Précision numérique

Anne Cadiou

Laboratoire de Mécanique des Fluides et d'Acoustique

Ateliers et Séminaires Pour l'Informatique et le Calcul Scientifique
PMCS2I - LMFA
Vendredi 2 octobre 2020

CC BY SA

Introduction

Les machines actuelles sont binaires : leur arithmétique est en base 2, composée uniquement de 0 et de 1.

Exemple :

En base 10,

$$\begin{aligned} 13 &= 1 \times 10 + 3 \\ &= 1 \times 10^1 + 3 \times 10^0 \end{aligned}$$

En binaire,

$$\begin{aligned} 13_{|10} &= 8 + 4 + 1 \\ &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 1101_{|2} \end{aligned}$$

Énumérer en binaire

décimal	binaire
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
...	...

décimal	binaire
2^1	10
2^2	100
2^3	1000
2^4	10000
2^5	100000
2^n	$1\underbrace{0\ \ldots\ 0}_{n \text{ zéros}}$

décimal	binaire
10	1010
20	10100
30	11110
...	...

Arithmétique binaire

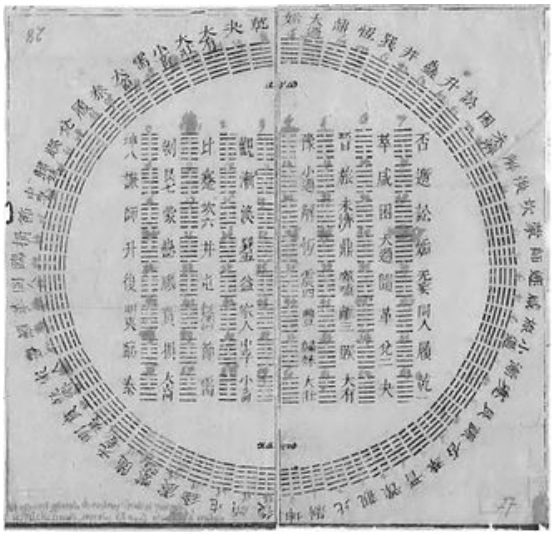


FIGURE – les 64 hexagrammes de Fuxi

Gottfried Wilhelm Leibniz (1646-1716) Explication de l'arithmétique binaire qui se sert des seuls caractères 0 et 1 (1652)

Georges Boole (1815-1864) Création d'une algèbre binaire (booléenne) sur laquelle repose l'électronique actuelle.

Exemple

Addition

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0 \text{ (avec retenue)}$$

$$\begin{array}{cccccccccc}
 & & * & * & & & & * & * & \\
 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
+ & & & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0
\end{array}
\qquad
\begin{array}{cccc}
 & (1) & (1) & \\
 & 3 & 7 & 9 \\
+ & & 9 & 7 \\
\hline
 & 4 & 7 & 6
\end{array}$$

Multiplication

$$
\begin{array}{cccc}
 & & (1) & \\
 & & 1 & 3 \\
\times & & 2 & 5 \\
\hline
 & & 6 & 5 \\
 & 2^1 & 6 & . \\
\hline
 & 3 & 2 & 5
\end{array}
$$

$$
\begin{array}{ccccccccc}
 & & & & 1 & 1 & 0 & 0 & 1 \\
 & & & \times & & 1 & 1 & 0 & 1 \\
\hline
 & & & & 1^1 & 1 & 0 & 0 & 1 \\
 & & & 0^1 & 0 & 0 & 0 & 0 & 0 \\
 & & 1^1 & 1 & 0 & 0 & 1 & 0 & 0 \\
 & 1^1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1
\end{array}
$$

Note sur les nombres réels

Exemples :

En base 10,

$$5.25 = 5 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2}$$

En binaire,

$$\begin{aligned} 5.25_{|10} &= 4 + 1 + 0.25 \\ &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 101.01_{|2} \end{aligned}$$

Conversion en binaire

Convertir $28.8625_{|10}$

$$28_{|10} = 11100_{|2}$$

Conversion de 0.8625

$$0.8625 \times 2 = 1.725 = 1 + 0.725$$
$$0.725 \times 2 = 1.45 = 1 + 0.45$$
$$0.45 \times 2 = 0.9 = 0 + 0.9$$
$$0.9 \times 2 = 1.8 = 1 + 0.8$$
$$0.8 \times 2 = 1.6 = 1 + 0.6$$
$$0.6 \times 2 = 1.2 = 1 + 0.2$$
$$0.2 \times 2 = 0.4 = 0 + 0.4$$
$$0.4 \times 2 = 0.8 = 0 + 0.8$$
$$0.8 \times 2 = 1.6 = 1 + 0.6$$
$$\ldots$$

Résultat

$$28.8625_{|10} = 11100.1101 \underbrace{1100}_{\text{motif répété}} 1100 \ldots_{|2}$$

Représentation en vigule fixe

Limites : nécessite de nombreux chiffres pour les très grands ou très petits nombres.

