Représentation des nombres flottants suivant le standard IEEE 754

Frédéric Goualard

2020-10-24, v. 2.0

Le standard IEEE 754, publié pour la première fois en 1985[1] et revisé en 2008[2] et 2019[3], décrit la représentation interne de l'arithmétique flottante ainsi que les propriétés devant être vérifiées par les opérateurs arithmétiques.

Cette note rappelle les principales caractéristiques du stockage des nombres flottants suivant le standard IEEE 754, puis donne des exemples de passage d'une représentation hexadécimale ou binaire au nombre exprimé en base 10 et inversement d'un nombre en base 10 à sa représentation IEEE 754.

[1] *IEEE Standard for Binary Floating-Point Arithmetic.* American National Standard (ANSI) IEEE Std 754-1985. The Institute of Electrical et Electronics Engineers, Inc, 1985

[2] *754-2008 - IEEE Standard for Floating-Point Arithmetic.* Rapp. tech. IEEE Std 754-2008. The Institute of Electrical et Electronics Engineers, Inc, 2008

[3] *IEEE Standard for Floating-Point Arithmetic.* Rapp. tech. IEEE Std 754-2019 (Revision of IEEE 754-2008). Conference Name : IEEE Std 754-2019 (Revision of IEEE 754-2008). Juil. 2019

1 Représentation en mémoire

Afin de préserver le maximum de bits de précision pour une taille de stockage donnée k, un nombre à virgule flottante v (ou à *point flottant* dans la version anglo-saxonne) est habituellement représenté sous la *forme normalisée* :

$$v = \pm \mathbf{1}.x_{k-1}x_{k-2}\cdots x_0 \times 2^E \tag{1}$$

avec les x_i des chiffres binaires.

Le chiffre avant la virgule n'est jamais stocké et ne compte donc pas dans le calcul de la taille du format de représentation.

La mise en forme normalisée peut nécessiter de déplacer le point flottant vers la gauche (lorsque v est plus grand que 1 en valeur absolue) ou vers la droite (lorsque v est plus petit que 1 en valeur absolue).

Exemple 1.
- *Pour la valeur* $v_1 = 1101{,}101_2$[4]*, on doit déplacer le point flottant de trois positions vers la gauche pour obtenir la forme normalisée de l'équation* (1)*. En conséquence, on doit multiplier la nouvelle valeur* $1{,}101101$ *par* 2^3 *pour conserver la valeur originale :* $v_1 = 1101{,}101 = 1{,}101101 \times 2^3$ *;*
- *Pour la valeur* $v_2 = 0{,}011101$*, on doit déplacer le point flottant de deux positions vers la droite pour obtenir la forme de l'équation* (1)*. On doit donc diviser la nouvelle valeur par* 2^2 *(ou, de façon équivalente, la multiplier par* 2^{-2}*) pour conserver la valeur originale :* $v_2 = 0{,}011101 = 1{,}1101 \times 2^{-2}$.

[4] **Notation.** On notera x_b la représentation du nombre x en base b. Exemple : la valeur 1101_2 équivaut à 13_{10} et à D_{16}.

On constate que l'exposant E dans l'équation (1) doit être un *entier signé* (c'est à dire, potentiellement positif ou négatif). La représentation habituellement utilisée pour un entier signé est celle par complément à 2. Cette représentation permet de réaliser des calculs sur les entiers signés de la même façon qu'avec des entiers non signés, ce qui simplifie le *design* des unités arithmétiques d'un processeur. Par contre, la comparaison de deux entiers signés en complément à 2 requiert plus de travail que pour des entiers non signés car le complément à 2 ne préserve pas l'ordre des chaînes binaires (les entiers négatifs apparaissent après les entiers positifs).

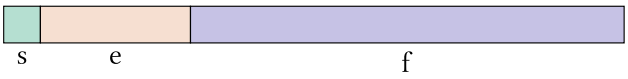


FIGURE 1 : Représentation en mémoire d'un nombre au format IEEE 754

Exemple 2.

