Précision en virgule flottante

Vincent Lafage

IJCLab, CNRS/IN2P3 & Université Paris-Saclay, Orsay, France

jeudi 8 février 2024

Calculs du monde flottant

Revisiting "What Every Computer Scientist Should Know About Floating-point Arithmetic"

- Les nombres : réels, algébriques, constructibles, rationnels, décimaux, binaires, flottants…
- Des types « primitifs » : `float`, `double`, `long double`, `quad`, `half`…
- Des calculs qui ne se passent pas comme prévu…(pourquoi, comment)
  - erreurs d'arrondis
  - erreurs de conversion
  - propagation d'erreurs
  - composition d'erreurs
- Adimensionalisation et réduction d'entropie des formules
- Pourquoi les calculs de géométrie sont compliqués

Des nombres qui tuent et qui coûtent

- Missiles Patriot, première guerre du Golfe, 1991 :
  erreur de $600\,\mathrm{m}$ sur l'interception : 28 morts, une centaine de blessés
- Bourse de Vancouver, 1982 :
  erreur accumulée pendant deux ans sur la valeur d'un indice boursier
  $52\,\%$ d'erreur : $524.811\,\$$ au lieu de $1098.892\,\$$

https://doi.org/10.1145/103162.103163

https://www.validlab.com/goldberg/paper.pdf (avec annexe)

"Floating-point arithmetic is considered an esoteric subject by many people"

What Every Computer Scientist Should Know About Floating-Point Arithmetic

DAVID GOLDBERG

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304

Floating-point arithmetic is considered an esoteric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point datatype; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding

Formats

« computing is about representation »

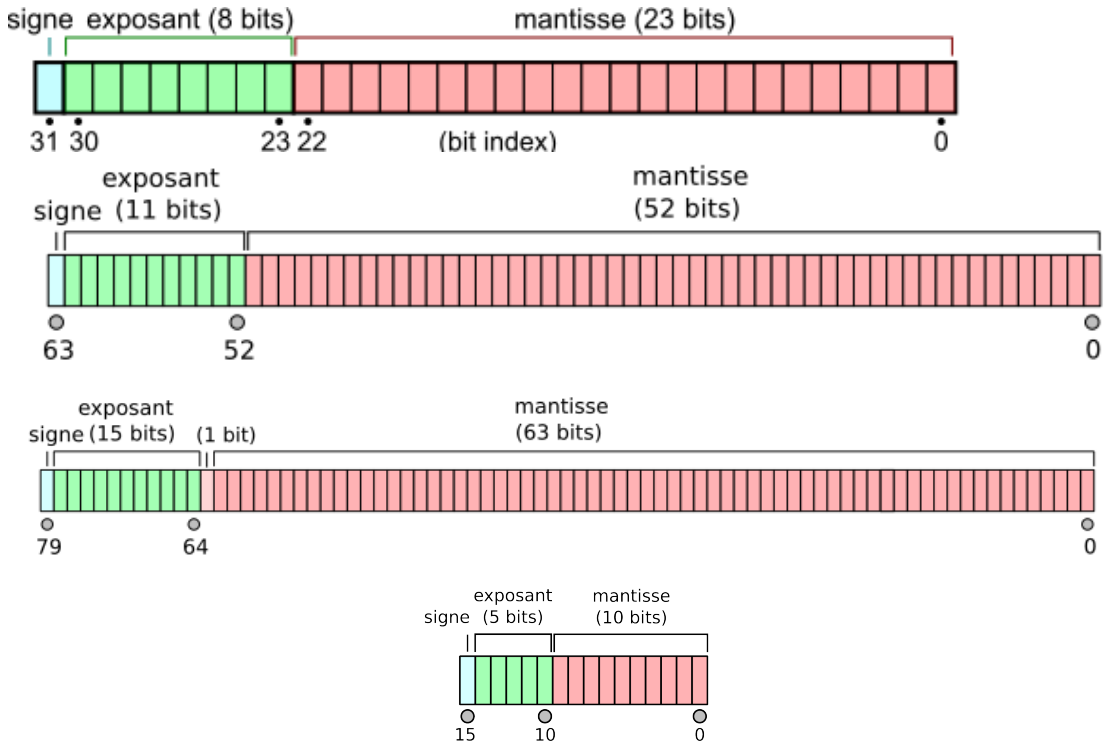
Notation scientifique :

significande $\times$ base$^{\text{exposant}}$ significande $\in \mathbb{Z}$, exposant $\in \mathbb{Z}$

Forme standard : mantisse, alias *significande normalisée*

mantisse $\times$ base$^{\text{exposant}}$ mantisse $\in [1\,;\,\text{base}[$, exposant $\in \mathbb{Z}$

Astuce, en base 2 : le chiffre le plus significatif est toujours 1…



Dans les registres du processeur, la mantisse est élargie de trois bits : bit de garde, bit d'arrondi et "sticky" bit