- Nombre d'Avogadro : $N_A \sim 6.022 \times 10^{23}$

  nécessite au moins 24 chiffres après la virgule pour une approximation à l'entier près
- Charge élémentaire d'un électron : $e \sim 1.602 \times 10^{-19}$ Coulombs

  nécessite au moins 20 chiffres

Représentation en virgule flottante

La représentation des réels est basée sur des nombres flottants pour en obtenir une représentation approchée. Cette représentation limite la précision (nombre de chiffres utilisés dans les flottants).

$$\pi \sim 3.14$$
$$\frac{1}{3} = 0.33333... \sim 3.33 \times 10^{-1}$$
$$\frac{1}{17} = 0.58823529... \sim 5.88 \times 10^{-2}$$

en base 10, avec 2 chiffres après la virgule (3 chiffres en tout).
Par convention, le chiffre avant la virgule est non nul.

Les machines actuelles utilisent la base 2 avec 24 ou 53 chiffres (simple ou double précision).

Norme IEEE-754

La représentation des nombres suit la norme IEEE-754 (1985, révisée en 2008 et 2019) qui fixe un format pour les nombres flottants, afin qu'ils puissent être stockés en machine. Elle est basée sur le triplet (s, e, m) pour signe, exposant et mantisse (entiers stockés en base 2, donc avec de 0 et des 1).

$$x = (-1)^s \, m \, \beta^e \quad \text{où} \quad m = x_0.x_1x_2...x_n$$

$n-1$ est la précision.
Les x_i sont des chiffres compris entre 0 et $\beta - 1$. Par convention x_0 est non nul (normalisation) : en base binaire $\beta = 2$, si bien que x_0 est toujours 1 (on ne le stocke pas en mémoire).

Machine 32 bits ? 64 bits ?

bit et octet

- bit (b) : *binary digit*. vaut 0 ou 1.

  plus petite unité d'information d'un ordinateur (binaire)
- octet (o) ou byte (B) : 8 bits
- 1Ko = 1024 octets (en fait de kilo, ce sont des kibi)
  - 1000 en binaire, c'est 1111101000
  - 1024 en binaire, c'est 10000000000 = 2^{10}
  - binaire : système en base 2 et non en base 10

- avec 1 bit, le processeur ne peut adresser que 2 zones mémoire (0 ou 1)
- avec 2 bits, le processeur peut adresser 4 zones mémoire (00, 01, 10, 11)
- avec 3 bits, 8 zones mémoire (000, 001, 010, 011, 100, 101, 110, 111)
- avec n bits, 2^n zones sont adressables
  - $2^6 = 64$ zones adressables
  - $2^8 = 256$ zones adressables
  - $2^{10} = 1024$ zones adressables

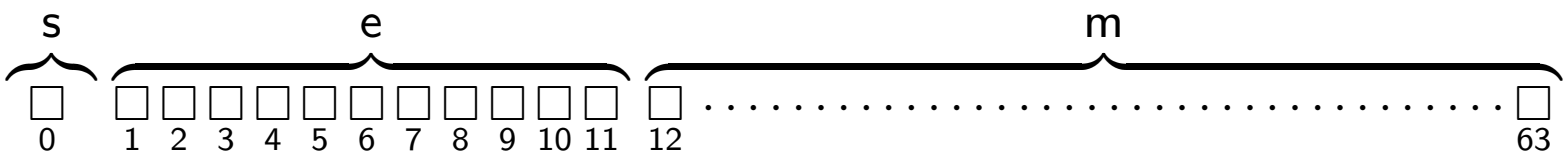
Un processeur 32 (resp. 64) bits peut adresser 4 Gio - gibioctet, i.e. 2^{32} octets (resp. 16 Eio - exbioctets, i.e. 2^{64} octets).

