Peut-on faire confiance au calcul “flottant” ?

Paul Zimmermann

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE | INRIA
centre de recherche NANCY - GRAND-EST

Institut de Physique du Globe de Strasbourg
16 janvier 2009

Plan de l'exposé

- Motivation
- Le standard IEEE 754
- La bibliothèque MPFR

Motivation

Useful Computations Need Useful Numbers
David R. Stoutemyer
ACM Communications in Computer Algebra
September 2007.

> *Most of us have taken the exact rational and approximate numbers in our computer algebra systems for granted for a long time, not thinking to ask if they could be significantly better.*

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                    -126
                                  0.2604007480 10
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                  -126
                                0.2604007480 10
```

```
> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                        -126
                      0.2604007480 10
```

```
> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                          -126
                0.34288028340847034512 10
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                        -126
                      0.2604007480 10
```

```
> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                          -126
                0.34288028340847034512 10
```

```
> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                        -126
                      0.2604007480 10
```

```
> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                          -126
                0.34288028340847034512 10
```

```
> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                                  -128
0.49076783443012876473973482836733778547443399549250 10
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                        -126
                      0.2604007480 10

> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                         -126
                0.34288028340847034512 10

> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                              -128
0.49076783443012876473973482836733778547443399549250 10

> Digits:=100: evalf(Int(exp(-x^2)*ln(x),x=17..42));
```

Augmenter la précision suffit en général . . .

```
    |\^/|     Maple 10 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2005
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                          -126
                      0.2604007480 10

> Digits:=20: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                           -126
                0.34288028340847034512 10

> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                                -128
0.49076783443012876473973482836733778547443399549250 10

> Digits:=100: evalf(Int(exp(-x^2)*ln(x),x=17..42));
0.490767834430128764739734828367337785474433995492503842\
                                                             -128
   1435498665013506331285359635525372050785062212 10
```

. . . mais pas toujours !

```
> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                          -128
0.49076783443012876473973482836733778547443399549250 10

> Digits:=100: evalf(Int(exp(-x^2)*ln(x),x=17..42));
0.490767834430128764739734828367337785474433995492503842\
                                                            -128
    1435498665013506331285359635525372050785062212 10
```

. . . mais pas toujours !

```
> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                          -128
0.49076783443012876473973482836733778547443399549250 10

> Digits:=100: evalf(Int(exp(-x^2)*ln(x),x=17..42));
0.4907678344301287647397348283673377854744339954925038\
                                                            -128
    1435498665013506331285359635525372050785062212 10

> Digits:=150: evalf(Int(exp(-x^2)*ln(x),x=17..42));
```

. . . mais pas toujours !

```
> Digits:=50: evalf(Int(exp(-x^2)*ln(x),x=17..42));
                                                              -128
0.49076783443012876473973482836733778547443399549250 10

> Digits:=100: evalf(Int(exp(-x^2)*ln(x),x=17..42));
0.4907678344301287647397348283673377854744339954925038\
                                                                -128
    1435498665013506331285359635525372050785062212 10

> Digits:=150: evalf(Int(exp(-x^2)*ln(x),x=17..42));
0.2565728500561051482917356395759084310260126667622196\
    1677972851947606951699614997563055992258263496188670\

                                                               -126
    5745187518615884166042779597226368281066261 3 10
```

It makes me nervous to fly on airplanes, since I know they are designed using floating-point arithmetic.

Alston Householder

(concepteur d'algorithmes flottants et de leur analyse d'erreur)

```
> evalf(sin(2^100));
```

```
> evalf(sin(2^100));
                    0.4491999480
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
              -0.58645356896925826300
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
              -0.58645356896925826300

> evalf(sin(2^100),30);
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
              -0.58645356896925826300

> evalf(sin(2^100),30);
         0.19988562165362573821513281152 5
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
              -0.58645356896925826300

> evalf(sin(2^100),30);
          0.19988562165362573821513281152 5

> evalf(sin(2^100),40);
```

```
> evalf(sin(2^100));
                    0.4491999480

> evalf(sin(2^100),20);
              -0.58645356896925826300

> evalf(sin(2^100),30);
          0.19988562165362573821513281152 5

> evalf(sin(2^100),40);
    -0.8721836054182673097807197782134705593243
```

Que dit la documentation ?

```
> ?evalf
- The evalf command numerically evaluates expressions
  (or subexpressions) involving constants (for
  example, Pi, exp(1), and gamma) and mathematical
  functions (for example, exp, ln, sin, arctan, cosh,
  GAMMA, and erf).

Output
- The evalf command returns a floating-point or
  complex floating-point number or expression.
```

Que dit la documentation ?

```
> ?evalf
- The evalf command numerically evaluates expressions
  (or subexpressions) involving constants (for
  example, Pi, exp(1), and gamma) and mathematical
  functions (for example, exp, ln, sin, arctan, cosh,
  GAMMA, and erf).

Output
- The evalf command returns a floating-point or
  complex floating-point number or expression.

  For detailed information including:
- Complete description of all parameters
- Controlling numeric precision of computations
- Special evaluation for user-defined constants and
  functions
  see the ?evalf/details (evalf,details) help page.
```

Regardons les détails

```
> ?evalf/details
- The evalf command numerically evaluates expressions
  (or subexpressions) involving constants (for example,
  Pi, exp(1), and gamma) and mathematical functions
  (for example, exp, ln, sin, arctan, cosh, GAMMA, and
  erf.

