Chapitre 13 :
Représentation des nombres flottants

I Généralités

1.1 Ecriture décimale

L'écriture décimale d'un nombre ne concerne pas que les nombres entiers mais elle est également valable pour les nombres réels :

Proposition 1

Tout réel strictement positif s'écrit sous la forme :

$$x = a_r \times 10^r + a_{r-1} \times 10^{r-1} + \ldots + a_1 \times 10 + a_0 + \frac{a_{-1}}{10} + \frac{a_{-2}}{10^2} + \ldots + \frac{a_{-k}}{10^k} + \ldots$$

Dans cette écriture $r \in \mathbb{Z}$, les a_i appartiennent à $[\![0,9]\!]$ et $a_r \neq 0$. Les ... à la fin signifient que cette écriture peut être infinie (par exemple $1/3 = 0{,}33333\ldots$).

Un nombre est dit décimal si son écriture décimale est finie. Ces nombres sont de la forme $\frac{a}{10^p}$ où $p \in \mathbb{N}$ et $a \in \mathbb{Z}$.

1.2 Ecriture binaire

On peut également écrire en binaire (base 2) des nombres réels :

Proposition 2

Tout réel $x \in \mathbb{R}_+^*$ s'écrit :

$$x = 1 \times 2^r + a_{r-1} \times 2^{r-1} + \ldots + a_1 \times 2 + a_0 + \frac{a_{-1}}{2} + \frac{a_{-2}}{2^2} + \ldots + \frac{a_{-k}}{2^k} + \ldots$$

où $r \in \mathbb{Z}$, les a_i valent 0 ou 1.
Cette écriture est appelée écriture en base 2 de x.
L'écriture $\overline{1a_{r-1}\ldots a_0, a_{-1}a_{-2}\ldots}^2$ représentera le nombre $1 \times 2^r + a_{r-1} \times 2^{r-1} + \ldots + a_1 \times 2 + a_0 + \frac{a_{-1}}{2} + \frac{a_{-2}}{2^2} + \ldots$.

Un nombre est dit dyadique si son écriture en base 2 est finie. Ces nombres sont de la forme $\frac{a}{2^p}$ où $p \in \mathbb{N}$ et $a \in \mathbb{Z}$. Les nombres dyadiques sont décimaux mais la réciproque est fausse.

➪ **Exemple 1 :**

- $a = \overline{101,1}^2$

- $b = \overline{1011,01}^2$

- $c = 22,625$

- $d = 125,75$

- $e = 21,375$

II Représentation des flottants

2.1 Nombres flottants

Le principe de la représentation décimale en virgule flottante est d'utiliser un nombre pour en représenter plusieurs : 2.7 peut représenter 27 ou 0.27 ou 270 ...

Les nombres flottants sont des nombres décimaux représentés en virgule flottante avec une taille imposée. Il s'agit d'une représentation approchée des nombres réels.

2.2 Représentation des flottants sur des mots de taille fixe

Une partie des bits est réservée pour coder la partie entière et une partie pour la partie décimale (plus un bit de signe). La réalisation d'opérations est alors très simple à réaliser. Cependant ceci est contraignant car il est très probable que de la place mémoire soit gâchée par une telle représentation. Par exemple, les nombres strictement inférieurs à 1 ont leur partie entière nulle, l'espace utilisé pour stocker la partie entière pourrait être utilisé pour représenter plus de décimales et donc obtenir une plus grande précision. Ce sera l'idée de la représentation en virgule flottante.

2.3 Représentation en virgule flottante normalisée

La norme IEEE-574 est actuellement le standard pour la représentation des nombres à virgule flottante en binaire. Cette norme décrit entre autre le stockage du signe, de la mantisse, de l'exposant et définit également le 0, l'infini et les NaN (Not a number).

Nous allons plutôt utiliser l'idée de la notation scientifique mais en base 2.
Un nombre réel non nul sera représenté en machine sous la forme $s \times m \times 2^e$ où

- $s \in \{+1, -1\}$ représente le signe.
- $m = \overline{1c_1...c_n}^2$ avec les $c_k \in \{0,1\}$ et n fixée, précisée dans le paragraphe suivant. m est appelé mantisse.
- e est un entier relatif appelé exposant appartenant à une plage de valeurs fixée.

L'ensemble des nombres de cette forme est appelé ensemble des nombres flottants ou des nombres à virgule flottante.