Chaîne binaire	*Interprétation non signée*	*Interprétation signée en complément à 2*
000	0	0
001	1	1
010	2	2
011	3	3
100	4	−4
101	5	−3
110	6	−2
111	7	−1

Lors des calculs sur les nombres à virgule flottante (ci-après, *nombres flottants*), on a souvent à comparer des exposants alors qu'ils interviennent dans peu de calculs. La représentation en complément à 2 n'est donc pas la plus adaptée. C'est pourquoi on a choisi d'utiliser la *représentation biaisée* : avant de stocker sous forme binaire un exposant E, on lui ajoute une valeur b, le *biais*, qui garantit que le résultat $e = E + b$ est positif ou nul pour toutes les valeurs possibles de E. C'est cette valeur e qui sera stockée.

Suivant le standard IEEE 754, un nombre flottant v est représenté en mémoire par une chaîne binaire de trois champs :

1. Le bit de signe s, valant 0 si v est positif et 1 sinon ;
2. L'exposant biaisé e ;
3. La partie fractionnaire f, correspondant aux bits à droite du point flottant dans la représentation en forme normalisée (1).

Ces trois champs sont stockés dans l'ordre (s, e, f) (voir la figure 1) de façon à faciliter les comparaisons : étant donnés deux nombres flottants v_1 et v_2, on peut les comparer en comparant deux à deux les bits de leur représentation de la gauche vers la droite.

Le signe est toujours stocké sur un seul bit. On choisit en général pour les champs e et f des tailles telles que la somme des tailles des trois champs soit un multiple de 8, pour utiliser un nombre entier d'octets. Le standard IEEE 754 définit, entre autres, deux formats : *simple précision* et *double précision*[5]. On a vu en cours magistral le format *Tiny*, qui est compatible IEEE 754 sans être défini formellement par le standard. Ce dernier format est à visée purement pédagogique et s'affranchit de la règle de divisibilité par 8 :

	Taille (bits)		
Format	**s**	**e**	**f**
Tiny	1	2	2
Simple précision (32 bits)	1	8	23
Double précision (64 bits)	1	11	52

La « mantisse ».

Bien que le terme « mantisse » (ou « mantissa », en anglais) soit employé dans de nombreux textes relatifs à la représentation des nombres flottants, il n'est jamais utilisé par le standard IEEE 754 lui-même, qui lui préfère le terme de « *significand* » (« significande » ou « signifiant », en français). La mantisse correspondant historiquement à la partie fractionnaire du logarithme en base 10 d'un nombre, on s'abstiendra d'utiliser le terme dans un contexte où un autre mot a *de facto* plus de légitimité.

[5] Le format *simple précision* correspond au type `float` en C, C++ et Java. Le format *double précision* correspond au type `double` en C, C++ et Java.

Interprétation des chaînes binaires Le standard IEEE 754 assure que chaque calcul retourne une valeur, même si aucun résultat n'est mathématiquement défini sur les nombres réels (exemple : la division de 0,0 par 0,0, ou la racine carrée de $-1{,}0$), ceci afin d'éviter une interruption de calcul dans un contexte où il n'est pas possible d'arrêter brutalement un programme (dans un système embarqué, par exemple). Pour cela, la notion de valeur exceptionnelle (*Not a Number* — ou *NaN* — en anglais) est définie.

De même, le standard garantit un résultat même en cas de dépassement de capacité (exemple : calcul de $e^{100\,000}$). Pour cela, les valeurs $-\infty$ et $+\infty$ sont définies.

Pour signaler ces valeurs exceptionnelles, on utilise un exposant e où tous les bits sont à 1. Si la partie fractionnaire f est nulle, on a un infini dont le signe dépend de s ; sinon, on a un *Not a Number*.

De la même façon, il est nécessaire de réserver une valeur particulière de e pour coder la valeur 0,0, qui ne peut être représentée sous la forme de l'équation (1). Lorsque le champ e est composé uniquement de 0 et que la partie fractionnaire est nulle, on considère que l'on a 0,0 ou $-0{,}0$ en fonction du bit de signe s.

Exemple 3. *Dans le format* simple précision, *l'exposant biaisé est codé sur 8 bits, ce qui permet de représenter tous les entiers non signés entre* 0 *et* 255. *Comme on veut pouvoir représenter un nombre à peu près égal de nombres négatifs et de nombres positifs, on a le choix entre un biais de* 127, *permettant de représenter tous les entiers dans l'intervalle* $[-127, +128]$ *et un biais de* 128 *pour représenter tous les entiers dans l'intervalle* $[-128, +127]$. *Le standard IEEE 754 choisit de privilégier les nombres positifs pour pouvoir représenter plus de grands nombres et adopte donc un biais de* 127.

