Leçon d'informatique : Représentation des flottants sur des mots de tailles finies

S. Benlhajlahsen - PCSI$_1$

Sommaire

Extrait du programme :

Notions	Commentaires
Distinction entre nombres réels, décimaux et flottants.	On montre sur des exemples l'impossibilité de représenter certains nombres réels ou décimaux dans un mot machine.
Représentation des flottants sur des mots de taille fixe. Notion de mantisse, d'exposant.	On signale la représentation de 0 mais on n'évoque pas les nombres dénormalisés, les infinis ni les NaN. Aucune connaissance liée à la norme IEEE-754 n'est au programme.
Précision des calculs en flottants.	On insiste sur les limites de précision dans le calcul avec des flottants, en particulier pour les comparaisons. Le comparatif des différents modes d'arrondi n'est pas au programme.

I Nombres réels, nombres décimaux et flottants

I.A définitions

À retenir : En mathématiques, on définit :

- l'ensemble $\mathbb{Q}$ des rationnels. $x \in \mathbb{Q}$ s'il peut se mettre sous la forme $x = p/q$ avec $(p,q) \in \mathbb{Z} \times \mathbb{Z}^*$. $\dfrac{1}{3}$ est un rationnel mais $\dfrac{1}{\sqrt{2}}$ ne l'est pas.
- l'ensemble $\mathbb{D}$ des nombres décimaux. Un nombre décimal est un nombre rationnel qui peut s'écrire sous la forme d'une fraction **dont le dénominateur est une puissance de 10**. Ainsi, $2{,}5 = \dfrac{25}{10}$ est un nombre décimal et rationnel mais $1/3$ n'est pas un nombre décimal.

Remarque : Un nombre décimal x pourra s'écrire sous la forme :

$$x = m \times 10^e$$

L'écriture scientifique correspond à l'unique écriture de x telle que $m \in [1, 10[$. On appellera alors *mantisse* le réel m et e sera l'exposant en base 10.

Remarque : En python, le flottant `x = 1.5e-4` ou `x = 1.5*10**(-4)` correspond au nombre décimal $1{,}5 \times 10^{-4}$.

Nombre à virgule : C'est un nombre dans lequel la partie entière est séparée de la partie décimale par une virgule. Les nombres à virgule sont les nombres réels écrits en notation décimale.

I.B Écriture en base 2

Pour faire le parallèle avec les entiers, les nombres à virgules sont aussi en base 2.

Idée : Seuls les nombres qui s'écrivent en base 2 avec un nombre fini de chiffres après la virgule seront représentables en machine.

Exemples :

- 9,0 est un nombre à virgule. On constate qu'il s'écrit :

$$9{,}0 = 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 1\times 2^0 + 0\times 2^{-1} = \underline{1001{,}0}_2$$

- 1,375 est aussi un nombre à virgule. On constate qu'il s'écrit :

$$1{,}375 = 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} + 1\times 2^{-3} = 1\times 2^0 + 0\times \frac{1}{2^1} + 1\times \frac{1}{2^2} + 1\times \frac{1}{2^3} = \underline{1{,}011}_2$$

- 2,25 est aussi un nombre à virgule. On constate qu'il s'écrit :

$$2{,}25 = 1\times 2^1 + 0\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2} = 1\times 2^1 + 0\times 2^0 + 0\times \frac{1}{2^1} + 1\times \frac{1}{2^2} = \underline{10{,}01}_2$$

Ecriture « scientifique » en base 2 : Par analogie avec l'écriture scientifique en base 10, on propose que la mantisse m soit dans $[1;2[$. Ainsi, on pourra noter :

- $9{,}0 = \underline{1{,}0010}_2 \times 2^3$;
- $2{,}25 = \underline{1{,}001}_2 \times 2^1$.

Nombres décimaux en base 2 : En base 10, les nombres décimaux s'écrivent comme une somme finie de nombre du type :

$$\dots 100, 10, 1, \frac{1}{10}, \frac{1}{100}\dots$$

En base 2, les nombres décimaux s'écrivent comme une somme finie de nombre du type :

$$\dots 4, 2, 1, \frac{1}{2}, \frac{1}{4}\dots$$

Exemple : $\frac{1}{10} = 0{,}1$ est un nombre décimal en base 10 mais **n'est pas un nombre décimal en base 2**. Comme $\frac{1}{2^3} = 0{,}125$ et $\frac{1}{2^4} = 0{,}0625$, on peut commencer par écrire $0{,}1 = \frac{1}{2^4} + a$ avec $a = 0{,}0375$.
Comme $\frac{1}{2^5} = 0{,}03125$, on peut ensuite écrire $0{,}1 = \frac{1}{2^4} + \frac{1}{2^5} + b$ avec $b = 0{,}00625$. Cette décomposition amène à la série numérique :

$$0{,}1 = \left(\frac{1}{2^4} + \frac{1}{2^5}\right)\sum_{k=0}^{+\infty}\frac{1}{2^{4k}} = \underline{0{,}0001100110011001\dots}_2$$

Méthode de décomposition en base 2 La méthode est la même que dans le chapitre précédent. On commence par encadre le nombre x entre deux puissances de 2 : $2^{p-1} \leqslant x < 2^p$ où p est un entier relatif. Tant que le reste n'est pas nul, on ajoute à l'écriture binaire les quotients de la division par 2^{p-1} puis $2^{p-2}, \dots$

II Représentation des flottants sur des mots de taille fixe.

En 64 bits, l'encodage des flottants est donnée par la norme IEEE-754. Un flottant pourra s'écrire de manière générale :

$$a = \pm m \cdot 2^e$$

On voit donc qu'il faut stocker :

- le signe $\pm$;
- l'information sur la mantisse $m \in [1;2[$;
- l'information sur l'exposant e.

