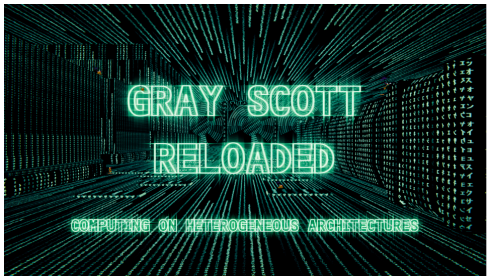
Précision en virgule flottante

Vincent Lafage

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



1er juillet 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
- L'heuristique au service de la précision :
  comment une approche à la louche sauve les epsilons
- Adimensionalisation et réduction d'entropie des formules
- Comment concilier adimensionalisation et performance
- Comment concilier abstraction et précision, les fonctions d'une variable complexe
- Pourquoi les calculs de géométrie sont compliqués
- La face cachée du fonctionnel : vers des fonctions totales

Des nombres qui coûtent et qui tuent

- 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

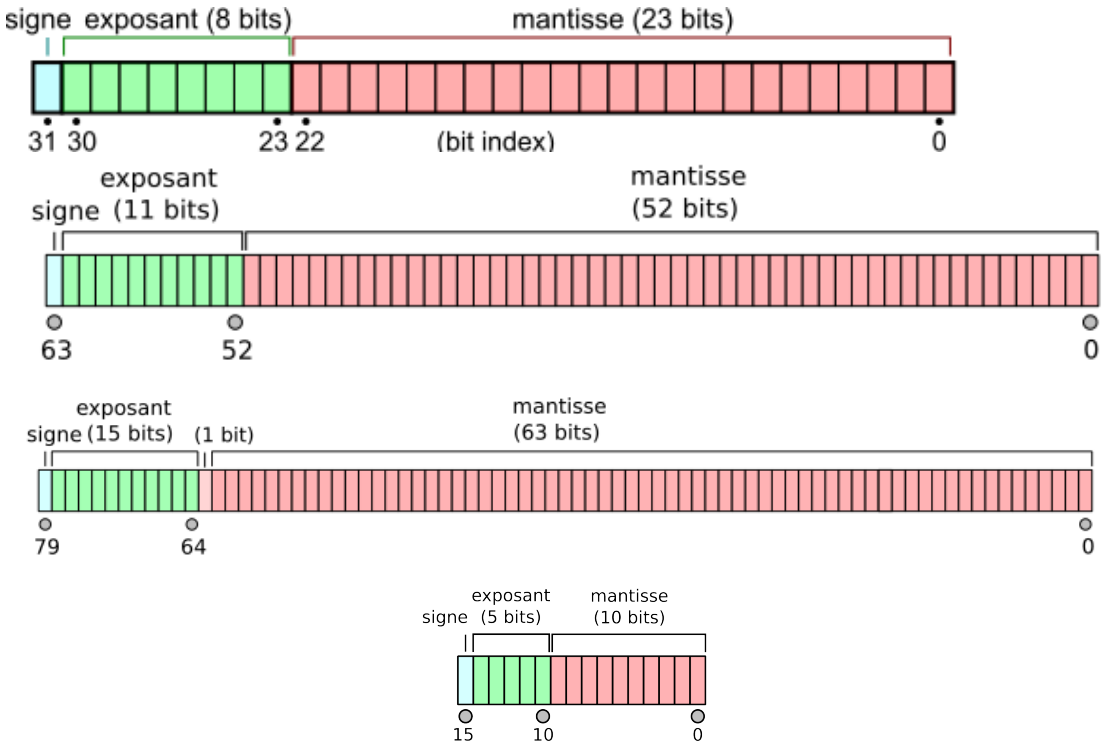
Formats

« 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	$\texttt{float} = (-1)^S \times 2^{E-127} \times (1+M)$
0	1000 0000	000 0000 0000 0000 0000 0000	$1 = 2^0 \times (1+0)$
0	1000 0000	100 0000 0000 0000 0000 0000	$2 = 2^1 \times (1+0)$
0	1000 0000	110 0000 0000 0000 0000 0000	$3 = 2^1 \times (1+1/2)$
0	1000 0000	111 0000 0000 0000 0000 0000	$3.5 = 2^1 \times (1+1/2+1/4)$
1	0111 1111	000 0000 0000 0000 0000 0000	$3.75 = 2^1 \times (1+1/2+1/4+1/8)$
0	0111 1110	000 0000 0000 0000 0000 0000	$-1 = -2^0 \times (1+0)$
0	0111 1100	100 1100 1100 1100 1100 1101	$1/2 = 2^{-1} \times (1+0)$
0	0111 1101	010 1010 1010 1010 1010 1011	$0.2 = 2^{-3} \times (1+1/2) \times \sum_n 1/16^n$
0	0111 1111	011 0101 0000 0100 1111 0011	$1/3 = 2^{-2} \times (1+1/4) \times \sum_n 1/16^n$
0	1000 0000	100 1001 0000 1111 1101 1011	$\sqrt{2}$
0	0000 0000	000 0000 0000 0000 0000 0000	$\pi \simeq 2^1 \times (1+1/2+1/16+1/128+\cdots)$
1	0000 0000	000 0000 0000 0000 0000 0000	0 special representation
0	1111 1110	111 1111 1111 1111 1111 1111	0_- special representation
0	1111 1111	000 0000 0000 0000 0000 0000	largest float $3.402823466 \times 10^{38}$
0	1111 1111	1xx xxxx xxxx xxxx xxxx xxxx	$+\infty$ = `Inf` special representation
0	1111 1111	1xx xxxx xxxx xxxx xxxx xxxx	`NaN` special representation
0	1111 1111	01x xxxx xxxx xxxx xxxx xxxx	`qNaN` *quiet* special representation
0	0000 0001	000 0000 0000 0000 0000 0000	`sNaN` *signaling* special representation
0	0000 0000	000 0000 0000 0000 0000 0001	smallest positive float $1.17549435 \times 10^{-38}$
0	0000 0000	111 1111 1111 1111 1111 1111	smallest **denormal** positive float 1.401×10^{-45}
			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 = 3.1415926535897793f;
    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

```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
```