Comme les valeurs 0 *et* 255 *(c'est à dire,* -127 *et* 128 *avant ajout du biais) sont réservées pour signaler des valeurs particulières (*$\pm 0{,}0$, *et* NaN *et* $\pm\infty$, *respectivement), le plus petit exposant utilisable est* -126 *(*1, *avec le biais) et le plus grand est* $+127$ *(*254, *avec le biais).*

Outre le problème de la représentation de la valeur 0,0, la contrainte de l'utilisation d'une représentation normalisée (c'est à dire, avec un « 1 » avant le point flottant) crée un grand trou entre 0 et le premier nombre normalisé représentable.

Exemple 4. *Dans le format* simple précision, *le plus petit nombre normalisé positif, souvent appelé* λ, *vaut* $1{,}0 \times 2^{-126}$. *Le flottant immédiatement supérieur vaut :*

$$1{,}00000000000000000000001 \times 2^{-126},$$

soit $2^{-126} + 2^{-149}$. *On a donc un trou à gauche de* λ *de* 2^{-126} *alors qu'il est de* 2^{-149} *seulement à droite. Pour réduire le trou autour de* 0,0, *le standard a choisi d'autoriser la présence de* nombres dénormalisés *(c'est à dire où le bit avant le point flottant est un* 0 *et non un* 1*). Pour signaler cette particularité, on fixe arbitrairement l'exposant biaisé* e *à* 0, *comme pour représenter* $\pm 0{,}0$. *Le flottant simple précision immédiatement inférieur à* λ *devient donc* $0{,}11111111111111111111111 \times 2^{-126}$ *(ou* $1{,}11111111111111111111110 \times 2^{-127}$ *si l'on pouvait le représenter sous forme normalisée, ce que l'on ne peut faire car l'exposant* -127 *nécessaire est trop négatif pour être représenté).*

Pour résumer, étant donné un nombre $v = (s, e, f)$ dans un format IEEE 754 avec k_e bits pour l'exposant, on a les interprétations :

$$\begin{cases} e = 0, f = 0 & : \quad v = (-1)^s \times 0{,}0 \\ e = 0, f \neq 0 & : \quad v = (-1)^s \times 0.f \times 2^{1-b} \\ 0 < e < 2^{k_e} - 1 & : \quad v = (-1)^s \times 1.f \times 2^{e-b} \\ e = 2^{k_e} - 1, f = 0 : & \quad v = (-1)^s \infty \\ e = 2^{k_e} - 1, f \neq 0 : & \quad v = \text{NaN} \end{cases}$$

avec $b = 2^{k_e - 1} - 1$ le biais associé au format.

2 Exemples de décodage d'une chaîne binaire

Exemple 5. *On a la chaîne hexadécimale* $v = C0AC0000_{16}$ *représentant un nombre flottant au format* simple précision. *On souhaite connaître la valeur décimale codée.*

— *On commence par calculer la représentation binaire de* v *en séparant les champs* s, e *et* f *:*

$$v = \underbrace{1}_{s}\ \underbrace{10000001}_{e}\ \underbrace{01011000000000000000000}_{f}$$

— *On regarde ensuite la valeur de* e *pour déterminer si l'on a affaire à une exception (cas où* $e = 0$ *ou* $e = 255$ *pour le format simple précision). Ici, ce n'est pas le cas donc on peut continuer à interpréter le nombre comme une valeur au format normalisé ;*

— *Le signe* s *vaut* 1 *donc le nombre est négatif ;*

— *L'exposant biaisé* e *vaut* $10000001_2 = 129_{10}$. *On retire le biais de* 127 *pour retrouver l'exposant original* $E = 129 - 127 = 2$ *;*

— *On ajoute* 1 *à la partie fractionnaire* f *pour obtenir :*

$$1{,}01011000000000000000000$$

— *On construit le nombre complet normalisé :* $v = -1{,}01011 \times 2^2$. *Comme l'exposant est petit, on peut s'en débarrasser en déplaçant le point flottant de deux positions vers la droite :*

$$v = -101{,}011$$

La partie entière vaut $101_2 = 5_{10}$. *La partie fractionnaire vaut* $2^{-2} + 2^{-3} = 0.375$. *On en déduit que* $v = -5{,}375$.