Exemples de `float32`

$$\texttt{float} = (-1)^S \times 2^{E-127} \times (1+M), \quad M \in [0,1[$$

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

S	E	M	
0	0111 1111	000 0000 0000 0000 0000 0000	$1 = 2^0 \times (1+0)$
0	1000 0000	000 0000 0000 0000 0000 0000	$2 = 2^1 \times (1+0)$
0	1000 0000	100 0000 0000 0000 0000 0000	$3 = 2^1 \times (1+1/2)$
0	1000 0000	110 0000 0000 0000 0000 0000	$3.5 = 2^1 \times (1+1/2+1/4)$
0	1000 0000	111 0000 0000 0000 0000 0000	$3.75 = 2^1 \times (1+1/2+1/4+1/8)$
1	0111 1111	000 0000 0000 0000 0000 0000	$-1 = -2^0 \times (1+0)$
0	0111 1110	000 0000 0000 0000 0000 0000	$1/2 = 2^{-1} \times (1+0)$
0	0111 1100	100 1100 1100 1100 1100 1101	$0.2 = 2^{-3} \times (1+1/2) \times \sum_n 1/16^n$
0	0111 1101	010 1010 1010 1010 1010 1011	$1/3 = 2^{-2} \times (1+1/4) \times \sum_n 1/16^n$
0	0111 1111	011 0101 0000 0100 1111 0011	$\sqrt{2}$
0	1000 0000	100 1001 0000 1111 1101 1011	$\pi \simeq 2^1 \times (1+1/2+1/16+1/128+\cdots)$
0	0000 0000	000 0000 0000 0000 0000 0000	0 special representation
1	0000 0000	000 0000 0000 0000 0000 0000	0_- special representation
0	1111 1110	111 1111 1111 1111 1111 1111	largest float $3.402823466 \times 10^{38}$
0	1111 1111	000 0000 0000 0000 0000 0000	$+\infty$ = `Inf` special representation
0	1111 1111	1xx xxxx xxxx xxxx xxxx xxxx	`NaN` special representation
0	1111 1111	1xx xxxx xxxx xxxx xxxx xxxx	`qNaN` *quiet* special representation
0	1111 1111	01x xxxx xxxx xxxx xxxx xxxx	`sNaN` *signaling* special representation
0	0000 0001	000 0000 0000 0000 0000 0000	smallest positive float $1.17549435 \times 10^{-38}$
0	0000 0000	000 0000 0000 0000 0000 0001	smallest **denormal** positive float 1.401×10^{-45}
0	0000 0000	111 1111 1111 1111 1111 1111	largest **denormal** positive float $1.175\,493\,79 \times 10^{-38}$

⇒ *représentez vos nombres favoris avec* `https://www.h-schmidt.net/FloatConverter/IEEE754.html`

Affichez en Hexadécimal C

```c
#include <stdio.h>

int main ()
{
    float x = 1.0f;
    printf ("%f = %a\n", x, x);
    x = 2.0f;
    printf ("%f = %a\n", x, x);
    x = 3.0f;
    printf ("%f = %a\n", x, x);
    x = 3.1415926535897 93f;
    printf ("%f = %a\n", x, x);
}
```

```
1.000000 = 0x1p+0
2.000000 = 0x1p+1
3.000000 = 0x1.8p+1
3.141593 = 0x1.921fb6p+1
```

Affichez en Hexadécimal C++

```cpp
#include <iostream>

int main ()
{
    float x = 1.0f;
    std::cout << x << " = " << std::hexfloat << x << std::defaultfloat << '\n';
    x = 2.0f;
    std::cout << x << " = " << std::hexfloat << x << std::defaultfloat << '\n';
    x = 3.0f;
    std::cout << x << " = " << std::hexfloat << x << std::defaultfloat << '\n';
    x = 3.141592653589793f;
    std::cout << x << " = " << std::hexfloat << x << std::defaultfloat << '\n';
}
```

```
1 = 0x1p+0
2 = 0x1p+1
3 = 0x1.8p+1
3.14159 = 0x1.921fb6p+1
```

Affichez en Hexadécimal Fortran

```
program hexfloat
  use, intrinsic :: iso_fortran_env, only: real32
  implicit none
  real (real32) :: x

  x = 1
  write (*, '(F10.6,A,Z16)') x, ' = ', x
  x = 2
  write (*, '(F10.6,A,Z16)') x, ' = ', x
  x = 3
  write (*, '(F10.6,A,Z16)') x, ' = ', x
  x = acos (-1.0_real32)
  write (*, '(F10.6,A,Z16)') x, ' = ', x
end program hexfloat
```

```
  1.000000 =         3F800000
  2.000000 =         40000000
  3.000000 =         40400000
  3.141593 =         40490FDB
```

Affichez en Hexadécimal Python

```
#!/usr/bin/python3

x = 1.0
print (x, " = ", float.hex(x))
x = 2.0
print (x, " = ", float.hex(x))
x = 3.0
print (x, " = ", float.hex(x))
x = 3.141592653589793
print (x, " = ", float.hex(x))
```

```
1.0  =  0x1.0000000000000p+0
2.0  =  0x1.0000000000000p+1
3.0  =  0x1.8000000000000p+1
3.141592653589793  =  0x1.921fb54442d18p+1
```

Interface

Table – *leaky abstraction* : standardiser l'interface

C / C++	Fortran'90	ieee_arithmetic	Ada
`copysign (d x, d y)`	`sign (x, y)`	`ieee_copy_sign (x, y)`	`F'Copy_Sign (value, sign)`
`frexp (d x, i *exp)`	`exponent (x)` `fraction (x)`	`ieee_logb (x)`	`F'Exponent (x)` `F'Fraction (x)`
`ldexp (d x, i exp)` `scalbn (d x, i exp)`	`set_exponent (x, i)`	`ieee_scalb (x, i)`	`F'Scaling (x, adjustment)`
`nextafter(d x, d y)`	`nearest (x, s)`	`ieee_next_after (x, y)`	`F'Adjacent (x, towards)`
`numeric_limits::radix`	`radix (x)`		`F'Machine_Radix`
`numeric_limits::epsilon ()`	`epsilon (x)`		`F'Model_Epsilon`
	`precision (x)`		
`numeric_limits::digits`	`digits (x)`		
	`range (x)`		
			`F'Machine_Mantissa`
`numeric_limits::min_exponent`	`minexponent (x)`		`F'Machine_Emin`
`numeric_limits::max_exponent`	`maxexponent (x)`		`F'Machine_Emax`
	`spacing (x)`		
	`rrspacing (x)`		
`nearbyint (d x)` `rint(d x)`	`nint (x)`	`ieee_rint (x)`	`F'Rounding (x)`
`floor (d x)`	`floor (x)`		`F'Floor (x)`
`ceil (d x)`	`ceiling (x)`		`F'Ceiling (x)`
		`ieee_rem (x, y)`	`F'Remainder (x, y)`

Unfortunately the C/C++ API doesn't vectorise well.

