Chapitre n° 3 : Représentation binaire des nombres réels

On retrouve le même problème que pour les entiers relatifs mais en plus compliqué : Il n'y a qu'une place finie pour une infinité de réels (comme pour $\mathbb{Z}$) mais en plus on ne sait pas représenter complètement un réel : autant tout entier relatif peut, théoriquement, être représenté exactement (en y consacrant le nombre de bits nécessaires), autant ce n'est pas le cas pour tous les réels :

- 16568697987897967856800899 8 grand mais complètement déterminé ;
- $\frac{1}{3} = 0.33333\cdots$ pas de représentation décimale finie ;
- $\sqrt{2} = 1.1414\cdots$ pas de représentation rationnelle ;
- $\pi = 3.14159\cdots$ pas de représentation algébrique.

1 Écriture en virgule fixe

C'est la méthode utilisée sur les premières machines : on convient de fixer un nombre de bits pour représenter la partie entière du réel et un nombre de bits pour la partie décimale.

Voyons, par exemple, comment coder certains réels positifs sur 8 bits : convenons que les 4 premiers bits (ceux de gauche) représente la partie entière et les 4 derniers bits la partie décimale.

L'octet 10111101 code donc le réel $2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-4}$ c'est-à-dire $11,8125$

Le problème est qu'il est difficile de coder à la fois des nombres très grands, comme par par exemple le nombre d'Avogadro environ égal à $6,02 \times 10^{23}\,\text{mol}^{-1}$ ou très petits, comme la constante gravitationnelle environ égale à $6,67 \times 10^{-11}\,\text{m}^3.\text{kg}^{-1}.\text{s}^{-2}$: cette structure de codage est trop rigide.

2 Écriture normalisée norme IEEE 754

2.1 Définition et exemples

En base 10, en utilisant exactement 4 chiffres, $0,031 \times 10^2$, $3,142 \times 10^0$, $0,003 \times 10^3$ sont trois approximations de π mais qui ne donne pas la même précision.

En base 10, l'écriture scientifique d'un réel consiste à l'écrire sous la forme $m \times 10^e$ où $1 \leqslant |m| < 10$, ainsi l'écriture scientifique de $-317,43$ est $-3,1743 \times 10^2$ (On remarquera que 0 n'a pas d'écriture scientifique)

Dans l'écriture précédente, m porte le nom de mantisse et e celui d'exposant.

La représentation en virgule flottante normalisée IEEE 754 (quasiment universellement utilisée dorénavant) s'inspire de cette écriture scientifique.

Soit x un réel non nul à coder en base 2, on a $x = (-1)^s \times m \times 2^e$ où :

- s est égal à 1 si le nombre est négatif, 0 dans le cas contraire.
- m désigne la mantisse (en binaire)
- e désigne l'exposant.

Comme pour l'écriture scientifique, on choisit m de telle sorte que $1 \leqslant m < 2$: le premier chiffre (celui de gauche) de la mantisse sera donc nécessairement 1, ce qui à son importance pour la suite.

La norme IEEE 754 pour un codage sur 32 bits d'un réel fixe le nombre de bits pour chacun des 3 paramètres précédents :

Définition 1: Écriture virgule flottante normalisée simple précision (32 bits)
De la gauche vers la droite :

- 1 bit (celui de poids fort) pour le signe : 1 pour un réel négatif, 0 pour un réel positif.
- les 8 bits suivant pour l'exposant, pouvant aller de -126 à 127 :
  (les valeurs -127 et 128 sont réservés pour des cas spéciaux dont nous reparlerons brièvement un peu plus tard)
  Plutôt que d'adopter, pour le codage de cet exposant, pouvant être négatif, la méthode du complément à 2 comme vu dans le chapitre précédent, le choix a été fait de biaiser cet exposant en lui ajoutant $2^7 - 1 = 127$, revenant donc ainsi à coder un entier positif.
  ⚠ne pas oublier de « débiaiser » au décodage !
- les 23 bits restant pour la mantisse :
  En fait on code sur 24 bits : comme le chiffre de gauche de la mantisse est 1, il est omis dans le codage
  ⚠ne pas oublier de rajouter ce 1 au décodage !

Exemples :

1. Quel nombre est représenté par :

$$\underbrace{1}_{\text{signe}}\ \underbrace{10000010}_{\text{exposant}}\ \underbrace{01010101010101010101010}_{\text{mantisse}}$$

   - le bit de signe est 1 : le nombre est négatif.
   - l'exposant biaisé est 10000010 correspondant à 130 : l'exposant non biaisé est donc $130 - 127 = 3$
   - la mantisse codée est 01010101010101010101010 correspondant donc à 1,01010101010101010101010 c'est-à-dire :

$$1 + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + 2^{-12} + 2^{-14} + 2^{-16} + 2^{-18} + 2^{-20} + 2^{-22} \approx 1.3333333134651184$$

   Le nombre codé est donc environ égal à $-1.3333333134651184 \times 2^3$ c'est-à-dire environ égal à $-10,66666$

