Changement de base pour l'écriture et le codage des nombres

Solution proposée par Stéphan Tassart
IRCAM, 1 place Igor Stravinsky, 75004 PARIS

tassart@ircam.fr http://www-stud.enst.fr/anciens/anciens/tassart

Juin 1996

Résumé

L'usage des nombres dans la vie courante repose essentiellement sur la base dix (surement à cause des dix doigts des deux mains) mais on trouve un certain nombre d'occurences de nombres exprimés dans d'autres systèmes de notation, telle que la base soixante pour compter les secondes et les minutes, la base douze pour compter les heures, etc. Le développement des ordinateurs a également permis l'avénement d'un nouveau type d'arithmétique en base 2, c'est-à-dire la manipulation de nombres composé uniquement des chiffres 0 et 1. Il apparaît clairement qu'il a fallu développer des techniques de *conversions de notation* afin de pouvoir transcrire des nombres d'un système de notation dans un autre. À titre d'exemple, les ordinateurs actuels convertissent constamment les nombres représentés en base dix vers un système en base deux puisque leurs circuits ne sont capables d'effectuer des opérations arithmétiques qu'en base deux. Cet exercice se propose de mettre en évidence, de façon concrète, quelques algorithmes de conversion.

1 Énoncé

La façon dont nous pensons l'arithmétique est intimement lié à la manière dont nous représentons les nombres. Le système positionnel utilisant la base b (système également connu sous le nom de *radix* b) est définie par la règle suivante:

$$(\ldots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} \ldots)_b = \ldots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 + a_{-1} b^{-1} + a_{-2} b{-}2 \ldots \tag{1}$$

Par exemple, $(520.3)_6 = 5 \cdot 6^2 + 2 \cdot 6^1 + 0 + 36^{-1} = 192\frac{1}{2}$. Notre système de notation décimal conventionnel est bien sur le cas particulier où b vaut 10, et les a sont choisis parmis les 10 "chiffres décimaux" 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; dans ce cas l'indice 10 dans l'équation (1) peut être omis.

1.1 Base entière naturelle

La plus simple des généralisation du système décimal consiste à prendre b entier supérieur à 1, et à contraindre les a à être des entiers compris dans l'intervalle $0 \leq a_k < b$. Cela nous donne pour $b = 2$ le système binaire, pour $b = 3$ le système ternaire, pour $b = 4$ le système quaternaire, pour $b = 8$ le système octal et pour $b = 16$ le système hexadécimal.

Les a_k sont couramment appelés les *chiffres* (ou *digits*) de la représentation. Les chiffres a_k correspondant à des k grands sont dits "plus significatifs" que les chiffres a_k correspondant à des k petits. En notation binaire, les chiffres sont également appelés des *bits*. En système hexadécimal, les 16 chiffres sont le plus souvent notés: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Les algorithmes suivants permettent de convertir un nombre entier ou fractionnaire dans une base quelconque. On remarquera également que les M premiers chiffres du nombre fractionnaire u en base B correspondent exactement aux M premiers chiffres du nombre entier naturel $\lfloor uB^M \rfloor$ en base B. ($\lfloor x \rfloor$: le plus grand entier inférieur à x et $\{x\}$ partie fractionnaire de x, i.e. $\{x\} = x - \lfloor x \rfloor = x \bmod 1$).

Conversion des nombres entiers Soit un entier naturel u, sa représentation *radix* B $(U_M \ldots U_1 U_0)_B$ s'obtient comme suit:

- $U_0 = u \bmod B$
- $U_1 = \lfloor u/B \rfloor \bmod B$
- $U_2 = \lfloor \lfloor u/B \rfloor / B \rfloor \bmod B$ ...

Conversion des nombres fractionnaires Soit un entier naturel u, sa représentation *radix* B $(0.U_{-1}U_{-2}\ldots)_B$ s'obtient comme suit[1]:

- $U_{-1} = \lfloor uB \rfloor$
- $U_{-2} = \lfloor \{uB\}\, B \rfloor$
- $U_{-3} = \lfloor \{\{uB\}\, B\}\, B \rfloor$ $\ldots$

1.2 Exercices

`[convert(n,base,b), convert(n,binary), convert(n,decimal), convert(n,octal)]` Convertir les nombres suivants en système décimal (base 10):

- $(524)_6$,
- $(B160)_{16}$,
- $(100111010)_2$,
- $(524)_8$.

