On s'intéresse dans ce chapitre aux nombres à virgule flottante (ou simplement flottants) qu'on peut voir en première approche comme des nombres à virgule.

1 Écriture scientifique binaire

1.1 Nombres à virgule binaires

Tout comme on peut écrire des nombres à virgule en base dix, il est possible d'écrire des nombres à virgule en base deux. Rappelons d'abord la signification d'une écriture décimale à virgule à travers un exemple :

Le nombre 123,456 peut être décomposé en deux parties : sa partie entière 123 et sa partie décimale 0,456.

- Pour l'interprétation de la partie entière, cf chapitre ARD1.
- Pour l'interprétation de la partie décimale, on écrit 0,456 comme :
  - 4 dixièmes (10^{-1}) ;
  - 5 centièmes (10^{-2}) ;
  - 6 millièmes (10^{-3})

  D'où $0,456 = 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}$.

On va encore une fois transposer ce système d'écriture à la base deux, toujours en remplaçant les occurrences de la base 10 par des 2. Plutôt que d'avoir après la virgule des nombres de dixième, centième, etc., on aura des nombres de demi, quart, huitième, etc.

Ainsi le nombre écrit en base deux $(1001,1101)_2$ s'interprète comme :

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$$

Le calcul donne alors $(1001,1101)_2 = 9,8125$.

> Pour écrire un nombre à virgule binaire en base dix, on procède de la même manière que pour les entiers, les bits après la virgule représentant les puissances négatives de deux : $2^{-1} = 1/2$, puis $2^{-2} = 1/4$, puis $2^{-3} = 1/8$ etc.

Voyons maintenant comment retrouver l'écriture binaire d'un nombre réel quelconque. Par exemple $9,8125 = (1001,1101)_2$.

1. La partie entière est 9 soit $(1001)_2$.
2. — Enlevons la partie entière, $0,8125 = (0,1101)_2$ puis multiplions par deux : $1,625 = (1,101)_2$
   — On trouve un nombre plus grand que 1 et un 1 dans l'écriture binaire. Ce 1 était au départ le premier bit après la virgule.
3. — Enlevons la partie entière, $0,625 = (0,101)_2$ puis multiplions par deux : $1,25 = (1,01)_2$.
   — On trouve un nombre plus grand que 1 et un 1 dans l'écriture binaire. Ce 1 était au départ le deuxième bit après la virgule.
4. — Enlevons la partie entière, $0,25 = (0,01)_2$ puis multiplions par deux : $0,5 = (0,1)_2$.
   — On trouve un nombre plus petit que 1 et un 0 dans l'écriture binaire. Ce 0 était au départ le troisième bit après la virgule.
5. — Enlevons la partie entière, $0,5 = (0,1)_2$ puis multiplions par deux : $1 = (1)_2$.
   — On trouve un dernier 1 correspondant au dernier bit de l'écriture binaire.

On voit que ces calculs auraient pu être faits sans connaître l'écriture binaire du nombre. Le fait de retirer la partie entière et de multiplier par deux donne le premier bit après la virgule de l'écriture binaire du nombre considéré. Répéter ces deux étapes jusqu'à trouver un 1 garantit donc de retrouver tous les bits de l'écriture binaire.

> Pour écrire un nombre réel en base 2, on procède comme suit :
> 1. Donner l'écriture de la partie entière.
> 2. Tant qu'on ne trouve pas exactement 1 :
>   — enlever la partie entière éventuelle et multiplier par deux ;
>   — si le résultat est supérieur ou égal à 1, le bit suivant est 1 ;
>   — si le résultat est inférieur à 1, le bit suivant est 0.

Un exemple : 7,1875 :

— $7 = (111)_2$.

— $0.1875 \times 2 = 0.375 < 1$: bit 0 ;

— $0.375 \times 2 = 0.75 < 1$: bit 0 ;

— $0.75 \times 2 = 1.5 \geq 1$: bit 1 ;

— $0.5 \times 2 = 1 \geq 1$: bit 1, STOP.

On en déduit que $7,1875 = (111,0011)_2$.

1.2 Écriture finie ou infinie

On sait qu'en base dix, certains nombres s'écrivent avec une infinité de chiffres après la virgule (π, $\sqrt{2}$, $\frac{1}{3}$, etc.), il en est de même en base deux. Voyons les exemples de $\frac{1}{3}$ et $0,2$:

— $1/3$ a une partie entière nulle.

— $1/3 \times 2 = 2/3 < 1$: bit 0 ;

— $2/3 \times 2 = 4/3 = 1 + 1/3 \geq 1$: bit 1 ;

— $1/3 \times 2 = 2/3 < 1$: bit 0 ;

— $2/3 \times 2 = 4/3 = 1 + 1/3 \geq 1$: bit 1 ;

— ...