$0.1 + 0.2 \neq 0.3$?

$$\Sigma = a + b =^{?} c \qquad \Delta = a + b - c$$

with

$$a = 0.1 \qquad b = 0.2 \qquad c = 0.3$$

$0.1 + 0.2 \neq 0.3$?

$$\Sigma = a + b =^{?} c \qquad \Delta = a + b - c$$

with

$$a = 0.1 \qquad b = 0.2 \qquad c = 0.3$$

	a	b	c	Σ	Δ
fp32	0.100000001	0.200000003	0.300000012	0.300000012	0
fp64	0.10000000000000001	0.20000000000000001	0.29999999999999999	0.30000000000000004	$5.551\cdots10^{-17}$
fp80	0.100000000000000000001	0.200000000000000000003	0.300000000000000000011	0.300000000000000000011	0

$0.1 + 0.2 \neq 0.3$?

$$\Sigma = a + b =^{?} c \qquad \Delta = a + b - c$$

with

$$a = 0.1 \qquad b = 0.2 \qquad c = 0.3$$

	a	b	c	Σ	Δ
fp32	0.100000001	0.200000003	0.300000012	0.300000012	0
fp64	0.10000000000000001	0.20000000000000001	0.29999999999999999	0.30000000000000004	$5.551 \cdots 10^{-17}$
fp80	0.100000000000000000001	0.200000000000000000003	0.300000000000000000011	0.300000000000000000011	0

$\Rightarrow \mathbb{D} \not\subset \mathbb{B}$: certains décimaux ne sont pas binaires

$\Rightarrow$ la conversion binaire nécessite un arrondi

$\frac{1}{5} = 0.2_{10} = 0.00\overline{1100}_2 \cdots \ominus 13421773 \times 2^{-26} = 0.2 + 2,98 \times 10^{-9}$

« Dieu a fait les nombres entiers, tout le reste est l'œuvre de l'Homme. »

KRONECKER

Décimaux *vs.* binaires... et binaires *VS.* flottants

$\mathbb{D} = \left\{ \frac{n}{10^p}, n \in \mathbb{Z}, p \in \mathbb{N} \right\} = \mathbb{Z}[1/10]$ (décimal)
$\mathbb{B} = \left\{ \frac{n}{2^p}, n \in \mathbb{Z}, p \in \mathbb{N} \right\} = \mathbb{Z}[1/2]$ (binaire)
$\mathbb{B} \subset \mathbb{D}$ mais $\mathbb{D} \not\subset \mathbb{B}$: $\frac{1}{5} \in \mathbb{D}$, $\frac{1}{5} \notin \mathbb{B} \Rightarrow 0.1 + 0.2 \neq 0.3$ $\left(\frac{1}{5} = 0.00\overline{1100}_2... \right) \Rightarrow$ Pas de calcul financiers...

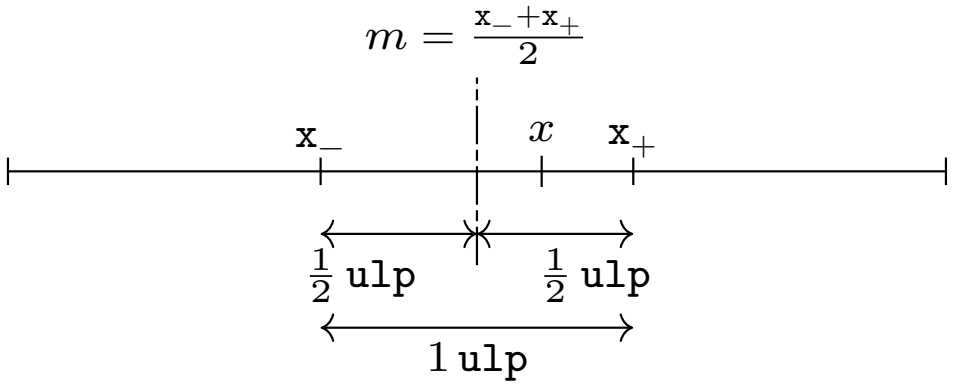
- clôture :
  $\forall (x, y) \in \mathbb{B}^2, \quad x + y \in \mathbb{B}$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y \in \mathbb{B}$
- commutativité :
  $\forall (x, y) \in \mathbb{B}^2, \quad x + y = y + x$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y = y \times x$
- associativité :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x + (y + z) = (x + y) + z$,
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y \times z) = (x \times y) \times z$
- distributivité :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y + z) = x \times y + x \times z$
- ordre total :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \leq y$ et $y \leq z \Rightarrow x \leq z$ (transitivité) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ et $y \leq x \Rightarrow x = y$ (antisymétrie) ;
  $\forall x \in \mathbb{B}, \quad x \leq x$ (réflexivité) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ ou $y \leq x$ (totalité).
- topologie :
  $\mathbb{B} \subset \mathbb{D} \subset \mathbb{Q}$ sont denses dans $\mathbb{R} \Rightarrow$ approximations arbitrairement proches des réels