Exemple 6. *On a la chaîne hexadécimale* $7FA00000_{16}$ *représentant un nombre flottant* v *au format simple précision.*

— *On calcule la représentation binaire de* v *:*

$$v = \underbrace{0}_{s}\ \underbrace{11111111}_{e}\ \underbrace{01000000000000000000000}_{f}$$

— *L'exposant biaisé* e *vaut* 255, *donc on est en présence d'une exception ; le champ* f *n'est pas nul, donc* v *est un* NaN.

Exemple 7. *On a la chaîne hexadécimale* 80100000_{16} *représentant un nombre flottant* v *au format simple précision.*

— *On détermine la représentation binaire de* v *:*

$$v = \underbrace{1}_{s}\ \underbrace{00000000}_{e}\ \underbrace{00100000000000000000000}_{f}$$

— *L'exposant biaisé* e *est nul. On a donc affaire à une exception. La partie fractionnaire n'est pas nulle donc* v *est un nombre dénormalisé négatif (car* s *vaut 1) de la forme :*

$$v = -0{,}001 \times 2^{-126}$$

que l'on peut aussi écrire :

$$v = -2^{-3}2^{-126} = -2^{-129}$$

3 Exemples de codage au format IEEE 754

Pour coder un nombre décimal dans un format IEEE 754, on commence par le mettre sous forme scientifique comme dans l'équation (1). On peut alors déterminer les champs s, e, et f.

Exemple 8. *On veut coder le nombre* $21{,}59375_{10}$ *dans le format* simple précision.

— *On a :*

$$21_{10} = 10101_2$$

Pour connaître la représentation binaire d'un nombre inférieur à 1 *(ici,* 0,59375*), on le multiplie par* 2 *pour obtenir un résultat sous la forme «* $0 + x$ *» ou «* $1 + x$ *» avec* x *un nombre inférieur à* 1*. Si* x *vaut* 0*, on s'arrête et l'on reprend chaque chiffre* 0 *ou* 1 *dans l'ordre dans lesquels on les a trouvés ; sinon, on multiple* x *par 2 comme au départ. Ici, on a :*

$$\begin{aligned} 0{,}59375 \times 2 &= 1 + 0{,}1875 \\ 0{,}1875 \times 2 &= 0 + 0{,}375 \\ 0{,}375 \times 2 &= 0 + 0{,}75 \\ 0{,}75 \times 2 &= 1 + 0{,}5 \\ 0{,}5 \times 2 &= 1 + 0{,}0 \end{aligned}$$

On a donc :

$$0{,}59375_{10} = 0.10011_2$$

En réunissant la partie entière et la partie fractionnaire, on obtient :

$$21{,}59375_{10} = 10101{,}10011_2$$

— *On met sous forme normalisée le résultat précédent en faisant attention à ce que l'exposant ne dépasse pas les bornes (pour le format simple précision :* $E \in [-126, 127]$*) :*

$$\begin{aligned} 21{,}59375_{10} &= 10101{,}10011_2 \\ &= 1{,}010110011_2 \times 2^4 \end{aligned}$$

Si nécessaire, on arrondit la partie fractionnaire au nombre de chiffres stockables dans f*. Ici, la partie fractionnaire ne contient que* 9 *chiffres alors que l'on peut en stocker* 23*. On complète avec des zéros ;*

— *On a :*

$$\begin{cases} s = 0 \text{ car } 21{,}59375 \text{ est positif,} \\ e = 4 + 127 = 131_{10} = 10000011_2, \\ f = 01011001100000000000000; \end{cases}$$

— *On construit la représentation IEEE 754 en insérant chaque champ à sa place :*

$$21{,}59375 \to \underbrace{0}_{s}\ \underbrace{10000011}_{e}\ \underbrace{01011001100000000000000}_{f}$$

Exemple 9. *On souhaite connaître la représentation du nombre* 13,1 *dans le format* simple précision.