Convertir le nombre décimal 19:

- en binaire,
- en octal,
- en hexadécimal.

Programmer la fonction `radix_frac` permettant de convertir des nombres fractionnaires dans une base donnée en vous inspirants des algorithmes ou des indications donnés au chapitre précédant, `[iquo, irem, frac, floor, while]` puis convertir les nombres suivants en système décimal:

- $1^\circ\, 22'\, 7''$ (base 60)
- $(11.00100100001111110110101010001000)_2$
- $(520.2)_6$
- $(0.1)_3$
- Écrire les 8 premiers chiffres de π en hexadécimal.
- Convertir $\frac{1}{3}$ en base 2 puis en base 3, puis en base 8.

1.3 Nombre négatifs, base négative

Habituellement, le signe des nombre est indiqué à l'aide d'un symbole supplémentaire (par exemple le symbole "-"). Toutefois il y existe des systèmes de notation qui permettent de prendre en compte la signature (i.e le signe) directement dans l'écriture du nombre. Par exemple, considérons les nombres en base (-10) de telle façon que:

$$\begin{aligned}(\ldots a_3 a_2 a_1 a_0 . a_{-1} a_{-2} \ldots)_{-10} &= \ldots + a_3(-10)^3 + a_2(-10)^2 + a_1(-10)^1 + a_0 + \ldots \\ &= \ldots - 1000a_3 + 100a_2 - 10a_1 + a_0 - \frac{1}{10}a_{-1} + \frac{1}{100}a_{-1} \ldots\end{aligned}$$

Les chiffres satisfont toujours $0 \leq a_k < 10$, comme dans le cas du système décimal normal, mais le signe est intégré directement dans la notation. Par exemple, le nombre -17 s'écrit en base (-10): $(23)_{-10}$, c'est-à-dire $17 = -10 \cdot 2 + 3$.

- Modifier les algorithmes précédants pour qu'ils prennent en comptes les bases entières négatives.
- Convertir π en base (-16) avec 4 chiffres significatifs (après la virgule).
- Convertir π puis $-\pi$ en base (-2) avec 4 chiffres significatifs.

1.4 Nombres complexes, base complexe (optionnel)

Le même principe s'applique également à des base complexes. En base $2i$ avec les chiffres 0,1,2 et 3, on représente tous les nombres complexes sans utiliser ni le symbole de signe, ni le symbole i représentant le nombre imaginaire pure. Par exemple $(11210.31)_{2i}$ correspond au nombre complexe $7\frac{3}{4} - 7\frac{1}{2}i$. En base $i-1$ on représente tous les nombres complexes en utilisant exclusivement les chiffre 0 et 1.

1. On remarquera au passage qu'un nombre s'exprimant en un nombre fini de chiffres dans un système de notation peut s'exprimer avec un nombre infini de chiffre dans un autre système de notation.

– Évaluer $e^{i\frac{\pi}{6}}$ avec 6 chiffres significatifs en base $2i$,

– Écrire i en base $1+i$.

2 Solutions

2.1 Base entière naturelle

La conversion d'un nombre d'une base étrangère à la notation décimale revient à évaluer les quantités suivantes:

```
> 5*6^2 + 2*6 + 4;   # (524)_base 6
196

> 2+2^3+2^4+2^5+2^8; # (100111010)_base 2
 314
```

```
> 6*16 + 16^2 + 11*16^3; # (B160)_base 16
 45408

> 5*8^2 + 2*8 + 4;       # (524)_base 8
  340
```

La conversion inverse demande l'utilisation des fonctions `convert()`, mais leur utilisation est très instinctive:

```
> convert(19,binary);
  10011

> convert(19,octal);
  23
```

```
> convert(19,base,16);
  [1,3]
```

La dernière se lit de gauche à droite, c'est à dire que 19 s'écrit bien $(13)_{16}$ en base 16.