En machine

1 octet (1 Byte) c'est 8 bits

- réel (simple précision) 4 octets
- réel (double précision) 8 octets
- entier 1, 2, 4 ou 8 octet(s)

Un réel de type **double précision** codé sur 8 octets adresse 64 bits



- s : signe, sur 1 bit
- e : exposant, sur 11 bits (fixe le rang de nombres couverts)
- m : mantisse, sur 52 bits

  (fixe la précision de la représentation, $2^{-52} \sim 10^{-16}$)

Le codage étant fixé sur un nombre de bits fixé, la représentation est un compromis entre l'étendue de la plage des réels représentés (plus grande, plus petite valeur) et l'espacement entre deux réels (espace discret)
La norme IEEE-754 privilégie la précision pour le calcul scientifique

Formats des réels

Les **réels** sont codés sur $n = 32$ (resp. $n = 64$) bits en simple (resp. double) précision.

Réels	codage	exposant	mantisse	décalage
Simple	32	8	23	127
Double	64	11	52	1023

Les réels en simple précision (`float`) :

- plus petite valeur absolue représentable : $(1\underbrace{000...000}_{23 \text{ fois } 0})_2 \times 2^{-126} = 2^{-126} \approx 1.17 \times 10^{-38}$
- plus grande : $(1\underbrace{111...111}_{23 \text{ fois } 1})_2 \times 2^{127} \approx 2^{128} - 2^{104} \approx 3.4 \times 10^{38}$

avec 6 chiffres significatifs.
En double précision (`double`) :

- plus petite valeur absolue représentable : $2^{-1022} \approx 2.23 \times 10^{-308}$
- plus grande : $(1\underbrace{111...111}_{52 \text{ fois } 1})_2 \times 2^{1023} \approx 1.80 \times 10^{308}$

avec 15 chiffres significatifs en décimale.

Exemple

En binaire à virgule fixe :

$$5.75_{|10} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 101.11_{|2}$$

En flottant :

$$101.11_{|2} = 1.0111 \times 2^{10} \text{car } 10_{|2} = 2_{|10}$$

En pratique, l'exposant n'est pas signé. En simple précision, il est codé sur 8 bits, et permet donc de représenter $2^8 = 256$ valeurs. Elles varient entre $-126_{|10}$ et $+127_{|10}$ (-127 et $+128$ sont réservées). On ajoute alors 127 à la valeur de l'exposant avant de repasser en binaire :

$$2 + 127 = 129_{|10} = 10000001_{|2}$$

Alors, en machine, en simple précision :

$$5.75_{|10} = 0\ \underbrace{10000001}_{\text{8 chiffres}}\ \underbrace{01110000000000000000000}_{\text{23 chiffres}}$$

En double précision :

$$2 + 1023 = 1025_{|10} = 10000000001_{|2}$$

$$5.75_{|10} = 0\ \underbrace{10000000001}_{\text{11 chiffres}}\ \underbrace{0111000000000000000000000000000000000000000000000000}_{\text{52 chiffres}}$$

Formats des entiers

Un **entier naturel** (positif) codé sur n bits varie entre 0 et $2^n - 1$:

- sur 64 bits (8 octets) : entre 0 et $2^{64} - 1$ (=18 446 744 073 709 551 615)
- sur 32 bits (4 octets) : entre 0 et $2^{32} - 1$ (=4 294 967 295)
- sur 16 bits (2 octets) : entre 0 et $2^{16} - 1$, soit entre 0 et 65 535 (entier de type `int` en C).

Pour représenter un **entier relatif**, il est nécessaire de coder le signe sur un bit (0 pour les nombres positifs et 1 pour les nombres négatifs). Le plus grand entier relatif représentable sur n bits est donc $2^{n-1} - 1$ et le plus petit entier négatif est -2^{n-1}. Il varie entre $-2\,147\,483\,648$ et $2\,146\,483\,647$ sur 32 bits (`long int`)

Nan et Inf

- `Nan` : lorsqu'une opération invalide est effectuée, la mantisse est non nulle et l'exposant ne continent que des 1. (Not a Number)
- `Inf` : lorsqu'un nombre dépasse le plus grand nombre représentable, la mantisse est nulle. L'exposant est $(\underbrace{111...111}_{11 \text{ fois } 1})_2$ (Infinity)

Arrondi

Exemple sur 3 chiffres

- troncature :

$$0.8573 \text{ devient } 0.857$$

- au plus près :

$$0.8573 \text{ devient } 0.857$$

- au représentant le plus proche dont la dernière décimale est paire

$$0.8573 \text{ devient } 0.858$$

Erreur d'arrondi

Soit x un réel. Si on note $\mathrm{fl}(x)$ l'arrondi de x dans sa représentation en virgule flottante, alors il exite un réel δ tel que $|\delta| < u = 2^{-n}$ qui permet d'écrire $\mathrm{fl}(x) = x(1+\delta)$.

u est l'unité d'erreur d'arrondi.