► En binaire, le premier chiffre de la mantisse est nécessairement 1 (hormis pour 0). Il n'a donc pas besoin d'être stocké. On parle de bit implicite. On ne code donc que la mantisse réduite m' telle que $m = 1 + m'$.
► Le bit de signe est égal à 0 si le nombre est positif et à 1 si le nombre est négatif.
► L'exposant pouvant être négatif, pour le coder, on aurait pu utiliser l'astuce du complément à 2 mais c'est un décalage qui est réalisé pour stocker ce nombre. Ainsi, on dispose de 2^{n_e} valeurs pour l'exposant. Les valeurs $\overline{1111...11111}^2$ et $\overline{0000...00000}^2$ sont réservées pour des cas particuliers. On applique un décalage égal à $2^{n_e-1}-1$. L'exposant doit donc être compris entre $1-(2^{n_e-1}-1) = -2^{n_e-1}+2$ et $2^{n_e}-2-(2^{n_e-1}-1) = 2^{n_e-1}-1$.

2.4 Flottant simple précision : sur 32 bits

- 1 bit est utilisé pour représenter le signe.
- 8 bits sont utilisés pour représenter l'exposant.
- 23 bits sont utilisés pour représenter la mantisse.

► Le décalage pour l'exposant est égal à $2^{8-1}-1 = 127$. L'exposant e est donc un entier relatif compris entre -126 et 127 et les 8 bits de l'exposant donnent la représentation binaire de l'exposant décalé $e' = e + 127$.
► La mantisse est représentée avec 23 chiffres après la virgule en base 2.

Valeurs extrémales et précision

- Le plus petit (en valeur absolue) nombre représentable est $\pm\overline{1,000...0}^2 \times 2^{-126} \simeq \pm 1.2 \times 10^{-38}$
- Le plus grand (en valeur absolue) nombre représentable est $\pm\overline{1,111...1}^2 \times 2^{127} \simeq 2^{128} \simeq 3.4 \times 10^{38}$
- La précision de la mantisse est de l'ordre de $2^{-23} \simeq 1,2 \times 10^{-7}$, ce qui correspond à environ 7 chiffres significatifs (en base 10).

➪ **Exemple 2 :**

- On cherche le nombre représenté en mémoire par : 0 01111100 01000000000000000000000.
  - Le signe est positif.
  - L'exposant décalé est $\overline{01111100}^2 = 2^6+2^5+2^4+2^3+2^2 = 124$, l'exposant est $124-127 = -3$.
  - La mantisse réduite est $\overline{0,01}^2$, la mantisse est $\overline{1,01}^2 = 1+2^{-2} = 1,25$.
  - Le nombre est donc :
$$1,25 \times 2^{-3} = 0,15625.$$
- On cherche la représentation sur 32 bits de $c = 22,625$.
  - Le signe est positif donc le bit de signe vaut 0.
  - On a : $c = 22,625 = \overline{10110,101}^2 = \overline{1,0110101}^2 \times 2^4$.
  - L'exposant est 4, l'exposant décalé est $4+127 = 131 = \overline{10000011}^2$.
  - La mantisse est $\overline{1,0110101}^2$, la mantisse réduite est $\overline{0,0110101}^2$.
  - c est donc représenté en mémoire par :

0 10000011 01101010000000000000000

2.5 Flottant double précision : sur 64 bits

- 1 bit est utilisé pour représenter le signe.
- 11 bits sont utilisés pour représenter l'exposant.
- 52 bits sont utilisés pour représenter la mantisse.

► Le décalage pour l'exposant est égal à $2^{11-1}-1 = 1023$. L'exposant e est un entier relatif compris entre -1022 et $1023 = 2^{10}-1$ et les 11 bits de l'exposant donnent la représentation binaire de l'exposant décalé $e' = e + 1023$.
► La mantisse est représentée avec 52 chiffres après la virgule.

Valeurs extrémales et précision

- Le plus petit (en valeur absolue) nombre représentable est $\pm\overline{1,000...0}^2 \times 2^{-1022} \simeq \pm 2.2 \times 10^{-308}$
- Le plus grand (en valeur absolue) nombre représentable est $\pm\overline{1,111...1}^2 \times 2^{1023} \simeq \pm 2^{1024} \simeq \pm 1.8 \times 10^{308}$
- La précision de la mantisse est de l'ordre de $2^{-52} \simeq 2.2 \times 10^{-16}$, ce qui correspond à environ 16 chiffres significatifs (en base 10).