2.2 Programmation des algorithmes de conversion

`new_irem`, `new_iquo` Nouvelles définition pour la division entière et le reste de la division entière de telle façon que $m = nq + r$ avec $0 < r < |n|$ (et pas $|r| < |n|$ et $m * r \leq 0$ comme il est proposé en standart dans **Maple**).

```
new_irem:=proc(m,n)
local result;
option remember;
   result := irem(m,n);
   if result < 0 then result:=result-n; fi;
   if result>9 and (n=16 or n=-16) then
        result:=op(result-9,[A,B,C,D,E,F])
   fi;
   RETURN(result);
end:

new_iquo:=proc(m,n)
local result;
option remember;
   result := iquo(m,n);
   if irem(m,n) < 0 then result:=result+1; fi;
   RETURN(result);
end:
```

`radix_int` Cette fonction réalise trait pour trait l'algorithme de conversion du nombre entier u en base B. Le résultat est donné sous la forme d'une liste de chiffres rangés de telle façon que les chiffres les plus significatifs soient en tête de liste.

```
radix_int := proc(u,B)
local U, u_local;
option remember;
    if u < 0 and B > 0 then RETURN(-radix_int(-u,B)) fi;
    U:=[]; u_local:=u;
```

Tant que u est non nul, je rajoute à la liste U le reste de la division entière de u par B puis je remplace u par le résultat de la division entière de u par B.

```
    while u_local<>0 do
        U:=[new_irem(u_local,B), op(U)];
        u_local:=new_iquo(u_local,B);
    od;
    RETURN(U);
end:
```

`radix_frac1` Cette fonction réalise trait pour trait l'algorithme de conversion du nombre fractionnaire u en base B. Le résultat est donné sous la forme d'une liste de n chiffres rangés de telle façon que les chiffres les plus significatifs soient en tête de liste.

```
radix_frac1 := proc(u,B,n)
local U, index, u_local;
U:=[]; u_local:=u;
```

Je répète n fois l'opération qui consiste à rajouter la partie entière de uB à la liste U puis à remplacer u par la partie fractionnaire positive de uB.

```
    for index from 1 to n by 1 do
        U:=[op(U), floor(u_local*B)];
        u_local:=frac(u_local*B);
    od;
    RETURN(U);
end:
```

`radix_frac2` Cette fonction réalise en principe la même chose que la fonction précédante, mais de manière différente. En multipliant u par B sup n, je décalle la virgule de représentation de u en base B de n chiffres vers la gauche. En conséquence la représentation en base B de la partie entière de uB sup n correspond exactement aux n premiers chiffres de la représentation enbase base B du nombre fractionnaire u.

```
radix_frac2 := proc(u,B,n)
local result, OldDigits;
    oldDigits := Digits;
    Digits := max(10,ceil(evalf(log10(abs(B^n)))));
    result := radix_int(floor(B^n*u),B);
```

En théorie, la liste obtenue devrait contenir n chiffres. Mais comme on ne note jamais les premiers 0 dans une notation entière (on écrit pas 007 mais 7 - sauf cas très particuliers!), il faut rajouter les 0 qui ont disparu. C'est le sens de cette dernière instruction que de compléter les zéros manquants de la liste U.

```
    if nops(result) < n then
       result := [ 0$(n-nops(")),op(")];
    fi;
    Digits := oldDigits;
    RETURN(result);
end:
```

On vérifie que la fonction donne des résultats cohérents avec les conversions obtenues au chapitre précédant:

```
> radix_int(196,6);
[5, 2, 4]
> radix_int(314,2);
[1, 0, 0, 1, 1, 1, 0, 1, 0]
> radix_int(45408,16);
[B, 1, 6, 0]
> radix_int(340,8);
[5, 2, 4]
> radix_int(19,2);
[1, 0, 0, 1, 1]
> radix_int(19,8);
[2, 3]
> radix_int(19,16);
[1, 3]
```

2.3 Conversion directe de nombres fractionnaires

La conversion d'un nombre fractionnaire provenant d'une base de notation étrangère en notation décimale peut toujours se réaliser en évaluer les quantités suivantes:

```
>  1 + 22/60 + 7/60^2; evalf("); % 1 22' 7''
     4927/3600
     1.368611111

> 5*6^2+2*6+2/6; evalf("); % (520.2)_base 6
    577/3
    192.33333333

> 2+1+1/2^3+1/2^6+1/2^11+1/2^12+
> 1/2^13+1/2^14+1/2^15+1/2^16+
> 1/2^18+1/2^19+1/2^21+1/2^23+
> 1/2^25; evalf(");
    105414357/33554432
    3.141592651

> 1/3; evalf(");
    1/3
    0.33333333333
```