En précision de type

- **simple** : $u = 2^{-23} \sim 1.19\,10^{-7}$ (6 chiffres significatifs)
- **double** : $u = 2^{-52} \sim 2.22\,10^{-16}$ (15 chiffres significatifs)

Si $|\mathrm{fl}(x)|$ est

- plus grande que la plus grande valeur représentable, le dépassement de capacité est dit **overflow**,
- plus petite que la plus petite valeur représentable, le dépassement de capacité est dit **underflow**.

Notes sur l'algèbre IEEE-754

Les opérations de base ($\oplus$, $\ominus$, $\otimes$ et $\oslash$) ainsi que la racine carrée sont censés retourner les valeurs les plus proches du résultat mathématique.

Mais il y a des difficultés liées aux erreurs d'arrondi, par exemple :

- troncature : $a \oplus b \neq a + b$
- perte d'associativité : $(a \oplus b) \oplus c \neq a \oplus (b \oplus c)$
- $a^n \neq \underbrace{a \otimes \ldots \otimes a}_{\text{n times}}$
- ...

Il s'en suit des problèmes de reproductibilité (cf. réduction en calcul parallèle), de perte de précision dans les équations mal conditionnées.

Exemple de non-associativité

```
#include <stdio.h>

int main() {
  double x,y,z;

  x = 8.223457853123465;
  y = 0.000000000000317;
  z = 0.000000000000432;

  printf ("(x+y)+z = %.15f\n",(x+y)+z);
  printf ("x+(y+z) = %.15f\n",x+(y+z));

  return 0;
}
```

```
(x+y)+z = 8.223457853124213
x+(y+z) = 8.223457853124215
```

À la main :

```
x+y+z = 8.223457853124214
```

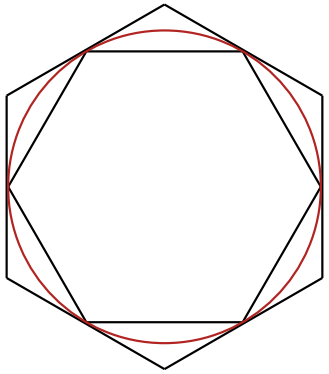
Exemple d'erreur d'arrondi sur une somme

```
program main

  implicit none

  real(kind=8) :: x

  print *, '1.+1e-8 =',1.+1e-8
  print *, '1d0+1d-16 =',1d0+1d-16

  x = 1.05d0
  print *,'x = 1.05d0; x-1 =', x-1

end program main
```

```
 1.+1e-8 = 1.00000000
 1d0+1d-16 = 1.0000000000000000
 x = 1.05d0; x-1 = 5.0000000000000044E-002
```

Conséquence sur le calcul d'une suite itérative

Calcul de π par la méthode d'Archimède
Vers 250 avant J-C, Archimède propose une méthode pour encadrer la valeur de π, à partir du calcul de l'aire d'un polygône inscrit dans le cercle et du correspondant exinscrit :

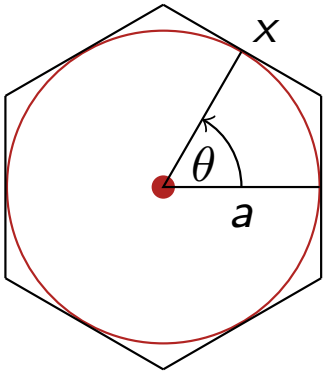


Cet algorithme préfigure le calcul intégral. En utilisant un polygône à 96 côtés, Archimède a pu encadrer π par

$$\frac{233}{71} \leq \pi \leq \frac{22}{7}$$

En écrivant un algorithme qui part d'un hexagone et subdivise chaque côté en deux, il suffit de calculer l'aire (ou le périmètre) d'un polygone à 6, 12, 24, 48, 96, etc. côtés.

Algorithme



a :apothème (rayon du cercle inscrit)

p :périmètre du polygône

x :côté du polygône

θ :angle

L'angle θ est divisé par deux et on calcule la longueur du côté correspondant pour faire varier le périmètre. En effet, si l'aire du cercle inscrit est

$$A_c = \pi\, a^2$$

l'aire du polygône régulier est

$$A_p = (a\, p)/2$$

et il suffit de calculer la variation du périmètre pour estimer π.