Décimaux *vs.* binaires... et binaires *VS.* flottants

- clôture :
  $\exists (x, y) \in \mathbb{F}^2, \quad x + y \notin \mathbb{F}$,
  $\exists (x, y) \in \mathbb{F}^2, \quad x \times y \notin \mathbb{F}$
  $\Rightarrow$ arrondi et extension $\overline{\mathbb{F}} = \mathbb{F} \cup \{\pm\texttt{Inf}\} \cup \{\texttt{NaN}\} \cup \{0\_\}$ overflow, underflow, inexact s
- commutativité :
  $\forall (x, y) \in \mathbb{F}^2, \quad x + y = y + x$,
  $\forall (x, y) \in \mathbb{F}^2, \quad x \times y = y \times x$
- associativité :
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x + (y + z) \neq (x + y) + z$,
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x \times (y \times z) \neq (x \times y) \times z$
- distributivité :
  $\exists (x, y, z) \in \mathbb{F}^3, \quad x \times (y + z) \neq x \times y + x \times z$
- ordre total :
  $\forall (x, y, z) \in \mathbb{F}^3, \quad x \leq y$ et $y \leq z \Rightarrow x \leq z$ (transitivité) ;
  $\forall (x, y) \in \mathbb{F}^2, \quad x \leq y$ et $y \leq x \Rightarrow x = y$ (antisymétrie) ;
  $\forall x \in \mathbb{F}, \quad x \leq x$ (réflexivité) ;
  $\forall (x, y) \in \mathbb{F}^2, \quad x \leq y$ ou $y \leq x$ (totalité).
  $\exists (x, y) \in \overline{\mathbb{F}}^2, \quad x \leq y$ et $y \leq x$ ($\texttt{NaN}$).
- topologie :
  $\mathbb{B} \subset \mathbb{D} \subset \mathbb{Q}$ sont denses dans $\mathbb{R} \Rightarrow$ approximations arbitrairement proches des réels
  mais
  $\mathbb{F}$: nombres à virgule flottante, les parties finies de $\mathbb{B}$ (ou $\mathbb{D}$) sont denses nulle part

Arrondi

$\forall x \in \mathbb{R}, \exists(\mathtt{x}_-, \mathtt{x}_+) \in \mathbb{F}^2 \,|\, \mathtt{x}_- \leq x \leq \mathtt{x}_+$ (représentables immédiatement voisins)



$\Rightarrow$ l'arrondi correct requiert au moins 2 bits supplémentaires au-delà de la précision (*cf* bit de garde, bit d'arrondi et "sticky" bit)
voire plus (*dilemme du fabricant de table*)
arrondi correct, arrondi fidèle, arrondi à la louche / à la hache

l'arrondi est non-linéaire mais complètement déterministe !

Conversion

- $\mathbb{D} \not\subset \mathbb{B}$: tout décimal n'est pas un binaire
  donc la conversion en binaire repose sur un arrondi
  $\frac{1}{5} = 0.2_{10} = 0.00\overline{1100}_2 \cdots \ominus 13421773 \times 2^{-26} = 0.2 + 2,98 \times 10^{-9}$

4 byte	`float`	$25.4E0 = 25.399999619\cdots$
8 byte	`double`	$25.4D0 = 25.39999999999999858\cdots$
10 byte	`long-double`	$25.4T0 = 25.399999999999999653\cdots$
16 byte	`quadruple`	$25.4Q0 = 25.399999999999999999999999999999877\cdots$

- $\mathbb{B} \subset \mathbb{D}$: tout binaire est un décimal
  Pourtant, la conversion d'un binaire, typiquement issu d'un calcul, typiquement pour affichage ou stockage, ne se fait pas vers le décimal exact qui requérerait trop de chiffres décimaux non-significatifs.
  $\frac{1}{8} = 0.001_2 = 0.125_{10} \ominus 0.1_{10} \cdots$
  $\Rightarrow$ la conversion en décimal repose aussi sur un arrondi

Conversion décimale

Si on n'a pas le droit à l'erreur de conversion de la base 10

- ⇒ utilisez les types flottants décimaux : `_Decimal32`, `_Decimal64`, `_Decimal128` (à partir de `C23`)
- ⇒ programmez en `SQL` ou `COBOL`...
- ⇒ changez d'échelle :
  comptez en 100^{e} entiers si vous avez besoin de 2 décimales exactes
- ⇒ virgule fixe

pi $\neq \pi$?

	exact	fp32	fp64	fp16*
$\sin \pi$	0	-8.7422777e-8	1.2246467991473532e-16	9.6750e-4
$\cos \pi$	-1	-1.000000	-1.000000000000000	-1.000
$\sin \frac{\pi}{6}$	$\frac{1}{2}$	0.5000000	0.4999999999999999	0.4998
$\cos \frac{\pi}{3}$	$\frac{1}{2}$	0.5000000	0.5000000000000001	0.5005
$\sin \frac{\pi}{3}$	$\frac{\sqrt{3}}{2}$	0.8660254	0.8660254037844386	0.8657
$\cos \frac{\pi}{6}$	$\frac{\sqrt{3}}{2}$	0.8660254	0.8660254037844387	0.8662

$$\begin{array}{rrcl} & \sin 0 = 0 & \Leftrightarrow & \texttt{sin}(\texttt{0.0}) = 0 \\ & \sin \pi = 0 & \text{mais} & \texttt{sin}\,(\texttt{pi}) \neq 0 \quad \text{pas de représentation finie...} \\ \texttt{pi} = \pi - \eta, & \sin \texttt{pi} & = & \sin \pi - \eta = \sin \eta \underset{0}{\sim} \eta \end{array}$$

$$|\eta| < \pi\varepsilon/2, \Rightarrow |\sin \texttt{pi}| < \frac{\pi}{2}\varepsilon$$

pi $\neq \pi$?

Si c'est un problème

- $\Rightarrow$ utilisez les fonctions trigonométriques en demi-tours : `sinpi`, `cospi`,...(à partir de `C23`...)
- $\Rightarrow$ utilisez les fonctions trigonométriques en degrés (tous les bons compilateurs `Fortran`...)

Addition

$$\sum_{n=1}^{N} 1/n \sim \ln N + \gamma$$

Table – *Somme Harmonique*

fp	N	up sum	down sum	theoretical sum
fp16	250	6.063	6.098	6.098
fp16	500	7.039	6.793	6.793
fp16	1 000	7.086	7.477	7.484
fp16	2 000	7.086	8.188	8.180
fp16	4 000	7.086	8.789	8.875
fp16	8 000	7.086	9.797	9.563
fp16	16 000	7.086	9.797	10.26
fp16	32 000	7.086	9.797	10.95
fp32	32 000	10.95073	10.95072	10.95071
fp32	3 200 000	15.55911	15.55588	15.55588

Hiérarchie d'opérations

- **arithmétique** : $+$, $-$, $\times$, $/$, puissance entières
- **algébrique** : $\sqrt{}$, $\sqrt[n]{}$, puissances fractionaires et racines de polynômes
- **fonctions (transcendentale) élémentaire** :
  exp, ln, sin, cos, puissances irrationnelles, toute la trigonométrie circulaire et hyperbolique
- **fonctions transcendantes d'ordre supérieur *alias* fonctions spéciales** :
  Bessel, Airy, Polylogarithme, intégrale elliptique, fonction Γ d'Euler, fonction ζ de Riemann,...