- You can control the precision of all numeric
  computations using the environment variable Digits.
  By default, Digits is assigned the value 10, so the
  evalf command uses 10-digit floating-point
  arithmetic.

See Also: numeric_overview
```

Lisons le « numeric overview »

```
> ?numeric_overview
  The Maple numeric computation environment is designed
  to achieve the following goals.

  1.   Consistency with IEEE standards.
```

Lisons le « numeric overview »

```
> ?numeric_overview
  The Maple numeric computation environment is designed
  to achieve the following goals.

  1.   Consistency with IEEE standards.

> Digits:=3:
> Rounding := 0:
> 1.0 - 9e-5;
                                  1.0
```

2. Consistency across different types of numeric computations (hardware, software, and exact).

2. Consistency across different types of numeric computations (hardware, software, and exact).

```
> evalf(sin(2^100));
                              0.4491999480
```

2. Consistency across different types of numeric computations (hardware, software, and exact).

```
> evalf(sin(2^100));
                          0.4491999480

> evalhf(sin(2^100));
                     -0.87218360541826738
```

Quelle est la sémantique des commandes `evalf()` en Maple, `N[]` en Mathematica, `RR()` en Magma, `n()` en Sage ?

```
----------------------------------------------------------------------
| SAGE Version 2.8.12, Release Date: 2007-11-06                      |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
| SAGE Version 2.8.12, Release Date: 2007-11-06                      |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
sage: n?
Definition:     n(x, prec=None, digits=None)
   Return a numerical approximation of x with
   at least prec bits of precision.
```

```
----------------------------------------------------------------------
| SAGE Version 2.8.12, Release Date: 2007-11-06                      |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
sage: n?
Definition:     n(x, prec=None, digits=None)
   Return a numerical approximation of x with
   at least prec bits of precision.

sage: f=exp(pi*sqrt(163))-262537412640768744
sage: n(f, digits=15)
-1024.00000000000
```

```
----------------------------------------------------------------------
| SAGE Version 2.8.12, Release Date: 2007-11-06                      |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------
sage: n?
Definition:     n(x, prec=None, digits=None)
   Return a numerical approximation of x with
   at least prec bits of precision.

sage: f=exp(pi*sqrt(163))-262537412640768744
sage: n(f, digits=15)
-1024.00000000000
sage: n(f, digits=30)
-0.000000000007673861546209082007408142089844
```

```
sage: f=sin(x)^2+cos(x)^2-1
```

```
sage: f=sin(x)^2+cos(x)^2-1
sage:  f.nintegrate(x,0,1)
(-1.1189600789284899e-18, 6.447843603224434e-18,
  8379, 5)
```

```
sage: f=sin(x)^2+cos(x)^2-1
sage:  f.nintegrate(x,0,1)
(-1.1189600789284899e-18, 6.447843603224434e-18,
  8379, 5)

sage: f.nintegrate?
OUTPUT:
  -- float: approximation to the integral
  -- float: estimated absolute error of the approximation
  -- the number of integrand evaluations
  -- an error code:
       0 -- no problems were encountered
       1 -- too many subintervals were done
       2 -- excessive roundoff error
       3 -- extremely bad integrand behavior
       4 -- failed to converge
       5 -- integral is probably divergent or slowly convergent
       6 -- the input is invalid
```

```
sage: f=exp(pi*sqrt(163))-262537412640768744
sage: f.nintegrate(x,0,1)
(-480.00000000000011, 5.3290705182007538e-12, 21, 0)
```