En partant d'un hexagone,

$$x = \frac{2}{\sqrt{3}} \ a$$

si bien que

$$A_p = 2\sqrt{3} \ a^2$$

soit :

$$\pi \sim \frac{6}{\sqrt{3}}$$

Puisque

$$\tan(\theta) = \frac{2 \ \tan(\theta/2)}{1 - \tan(\theta/2)}$$

la relation entre le côté x et le côté y, obtenu en divisant l'angle de moitié, est :

$$y = \frac{\sqrt{x^2 + 1} - 1}{x}$$

Programme possible

```
program main

  implicit none

  integer :: i,n
  real(kind=8) :: x,p,fac

  x = 1.0d0/dsqrt(3.0d0)
  fac = 6.0d0
  p = fac*x
  print *, 0, p

  n = 27
  do i=1,n
    x = (sqrt(x*x+1.0d0)-1.0d0)/x
    fac = 2.0d0*fac
    p = fac*x
    print *, i, p
  enddo

end program main
```

autre forme

```
program main

  implicit none

  integer :: i,n
  real(kind=8) :: x,p,fac

  x = 1.0d0/dsqrt(3.0d0)
  fac = 6.0d0
  p = fac*x
  print *, 0, p

  n = 27
  do i=1,n
    x = x/(sqrt(x*x+1.0d0)+1.0d0)
    fac = 2.0d0*fac
    p = fac*x
    print *, i, p
  enddo

end program main
```

Méthode 1

```
 0 3.4641016151377553
 1 3.2153903091734750
 2 3.1596599420975098
 3 3.1460862151314668
 4 3.1427145996455734
 5 3.1418730499798659
 6 3.1416627470550682
 7 3.1416101765995217
 8 3.1415970343233370
 9 3.1415937488168559
10 3.1415929278736332
11 3.1415927256225915
12 3.1415926717415452
13 3.1415926189008863
14 3.1415926717415452
15 3.1415919358819728
16 3.1415926717415452
17 3.1415810075793642
18 3.1415926717415452
19 3.1414061547376217
20 3.1405434924010995
21 3.1400068646909682
22 3.1349453756588517
23 3.1400068646909682
24 3.2245152435348192
25 2.7911172130586381
26 0.0000000000000000
27 NaN
```

Méthode 2

```
 0 3.4641016151377553
 1 3.2153903091734728
 2 3.1596599420975013
 3 3.1460862151314357
 4 3.1427145996453696
 5 3.1418730499798251
 6 3.1416627470568503
 7 3.1416101766046913
 8 3.1415970343215283
 9 3.1415937487713546
10 3.1415929273850987
11 3.1415927220386157
12 3.1415926707019999
13 3.1415926578678461
14 3.1415926546593083
15 3.1415926538571739
16 3.1415926536566401
17 3.1415926536065069
18 3.1415926535939738
19 3.1415926535908403
20 3.1415926535900569
21 3.1415926535898615
22 3.1415926535898127
23 3.1415926535898002
24 3.1415926535897976
25 3.1415926535897967
26 3.1415926535897967
27 3.1415926535897967
pi 3.1415926535897932384626433832 7...
```

Problème de compensation (catastrophic cancellation)

Se produit lorsqu'on tente de soustraire des nombres très proches.

```
program main

  implicit none

  real(kind=8) :: a,x,y

  a = 1.0d9
  print *, a
  x = dsqrt(a+1.0d0)
  y = dsqrt(a)
  print *, x-y

  a = 1.0d16
  x = dsqrt(a+1.0d0)
  y = dsqrt(a)
  print *, x-y

end program main
```

```
  1.5811390767339617E-005
   0.0000000000000000
```

```
program main

  implicit none

  real(kind=8) :: a,x,y

  a = 1.0d9
  x = dsqrt(a+1.0d0)
  y = dsqrt(a)
  print *, 1.0d0/(x+y)

  a = 1.0d16
  x = dsqrt(a+1.0d0)
  y = dsqrt(a)
  print *,x
  print *,y
  print *, 1.0d0/(x+y)

end program main
```

```
   1.5811388296889049E-005
   5.0000000000000001E-009
```

Exemple (Rump, 1988)

$$p = 333.75\, y^6 + x^2\,(11\, x^2\, y^2 - y^6 - 121\, y^4 - 2) + 5.5\, y^8 + x/(2y)$$

avec $x = 77617$ et $y = 33096$.

Solution(s) calculée(s) (en Fortran) :

- en simple précision : `p = 6.33825300E+29`
- en double précision : `p = 1.1726039400531787`
- en quadruple précision : `p = 1.1726039400531786318588349045201838 0`

Exemple (Rump, 1988)

$$p = 333.75\, y^6 + x^2\,(11\, x^2\, y^2 - y^6 - 121\, y^4 - 2) + 5.5\, y^8 + x/(2y)$$

avec $x = 77617$ et $y = 33096$.