2. Codons le nombre décimal $-118,625$

- Premièrement, nous avons besoin du signe, de l'exposant et de la partie fractionnaire. C'est un nombre négatif, le signe est donc "1".
- Puis nous écrivons le nombre (sans le signe) en binaire. Nous obtenons 1110110,101 (avec divisions par deux successives pour la partie décimale) Ensuite, nous décalons la virgule vers la gauche, de façon à ne laisser qu'un 1 sur sa gauche :

$$1110110,101 = 1,110110101 \times 2^6$$

  C'est un nombre flottant normalisé : la mantisse est la partie à droite de la virgule, remplie de 0 vers la droite pour obtenir 23 bits. Cela donne 11011010100000000000000 (on omet le 1 avant la virgule, qui est implicite). L'exposant est égal à 6, et nous devons le décaler puis le convertir en binaire : $6+127=133$ codé par 10000101.
  On a donc $-118,625$ qui est codé par 11000010111011010100000000000000

Exercice n° 1

Quel nombre est représenté par :

$$\underbrace{1}_{\text{signe}}\ \underbrace{00011110}_{\text{exposant}}\ \underbrace{01010101010101010101010}_{\text{mantisse}}$$

Exercice n° 2

Donner le codage norme IEE754 sur 32 bits de $-43,75$

2.2 Les limites d'une telle représentation

nombre représentable	signe	exposant	mantisse	valeur approchée
le plus grand	0	1111 1110	111 1111 1111 1111 1111 1111	$3,40282346 \times 10^{38}$
le positif non nul le plus proche de 0	0	0000 0001	000 0000 0000 0000 0000 0000	$1,17549435 \times 10^{-38}$
le négatif non nul le plus proche de 0	1	0000 0001	000 0000 0000 0000 0000 0000	$-1,17549435 \times 10^{-38}$
le plus petit	1	1111 1110	111 1111 1111 1111 1111 1111	$-3,40282346 \times 10^{38}$

2.3 Les codages spéciaux

Ce paragraphe est juste là à titre informatif.

- **Cas du 0 :** 0 ne peut pas avoir une mantisse commençant par un 1, il n'est donc pas possible de le représenter comme décrit ci-dessus : par convention, il a été décidé qu'un nombre vaut 0 si, et seulement si, tous les chiffres de son exposant et sa mantisse valent 0 (il y a donc +0 et −0)
- **Les NaN (not a number)** sont là pour signaler une erreur de calcul, comme une division par 0 ou la racine carrée d'un nombre négatif : ils sont représentés avec tous les chiffres de l'exposant égaux à 1 et une mantisse non nulle.

3 Représentation erronée de certains réels

Commençons tout d'abord par remarquer que la convention adoptée ne permet qu'un nombre fini de représentations et rend donc impossible la représentation de tous les réels (même en se restreignant à ceux d'un intervalle borné non vide)

Voyons quelques exemples :

Exemples :

- $1/3 = \sum_{k=1}^{+\infty} \frac{1}{2^{2k}}$: 1/3 ne peut donc pas être représenté sous forme exacte. Bon d'accord il fallait s'en douter puisque 1/3 n'est pas décimal$\cdots$
- Et par exemple 0,1 qui est un nombre décimal bien « anodin » ?
  On peut montrer que ce nombre ne peut pas se décomposer comme une somme finie de puissances de 1/2 !
  Sa représentation sur 32 bits d'après la norme évoquée (pour info 00111101110011001100110011001101) n'est donc exacte !! : le nombre codé est en fait environ égal à 0.100000001490116119384765625

Remarque : Par défaut la norme IEEE754 impose que les nombres décimaux (non entiers) soit arrondis à la valeur représentable la plus proche.

Donnons une illustration tragique de ce problème de représentation approchée :

Le 25 février 1991, pendant la Guerre du Golfe, une batterie américaine de missiles Patriot, à Dharan (Arabie Saoudite), a échoué dans l'interception d'un missile Scud irakien. Le Scud a frappé un baraquement de l'armée américaine et a tué 28 soldats. La commission d'enquête a conclu à un calcul incorrect du temps de parcours, dû à un problème d'arrondi. Les nombres étaient représentés en virgule fixe sur 24 bits. Le temps était compté par l'horloge interne du système en 1/10 de seconde. Malheureusement, comme nous venons de le voir, 1/10 n'a pas d'écriture finie dans le système binaire. L'ordinateur de bord arrondissait 1/10 à 24 chiffres, d'où une petite erreur dans le décompte du temps pour chaque 1/10 de seconde. Au moment de l'attaque, la batterie de missile Patriot était allumée depuis environ 100 heures, ce qui avait entraîné une accumulation des erreurs d'arrondi de 0,34 s. Pendant ce temps, un missile Scud parcourt environ 500 m, ce qui explique que le Patriot soit passé à côté de sa cible.

Nous reviendrons en détail prochainement et plusieurs fois dans l'année sur ce problème majeur du calcul numérique.