REPRESENTATION DES DONNEES

NSI | 03

REPRESENTATION DES NOMBRES RÉELS

[Marion SZPIEG – Frédéric PEURIERE]

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` ?

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 :

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.

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 :

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

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 =

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 : $(1034)_{10}=$.....
- 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.

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 :

3. Codage en virgule flottante (écriture scientifique)

3.1. Rappel du fonctionnement avec les puissances de 10

Vue en fin de collège, l'écriture scientifique permet d'écrire un nombre x sous la forme $x=\pm m\times 10^n$ avec :

- le signe du nombre (1 caractère + ou -)
- la mantisse du nombre . $m \in [1,10[$
- l'exposant n : entier relatif.

Exemples : écrire les nombres suivants en écriture scientifique :
2156 = -398457,5 = 0,000001452 = 1,34 =

Remarque : le nombre 0 n'a pas d'écriture scientifique.

3.2. Adaptation aux puissances de 2

3.2.1. Le principe

Nous allons pouvoir représenter des «nombres à virgule» écrits en base deux, sous forme «scientifique» de façon complètement analogue à ce qu'on fait en base 10.

Ainsi, un nombre réel *x* pourra s'écrire sous la forme : …….. où :

-
-

Le zéro sera codé à part.

Par exemple, écrivons l'«écriture scientifique binaire» du nombre *x* = 21, 5 :

21,5 =

La mantisse *m* est donc ……..... codée sur ... bits, l'exposant *e* est ... (donc en binaire ……..) codé sur ... bits.

3.2.2. La norme IEEE 754

La norme IEEE 754 est la norme la plus employée pour la représentation des nombres à virgule flottante dans le domaine informatique. La première version de cette norme date de 1985.

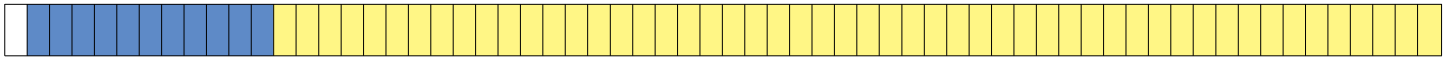
Nous allons étudier deux formats associés à cette norme : le format dit "simple précision" (utilise 32 bits pour écrire un nombre décimal) et le format dit "double précision" (utilise 64 bits pour écrire un nombre décimal).

En Python, et comme dans la plupart des langages de programmation ou de calcul scientifique, un nombre flottant *x* est représenté sur (32 bits ou) 64 bits selon la norme IEEE 754 de la façon suivante :

Codage en 64 bits

Cette norme se propose de coder le nombre sur 64 bits et définit trois composantes :

- le signe est représenté par un seul bit, le bit de poids fort . Si s=0 , le signe est ……., sinon il est …….. .
- l'entier ***e*** est l'exposant : codé sur les 11 bits consécutifs au signe par un nombre ***E***. On obtient e en faisant l'opération ***e=E*-1023** , qui correspond à un décalage . (voir remarques sur l'exposant ci-dessous)
- la mantisse (les bits situés après la virgule) sur les 51 bits restants. Si la partie décimale est constituée de moins de 51 nombres, on complétera par des zéros **à droite**.



Remarques sur l'exposant :

- Notre première intuition serait de dire que la partie "exposant" correspond simplement au "***e***" de $(-1)^s \times 1,(m_1 m_2 \ldots m_{52})_2 \times 2^e$ (dans l'exemple de la partie 3.2.1 (21,5), nous aurions "100"). Le problème c'est qu'on ne peut pas représenter un exposant négatif... Aucun bit pour le signe de l'exposant n'a été prévu dans le norme IEEE754, une autre solution a été choisie : 11 bits sont consacrés à l'exposant ***E***, il est donc possible de représenter 2048 valeurs dans l'intervalle $[0;2^{11}-1]=[0;2047]$
- l'exposant 00000000000 est réservé pour coder le nombre 0 (mantisse et exposant nuls)
- l'exposant 11111111111 est interdit. On s'en sert toutefois pour signaler des erreurs de dépassement de capacité, ou bien d'opérations impossibles ($\sqrt{-1}$ par exemple) : on appelle cette dernière configuration du nombre NaN, ce qui signifie ≪ Not a number ≫.
- ***E*** appartient donc à l'intervalle ……..
- en décalant ce nombre E de ……..., on atteint tous les entier entre …….. et …… …….. et donc ***e*** **∈ […….…….;…….]**