Solution(s) calculée(s) (en Fortran) :

- en simple précision : `p = 6.33825300E+29`
- en double précision : `p = 1.1726039400531787`
- en quadruple précision : `p = 1.17260394005317863185883490452018380`

Solution exacte :

$$p = -0.827396059946821368141 1\ldots$$

Exemple du calul de racines de polyômes du second degré

$$x^2 - 1634\,x + 2 = 0$$

Discriminant $\Delta = \sqrt{b^2 - 4\,ac}$

$$x_\pm = 817 \pm \sqrt{667487} \text{ avec } \sqrt{667487} = 816.998776008875...$$

Numériquement

```
xp = 1.633998776008875e+03
xp = 1633.998776008875211

xm = 1.223991124902568e-03
xm = 0.001223991124903
residual = 6.347744552215318e-11
```

$$x_- = 2/x_+$$

```
xm = 1.223991124941416e-03
xm = 0.001223991124941
residual = 4.440892098500626e-16
```

Problème d'instabilité numérique

Amplification d'erreur d'arrondi lors de calculs itératifs ou récurrents :

$$u_0 = \frac{1}{3}$$
$$u_{n+1} = 4\,u_n - 1$$

```
0 3.333333333333333e-01
1 3.333333333333333e-01
2 3.333333333333330e-01
3 3.333333333333321e-01
4 3.333333333333286e-01
5 3.333333333333144e-01
6 3.333333333332575e-01
7 3.333333333330302e-01
8 3.333333333321207e-01
9 3.333333333284827e-01
10 3.333333333139308e-01
11 3.333333332557231e-01
12 3.333333330228925e-01
13 3.333333320915699e-01
```

...

```
27 0.000000000000000e+00
```

$$u_0 = \frac{1}{2}$$
$$u_{n+1} = 3\,u_n - 1$$

```
0 5.000000000000000e-01
1 5.000000000000000e-01
2 5.000000000000000e-01
3 5.000000000000000e-01
4 5.000000000000000e-01
5 5.000000000000000e-01
6 5.000000000000000e-01
7 5.000000000000000e-01
8 5.000000000000000e-01
9 5.000000000000000e-01
10 5.000000000000000e-01
```

1/3 ne peut pas être représenté exactement en flottant : l'erreur initiale est amplifiée par la récurrence.
En base 2,

$$\frac{1}{3} = \frac{1}{4} \sum_{i=0}^{\infty} \frac{1}{4^i} = \frac{1}{4} \sum_{i=0}^{\infty} \frac{1}{2^{2i}}$$

En base 2,

- 1/2 s'écrit 0.1
- 3/2 s'écrit 1.1

donc le calcul est exact.

La précision des calculs numériques dépendra si le problème est bien posé, des données, de la stabilité de l'algorithme, etc, il est donc important d'analyser le comportement numérique du problème traité. (et de savoir s'il est bien posé, accessoirement ...)

Exemple de convergence numérique

Méthode de Newton-Raphson pour calculer $\sqrt{2}$, solution de $x^2 - 2 = 0$

```
Initial guess: x = 0.100000000000000E+01
After 1 iterations, x = 0.150000000000000E+01 (1 correct digit)
After 2 iterations, x = 0.141666666666667E+01 (3 correct digits)
After 3 iterations, x = 0.141421568627451E+01 (6 correct digits)
After 4 iterations, x = 0.141421356237469E+01 (12 correct digits)
After 5 iterations, x = 0.141421356237310E+01
solver returns x = 0.141421356237310E+01 after 5 iterations
the residual is 0.444089209850063E-15
```

Problème discret

Description d'un cercle par le système d'équations différentielles couplées :

