Représentation approximative des nombres réels : les nombres flottants

Spé NSI - Lycée du parc

Année 2020 - 2021

Introduction

Il est tout simplement impossible de représenter de manière exacte les nombres réels dans la mémoire d'un ordinateur. Les nombres flottants en constituent l'approximation usuellement utilisée, par exemple pour les calculs scientifiques.

I Écriture scientifique

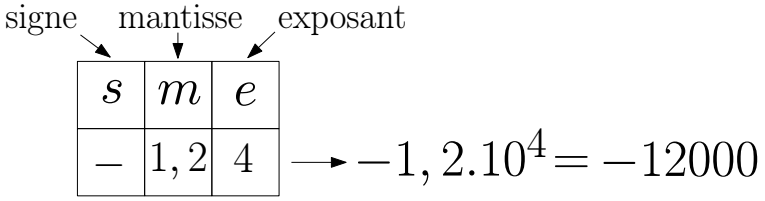
Exercice 1

1. Rappeler le principe de l'écriture scientifique des nombres utilisée en sciences physique.
2. Écrire les nombres suivants sous forme scientifique :

   a = 2021 b = 0,0000005468 c = -897164,236 d = -897164,236 , e = -4,236

3. On donne $x = 11,25 \times 10^{125}$ et $y = 2,6 \times 10^{-84}$. Que dire de $x + y$?

Pour comprendre, la représentation approximative des réels par les flottants, on étudie la situation simplifiée et plus familière des nombres en notation scientifique. On la reproduit à l'aide de trois éléments : un signe, une mantisse qui est un nombre décimal et un exposant :



Mais il faut fixer une taille limitée pour la mantisse et l'exposant afin de pouvoir les coder.

Si l'on suppose que la mantisse est codée sur $t+1$ chiffres (décimaux) et que l'exposant est un entier entre $e_{\min}$ et $e_{\max}$ on peut alors représenter tous les décimaux de la forme

$$\pm\left(c_0+\frac{c_1}{10}+\cdots+\frac{c_t}{10^t}\right)10^e$$

avec $e\in[\![e_{\min},e_{\max}]\!]$ et chaque chiffre $c_i\in[\![0,9]\!]$ avec $c_0\neq 0$ (excepté pour le nombre 0 où l'on ne peut pas imposer $c_0\neq 0$. Le nombre 0 a un traitement particulier).

Exercice 2

Si l'on prend $e_{\min}=-1$, $e_{\max}=1$ et $t=1$ alors on peut représenter les décimaux de la forme

$$\pm\left(c_0+\frac{c_1}{10}\right)10^{-1}\qquad\pm\left(c_0+\frac{c_1}{10}\right)10^{0}\qquad\pm\left(c_0+\frac{c_1}{10}\right)10^{1}$$

Quel est alors

1. le plus grand nombre représentable ?

2. le plus petit ?

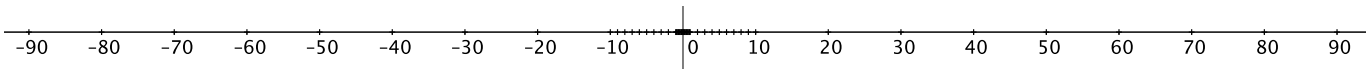
3. le plus petit strictement positif ?

4. le plus petit nombre après 1 ?

5. Combien de nombres différents sont représentable ?

Dans le cas où $t=0$, $e_{\min}=-1$ et $e_{\max}=1$, les nombres décimaux représentables sont :

-90, -80, -70, -60, -50, -40, -30, -20, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90



Il est important de constater que l'écart entre deux grands nombres représentables est bien plus grand que celui entre deux nombres proches de 0 :

La répartition des décimaux représentables n'est pas uniforme [1].

1. Il en sera de même pour les flottants.

II Norme IEEE 754

Exercice 3

L'écriture binaire des entiers a été vue vu dans le chapitre sur la représentation des nombres.

