable (et son opposé pour les nombres négatifs) et un « plus petit nombre positif représentable » (le plus proche de zéro possible). De plus **on a automatiquement des valeurs approchées si le nombre que l'on veut représenter n'est pas de de la forme $\frac{a}{2^n}$, avec $a \in \mathbb{Z}$ et $n \in \mathbb{N}$.**

Par exemple, un nombre aussi simple que 0,1 (c'est-à-dire $\frac{1}{10}$) n'est pas de la forme $\frac{a}{2^n}$, avec $a \in \mathbb{Z}$ et $n \in \mathbb{N}$, donc son écriture dyadique ne termine pas .
On peut montrer que :

$$(0,1)_{10} = (0,0001\ 1001\ 1001\ 1001\ \dots)_2$$

C'est l'équivalent dyadique de

$$\frac{1}{3} = (0,3333\dots)_{10}$$

Et puisque PYTHON ne peut pas représenter intégralement le nombre 0,1 il l'approche du mieux qu'il peut : en fait pour PYTHON la valeur de 0,1 est :

`0.1000000000000000055511151231257827021181583404541015625`

Mais celui-ci a la gentillesse d'afficher `0.1`.

Il faut se résigner à accepter les erreurs d'arrondis :

Shell Python

```
>>> 0.1 + 0.1 + 0.1
0.30000000000000004
```

Cet exemple prouve que **tester l'égalité de 2 `float` n'a pas d'utilité**. On aura plutôt intérêt à **tester si leur différence est très petite**.

De même, l'addition de plusieurs `float` donne un résultat qui peut dépendre de l'ordre dans lequel on les ajoute. Elle **n'est pas non plus associative** :

on peut avoir `a + (b + c) ≠ (a + b) + c`.

Les erreurs d'arrondis se cumulent. Pour les minimiser on aura intérêt à appliquer la règle suivante :

Propriété : Règle de la photo de classe

Dans une somme de `float`, l'erreur est minimisée quand on commence par ajouter les termes de plus petite valeur absolue.

Exercices

Exercice 1

Donner l'écriture décimale des nombres suivants

a. $(101,1)_2$

b. $(1,011)_2$

c. $(0,1111\ 111)_2$ en remarquant que c'est « $(111\ 1111)_2$ divisé par 2^7.

Exercice 2

Écrire en base 2 les nombres suivants :

a. 3,5

b. 7,75

c. 27,625

Exercice 3

On considère le format IEEE 754 64 bits : 1 bit de signe, 11 bits d'exposant (donc une plage de -1023 à 1024) et 52 bits de mantisse.

1. Quel est le plus grand nombre positif représentable ?

2. Quel est le plus petit nombre positif représentable ?

Exercice 4

Écrire un programme déterminant le plus petit entier n pour lequel PYTHON considère que $1 + 2^{-n} = 1$.
Faire le lien avec le format IEEE 754 64 bits.

Exercice 5

Faire calculer `1+2**(-53)-1` puis `1-1+2**(-53)`.
Que remarque-t-on ?
Comment, au regard de l'exercice 1, expliquer ce résultat ?

Exercice 6

Écrire un programme déterminant le plus petit entier n pour lequel PYTHON considère que $2^{-n} = 0$.
En faisant le lien avec le format IEEE 754 64 bits, quelle valeur devrait-on trouver ?
Quelle explication peut-on imaginer (sachant qu'en PYTHON, les `float` sont bien codés sur 64 bits) ?

Exercice 7

On peut prouver (c'est dur) que

$$\sum_{n=1}^{+\infty} \frac{1}{n^4} = \frac{\pi^4}{90}$$

Appelons c cette constante.

1. Calculer à l'aide d'un script $S = \sum_{n=1}^{10^6} \frac{1}{n^4}$.

Ajoute-t-on des termes de plus en plus petits ou de plus en plus grands ?
Continuer le script pour afficher $c - S$ (on pourra utiliser `from math import pi`).

2. Calculer S « dans l'autre sens ».
Afficher $c - S$.

3. Qu'illustre cet exercice ?

Exercice 8

1. Montrer qu'un triangle (3,4,5) est rectangle, ainsi qu'un triangle $(\sqrt{11}, \sqrt{12}, \sqrt{23})$.

2. Écrire une fonction `est_rectangle(a,b,c)` : qui renvoie `True` si `c**2 == a**2 + b**2` et, sinon, qui renvoie la différence entre `c**2` et `a**2+b**2`.

3. Tester la fonction `est_rectangle()` avec les deux triangles précédents.
Que remarque-t-on ? Comment modifier la fonction pour qu'elle soit plus satisfaisante ?

Exercice 9**

On aimerait trouver l'écriture dyadique (illimitée) de $\frac{1}{3}$. On note donc

$$\frac{1}{3} = (0, a_1 a_2 a_3 \ldots)_2$$

où a_i vaut 1 ou 0.

1. Expliquer pourquoi a_1 vaut *nécessairement 0*.
2. On note $x = \frac{1}{3}$. Montrer que x vérifie $4x = 1 + x$.
3. Quelle est l'écriture dyadique de $4x$?
4. Quelle est celle de $1 + x$?
5. En écrivant que ces 2 écritures représentent le même nombre, en déduire que

$$\frac{1}{3} = (0,0101\ 0101\ \ldots)_2$$