— *On commence par coder la partie entière* 13_{10} *et la partie fractionnaire* $0{,}1_{10}$ *en binaire. On obtient :*

$$13_{10} = 1101_2$$

et

$$\begin{aligned} 0{,}1 \times 2 &= 0 + 0{,}2 \\ 0{,}2 \times 2 &= 0 + 0{,}4 \\ 0{,}4 \times 2 &= 0 + 0{,}8 \\ 0{,}8 \times 2 &= 1 + 0{,}6 \\ 0{,}6 \times 2 &= 1 + \ \ 0{,}2 \\ 0{,}2 \times 2 &= \dots \\ &\vdots \end{aligned}$$

Il vient[6] *:*

$$0{,}1_{10} = 0.\overline{00011}_2$$

[6] **Notation.** On notera $\overline{xyz}$ la répétition à l'infini de la séquence xyz.

— *En combinant la partie entière et la partie fractionnaire, on obtient :*

$$13{,}1_{10} = 1101.0\overline{0011}$$

— *On représente le résultat sous forme normalisée :*

$$\begin{aligned} 13{,}1_{10} &= 1101.\overline{00011}_2 \\ &= 1.1010\overline{00011}_2 \times 2^3 \end{aligned}$$

*Le nombre de bits nécessaire pour la partie fractionnaire dépasse celui alloué par le format. On doit donc l'arrondir. On choisit l'*arrondi au plus proche pair *par défaut. On a la partie fractionnaire :*

$$\begin{aligned} f &= 1010\overline{00011} \\ &= \underbrace{10100011001100110011001}_{23\ bits}10011\ldots \end{aligned}$$

Le 24^e *bit est un* 1 *et il y a d'autres bits non nuls dans la suite, donc* 13,1 *est plus près de* $up(13{,}1)$ *que de* $dn(13{,}1)$[7]*. On arrondit donc le résultat à :*

$$f = 10100011001100110011010$$

[7] Étant donné un réel x, on notera $dn(x)$ le plus grand nombre flottant inférieur ou égal x et $up(x)$ le plus petit flottant supérieur ou égal à x.

— *On a :*

$$\begin{cases} s = 0 \text{ car } 13.1 \text{ est positif} \\ e = 3 + 127 = 130_{10} = 10000010_2 \\ f = 10100011001100110011010 \end{cases}$$

En concaténant les différents champs, il vient :

$$13.1 \rightarrow 0\,10000010\,10100011001100110011010$$

Exemple 10. *On veut coder au format simple précision le nombre* 18,13.

— *On code séparément en binaire la partie entière et la partie fractionnaire :*

$$18_{10} = 10010_2$$

et

$$\begin{aligned} 0{,}13 \times 2 &= 0 + 0{,}26 \\ 0{,}26 \times 2 &= 0 + 0{,}52 \\ 0{,}52 \times 2 &= 1 + 0{,}04 \\ 0{,}04 \times 2 &= 0 + 0{,}08 \\ 0{,}08 \times 2 &= 0 + 0{,}16 \\ 0{,}16 \times 2 &= 0 + 0{,}32 \\ 0{,}32 \times 2 &= 0 + 0{,}64 \\ 0{,}64 \times 2 &= 1 + 0{,}28 \\ 0{,}28 \times 2 &= 0 + 0{,}56 \\ 0{,}56 \times 2 &= 1 + 0{,}12 \\ &\vdots \end{aligned}$$

On a l'impression que le calcul de la représentation binaire de 0,13 *ne boucle pas mais ne va pas s'arrêter non plus. Comme on utilise le format* simple précision, *on peut stocker seulement* 23 *bits dans la partie fractionnaire. La représentation de la partie entière occupera déjà* 4 *bits (on exclut le bit de gauche de* 10010_2 *qui se*

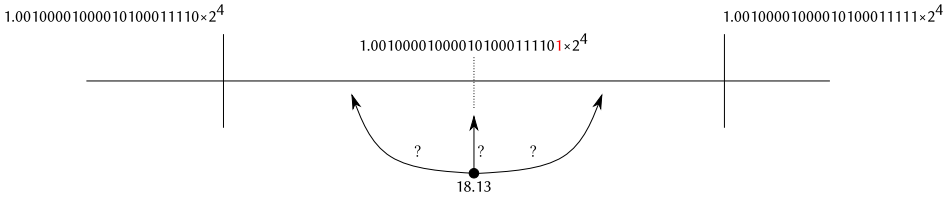
trouvera à gauche de la virgule et ne sera donc pas codé). Il reste donc 19 *bits à calculer pour la partie fractionnaire. On en a déjà* 11. *La suite est :*

$$
\begin{aligned}
0{,}12 \times 2 &= 0 + 0{,}24 \\
0{,}24 \times 2 &= 0 + 0{,}48 \\
0{,}48 \times 2 &= 0 + 0{,}96 \\
0{,}96 \times 2 &= 1 + 0{,}92 \\
0{,}92 \times 2 &= 1 + 0{,}84 \\
0{,}84 \times 2 &= 1 + 0{,}68 \\
0{,}68 \times 2 &= 1 + 0{,}36 \\
0{,}36 \times 2 &= 0 + 0{,}72 \\
&\vdots
\end{aligned}
$$