```
sage: f=exp(pi*sqrt(163))-262537412640768744
sage: f.nintegrate(x,0,1)
(-480.00000000000011, 5.3290705182007538e-12, 21, 0)
```

Est-ce un bug ?

Constat principal

pas de spécification $\Longrightarrow$ pas de bug

Constat principal

pas de spécification $\Longrightarrow$ pas de bug

. . . mais inutile

Constat principal

pas de spécification $\Longrightarrow$ pas de bug

. . . mais inutile

sémantique bien définie

Constat principal

pas de spécification $\Longrightarrow$ pas de bug

. . . mais inutile

sémantique bien définie

$\Longrightarrow$ peut être utile

Le standard IEEE 754

Le standard IEEE 754

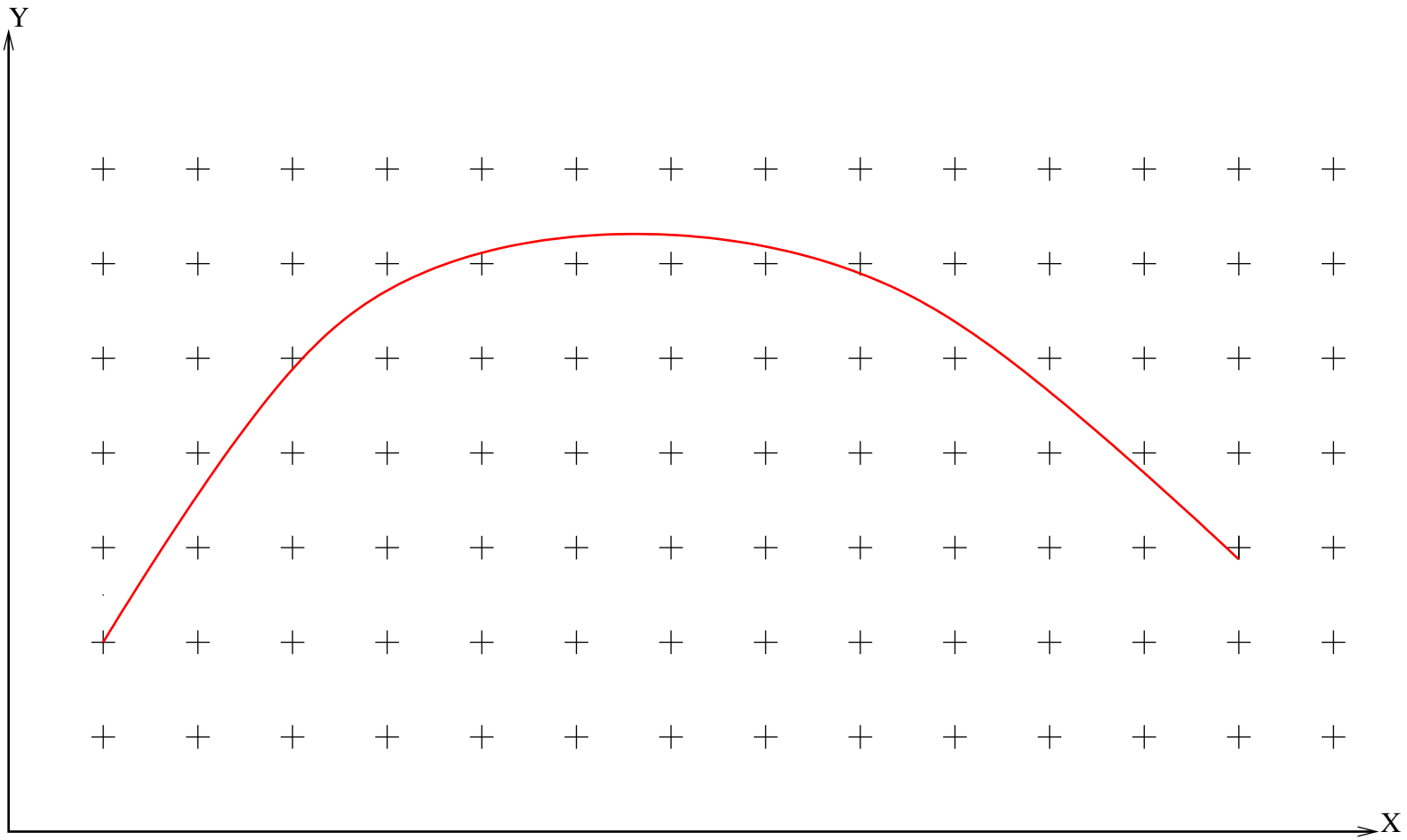
- première version en 1985, révisé en 2008
- 754-1985 : quatre formats binaires différents (simple, simple étendu, double, double étendu)
- quatre modes d'arrondi
- impose l'arrondi correct pour $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$
- exceptions (*underflow, overflow, inexact, invalid*)
- aujourd'hui largement répandu

Arrondi correct

	← Negative Infinity							Zero →	← Zero							Positive Infinity →
Mode	**-2.0**	**-1.7**	**-1.5**	**-1.3**	**-1.0**	**-0.7**	**-0.5**	**-0.3**	**0.0**	**0.3**	**0.5**	**0.7**	**1.0**	**1.3**	**1.5**	**1.7**	**2.0**
R-H-U (s)	-2	-2	-2	-1	-1	-1	-1	-0	0	0	1	1	1	1	2	2	2
R-H-U (a)	-2	-2	-1	-1	-1	-1	-0	-0	0	0	1	1	1	1	2	2	2
R-H-D (s)	-2	-2	-1	-1	-1	-1	-0	-0	0	0	0	1	1	1	1	2	2
R-H-D (a)	-2	-2	-2	-1	-1	-1	-1	-0	0	0	0	1	1	1	1	2	2
R-H-E	-2	-2	-2	-1	-1	-1	-0	-0	0	0	0	1	1	1	2	2	2
R-H-O	-2	-2	-1	-1	-1	-1	-1	-0	0	0	1	1	1	1	1	2	2
R-C	-2	-1	-1	-1	-1	-0	-0	-0	0	1	1	1	1	2	2	2	2
R-F	-2	-2	-2	-2	-1	-1	-1	-1	0	0	0	0	1	1	1	1	2
R-T-Z	-2	-1	-1	-1	-1	-0	-0	-0	0	0	0	0	1	1	1	1	2
R-A-F-Z	-2	-2	-2	-2	-1	-1	-1	-1	0	1	1	1	1	2	2	2	2

R-H-U = Round-Half-Up R-H-D = Round-Half-Down (s) = Symmetric
R-H-E = Round-Half-Even R-H-O = Round-Half-Odd (a) = Asymmetric
R-C = Round-Ceiling R-F = Round-Floor
R-T-Z = Round-Toward-Zero R-A-F-Z = Round-Away-From-Zero

Dilemme du fabricant de tables



Bornes d'erreur

$$y = \circ(x)$$

- arrondi au plus proche : $|y - x| \leq \frac{1}{2}\mathrm{ulp}(y)$

Bornes d'erreur

$$y = \circ(x)$$

- arrondi au plus proche : $|y - x| \leq \frac{1}{2}\mathrm{ulp}(y)$
- arrondis dirigés : $|y - x| \leq \mathrm{ulp}(y)$
  - vers zéro : $|y| \leq |x|$

Bornes d'erreur

$$y = \circ(x)$$

- arrondi au plus proche : $|y - x| \leq \frac{1}{2}\mathrm{ulp}(y)$
- arrondis dirigés : $|y - x| \leq \mathrm{ulp}(y)$
  - vers zéro : $|y| \leq |x|$
  - vers $-\infty$: $y \leq x$

Bornes d'erreur

$$y = \circ(x)$$

- arrondi au plus proche : $|y - x| \leq \frac{1}{2}\mathrm{ulp}(y)$
- arrondis dirigés : $|y - x| \leq \mathrm{ulp}(y)$
  - vers zéro : $|y| \leq |x|$
  - vers $-\infty$: $y \leq x$
  - vers $+\infty$: $y \geq x$

Lemme de Sterbenz

Lemme. *Si x et y sont deux flottants tels que $y/2 < x < 2y$, alors :*

$$\circ(x - y)$$

est exact.