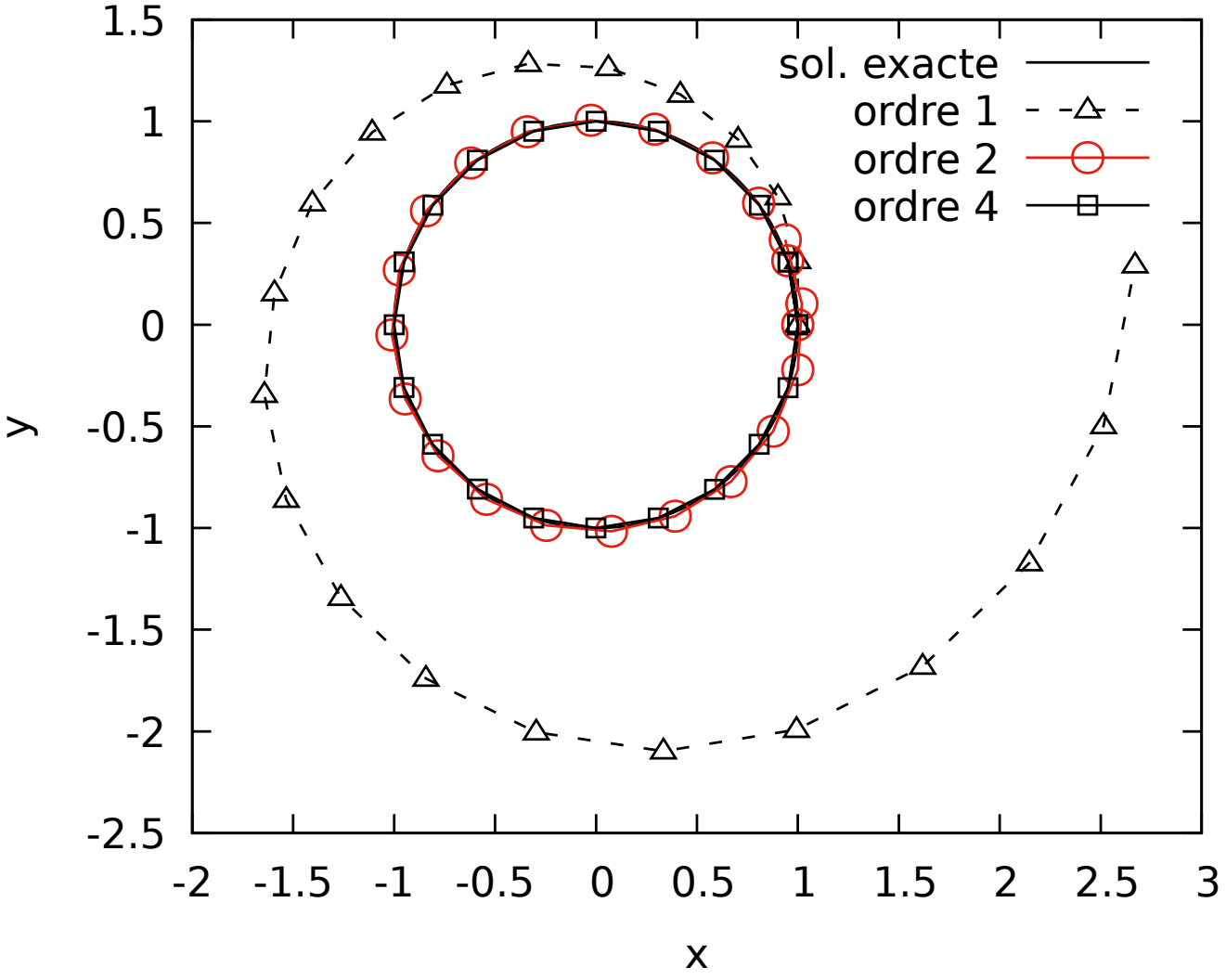
$$\frac{dx}{dt} + y = 0$$
$$\frac{dy}{dt} - x = 0$$