III Conséquences de la représentation des flottants

3.1 Dépassement de capacité ou overflow

Lorsque le résultat d'un calcul est supérieur au plus grand nombre représentable en machine, on parle de dépassement de capacité ou d'overflow.

```
>>> 10.**500
OverflowError : (34, 'Result too large')
```

3.2 Problèmes d'arrondi

L'ensemble $\mathbb{R}$ ne peut pas être représenté en machine sur un nombre fini de bits puisque l'information ne peut alors être que finie. Un réel est donc approché par un flottant. Ainsi, la plupart des nombres ne peuvent pas être représentés en machine de manière exacte. La conversion en flottant sera à l'origine d'erreurs d'arrondi. Ces approximations peuvent se cumuler dans les calculs et conduire à des résultats très imprécis voir erronés.

```
>>> 0.1+0.2
0.30000000000000004
>>> 0.1+0.2-0.3
5.551115123125783e-17
```

On considère une liste l d'**entiers** et un **entier** x. La fonction suivante renvoie True si x est un élément de l et False sinon.

```
1 def recherche(l,x) :
2     n=len(l)
3     v=False
4     for i in range(n) :
5         if l[i]==x :
6             v=True
7     return v
```

```
>>> l=[2/i+1/j for i in range(1,11) for j in range(1,11)]
>>> recherche(l,0.3)
False
```

Cette fonction renvoie False car 2/10+3/10=0.30000000000000004, il y a donc un problème d'arrondi. Afin de régler ce problème, on introduit une précision :

```
1 def recherchebis(l,x,p) :
2     n=len(l)
3     v=False
4     for i in range(n) :
5         if abs(l[i]-x)<=p :
6             v=True
7     return v
```

```
>>> l=[2/i+1/j for i in range(1,11) for j in range(1,11)]
>>> recherche(l,0.3,10**(-15))
True
>>> recherche(l,0.3,10**(-17))
False
```

En choisissant bien la précision, on peut obtenir un résultat cohérent.

3.3 Phénomène d'absorption

Ce problème apparait lors de l'addition de deux nombres ayant des ordres de grandeurs très différents.

```
>>> (1.+2**53)-2**53
0.0
```

L'écriture en base 2 de $1+2^{53}$ est $\overline{1\underbrace{000\ \ldots\ 000}_{52\text{ fois }0}1}^{2} = \overline{1,\underbrace{000\ \ldots\ 000}_{52\text{ fois }0}1}^{2} \times 2^{53}$. Or, la mantisse est limitée à 52 bits, ainsi, le dernier 1 sera perdu. Ce nombre sera donc représenté en machine par $\overline{1,\underbrace{000\ \ldots\ 000}_{52\text{ fois }0}}^{2} \times 2^{53}$. Ainsi, en représentation machine, $1+2^{53} = 2^{53}$, ce qui explique le résultat renvoyé par Python.

Conséquences : on évitera si possible d'additionner deux quantités aux ordres de grandeurs très différents.

3.4 Phénomène de cancellation

Ce problème se rencontre lorsque l'on calcule la différence de nombres très proches. Il y a cette fois une perte importante du nombre de chiffres significatifs.

Supposons que l'on connaisse x et y avec 25 chiffres significatifs après la virgule (en binaire) :

$$\begin{aligned} x &= \overline{1.10100100011011100101010 11}^{2} \\ y &= \overline{1.10100100011011100100101 00}^{2} \\ x-y &= \overline{0.00000000000000000000010 111}^{2} \end{aligned}$$

Le résultat de $x-y$ sera écrit sous la forme $\overline{1.0111}^{2} \times 2^{-21}$ pour être représenté en machine. La précision ne sera donc plus que de 4 chiffres significatifs en binaire (les chiffres suivants n'ont pas de signification). Cette perte importante de précision est appelée phénomène de cancellation ou d'annulation.

Conséquences : Afin d'éviter les problèmes de cancellation, on évitera si possible de soustraire deux nombres très voisins.

Il faut garder de phénomène à l'esprit afin d'organiser au mieux les calculs pour éviter ce phénomène.
Par exemple, il est préférable de calculer $\dfrac{1}{x(x+1)}$ plutôt que $\dfrac{1}{x} - \dfrac{1}{x+1}$ bien que ces quantités sont théoriquement égales. La seconde expression est plus sensible au phénomène de cancellation.

```
>>> 1/(10**20)-1/(10**20+1)
0.0
>>>  1/(10**20*(10**20+1))
1e-40
```