```
sage: R=RealField(42)
sage: x=R(catalan)
sage: y=R(euler_gamma)
sage: x, y
(0.915965594177, 0.577215664902)
sage: z=x-y
sage: z
0.338749929276
sage: x.exact_rational() - y.exact_rational()
1489837944587/4398046511104
sage: z.exact_rational()
1489837944587/4398046511104
```

FastTwoSum

Théorème. *Si* $|a| \geq |b|$, *et :*

$$\begin{aligned} s &\leftarrow \circ(a+b) \\ t &\leftarrow \circ(s-a) \\ u &\leftarrow \circ(b-t) \end{aligned}$$

alors

$$a+b = s+u.$$

```
sage: R=RealField(53,sci_not=1)
sage: a=R(pi)
sage: b=R(exp(1))
sage: s=a+b
sage: t=s-a
sage: u=b-t
sage: s, u
(5.85987448204884e0, 4.44089209850063e-16)
sage: a.exact_rational()+b.exact_rational()\
     -s.exact_rational()-u.exact_rational()
0
```

Le programme ci-dessous calcule la base interne :

```
sage: A = 1.0
sage: B = 1.0
sage: while ((A + 1.0) - A) - 1.0 == 0.0:
....:         A = 2.0 * A
sage: while ((A + B) - A) - B <> 0.0:
....:         B = B + 1.0
sage: B
2.00000000000000
```

```
> A := 1.0:
> B := 1.0:
> while (evalf(A + 1.0) - A) - 1.0 = 0.0 do
    A := 2.0 * A
  od:
> while ((A + B) - A) - B <> 0.0 do
    B := B + 1.0
  od:
> B;
                              10.0
```

Opteron, Linux 2.6.12, gcc 4.0.1, libc 2.3.5 :

```
Testing function atan for exponent 0.
 rounding mode GMP_RNDU:
  1.507141 ulp(s) for x=5.2734875051429341 8412e-01
  wrong DR: x=8.7115929270125391 7812e-01 [-0.505215]

Testing function cbrt for exponent 0.
 rounding mode GMP_RNDN:
  wrong monotonicity for x=8.9055049757491810 9578e-01
             f(x-)=9.6209845421919726 3271e-01
       not <= f(x)=9.6209845421919715 2248e-01
```

Sparc, SunOS 5.7, cc Sun WorkShop 6 :

```
Testing function exp for exponent 0.
 rounding mode GMP_RNDN:
  0.659120 ulp(s) for x=9.4334449125543784475 7e-01
 rounding mode GMP_RNDU:
  wrong DR: x=5.3382449867961789813 4e-01 [-0.295496]
Testing function pow for exponents 0 and 0.
 rounding mode GMP_RNDN:
  -0.522792 ulp(s) for x=9.9110907189521668669 8e-01
                        t=6.0662731225498922604 8e-01
Testing function tanh for exponent 0.
 rounding mode GMP_RNDN:
  1.771299 ulp(s) for x=5.1924036858115574233 4e-01
```

MIPS R16000, IRIX64 6.5, gcc 3.3 :

```
Testing function tan for exponent 10.
 rounding mode GMP_RNDZ:
  -6.143332 ulp(s) for x=5.2542719838976336000e+02
  wrong DR: x=7.5607852096729857000e+02 [-4.523771]
```

Itanium 1, Linux 2.4.20, gcc 3.2.3, libc 2.2.5 :

```
Testing function gamma for exponent 7.
 rounding mode GMP_RNDN:
  -610.873724 ulp(s) for x=1.2220157663154327565e+02
```

Pentium 4, Linux 2.6.11, gcc 3.4.3, libc 2.3.5 :