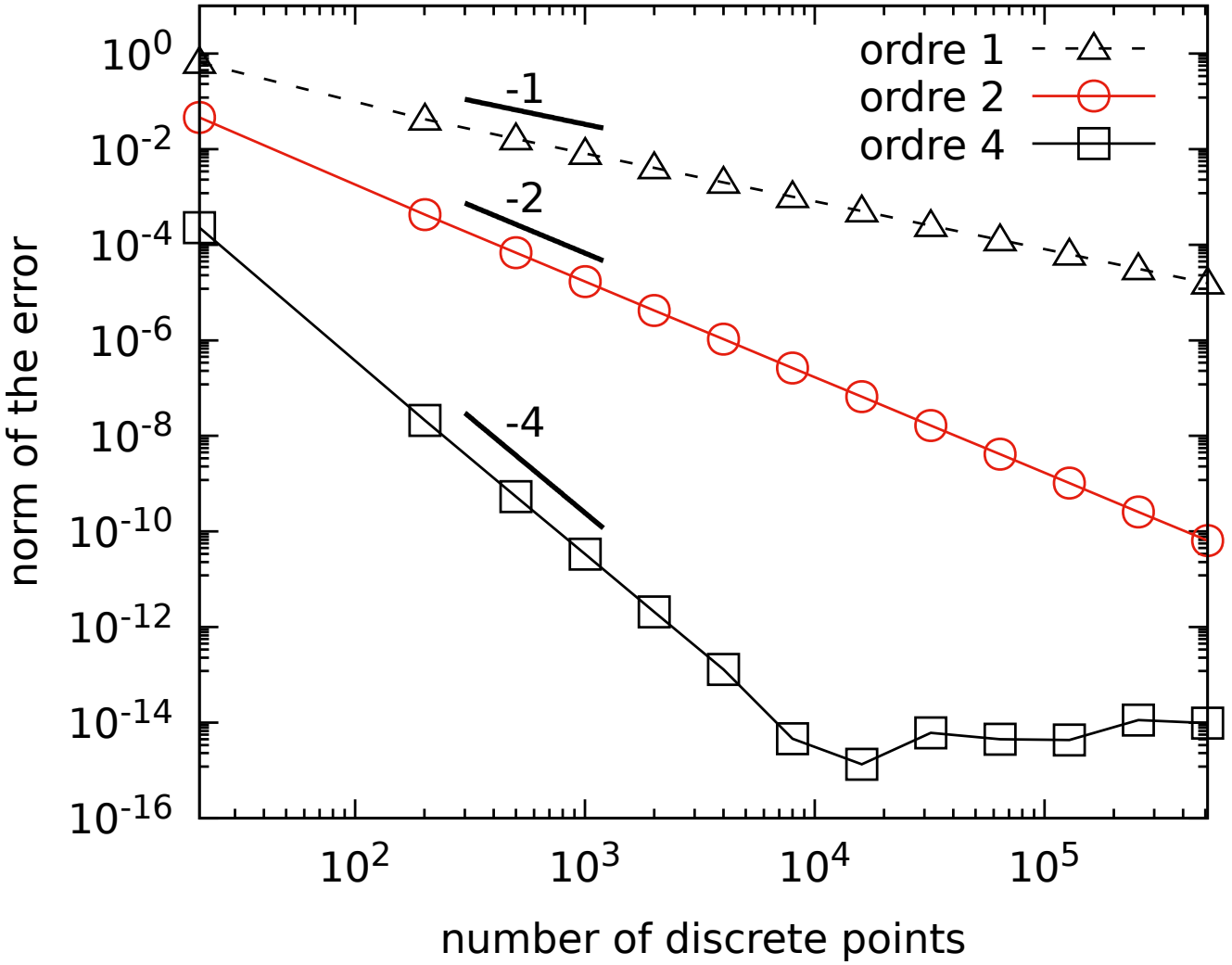
avec pour condition initiale $x(t = 0) = 1$, $y(y = 0) = 0$.

Solution exacte :

$$x(t) = \cos t, \;\; y(t) = \sin t$$

Résolution par un schéma d'Euler explicite, Runge-Kutta d'ordre 2 et 4


sol. exacte
ordre 1
ordre 2
ordre 4
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
y
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3
x


ordre 1
ordre 2
ordre 4
-1
-2
-4
norm of the error
number of discrete points
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{2}
10^{3}
10^{4}
10^{5}

En FORTRAN

Expression	Forme alternative possible
`X*Y`	`Y*X`
`-X+Y`	`Y-X`
`X + Y + Z`	`(X + Y) + Z`
`X - Y + Z`	`(X - Y) + Z`
`X * A / Z`	`X * (A / Z)`
`X * Y - X * Z`	`(X * Y) - (X * Z)`
`A / B / C`	`A / (B * C)`
`A / 5.0`	`0.2 * A`

L'ordre des opérations en FORTRAN suit ces règles hiérarchiques :

- exponentielles, multiplication et division, puis addition et soustraction
- à l'intérieur d'une hiérarchie, l'opération est effectuée de la gauche vers la droite

Expression	Forme alternative à proscrire
`I/2`	`0.5 * I`
`X*I/J`	`X * (I / J)`
`I/J/A`	`I / (J * A)`
`(X + Y) + Z`	`X + (Y + Z)`
`(X * Y) - (X * Z)`	`X * (Y - Z)`
`X * (Y - Z)`	`X*Y-X*Z`

`I, J` sont des entiers et `A, X, Y, Z` des réels ou complexes.

Références

- *Norme IEEE-754*
- *De calculer juste à calculer au plus juste*, Florent de Dinechin
- *Nombres, machines, calculs*, Thierry Dumont
- *Synchronous stochastic arithmetic*, Fabienne Jézéquel
- *Arithmétique virgule flottante*, Jean-Michel Muller
- *Les nombres et l'ordinateur*, Sylvie Boldo
- *Arithmétique des ordinateurs*, Nathalie Revol