1. Quelle est la « valeur » du premier chiffre (tout à droite) ?
2. Quelles sont les valeurs des chiffres suivants vers la gauche ?
3. Que représente en décimal le nombre $\overline{101001}^{(2)}$?
4. On peut écrire de la même manière des nombres avec une virgule. Quelle est alors la valeur du premier chiffre après la virgule ?
5. Que représente en décimal le nombre $\overline{1,101}^{(2)}$?

Exercice 4

Écrire une fonction python `ChaineBinDec(c)` qui prend en entrée une chaîne de caractères `c` représentant un nombre à virgule écrit en binaire et qui renvoie le nombre flottant de valeur correspondante.

```
>>> ChaineBinDec('1,101')
1.625
```

En décimal on est habitué à ce que certaines fractions simples aient une représentation illimitée :

$$\frac{1}{3} = 0,\underbrace{3333333333333333333333333333333333333...}_{\text{une infinité de 3}}$$

En binaire, le problème est simplement un peu plus fréquent. Ainsi, le nombre $\frac{1}{10}$ a une écriture binaire illimitée et il n'est donc pas représentable exactement en machine ! C'est ce qui explique le résultat :

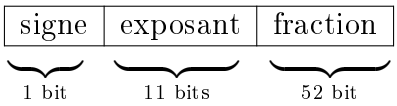
```
>>> 0.1*0.2
0.020000000000000004
```

La norme IEEE 754 standardise la représentation des flottants. C'est la norme la plus couramment utilisé dans les ordinateurs. Elle se décline en deux versions : le format *binary32* ou *simple précision* utilise 32 bits pour représenter un flottant, le format *binary64* ou *double précision* utilise 64 bits pour représenter un flottant. Comme Python utilise le deuxième type sur nos machines, on va se concentrer sur celui-ci.

Il y a principalement deux différences avec l'écriture scientifique :

— La mantisse est écrite en binaire, elle est donc dans l'intervalle $[1, 2[$

— l'exposant est un entier entre -1022 et 1023 qui est stocké de manière décalé d'une valeur $d = 1023$. On remarque que les 2048 valeurs représentables avec 11 bits ne sont pas toutes utilisées ; il « manque » les valeurs -1023 et 1024.



Si on note s la valeur 0 (pour +) ou 1 (pour -) du signe, e la valeur de l'exposant et $m \in [1, 2[$ celui de la mantisse, la valeur du nombre représenté est :

$$x = (-1)^s \times m \times 2^{e-d}$$

Enfin une dernière difficulté ; le chiffre des unités de la mantisse étant toujours égal à 1, il n'est pas représenté, ainsi les 52 bits de la mantisse représente en fait la *fraction* c'est à dire les bits de la partie fractionnaire de la mantisse.

On a donc :

s	e_{10}	e_9	$\dots$	e_1	e_0	f_1	f_2	$\dots$	f_{51}	f_{52}

On a donc $e = \sum_{k=0}^{10} e_k 2^k$ et $m = 1 + \sum_{n=1}^{52} f_n 2^{-n} = 1 + f$

Exercice 5

Calculer le nombre représenté par la suite de bits : | 1 | 10110100000 | 011101100000 . . . 0 |

Exercice 6

Déterminer les bits de la représentation du nombre x = 0,01203125

Exercice 7

Écrire une fonction `ChaineIEEE754(c)` qui prend en entrée une chaîne de caractères représentant les bits d'un nombre flottant en double précision et qui renvoie le nombre flottant de valeur correspondante.

Zéro et valeurs spéciales

Tel que vu jusqu'à présent, le nombre 0 n'est pas représentable en IEEE 754. On utilise les valeurs de l'exposants qui restent libres : il y a deux zéro +0 avec s = 0, e = 0 et f = 0 et -0 avec s = 1, e = 0 et f = 0.

Il y a aussi deux infinis $+\infty$ avec s = 0, e = 2047 et f = 0 et $-\infty$ avec s = 1, e = 2047 et f = 0. Ils servent notamment à indiquer un dépassement de capacité.

Enfin on dispose aussi d'une valeur spéciale, notée **NaN** (pour *Not a Number*) qui représente les résultats d'opération invalides comme 0/0 ou $\sqrt{-1}$ avec s = 0, e = 2047 et f *not* = 0.