```
Testing function tan for exponent 10.
 rounding mode GMP_RNDZ:
  -6.143332 ulp(s) for x=5.254271983897633600000e+02
  wrong DR: x=7.560785209672985700000e+02 [-4.523771]
```

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`
- deux encodages décimaux (DPD et BID)

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`
- deux encodages décimaux (DPD et BID)
- arrondi correct pour toutes les conversions

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`
- deux encodages décimaux (DPD et BID)
- arrondi correct pour toutes les conversions
- et pour les fonctions mathématiques [recommandé]

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`
- deux encodages décimaux (DPD et BID)
- arrondi correct pour toutes les conversions
- et pour les fonctions mathématiques [recommandé]
- évaluation des expressions

IEEE 754-2008

- nouveaux formats `decimal32`, `decimal64`, `decimal128`
- deux encodages décimaux (DPD et BID)
- arrondi correct pour toutes les conversions
- et pour les fonctions mathématiques [recommandé]
- évaluation des expressions
- résultats reproductibles

La bibliothèque MPFR

www.mpfr.org

A lot of code involving a little floating-point will be written by many people who have ***never attended*** *my (nor anyone else's) numerical analysis classes. We had to enhance the likelihood that* ***their programs would get correct results.*** *At the same time we had to ensure that people who really are expert in floating-point could write* ***portable software*** *and prove that it worked, since so many of us would have to rely upon it. There were a lot of almost* ***conflicting requirements*** *on the way to a balanced design.*

William Kahan, An Interview with the Old Man of Floating-Point, February 1998.

Historique de MPFR

- 1998 : texte fondateur
- 1999 : premières lignes de code
- 2000 : arrivée de Vincent Lefèvre
- 2001 : postdoc de David Daney
- 2003-2005 : Patrick Pélissier (IA)
- 2004 : version 2.1.0, **gfortran** utilise MPFR
- 2005 : MPFR gagne la compétition *Many Digits*
- 2007-2009 : Philippe Théveny, ODL MPtools
- 2008 : GCC 4.3 utilise MPFR, version 2.3.0
- janvier 2009 : sortie de GNU MPFR 2.4.0

MPFR est aujourd'hui dans la plupart des distributions Linux :

Modèle de calcul

Extension de IEEE 754 à la précision arbitraire :

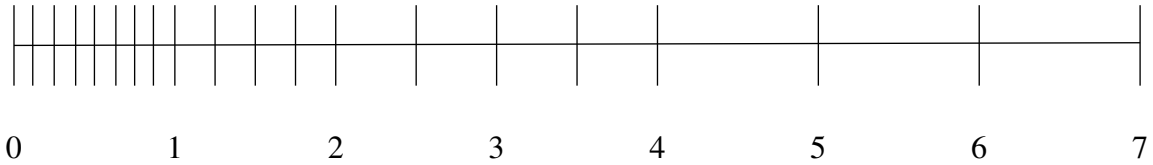
- formats : précision arbitraire p

$$x = \pm 0.\underbrace{b_1 b_2 \ldots b_p}_{p \text{ bits}} \cdot 2^e$$

  avec $E_{\min} \leq e \leq E_{\max}$;
- 5 nombres spéciaux ± 0, $\pm\infty$, NaN ;
- modes d'arrondi : quatre modes IEEE 754.

Limites de MPFR

- la base interne est fixée (2) ;
- précision $p \geq 2$;
- $E_{\min}$ et $E_{\max}$ sont globales (par défaut $E_{\min} = -2^{30} + 1$, $E_{\max} = 2^{30} - 1$) ;
- pas de dénormalisés (mais `mpfr_subnormalize`) ;



- les opérations sont **atomiques** (comme IEEE 754).

Différences avec IEEE 754

- chaque variable a sa propre précision :

```
mpfr_init2 (a, 17);
mpfr_init2 (b, 42);
```

Différences avec IEEE 754

- chaque variable a sa propre précision :

```
mpfr_init2 (a, 17);
mpfr_init2 (b, 42);
```

- opérations « mixtes » autorisées (sans double arrondi)

```
mpfr_sqrt (a, b, GMP_RNDN);
```

Différences avec IEEE 754

- chaque variable a sa propre précision :

```
mpfr_init2 (a, 17);
mpfr_init2 (b, 42);
```

- opérations « mixtes » autorisées (sans double arrondi)

```
mpfr_sqrt (a, b, GMP_RNDN);
```

- opérations « en place » autorisées

```
mpfr_sqrt (a, a, GMP_RNDN);
```

Arrondi correct

Pour chaque opération, MPFR garantit **l'arrondi correct** :

- opérations arithmétiques de base $(+, -, \times, \div)$;
- fonctions algébriques $\sqrt{\cdot}$, $\sqrt{x^2 + y^2}$, x^n, ... ;
- fonctions élémentaires et spéciales : exp, log, sin, ..., erf, Bessel, ...
- **conversions** (types `long`, `char*`, `double`, `long double`, `mpz_t`, `mpq_t`).

Conséquence : **un** résultat correct **et un seul !**

Arrondi correct

Pour chaque opération, MPFR garantit **l'arrondi correct** :

- opérations arithmétiques de base $(+, -, \times, \div)$;
- fonctions algébriques $\sqrt{\cdot}$, $\sqrt{x^2 + y^2}$, x^n, ... ;
- fonctions élémentaires et spéciales : exp, log, sin, ..., erf, Bessel, ...
- **conversions** (types `long`, `char*`, `double`, `long double`, `mpz_t`, `mpq_t`).

Conséquence : **un** résultat correct **et un seul !**
Corollaire 1 : portabilité du code (processeurs, compilateurs, versions de MPFR)

Arrondi correct

Pour chaque opération, MPFR garantit **l'arrondi correct** :

- opérations arithmétiques de base $(+, -, \times, \div)$;
- fonctions algébriques $\sqrt{\cdot}$, $\sqrt{x^2+y^2}$, x^n, ... ;
- fonctions élémentaires et spéciales : exp, log, sin, ..., erf, Bessel, ...
- **conversions** (types `long`, `char*`, `double`, `long double`, `mpz_t`, `mpq_t`).

Conséquence : **un** résultat correct **et un seul !**
Corollaire 1 : portabilité du code (processeurs, compilateurs, versions de MPFR)
Corollaire 2 : arrondis dirigés $\Longrightarrow$ arithmétique d'intervalles (MPFI, P1788)

MPFR : architecture interne

3 niveaux de fonctions :

- bas niveau (addition, soustraction, multiplication, division, racine carrée) ;
- fonctions élémentaires primitives ($\exp, \log, \sin$) ;
- autres fonctions élémentaires et spéciales.

Fonctions de bas niveau

Implantation native au-dessus de la couche `mpn` de GMP :

$$0.\underbrace{1101011100}_{10} + 0.\underbrace{11111011110110}_{13} \cdot 2^{-3} \rightarrow \underbrace{xxxxxx}_{6} \cdot 2^{y}$$