L'arrondi correct est garanti par le standard pour :

- **arithmétique**
- **racine carrée**

Dilemme du fabricant de table

fonctions **transcendante**

- ... coûteuse
- ... **arrondi correct** pas garanti

L'arrondi est **non-linéaire**

$\Rightarrow$ il mélange des échelles distinctes

$\Rightarrow$ il déclenche des cascades (effet papillon)

pour un arrondi correct à n chiffres/bits... `https://members.loria.fr/PZimmermann/wc/decimal32.html`

$$\exp(0.5091077534282133) = \underbrace{1.663806007261509}_{\text{16 chiffres}}\underbrace{5000000000000000}_{\text{16 digits}}49\ldots$$

$$\exp(0.7906867968553504) = \underbrace{2.204910231771509}_{\text{16 chiffres}}\underbrace{4999999999999999}_{\text{16 digits}}16\ldots$$

Exceptionnellement base 10 (pas binaire) ! mantisse : 3 chiffres
Pour $a = 3.34$ et $b = 3.33$

- $a \ominus b = 0.01 \Rightarrow$ **compensation** (réduction de la précision relative)
  mais **bénigne** (le résultat flottant est exact : $a \ominus b = a - b$)
- $\begin{cases} a^2 - b^2 & = 0.0667 = 6.67 \times 10^{-2} \\ a \otimes a \ominus b \otimes b & = 0.1 = 1.00 \times 10^{-1} \end{cases}$
  50% d'erreur relative du résultat, ou 333 `ulp`, aucun chiffre n'est correct : **compensation calamiteuse**
- Quand advient-elle ?
- Combien de chiffres sont perdus ?

Plus, le risque d'***overflow***
$\Rightarrow$ Factorisons !

$$(a \oplus b) \otimes (a \ominus b) = 6.67 \otimes 0.01 = 6.67 \times 10^{-2} \qquad \text{exact}$$

$\Rightarrow$ The Right Way™

Compensation maudite

$\Rightarrow$ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

Compensation maudite

$\Rightarrow$ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` : $P = -6.33825300e + 29$

Compensation maudite

⇒ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` : $P = -6.33825300e + 29$
`double` : $P = -1.18059162071741 13e + 021$

Compensation maudite

⇒ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` :	$P = -6.33825300e + 29$
`double` :	$P = -1.18059162071741 13e + 021$
`long double` :	$P = +5.76460752303423489188e + 17$

Compensation maudite

$\Rightarrow$ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` :	$P = -6.33825300e + 29$
`double` :	$P = -1.18059162071741 13e + 021$
`long double` :	$P = +5.76460752303423489188e + 17$
`quad` :	$P = +1.17260394005317863185883490452018380$

Compensation maudite

⇒ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* RUMP)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. RUMP, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` :	$P = -6.33825300e + 29$
`double` :	$P = -1.18059162071741 13e + 021$
`long double` :	$P = +5.76460752303423489188e + 17$
`quad` :	$P = +1.17260394005317863185883490452018380$
`fp16` :	$P =$ `NaN`

Compensation maudite

$\Rightarrow$ les polynômes de degré élevé peuvent dégrader des résolutions élevées (*cf* Rump)

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

avec $x = 77617$, $y = 33096$ (premiers entre eux)

[S.M. Rump, 1983, *"How reliable are results of computers"*
`https://www.tuhh.de/ti3/paper/rump/Ru83b.pdf`]

`float` :	$P = -6.33825300e + 29$
`double` :	$P = -1.18059162071741 13e + 021$
`long double` :	$P = +5.76460752303423489188e + 17$
`quad` :	$P = +1.17260394005317863185883490452018380$
`fp16` :	$P =$ `NaN`
exact :	$P \approx -0.82739605994682136814116509547981629 2$
	$P = -\frac{54767}{66192}$

Comment contrôler les erreurs d'arrondi ?

Variance ... et tous les moments statistiques

- Une différence...
- ...de carrés...
- ...de sommes...
- ...et de sommes...
- ...de carrés...

- approche en deux passes
- décalage arbitraire des données vers une valeur moyenne attendue
- algorithme de Welford en une passe (une division par itération)

Calcul typique

- produit scalaire
- produit de convolution (produit scalaire “tête bêche”)
- transformée de Fourier
- produit de matrice = une matrice de produits scalaires

en pratique : somme de simples produits (quadratique dans le fond).
⇒ on s'attend à rencontrer des problèmes similaires à des différences de carrés et des calculs de variance. Mais ici on ne peut pas utiliser l'astuce de factorisation...

- précision mixte
- `fma` (fused multiply accumulate)
- `fma` utilisée pour extraire un produit exact
- combinée avec des sommes compensées de Kahan ou autres

Quadratique

$$\begin{aligned} ax^2 + bx + c &= 0 \qquad (a \neq 0) \\ \Delta &= b^2 - 4ac \\ x_\pm &= \frac{-b \pm \sqrt{\Delta}}{2a} \end{aligned}$$

2 *compensation calamiteuse* (« *catastrophic cancelation* ») possibles

- entre $-b$ and $\sqrt{\Delta}$

$$\Rightarrow q = -b - \mathrm{sgn}(b)\sqrt{\Delta} = -\mathrm{sgn}(b)\left(|b| + \sqrt{\Delta}\right)$$

$$\begin{cases} x_1 = \frac{q}{2a} \\ x_2 = \frac{2c}{q} = \frac{c}{ax_1} \end{cases}$$

- discriminant $\Delta = b^2 - 4ac$ $\Rightarrow$ `fma`

4 possible *overflow* :

