REPRESENTATION DES DONNEES

NSI | 03

REPRESENTATION DES NOMBRES REELS

[Marion SZPIEG]

Connaître la représentation approximative des nombres réels (à virgule flottante) et en mesurer les conséquences (capacité et arrondis)

1. Petits tests en Python...

Que fait la commande suivante en Python, si a et b sont deux nombres `>>> a - b == 0` ?
Elle test si les nombres a et b sont égaux : elle renvoie « True » si c'est le cas et « False » sinon.

Taper puis noter ce que renvoie Python :

`>>> 4 - 4 == 0` `>>> 4 - 4.0 == 0` `>>> 0.2 + 0.8 -1 == 0` `>>> 0.2 + 0.6 – 0.8 == 0`

`>>> 0.1 + 0.5 – 0.6 == 0` `>>> 0.1 + 0.2 - 0.3 == 0` `>>> 0.7 + 0.2 - 0.9 == 0`

Êtes vous surpris ? Pourquoi ?

Chercher d'autres cas surprenants et écrivez en quelques uns ici :

Il faut donc ÉVITER de TESTER l'égalité des nombres de type float

Afin de remédier à cela, nous allons effectuer un test du type $|a-b|<\epsilon$ où ϵ est une valeur proche de zéro à définir. Taper la commande suivante :

```
>>>abs(0.1+ 0.2 - 0.3)<1E-5      # test adapté aux flottants
```

Quel résultat est affiché ? Quelle information cela donne t il ?

Créer un programme en Python afin de coder le test ci-dessous. Votre programme contiendra une fonction **égal(***a,b,y,eps***)** qui prendra en argument 4 nombres : *a* et *b* dont on fera la somme qu'on comparera avec *y* le tout avec une précision) *eps* près (10^{-5} dans l'exemple précédent).

Tester ce programme avec *a*=0.1, *b*=0.2 et *y*=0.3 et différentes valeurs de *eps*, et trouver à partir de quelle précision (sous la forme d'une puissance de 10) le test renvoie False.

La représentation des nombres réels est **impossible** car l'ensemble des réels est infini et non dénombrable. (C'est à dire on ne peut pas numéroter les nombres réels). Aucun codage ne peut représenter un intervalle des réels aussi petit qu'il soit. Ainsi la représentation d'un nombre réel est une approximation par un nombre proche. Cela explique les erreurs que vous venez d'observer.

2. Codage des nombres réels en virgule fixe

2.1. Le principe

En base 10, l'expression 652,375 est une manière abrégée d'écrire :

$$6\times10^{2}+5\times10^{1}+2\times10^{0}+3\times10^{-1}+7\times10^{-2}+5\times10^{-3}.$$

On fait exactement pareil pour la base 2. Ainsi, l'expression 110,101 signifie :

$$1\times2^{2}+1\times2^{1}+0\times2^{0}+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}.$$

2.2. Passer de la base 2 à la base 10

Pour convertir un nombre réel de la base 2 vers la base 10, on ne décompose à l'aide de puissances de 2 avec des exposants positifs et négatifs. Par exemple :

$110{,}101 = 1\times2^{2}+1\times2^{1}+0\times2^{0}+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=4+2+0{,}5+0{,}125=6{,}625$

2.3. Passer de la base 10 à la base 2

Le passage de base 10 en base 2 est plus subtil. Par exemple : convertissons 1234,347 en base 2.

- La partie entière se transforme comme vu sur les entiers : $(1034)_{10} = 1024 + 8 + 2 = (10000001010)_2$
- On transforme la partie décimale selon le schéma suivant : on multiplie la partie fractionnaire par 2 et on note la partie entière du résultat, on recommence cette opération avec la partie fractionnaire du résultat. On continue ainsi jusqu'à la précision désirée...On concatène les parties entières obtenues dans l'ordre après la virgule.

$0{,}347 \times 2 = \mathbf{0}{,}694$	0,347 = 0,**0**...
$0{,}694 \times 2 = \mathbf{1}{,}388$	0,347 = 0,0**1**...
$0{,}388 \times 2 = \mathbf{0}{,}776$	0,347 = 0,01**0**...
$0{,}776 \times 2 = \mathbf{1}{,}552$	0,347 = 0,010**1**...
$0{,}552 \times 2 = \mathbf{1}{,}104$	0.347 = 0,0101**1**...
$0{,}104 \times 2 = \mathbf{0}{,}208$	0,347 = 0,01011**0**...
$0{,}208 \times 2 = \mathbf{0}{,}416$	0,347 = 0,0101100...
$0{,}416 \times 2 = \mathbf{0}{,}832$	0,347 = 0,01011000...
$0{,}832 \times 2 = \mathbf{1}{,}664$	0,347 = 0,010110001...
$0{,}664 \times 2 = \mathbf{1}{,}328$	0,347 = 0,0101100011...
$0{,}328 \times 2 = \mathbf{0}{,}656$	0,347 = 0,01011000110...

Attention ! Un nombre à développement décimal fini en base 10 ne l'est pas forcément en base 2 (on dit alors que le nombre n'est pas dyadique)

On dit que cette représentation est **à virgule fixe** car il suffit de définir la position de la virgule et on sait instantanément la valeur du nombre. Par exemple, pour un nombre stocké sur un octet, soit huit bits, si on définit arbitrairement la position de la virgule juste après le quatrième bit, alors on sait que 0110 1001 = 0110,1001.

C'est extrêmement simple **MAIS**, l'inconvénient de cette méthode est que, pour un nombre avec peu de chiffres après la virgule, on perd un espace de stockage significatif. Si le nombre en question est 0110 1000, on perd trois bits "inutilement". C'est pour cela qu'on cherché une autre façon de faire :