```
.1101011100
    .1111011110110
.1111010111110110
```

Fonctions élémentaires primitives

Exemple : exp en précision n.

- réduction d'argument :

$$x \to r = x/2^k$$

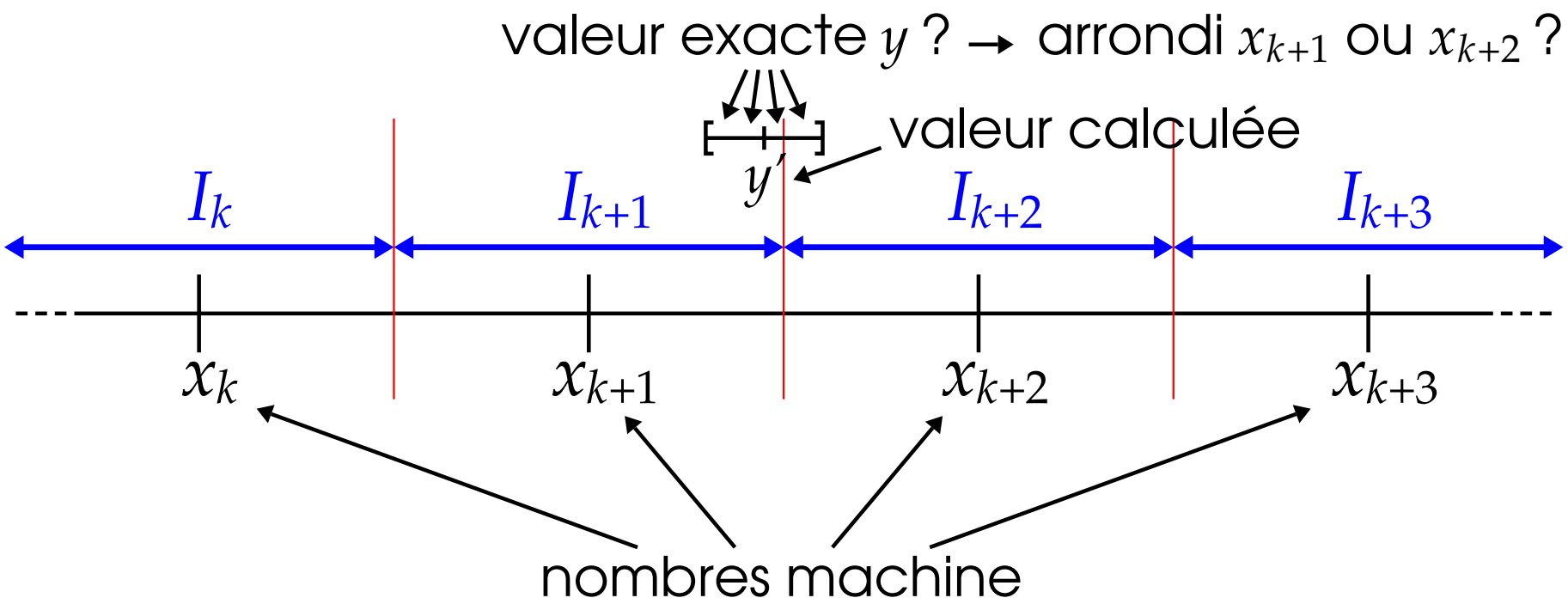
- série de Taylor :

$$\exp r \approx 1 + r + \frac{r^2}{2!} + \frac{r^3}{3!} + \cdots + \frac{r^\ell}{\ell!}$$

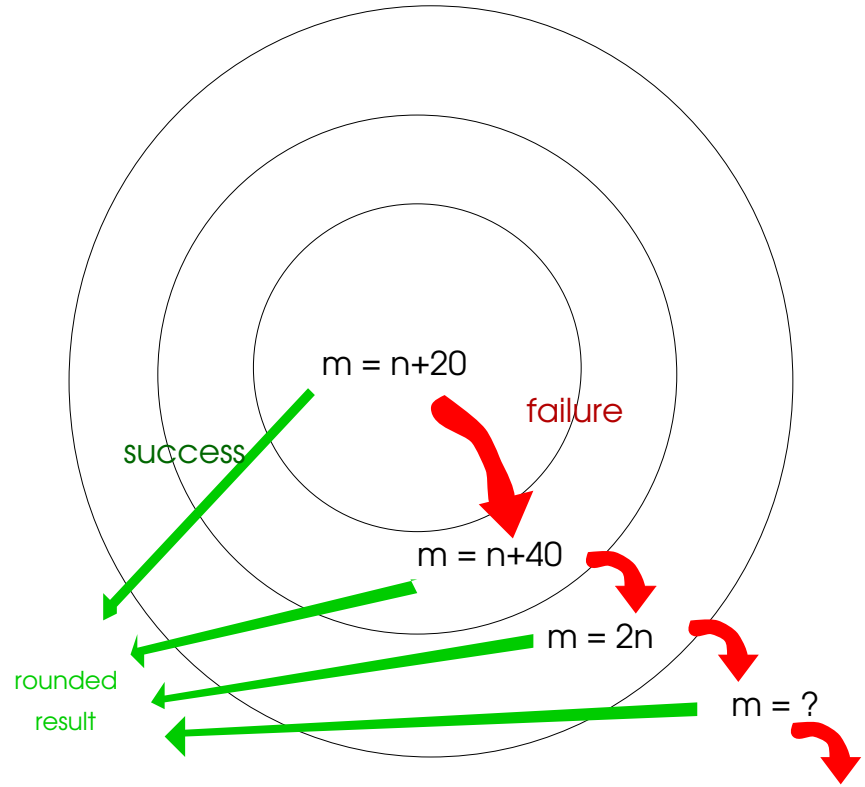
- reconstruction :

$$x = r2^k \Longrightarrow \exp x = (\exp r)^{2^k}$$

Comment garantir l'arrondi correct ?



Stratégie de Ziv



- en cas d'échec, $p \leftarrow p + 32$ ou $p + 64$;
- échecs multiples : $p \leftarrow p + \lfloor p/2 \rfloor$.

Autres fonctions mathématiques

Réduction aux fonctions primitives, avec la stratégie de Ziv :

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\begin{aligned} u &\leftarrow \circ(e^x) \\ v &\leftarrow \circ(u^{-1}) \\ w &\leftarrow \circ(u + v) \\ s &\leftarrow \frac{1}{2} w \qquad [exact] \end{aligned}$$

Tests

- tests de couverture du code ($> 95\%$) ;
- tests de non-régression (bugs corrigés) ;
- tests aléatoires (toutes les nuits) :

$$\begin{aligned} y &\leftarrow \circ_p(f(x)) \\ t &\leftarrow \circ_{p+10}(f(x)) \\ z &\leftarrow \circ_p(t) \end{aligned}$$

  Si pas de problème de double arrondi, on doit avoir $y = z$.
- bases de données (warning . . .).

Efficacité (petite précision)

Précision 53 bits sur Pentium 4 et Athlon (cycles) :

version	machine	add	sub	mul	div	sqrt
2.0.1	Pentium 4	298	398	331	1024	1211
2.1.0	Pentium 4	211	213	268	549	1084
2.0.1	Athlon	222	323	270	886	975
2.1.0	Athlon	132	151	183	477	919

Comparaison entre MPFR, CLN, PARI, NTL

Athlon 1.8Ghz, millisecondes, $x = \sqrt{3} - 1$, $y = \sqrt{5}$:

	chiffres	MPFR 2.2.0	CLN 1.1.11	PARI 2.2.12-beta	NTL 5.4
$x \cdot y$	10^2	**0.00048**	0.00071	0.00056	0.00079
	10^4	**0.48**	0.81	0.58	0.57