- b^2 : *spurious overflow* (if $|b| > 10^{19}$, $\Delta =$ `Inf`, $|q| =$ `Inf` while $|q| \sim 2 \times 10^{19}$)
- ac
- b/a
- c/b

$$
\begin{aligned}
Q &= \frac{\sqrt{|ac|}}{b} \\
F &= \frac{1}{2}\left(1+\sqrt{1-4\sigma Q^2}\right) \\
x_1 &= -\frac{b}{a}F \\
x_2 &= \frac{c}{b}\frac{1}{F}
\end{aligned}
$$

$$
\begin{aligned}
\text{si } \sigma = +1 \qquad & F = \frac{1}{2}\left(1+\sqrt{(1-2Q)\,(1+2Q)}\right) \\
\text{si } \sigma = -1 \qquad & F = \frac{1}{2}\left(1+\sqrt{\texttt{fma}(2Q,2Q,1)}\right)
\end{aligned}
$$

- Formule de basse entropie
- Importance de l'analyse dimensionnelle (mise en œuvre de nombres sans dimensions)

Middlebrook, R.D., *"Methods of Design-Oriented Analysis : The Quadratic Equation Revisited"*,

Adimensionalisation

L'analyse dimensionelle, sépare

- les paramètres d'échelle, ou **échelles caractéristiques** du problème
- ... des paramètres de forme **sans dimension** (nombres purs)
- réduit l'entropie des formules
- souvent plusieurs manières de faire
  - symétries du problème,
  - limiter la complexité du calcul,
  - limiter les exceptions du calcul.
- si la solution n'est pas représentable, les calculs intermédiaires sont autorisés à ne pas l'être
- ramène les valeurs autour de l'unité
  c'est là où la densité des flottants est maximum !

Diffusion Compton

$$\theta = \arccos\left[1 + m_e c^2 \left(\frac{1}{E_1 + E_2} - \frac{1}{E_2}\right)\right]$$

- en fait, 2 soustractions (de positifs)

$$\theta = \arccos\left[1 - m_e c^2 \left(\frac{1}{E_2} - \frac{1}{E_1 + E_2}\right)\right]$$

- algèbre de base :

$$\theta = \arccos\left[1 - \frac{m_e c^2 E_1}{E_2 (E_1 + E_2)}\right]$$

- ...reste 1 soustraction (de positifs)
- trigonométrie de base : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\text{versin}\,\theta}{2}$

$$\theta = 2\arcsin\sqrt{\frac{m_e c^2 E_1}{2E_2 (E_1 + E_2)}}$$

- ...reste 0 soustraction (de positifs)

Diffusion Compton

$$\theta = \arccos\left[1 + m_e c^2 \left(\frac{1}{E_1 + E_2} - \frac{1}{E_2}\right)\right]$$

- en fait, 2 soustractions (de positifs)

$$\theta = \arccos\left[1 - m_e c^2 \left(\frac{1}{E_2} - \frac{1}{E_1 + E_2}\right)\right]$$

- algèbre de base :

$$\theta = \arccos\left[1 - \frac{m_e c^2 E_1}{E_2 (E_1 + E_2)}\right]$$

- ...reste 1 soustraction (de positifs)
- trigonométrie de base : $\cos 2\alpha = 1 - 2\sin^2\alpha \Leftrightarrow \sin^2\frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\mathrm{versin}\,\theta}{2}$

$$\theta = 2\arcsin\sqrt{\frac{m_e c^2 E_1}{2E_2 (E_1 + E_2)}}$$

- ...reste 0 soustraction (de positifs)

Diffusion Compton

$$\theta = \arccos\left[1 + m_e c^2 \left(\frac{1}{E_1 + E_2} - \frac{1}{E_2}\right)\right]$$

- en fait, 2 soustractions (de positifs)

$$\theta = \arccos\left[1 - m_e c^2 \left(\frac{1}{E_2} - \frac{1}{E_1 + E_2}\right)\right]$$

- algèbre de base :

$$\theta = \arccos\left[1 - \frac{m_e c^2 E_1}{E_2 \left(E_1 + E_2\right)}\right]$$

- ...reste 1 soustraction (de positifs)
- trigonométrie de base : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\text{versin}\,\theta}{2}$

$$\theta = 2\arcsin\sqrt{\frac{m_e c^2 E_1}{2E_2 \left(E_1 + E_2\right)}}$$

- ...reste 0 soustraction (de positifs)

Diffusion Compton

$$\theta = \arccos\left[1 + m_e c^2 \left(\frac{1}{E_1 + E_2} - \frac{1}{E_2}\right)\right]$$

- en fait, 2 soustractions (de positifs)

$$\theta = \arccos\left[1 - m_e c^2 \left(\frac{1}{E_2} - \frac{1}{E_1 + E_2}\right)\right]$$

- algèbre de base :

$$\theta = \arccos\left[1 - \frac{m_e c^2 E_1}{E_2 \left(E_1 + E_2\right)}\right]$$

- ...reste 1 soustraction (de positifs)
- trigonométrie de base : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\text{versin}\,\theta}{2}$

$$\theta = 2\arcsin\sqrt{\frac{m_e c^2 E_1}{2E_2 \left(E_1 + E_2\right)}}$$

- ...reste 0 soustraction (de positifs)

$$\theta = \arccos\left[1 + m_e c^2 \left(\frac{1}{E_1 + E_2} - \frac{1}{E_2}\right)\right]$$

- en fait, 2 soustractions (de positifs)

$$\theta = \arccos\left[1 - m_e c^2 \left(\frac{1}{E_2} - \frac{1}{E_1 + E_2}\right)\right]$$

- algèbre de base :

$$\theta = \arccos\left[1 - \frac{m_e c^2 E_1}{E_2 (E_1 + E_2)}\right]$$

- ...reste 1 soustraction (de positifs)
- trigonométrie de base : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\text{versin}\,\theta}{2}$

$$\theta = 2\arcsin\sqrt{\frac{m_e c^2 E_1}{2E_2 (E_1 + E_2)}}$$

- ...reste 0 soustraction (de positifs)