Problèmes : On voit tout de suite que cet norme amène à définir deux zéros !
L'exposant peut être positif pour les grand nombres et négatif pour les petits nombres. Ce problème a déjà été rencontrés pour les entiers signés. Il a été résolu par le complément à deux. Il sera ici résolu dans la norme IEEE-754 par un exposant biaisé.

Remarque hors-programme : La norme IEEE-754 attribue 1 bit au signe puis 11 bits d'exposant puis 52 bits de mantisse. On rappelle que $2^{11} = 2048$.
Si on utilisait le complément à deux, on aurait des exposants entre $-2^{-10} = -1024$ et $2^{10} - 1 = 1023$. On utilise un **exposant biaisé**, le nombre s'écrira :

$$x = \text{signe} \times \text{mantisse} \times 2^{\text{e}-\text{biais}}$$

Le biais vaut ici $2^{10} - 1 = 1023$. e est ici l'exposant biaisé. Cela donne le tableau suivant :

Type	Mantisse	e	$\text{e} - \text{biais}$
zéro	0	-1023	0
Nombre normalisé	quelconque	entre 1 et 2046	entre -1022 et 1023
infinis	0	2047	1024
NaN	différent de 0	2047	1024

Remarque : Le module **sys** renvoie des informations sur l'implémentation de python. Le nombre le plus grand (voir ci-dessous) vaut :

`max=1.7976931348623157e+308`

ce qui est en accord avec la norme IEEE-754 car l'exposant le plus grand est 1023 et $2^{1023} \approx 0{,}89 \times 10^{308}$. De plus, si la mantisse est remplis de 52 chiffres un, on aura :

$$\texttt{max} = 2^{1023} \times \sum_{k=-51}^{0} 2^k = 1{,}7976931348623157 \times 10^{308}$$

```python
import sys                              # module sys
print(sys.float_info)
'''
max=1.7976931348623157e+308,            # flottant le plus grand
max_exp=1024,                           # exposant de 2 le plus grand
max_10_exp=308,                         # exposant de 10 le plus grand
min=2.2250738585072014e-308,            # flottant positif le plus petit
min_exp=-1021,                          # exposant de 2 le plus petit
min_10_exp=-307,                        # exposant de 10 le plus petit
dig=15,                                 # nombre de chiffre après la virgule
mant_dig=53,                            # nombre de bit dans la mantisse
epsilon=2.220446049250313e-16           # ecart entre deux flottants
'''
```

III Précision des calculs en flottants

Le nombre flottant (normalisé) le plus grand a la représentation suivante :

$$\underline{1\ 11111111110\ 1111111111111111111111111111111111111111111111111111}_2 = \left(2 - 2^{-52}\right) \times 2^{1023}$$

Le nombre qui vient juste avant est :

$$\underline{1\ 11111111110\ 1111111111111111111111111111111111111111111111111111}_2 = \left(2 - 2^{-51}\right) \times 2^{1023}$$

Ces deux nombres sont séparés de $2^{-52} \times 2^{1023}$ (environ $2{,}2 \times 10^{307}$) du précédent. C'est vraiment très grand ! Les grands nombres flottants sont « rares », ils sont très espacés.
Par contre, au voisinage de 0 la différence entre les deux plus petits flottants successifs est très petite, de l'ordre de $2^{-52} \times 2^{-1022}$ soit environ 5×10^{-324}.

III.A Overflow (ou dépassement de capacité)

Une conséquence immédiate de ce qu'on vient de voir est que l'opération `10*2.0**1022` produit un dépassement de mémoire. On devra donc veiller à ce que les programmes n'utilisent jamais de flottants trop grands.

III.B Comparaison des flottants

Du fait qu'un nombre réel est codé par un `float` qui en est une valeur approchée, la comparaison des nombres peut être surprenante.
Par exemple `0.1+0.1+0.1-0.3 == 0` prend la valeur `False` car le calcul est fait à l'aide des valeurs arrondies de `0.1` et de `0.3`, le résultat de l'opération `0.1+0.1+0.1-0.3` est petit mais il est de l'ordre de l'erreur faite lors de l'arrondi de `0.3`.

> **À retenir :** On ne fera donc pas de comparaison entre des flottants.
> C'est une difficulté qui peut apparaître dans un algorithme de recherche de racine d'une équations sur les flottants.

III.C Comment opérer malgré tout une comparaison entre flottants ?

Si on choisit un flottant a et qu'on note b son suivant, la différence relative entre a et b est $\dfrac{b-a}{\max(|a|,|b|)}$, elle est de l'ordre de 2^{-52} soit environ 2×10^{-16}. Cette différence relative ne dépend pas de la taille de a.
Plutôt que comparer directement deux nombres non nuls a et b, on pourra tester si leur différence relative est inférieure à un seuil choisi, par exemple 10^{-9}.
On observe alors que `(0.1+0.1+0.1-0.3)/0.3 < 10**(-9)` est vrai alors que `(0.1+0.1+0.1-0.3)/0.3 < 10**(-16)` est faux.
Pour comparer un nombre a à 0, on cherchera par exemple si la valeur absolue de a est inférieure à un seuil choisi (qui peut être théoriquement choisi aussi petit que 10^{-323}).

> **Remarque :** Si on veut tester si deux nombres a et b sont proches, le module `math` contient la fonction `isclose` qui teste si la différence relative entre a et b est inférieure à 10^{-9} et renvoie `True` si c'est le cas.

Conclusion

On retiendra la distinction entre nombres réels, décimaux et flottants. On remarquera que la représentation des flottants se fait, par analogie avec l'écriture scientifique à l'aide d'une mantisse et d'un exposant. Enfin, on n'oubliera pas que le calcul sur les flottants est soumis à des erreurs qui empêche la comparaison entre eux.