x/y	10^2	**0.0010**	0.0013	0.0011	0.0020
	10^4	**1.2**	2.4	**1.2**	**1.2**
$\sqrt{x}$	10^2	**0.0014**	0.0016	0.0015	0.0037
	10^4	**0.81**	1.58	0.82	1.23

Comparaison entre MPFR, CLN, PARI, NTL

	chiffres	MPFR 2.2.0	CLN 1.1.11	PARI 2.2.12-beta	NTL 5.4
exp x	10^2	**0.017**	0.060	0.032	0.140
	10^4	**54**	70	68	1740
log x	10^2	**0.031**	0.076	0.037	0.772
	10^4	**34**	79	40	17940
sin x	10^2	**0.022**	0.056	0.032	0.155
	10^4	**78**	129	134	1860
atan x	10^2	0.28	**0.067**	0.076	NA
	10^4	610	**149**	151	NA

Limites de MPFR

- arrondi exact seulement pour opérations **atomiques** : arithmétique d'intervalles (MPFI), RealRAM (iRRAM, RealLib)

Limites de MPFR

- arrondi exact seulement pour opérations **atomiques** : arithmétique d'intervalles (MPFI), RealRAM (iRRAM, RealLib)
- pas de précision (*precision*) automatique en fonction de la précision (*accuracy*) des résultats ;
- base fixée à 2 (cf `decNumber` pour base 10)

Limites de MPFR

- arrondi exact seulement pour opérations **atomiques** : arithmétique d'intervalles (MPFI), RealRAM (iRRAM, RealLib)
- pas de précision (*precision*) automatique en fonction de la précision (*accuracy*) des résultats ;
- base fixée à 2 (cf `decNumber` pour base 10)
- pas d'algorithmes de « haut niveau » : racine d'un polynôme (MPC), algèbre linéaire (ALGLIB.NET), intégration numérique (CRQ)

Limites de MPFR

- arrondi exact seulement pour opérations **atomiques** : arithmétique d'intervalles (MPFI), RealRAM (iRRAM, RealLib)
- pas de précision (*precision*) automatique en fonction de la précision (*accuracy*) des résultats ;
- base fixée à 2 (cf `decNumber` pour base 10)
- pas d'algorithmes de « haut niveau » : racine d'un polynôme (MPC), algèbre linéaire (ALGLIB.NET), intégration numérique (CRQ)
- preuves « à la main »

Premier programme MPFR