Intérêts composés

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

$$I = P\left(1 + \frac{r}{n}\right)^{nt} - P$$

$$I = P\left[\left(1 + \frac{r}{n}\right)^{nt} - 1\right]$$

$$I = P\left[\texttt{pow}\left(\left(1 + \frac{r}{n}\right), nt\right) - 1\right]$$

$$I = P\left[\exp\left(nt \ln\left(1 + \frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\exp\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\texttt{expm1}\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right)\right]$$

Intérêts composés

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

$$I = P\left(1 + \frac{r}{n}\right)^{nt} - P$$

$$I = P\left[\left(1 + \frac{r}{n}\right)^{nt} - 1\right]$$

$$I = P\left[\texttt{pow}\left(\left(1 + \frac{r}{n}\right), nt\right) - 1\right]$$

$$I = P\left[\exp\left(nt \ln\left(1 + \frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\exp\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\texttt{expm1}\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right)\right]$$

Intérêts composés

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

$$I = P\left(1 + \frac{r}{n}\right)^{nt} - P$$

$$I = P\left[\left(1 + \frac{r}{n}\right)^{nt} - 1\right]$$

$$I = P\left[\mathsf{pow}\left(\left(1 + \frac{r}{n}\right), nt\right) - 1\right]$$

$$I = P\left[\exp\left(nt \ln\left(1 + \frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\exp\left(nt \,\mathtt{log1p}\left(\frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\mathtt{expm1}\left(nt \,\mathtt{log1p}\left(\frac{r}{n}\right)\right)\right]$$

Intérêts composés

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

$$I = P\left(1 + \frac{r}{n}\right)^{nt} - P$$

$$I = P\left[\left(1 + \frac{r}{n}\right)^{nt} - 1\right]$$

$$I = P\left[\texttt{pow}\left(\left(1 + \frac{r}{n}\right), nt\right) - 1\right]$$

$$I = P\left[\exp\left(nt \ln\left(1 + \frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\exp\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\texttt{expm1}\left(nt \,\texttt{log1p}\left(\frac{r}{n}\right)\right)\right]$$

Intérêts composés

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

$$I = P\left(1 + \frac{r}{n}\right)^{nt} - P$$

$$I = P\left[\left(1 + \frac{r}{n}\right)^{nt} - 1\right]$$

$$I = P\left[\mathtt{pow}\left(\left(1 + \frac{r}{n}\right), nt\right) - 1\right]$$

$$I = P\left[\exp\left(nt \ln\left(1 + \frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\exp\left(nt \,\mathtt{log1p}\left(\frac{r}{n}\right)\right) - 1\right]$$

$$I = P\left[\mathtt{expm1}\left(nt \,\mathtt{log1p}\left(\frac{r}{n}\right)\right)\right]$$

Intérêts composés

Si `log1p` n'est pas disponible (*cf.* Goldberg)

$$\ln(1+x) = \begin{cases} x & \text{si } 1 \oplus x = 1 \\ \frac{x \ln(1+x)}{(1+x)-1} & \text{sinon.} \end{cases}$$

Aire du triangle

aire S en fonction des longueurs a, b et c des cotés

$$S = \sqrt{p(p-a)(p-b)(p-c)} \qquad \text{(Héron d'Alexandrie, } \textit{Stereometrica}\text{)}$$

$p = \frac{a+b+c}{2}$ demi-périmètre

Symétrique, mais instable numériquement, pour les triangles en épingle (confrontation de grandes et de petites valeurs)

Kahan Ré-étiquetage : $a > b > c$

$$\frac{1}{4}\sqrt{[a+(b+c)]\,[c-(a-b)]\,[c+(a-b)]\,[a+(b-c)]}$$

Symétrie apparente perdue, mais formule beaucoup plus robuste

Origine déterminantale

$$S = \frac{1}{4}\sqrt{\begin{vmatrix} 0 & a^2 & b^2 & 1 \\ a^2 & 0 & c^2 & 1 \\ b^2 & c^2 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{vmatrix}}$$

Volume du tétraèdre

$$V = \sqrt{\frac{1}{288}\begin{vmatrix} 0 & a^2 & b^2 & c^2 & 1 \\ a^2 & 0 & C^2 & B^2 & 1 \\ b^2 & C^2 & 0 & A^2 & 1 \\ c^2 & B^2 & A^2 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{vmatrix}}$$

$$X = (c - A + b)(A + b + c) \qquad x = (A - b + c)(b - c + A)$$
$$Y = (a - B + c)(B + c + a) \qquad y = (B - c + a)(c - a + B)$$
$$Z = (b - C + a)(C + a + b) \qquad z = (C - a + b)(a - b + C)$$

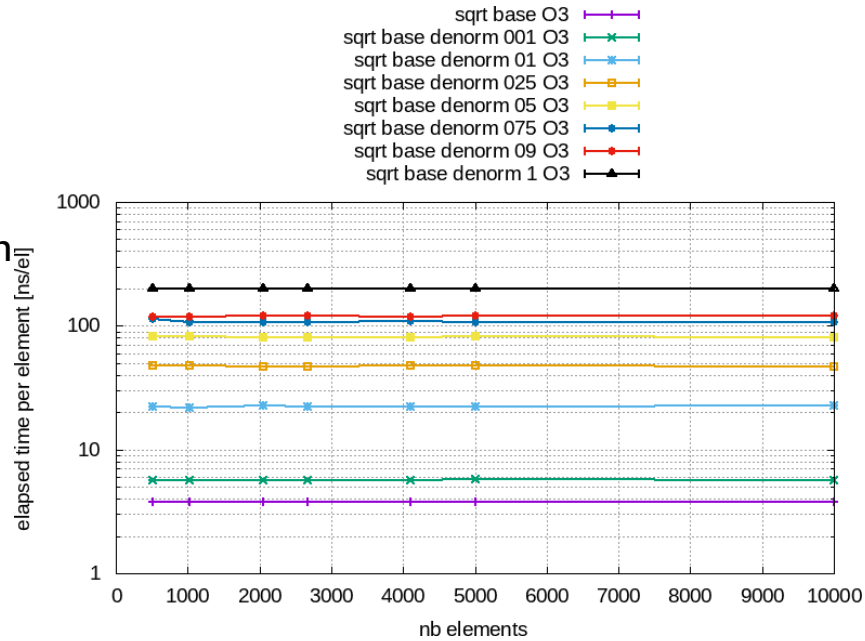
$$\xi = \sqrt{xYZ} \qquad \eta = \sqrt{yZX} \qquad \zeta = \sqrt{zXY} \qquad \lambda = \sqrt{xyz}$$