On voit que notre algorithme ne termine jamais. Cependant, à partir du premier bit après la virgule, on voit aussi qu'on aura à l'infini une succession de 01. On écrira dans ce cas :

$$\frac{1}{3} = (0,[01])_2$$

Continuons avec 0,2 :

— $0,2$ a une partie entière nulle.

— $0,2 \times 2 = 0,4 < 1$: bit 0 ;

— $0,4 \times 2 = 0,8 < 1$: bit 0 ;

— $0,8 \times 2 = 1,6 \geq 1$: bit 1 ;

— $0,6 \times 2 = 1,2 \geq 1$: bit 1 ;

— $0,2 \times 2 = 0,4 < 1$: bit 0 ;

— ...

Là encore, l'algorithme ne termine jamais. Cependant, à partir du premier bit après la virgule, on aura à l'infini une succession de 0011. On écrit donc :

$$0,2 = (0,[0011])_2$$

S'il n'est pas surprenant que le nombre $\frac{1}{3}$ n'ait pas d'écriture finie en base deux, le fait que $0,2$ n'en ait pas non plus montre qu'il existe des nombres ayant une écriture finie en base dix mais infinie en base deux. On pourra cependant retenir le résultat suivant :

Propriété : Tout nombre ayant une écriture finie en base deux a une écriture finie en base dix (la réciproque étant fausse).

Remarque : mathématiquement, l'ensemble des nombres ayant une écriture binaire finie s'écrit $B = \{\frac{a}{2^k} | a \in \mathbb{Z}, k \in \mathbb{N}\}$ et l'ensemble des nombres ayant une écriture décimale finie $\mathbb{D} = \{\frac{a}{10^k} | a \in \mathbb{Z}, k \in \mathbb{N}\}$. On peut alors montrer que $B \subseteq \mathbb{D}$.

1.3 Écriture scientifique binaire

En base dix, l'écriture scientifique d'un nombre réel x est l'écriture de x sous la forme :

$$x = s.m.10^e$$

Où :

— $s = \pm$ est le signe de x ;

— $m \in [0, 10[$ est la mantisse de x ;

— $e \in \mathbb{Z}$ est l'exposant de x.

Par exemple, $1234 = +1,234.10^3$, $-0,0321 = -3,21.10^{-2}$.
En transposant ce principe en base deux, on obtient l'écriture scientifique binaire de x :

$$x = s.m.2^e$$

Où :

— $s = \pm$ est le signe de x ;
— $m \in [0, 2[$ est la mantisse de x ;
— $e \in \mathbb{Z}$ est l'exposant de x.

Attention : Bien qu'on leur ait donné le même nom, les nombres présentés ici différent entre la base 10 et la base 2.

Pour écrire un nombre en écriture scientifique binaire on procède comme suit :
— on écrit le nombre en base 2 ;
— on trouve l'exposant en décalant la virgule comme en base dix.

Exemples :

1. $9,8125 = (1001,1101)_2 = (1,0011101)_2.2^3$
2. $-7,1875 = -(111,0011)_2 = -(1,110011)_2.2^2$
3. $1/3 = (0,[01])_2 \simeq +(1,0101010)_2.2^{-2}$ (8 chiffres significatifs)
4. $0,2 = (0,[0011])_2 \simeq +(1,1001100)_2.2^{-3}$ (8 chiffres significatifs)

2 Nombres flottants, norme IEEE 754

2.1 Norme IEEE 754 : description

La norme IEEE 754 décrit la manière dont sont encodés les réels dans la plupart des langages informatiques et se base sur l'écriture scientifique binaires. Il en existe plusieurs formats dont :

— le format simple, sur 32 bits ;
— le format double sur 64 bits.

Décrivons le format simple, le format double fonctionnant de la même manière. Soit $x \in \mathbb{R}$, mis sous forme scientifique binaire $x = s.m.2^e$, on décrit x en utilisant 32 bits organisés de la manière suivante :

— le premier bit représente le signe s ;
— les 8 bits suivants décrivent l'exposant e si celui-ci est compris entre -126 et $+127$ (sinon x n'est pas représentable dans ce format) ;
— les 23 bits restant décrivent la mantisse m.

Description du signe : Pour le bit de signe on écrit `0` pour les nombres positifs et `1` pour les nombres négatifs.

Description de l'exposant : On donne l'écriture binaire 8 bit de $e + 127$ (on parle alors d'exposant biaisé de $+127$).

Description de la mantisse : Remarquons que la mantisse de tout nombre non-nul s'écrit par définition $m = 1, ...$, aussi il n'est pas nécessaire de représenter le premier `1`. On écrit donc les 23 premiers bits de m après la virgule.

Un nombre encodé de la sorte est appelé un nombre flottant (ou nombre à virgule flottante). Python, utilise lui le format double (64 bits : 1 pour le signe, 11 pour l'exposant et 52 pour la mantisse) le type de données associé étant le type `float`.