```
#include <stdio.h>
#include "mpfr.h"

int main ()
{
   unsigned long i;
   mpfr_t s, t;
   mpfr_init2 (s, 100); mpfr_init2 (t, 100);
   mpfr_set_ui (t, 1, GMP_RNDN);
   mpfr_set (s, t, GMP_RNDN);
   for (i = 1; i <= 29; i++)
      {
         mpfr_div_ui (t, t, i, GMP_RNDN);
         mpfr_add (s, s, t, GMP_RNDN);
      }
   mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
   printf ("\n");
   mpfr_clear (s); mpfr_clear (t);
}
```

```
#include "mpfr.h"
```

Inclut le fichier d'en-tête MPFR.

```
#include "mpfr.h"
```

Inclut le fichier d'en-tête MPFR.

```
mpfr_t s, t;
```

Déclare deux variables *s* et *t*.

```
#include "mpfr.h"
```

Inclut le fichier d'en-tête MPFR.

```
   mpfr_t s, t;
```

Déclare deux variables *s* et *t*.

```
   mpfr_init2 (s, 100); mpfr_init2 (t, 100);
```

Initialise *s* et *t*, avec une précision de 100 bits.

```
#include "mpfr.h"
```

Inclut le fichier d'en-tête MPFR.

```
mpfr_t s, t;
```

Déclare deux variables *s* et *t*.

```
mpfr_init2 (s, 100); mpfr_init2 (t, 100);
```

Initialise *s* et *t*, avec une précision de 100 bits.

```
mpfr_set_ui (t, 1, GMP_RNDN);
mpfr_set (s, t, GMP_RNDN);
```

Met *t* à 1, arrondi au plus proche, et copie *t*, arrondi au plus proche, dans *s*.

```
mpfr_div_ui (t, t, i, GMP_RNDN);
```

Divise *t* par *i*, arrondi au plus proche.

```
mpfr_div_ui (t, t, i, GMP_RNDN);
```

Divise *t* par *i*, arrondi au plus proche.

```
mpfr_add (s, s, t, GMP_RNDN);
```

Ajoute *t* à *s*, avec arrondi au plus proche.

```
mpfr_div_ui (t, t, i, GMP_RNDN);
```

Divise *t* par *i*, arrondi au plus proche.

```
mpfr_add (s, s, t, GMP_RNDN);
```

Ajoute *t* à *s*, avec arrondi au plus proche.

```
mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
```

Affiche *s* en décimal, avec arrondi au plus proche (nombre de chiffres décimaux déduit de la précision de *s*).

```
mpfr_div_ui (t, t, i, GMP_RNDN);
```

Divise *t* par *i*, arrondi au plus proche.

```
mpfr_add (s, s, t, GMP_RNDN);
```

Ajoute *t* à *s*, avec arrondi au plus proche.

```
mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
```

Affiche *s* en décimal, avec arrondi au plus proche (nombre de chiffres décimaux déduit de la précision de *s*).

```
mpfr_clear (s); mpfr_clear (t);
```

Libère la mémoire occupée par *s* et *t*.

```
mpfr_div_ui (t, t, i, GMP_RNDN);
```

Divise t par i, arrondi au plus proche.

```
mpfr_add (s, s, t, GMP_RNDN);
```

Ajoute t à s, avec arrondi au plus proche.

```
mpfr_out_str (stdout, 10, 0, s, GMP_RNDN);
```

Affiche s en décimal, avec arrondi au plus proche (nombre de chiffres décimaux déduit de la précision de s).

```
mpfr_clear (s); mpfr_clear (t);
```

Libère la mémoire occupée par s et t.

Si on remplace `GMP_RNDN` par `GMP_RNDZ`, on obtient une borne inférieure de $e = \sum_{n \geq 0} \frac{1}{n!}$.

Compilation et exécution

```
$ gcc sample.c -lmpfr -lgmp
```

Compilation et exécution

```
$ gcc sample.c -lmpfr -lgmp
```

ou :

```
$ echo $MPFR
/usr/local/mpfr-2.3.0
$ gcc -I$MPFR/include sample.c $MPFR/lib/libmpfr.a
```

Compilation et exécution

```
$ gcc sample.c -lmpfr -lgmp
```

ou :

```
$ echo $MPFR
/usr/local/mpfr-2.3.0
$ gcc -I$MPFR/include sample.c $MPFR/lib/libmpfr.a
```

et :

```
$ ./a.out
2.71828182845904523536028747134 81
```

Applications

Bibliothèque CRQ (L. Fousse) : intégration numérique avec borne d'erreur garantie

Applications

Bibliothèque CRQ (L. Fousse) : intégration numérique avec borne d'erreur garantie

FPLLL (D. Stehlé) : réduction de réseau utilisant des flottants.

$$\mu_{ij} = \lfloor \frac{b_i b_j^*}{||b_j^*||^2} \rceil$$

David R. Stoutemyer :

> *The astounding increase in computer speed and memory size since floating-point arithmetic was first implemented makes it affordable to use interval arithmetic and self-validating algorithms for almost all approximate scientific computation. It should be the default approximate arithmetic — especially in computer algebra [. . .]*