Méthode pour coder un nombre en virgule flottante en double précision : exemple : codons 5,25

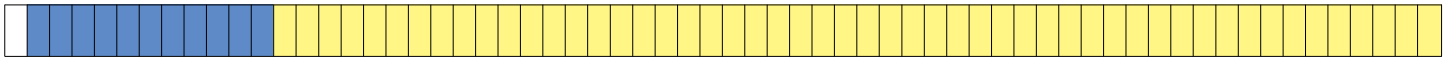
étape 1 : écrire le nombre **non signé** en binaire (virgule fixe)

étape 2 : écrire le nombre en « notation scientifique binaire »

étape 3 : calculer s, m et E :

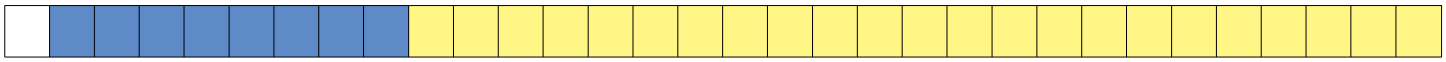
-
- la mantisse est ce qui se trouve derrière la virgule , donc m= ……...
- l'exposant vaut E=…….. or e=…….. donc E=.....

étape 4 : reprendre les éléments précédents : 5,25 sera représenté par :



Codage 32 bits (simple précision)

- l'exposant ***e*** est codé sur 8 bits. La relation entre ***E*** et ***e*** devient : ***e*=……**
- un codage sur 8 bits nous donne un intervalle [0,255] ; on garde comme précédemment les exposants 0 et 255 pour des cas particuliers, donc ***E*** **∈ [……..] et *e* ∈ [……..]**
- la mantisse n'est plus codée que sur 23 bits, ce qui nous donne le schéma :



(même méthode que précédemment, en changeant 1023 par ……. pour le calcul de l'exposant)

Petit récap'

Endodage	Signe s	Exposant E	Mantisse m	Écriture selon la norme IEEE 754
32 bits (simple précision)				
64 bits (double précision)				

3.3. Conclusion

- Les nombres décimaux et réels n'ont pas tous un codage exact en machine.
- Les calculs et représentations sont nécessairement arrondis et la propagation des erreurs d'arrondi est une problématique délicate.
- Travailler quand on peut avec des entiers plutôt qu'avec des décimaux.
- Il faudra être prudent sur les tests (par exemple , ne pas tester si un nombre flottant A==0, mais plutôt tester si $A<10^{-10}$)
- Les résultats dépendent de propagation d'arrondis faits par la machine.
- On évitera d'additionner deux quantités dont l'écart relatif est très important
- On évitera de soustraire deux quantités très proches

3. A vos machines

Connectez vous alors à https://baseconvert.com/ieee-754-floating-point

1. Que pouvez vous dire de la représentation de 0 ?.....

2. a) Copier la représentation décimale de 0.1 en 32 bits (on peut faire de même en 64 bits) et coller la dans une calculatrice (par exemple https://web2.0calc.fr/)

   b) Additionner de même la représentation de 0.2. Enfin comparez cette valeur avec celle de 0.3 (en faisant pas exemple une soustraction)

   c) Expliquez ce que vous avez trouvé dans la partie 1 (avec le programme)

3. Testez et observez aussi les exemples proposés en bas de page

decimal: -0.1 **decimal:** 1e+100 **decimal:** NaN