Les lignes précédantes nous permettent de constater que certains nombres peuvent s'écrire suivant la base de notation envisagée, soit avec un nombre infini de symboles, soit avec un nombre fini de symboles. Par exemple $\frac{1}{3}$ s'écrit très simplement $(0.1)_3$ en base trois, mais s'écrit $0.333333333\ldots$ en base décimale. Au passage, vous aurez remarqué les 20 premières bits codant les 9 premières décimales de π.

2.4 Conversion inverse de nombres fractionnaires

Les conversion inverses se réalisent beaucoup plus simplement à l'aide des fonctions `radix_frac1` ou `radix_frac2`.

```
> radix_frac2(Pi,16,10);
   [3, 2, 4, 3, F, 6, A, 8, 8, 7, F]
   % 3.243F6A887F

> radix_frac1(1/3,2,10);
   [0,1,0,1,0,1,0,1,0,1]

> radix_frac1(1/3,3,10);
   [1,0,0,0,0,0,0,0,0,0]

> radix_frac1(1/3,8,10);
   [2,5,2,5,2,5,2,5,2,5]
```

2.5 Nombre négatifs, base négative

Les algorithmes `radix_int` et `radix_frac2` tiennent déjà compte des bases négatives (de part l'utilisation de `new_irem` et `new_iquo` à la place de `irem` et `iquo`).

```
> radix_frac2(Pi,-16,4);
   [4, E, 5, D, F] % 4.E5DF

> radix_frac2(Pi,-2,4);
   [1, 1, 1, 0, 1, 1, 0] % 111.0110

> radix_frac2(-Pi,-2,4);
   [1, 1, 0, 1, 1, 1, 0, 1] % 1101.1101
```

2.6 Nombre complexes, base complexe

La décomposition des nombres complexes sur la base $2i$ est en fait très similaire à la décomposition des nombre sur la base -4. Il suffit en effet de faire le petit calcul suivant pour remarquer que l'entrelacement de la partie réelle radix -4 et de la moitié de l'opposée de la partie imaginaire radix -4 donne le nombre complexe radix $2i$.

$$\sum_{k\in\mathbb{Z}} a_k(2i)^k = \sum_{k\in\mathbb{Z}} a_{2k}(-4)^k - 2i\sum_{k\in\mathbb{Z}} a_{2k+1}(-4)^k \tag{2}$$

Par exemple la convertion de $e^{i\frac{\pi}{6}}$ en base $2i$ revient à convertir successivement avec 3 chiffres significatifs $\frac{sqrt3}{2}$ et $\frac{1}{4}$ en base (-4), puis d'entrelacer les deux nombres. On vérifie rapidement que $\frac{\sqrt{3}}{2}$ s'écrit $(1.120)_{-4}$ et que $-\frac{1}{4} = (0.1)_{-4}$. Donc nous trouvons: $e^{i\frac{\pi}{6}} = (1.110200)_{2i}$.

```
> radix_frac2(sqrt(3)/2,-4,3);
    [1,1,2,0] % 1.120
> radix_frac2(-1/4,-4,3);
    [1,0,0] % 0.100

> v:=[1,1,1,0,2,0,0]:       B:=2*I:
> BB:=[1,1/B,1/B^2,1/B^3,1/B^4,1/B^5,1/B^6]:
> linalg[dotprod](v,BB);
    7/8+1/2*I
> evalf(");evalf(exp(I*Pi/6));
```

```
.8750000000-.5000000000*I                    |     .8660254040+.5000000000*I
```

En ce qui concerne la base $1+i$, la situation est bien plus délicate, et nous nous contenterons de vous inviter à vérifier que $(100.111111\ldots)_{1+i}$ est la notation appropriée pour i en base $1+i$.

```
> B:=1+I:
> B^2 + sum(1/B^k,k=1..infinity);
      I
```