Exemple 1 : Représentons $x = 9,8125$, étant établi que :

$$9,8125 = (1001,1101)_2 = (1,0011101)_2.2^3$$

— $x > 0$: premier bit `0` ;
— $e = 3$: exposant biaisé $3 + 127 = 130 = (010\ 0001\ 1)_2$;
— $m = 1,0011101$, on va donner `001 1101 0000 ...`

La représentation dans la norme IEEE754 (simple) de $9,8125$ est donc :

```
0010 0001 1001 1101 0000 0000 0000 0000
```

Exemple 2 : À l'inverse, retrouvons le nombre x représenté par :

1001 0011 1101 1111 0000 0000 0000 0000

— Le premier bit étant 1, $x < 0$;

— l'exposant biaisé est :

$$e + 127 = (001\ 0011\ 1)_2 = 32 + 4 + 2 + 1 = 39$$

donc $e = 39 - 127 = -88$;

— la mantisse s'écrit $m = (1, 101\ 1111)_2$.

On en déduit que $x = (1, 101111)_2.2^{-88} = 1,734375.2^{-88}$

Cas particuliers : On voit que tous les nombres représentables ne sont pas utilisés dans les huit bits réservé à l'exposant biaisé. En effet, pour un exposant biaisé de 0, on obtient un exposant de -127 et pour un exposant biaisé de 255, on obtient un exposant de $+128$, ces deux exposants étant exclus.

En fait, ces valeurs servent à représenter des nombres particuliers tels que :

— 0 de code `0000 0000 0000 0000 0000 0000 0000 0000`

— $+\infty$ de code `0111 1110 0000 0000 0000 0000 0000 0000`

— $-\infty$ de code `1111 1110 0000 0000 0000 0000 0000 0000`

— et autres...

2.2 Approximations et erreurs

La norme IEEE 754 permet de représenter beaucoup de nombres sur une large amplitude puisque, sur 32 bits, on a un potentiel de $2^{32} \simeq 4.10^9$ configurations pour une amplitude allant d'environs -2^{128} à $+2^{128}$.

Cependant, tous les nombres de $\mathbb{R}$, ni même ceux de l'intervalle $[-2^{128}, +2^{128}]$, ne peuvent pas être représentés dans ce format qui ne propose qu'un nombre fini de nombres représentables.

On peut déjà en lister plusieurs catégories (pour le format simple) :

— les nombres hors de l'intervalle $[-2^{128}, +2^{128}]$;

— les nombres ayant une écriture binaire infinie ;

— les nombres ayant plus de 24 bits significatifs.

Pour les premiers, c'est la fin : la norme IEEE 754 ne permet tout simplement pas de travailler avec eux. On peut bien sûr augmenter le nombre de bits de la représentation, par exemple en passant au format double, mais le problème se posera toujours pour des nombres plus grands encore.

Pour les autres, la norme ne permet de manipuler que des approximations. Ainsi, lorsqu'on calcule 1/3, le résultat est stocké avec 24 chiffres significatifs sous la forme :

0011 1110 1010 1010 1010 1010 1010 1010

qui est en réalité la représentation exacte de 0,3333333134651184.

Ces approximations peuvent sembler anodines mais peuvent en pratique générer des erreurs lorsque l'on calcule avec des flottants.

Jouons avec ces erreurs d'approximation en ajoutant un nombre très grand à un nombre très petit par rapport au premier. Par exemple $2^{53} + 1$.

Ce nombre s'écrit en écriture scientifique binaire $(1, 0...01)_2.2^{53}$ avec 52 zéros. Pour le représenter avec ses 54 chiffres significatifs, on aurait donc besoin de 53 bits dans la partie mantisse. Or même le format double utilisé par Python n'en utilise que 52.

On peut donc tester les commandes suivantes en Python et constater que quelque chose ne va pas :

— `2.0**53+1-2.0**53`

— `2.0**53 == 2.0**53+1`

Un autre programme à tester :

```
x = 1
compteur = 0
while x !=0:
    compteur = compteur +1
    x= x/2
print("Je ne suis pas sense etre affiche car x!=0 !")
print("Pourtant me voici apres", compteur, "tours.")
```

Ici le problème provient de l'exposant qui baisse de 1 à chaque tour et qui finit par atteindre sa valeur minimale de -1023.

Travail de recherche : Expliquer alors la valeur finale de `compteur` en explorant la page Wikipédia de la norme IEEE 754.

> On retiendra que les nombres flottants sont la plupart du temps des approximations et qu'il est déconseillé de tester des égalités ou inégalités les impliquant, surtout si les nombres en question proviennent de calculs : plus les calculs sont nombreux, plus les approximations risquent de s'accumuler.
>
> En cas de calculs impliquant de grands nombres flottants ou nombres proches de zéro, on se souviendra que ces approximations peuvent générer des résultats aberrants auquel il faudra porter attention.