On a :

$$
\begin{aligned}
18{,}13_{10} &= 10010{,}00010000101000111 10\cdots_2 \\
&= 1{,}00100001000010100011110\cdots_2 \times 2^4 \qquad (2)
\end{aligned}
$$

Considérons la ligne réelle autour de la valeur 18,13 *:*



Les valeurs $1{,}00100001000010100011110 \times 2^4{}_2$ *et* $1{,}00100001000010100011111 \times 2^4{}_2$ *sont représentables au format simple précision. Malheureusement, si l'on choisit d'arrondir* au plus proche pair, *les bits de* 18,13 *de l'équation* (2) *ne permettent pas de savoir si ce nombre se trouve à gauche ou à droite du milieu* $1{,}00100001000010100011110 1 \times 2^4{}_2$. *On doit donc calculer suffisamment de bits supplémentaires pour savoir comment arrondir. On a :*

$$
\begin{aligned}
0{,}72 \times 2 &= 1 + 0{,}44 \\
0{,}44 \times 2 &= 0 + 0{,}88 \\
0{,}88 \times 2 &= 1 + 0{,}76
\end{aligned}
$$

On en déduit que 18,13 *se trouve à droite du milieu et qu'il faut donc l'arrondir à la valeur* $1{,}00100001000010100011111 \times 2^4{}_2$. *Il vient donc :*

$$
18{,}13_{10} = \underbrace{0}_{s}\ \underbrace{10000011}_{e}\ \underbrace{00100001000010100011111}_{f}
$$

Annexes

A Représentation finie et infinie

On a vu dans les exemples précédents que certains nombres dont la représentation est finie en base 10 ont une représentation infinie en base 2. Dans quelles conditions cela peut-il arriver ? À l'inverse, existe t-il des nombres avec une représentation finie en base 2 qui nécessiteraient un nombre infini de chiffres en base 10 ?

Pour répondre à ces questions, commençons par généraliser le problème et demandons-nous quelles sont les conditions pour qu'un nombre x ait une représentation finie dans

une base b. Afin de ne pas devoir intégrer le signe dans les notations, on ne considérera que des nombres positifs, le cas des nombres négatifs étant tout à fait analogue.

Représentation finie en base b

Soit $(c_{k-1}c_{k-2}\cdots c_0,c_{-1}c_{-2}\cdots)_b$ la représentation en base b d'un nombre x. Si x admet une représentation finie sur $n+k$ chiffres en base b, on peut décaler la virgule vers la droite jusqu'à la positionner après le dernier chiffre c_{-n} en partie fractionnaire, ce qui revient à multiplier le nombre par b^n :

$$\begin{aligned} x &= (c_{k-1}c_{k-2}\cdots c_0,c_{-1}c_{-2}\cdots c_{-n})_b \\ &= \frac{(c_{k-1}c_{k-2}\cdots c_0c_{-1}c_{-2}\cdots c_{-n})_b}{b^n} \end{aligned}$$

On aura donc :

Proposition 1. *Un nombre x admet une représentation finie en base b s'il existe un entier positif ou nul n tel que xb^n est un entier.*

Quelle que soit la base considérée, x ne peut admettre une représentation finie s'il est irrationnel. On peut donc se concentrer sur les nombres rationnels seulement. Dans ce cas, on peut exprimer x comme la *fraction irréductible* de deux entiers :

$$x = \frac{p}{q},\ (p,q) \in \mathbb{N}^2$$

[8] **Fraction irréductible.** Une fraction est irréductible s'il n'est pas possible d'obtenir le même rapport avec des termes plus petits. Exemple : $\frac{4}{3}$ est une fraction irréductible. Par contre, $\frac{12}{9}$ ne l'est pas car on peut la réécrire sous la forme $\frac{4}{3}$.