$$V = \frac{1}{192abc}\sqrt{(\xi + \eta + \zeta - \lambda)(\lambda + \xi + \eta - \zeta)(\eta + \zeta + \lambda - \xi)(\zeta + \lambda + \xi - \eta)}$$

Stable moyennant un ré-étiquetage : ordonner les paires de côtés de sorte à ce que les 3 plus petites des 12 différences faciales

Dénormaux

- en-dessous de 1.17×10^{-38} pour les `fp32`
- en-dessous de 2.22×10^{-308} pour les `fp64`
- en-dessous de 6.09×10^{-5} pour les `fp16`
  (jusqu'à 5.96×10^{-8})
- Pourquoi ?
  $\Rightarrow$ permettre un "gradual underflow"
- Pourquoi pas ?
  $\Rightarrow$ $100\times$ plus lent
  (merci Pierre !)
- Comment ?
  - différence de flottants au seuil du minimum
  - progression géométrique décroissante
    (merci Hadrien !)



Not metrology : we do not seek "precision for precision's sake"

The **functional** paradigm invites us to write computer function approaching mathematical functions, and we tend to focus on the aspect of **purity**.

But a mathematical function also seeks **totality** (being defined on the largest domain of definition) :

the function should be calculable for any argument for which it is defined.

- *removing non-jump and non-essential discontinuity* : $\Rightarrow \left.\frac{\sin x}{x}\right|_{x=0} = 1$ *(naively `sin (0.0) / 0.0 = NaN`)*
- *analytic continuation* : factorial $\Rightarrow \Gamma$, or Riemann ζ function

$\Rightarrow$ *maximal extension of function domain*
$\Rightarrow$ piecewise function definition, *casuistry*

Using IEEE-754 exceptional values, we can reach a "weak totality" :

- `log (0.0) = -Inf` (mathematically correct)
- `log (-1.0) = NaN` (mathematically correct ? more precisely `NaRN`)

Precision limitations lead to a gray zone in this kind of totality :

- `expf (88.72284) = + Inf` (but mathematically it's 2^{128} $\Rightarrow$ `domainException`)
- `expf (-103.972084) = 0.0f` (but mathematically it's just below 2^{-150} $\Rightarrow$ `domainException`)
- `gammaf (35.0401001) = + Inf` (but mathematically it's 2^{128})

OK with `double`, but not with `float`.

Not all `Inf` have the same meaning, not all `NaN` have the same meaning, *cf* `null` in `SQL`

« Why aiming for precision ? »

⇒ Implicit **contract** : the fonction will

1. (if the argument is inside the mathematical domain of the mathematical function)
2. (if the type representation of the argument is inside the domain of the function that has a representable image in the return type)
3. return a result
4. this result is relevant(?)
5. (ideally the returned value is the representation of the image of the mathematical function applied on the represented argument)

« Why aiming for precision ? »

totality (mathematical) *vs.* representable totality

A representable solution resulting from representable arguments CAN go through a non-representable intermediate calculation.
IEEE-754 exceptional values are not the value of the function, relative error of 100%, as in catastrophic cancelation.
least surprise principle

- we agree to compute erroneous results, because we know that we cannot compute exact results : exact results are rarely (= almost never) representable : π, e, $\sqrt{2}$, $1/3$, $1/5$ in base 2…
- On the other hand, we don't want things to be very wrong : mathematical result 2 but the function returns `NaN`

If the calculation is badly carried out, we can end up with

- infinite roots, where they exist and can be represented
- to an absence of roots, where they exist and are representable
- to a presence of roots, where they do not exist

a difference of degree generates a difference of nature (catastrophe theory, bifurcation, chaos)

The relative size of the danger zone in the parameter space will be much larger in low precision.

Annex for a less costly nondimensionalization :

« *You Could Learn a Lot from a Quadratic* » `doi:10.1145/609742.609746`, shows how to nondimensionalize with binary, much less costly in time and accuracy than divisions (and roots) in physicist nondimensionalization. Easy when knowing IEEE-754 API.

« precision ? » a take-away

PRECISE NUMBER + PRECISE NUMBER = SLIGHTLY LESS PRECISE NUMBER

PRECISE NUMBER × PRECISE NUMBER = SLIGHTLY LESS PRECISE NUMBER

PRECISE NUMBER + GARBAGE = GARBAGE

PRECISE NUMBER × GARBAGE = GARBAGE

$\sqrt{\text{GARBAGE}}$ = LESS BAD GARBAGE

$(\text{GARBAGE})^2$ = WORSE GARBAGE

$\frac{1}{N}\sum(\text{N PIECES OF STATISTICALLY INDEPENDENT GARBAGE})$ = BETTER GARBAGE

$(\text{PRECISE NUMBER})^{\text{GARBAGE}}$ = MUCH WORSE GARBAGE

GARBAGE − GARBAGE = MUCH WORSE GARBAGE

$\frac{\text{PRECISE NUMBER}}{\text{GARBAGE} - \text{GARBAGE}}$ = MUCH WORSE GARBAGE, POSSIBLE DIVISION BY ZERO

GARBAGE × 0 = PRECISE NUMBER

https://xkcd.com/2295/