D'après la proposition 1, pour que x ait une représentation finie en base b, on doit pouvoir trouver n tel que :

$$\frac{p}{q}b^n \in \mathbb{N}$$

Comme p/q est irréductible, p et q ne partagent aucun facteur premier. La seule possibilité pour que xb^n soit entier est donc que b^n contiennent tous les facteurs premiers de la décomposition en facteurs de q. Avec $b^n = qr$ (pour $r \in \mathbb{N}$), on obtient :

$$\frac{p}{q}b^n = pr \in \mathbb{N}$$

On en déduit :

Proposition 2. *Un nombre positif rationnel $x = p/q$ (avec $pgcd(p,q) = 1$[9]) admet une représentation finie en base b si et seulement si*[10] *:*

$$\mathcal{D}_q \subseteq \mathcal{D}_b$$

[9] La notation $\text{pgcd}(p,q) = 1$ indique que p et q sont premiers entre eux ; la fraction p/q est donc irréductible.

[10] **Notation.** Étant donné un entier positif a, on note $\mathcal{D}_a$ l'ensemble des diviseurs premiers de a. Exemple : $\mathcal{D}_{15435} = \{1,3,5,7\}$ car $15435 = 1 \times 3^2 \times 5 \times 7^3$.

Exemple 11. *Le nombre $x = 0{,}43512$ a une représentation finie en base 10 car on a :*

$$x = \frac{43512}{100000} = \frac{5439}{12500} = \frac{5439}{2^2 \times 5^5}$$

Il suffit de prendre $n = 5$ pour avoir :

$$x \times 10^5 = \frac{5439}{2^2 \times 5^5} \times (2 \times 5)^5 = 5439 \times 2^3 \in \mathbb{N}$$

À l'inverse, le nombre $x = \frac{1}{3}$ n'admet pas de représentation finie en base 10 car 3 et 10 n'ont aucun facteur premier en commun. Par contre, il a une représentation finie en base 3 puisque l'on peut prendre $n = 1$ pour avoir :

$$x \times 3^1 = \frac{1}{3} \times 3^1 = 1 \in \mathbb{N}$$

Représentation finie en bases b_1 et b_2

Sous quelles conditions sur b_1 et b_2, un nombre positif x ayant une représentation finie en base b_1 a t-il aussi une représentation finie dans une base b_2 ?

Considérons l'ensemble des nombres ayant une représentation finie en base b_1 :

$$\mathcal{U} = \left\{ p/q \mid \text{pgcd}(p,q) = 1 \text{ et } \mathcal{D}_q \subseteq \mathcal{D}_{b_1} \right\}$$

Une condition suffisante simple pour que tous les éléments de $\mathcal{U}$ aient une représentation finie en base b_2 est que tous les diviseurs de b_1 soient aussi des diviseurs de b_2 : $\mathcal{D}_{b_1} \subseteq \mathcal{D}_{b_2}$.

On peut facilement vérifier que c'est aussi une condition nécessaire. En effet, considérons le nombre $1/r$ avec r un diviseur de b_1 mais pas de b_2. Dans ce cas, d'après la proposition 2, $1/r$ est représentable en base b_1 mais pas en base b_2.

On a donc :

Proposition 3. *Tous les nombres entiers ayant une représentation finie en base b_1 ont une représentation finie en base b_2 si et seulement si tous les diviseurs de b_1 sont des diviseurs de b_2 :*

$$\mathcal{D}_{b_1} \subseteq \mathcal{D}_{b_2}$$

Application aux base 2 et 10

On peut appliquer directement la proposition 3 aux base 2 et 10 :

- Comme tous les diviseurs de 2 sont aussi des diviseurs de 10, tous les nombres ayant une représentation finie en base 2 ont aussi une représentation finie en base 10 ;
- À l'inverse, tous les diviseurs de 10 ne sont pas des diviseurs de 2. En particulier, comme l'on a :

$$\mathcal{D}_2 = \{1, 2\}$$
$$\mathcal{D}_{10} = \{1, 2, 5\}$$

tout nombre x s'exprimant sous la forme de la fraction irréductible p/q, avec $5 \in \mathcal{D}_q$ n'admet pas de représentation finie en base 2.

B Licence de ce document

Les remarques, suggestions et indications d'erreurs peuvent être transmises à l'auteur : frederic.goualard@univ-nantes.fr.