Affichez en Hexadécimal Ada

```
with Ada.Text_Io;

procedure Hexfloat is
    use Ada.Text_Io;
    X : Float := 1.0;
begin
    Put_Line (X'Image & " = 2^" & Float'Exponent (X)'Image & " × " & Float'Fraction (X)'Image);
    X := 2.0;
    Put_Line (X'Image & " = 2^" & Float'Exponent (X)'Image & " × " & Float'Fraction (X)'Image);
    X := 3.0;
    Put_Line (X'Image & " = 2^" & Float'Exponent (X)'Image & " × " & Float'Fraction (X)'Image);
    X := 3.141592653589793;
    Put_Line (X'Image & " = 2^" & Float'Exponent (X)'Image & " × " & Float'Fraction (X)'Image);
end Hexfloat;
```

```
 1.00000E+00 = 2^ 1 ×  5.00000E-01
 2.00000E+00 = 2^ 2 ×  5.00000E-01
 3.00000E+00 = 2^ 2 ×  7.50000E-01
 3.14159E+00 = 2^ 2 ×  7.85398E-01
```

Affichez en Hexadécimal Rust

```rust
fn extract_components(x: f32) -> (char, i32, u32) {
    let bits = x.to_bits();
    let sign = if (bits >> 31) & 1 == 0 { '+' } else { '-' };
    let mantissa = (bits & ((1 << 23) - 1)) * 2;
    let exponent = (((bits >> 23) & 0xFF) as i32) - 127;
    (sign, exponent, mantissa)
}

pub fn main() {
    let x: f32 = 1.0;
    let (sign, exponent, mantissa) = extract_components(x);
    println!("{} = {}0x1.{:x}p{}", x, sign, mantissa, exponent);
    let x: f32 = 2.0;
    let (sign, exponent, mantissa) = extract_components(x);
    println!("{} = {}0x1.{:x}p{}", x, sign, mantissa, exponent);
    let x: f32 = 3.0;
    let (sign, exponent, mantissa) = extract_components(x);
    println!("{} = {}0x1.{:x}p{}", x, sign, mantissa, exponent);
    let x: f32 = 3.141592653589793;
    let (sign, exponent, mantissa) = extract_components(x);
    println!("{} = {}0x1.{:x}p{}", x, sign, mantissa, exponent);
    let x: f32 = -0.3141592653589793;
    let (sign, exponent, mantissa) = extract_components(x);
    println!("{} = {}0x1.{:x}p{}", x, sign, mantissa, exponent);
}
```

```
1 = +0x1.0p0
2 = +0x1.0p1
3 = +0x1.800000p1
3.1415927 = +0x1.921fb6p1
-3.1415927 = -0x1.921fb6p1
```

Interface *leaky abstraction*

C / C++ (/ Python)	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) precision (x)		F'Model_Epsilon
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) spacing (x) rrspacing (x)		F'Machine_Emax
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)

$0.1 + 0.2 \neq 0.3$?

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

avec

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

$0.1 + 0.2 \neq 0.3$?

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

avec

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

	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`	0.10000000000000000001	0.20000000000000000003	0.30000000000000000011	0.30000000000000000011	0
`fp16`	0.099976	0.19995	0.30005	0.29980	$2.4414 \cdots 10^{-4}$

$0.1 + 0.2 \neq 0.3$?

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

avec

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

	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
fp16	0.099976	0.19995	0.30005	0.29980	$2.4414 \cdots 10^{-4}$

$\Rightarrow \mathbb{D} \not\subset \mathbb{B}$: 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 $\mathbb{D} \not\subset \mathbb{B}$: $\frac{1}{5} \in \mathbb{D}$, $\frac{1}{5} \notin \mathbb{B} \Rightarrow 0.1 + 0.2 \neq 0.3$ $\left(\frac{1}{5} = 0.00\overline{1100}_2 \ldots \right) \Rightarrow$ Pas de calcul financiers...

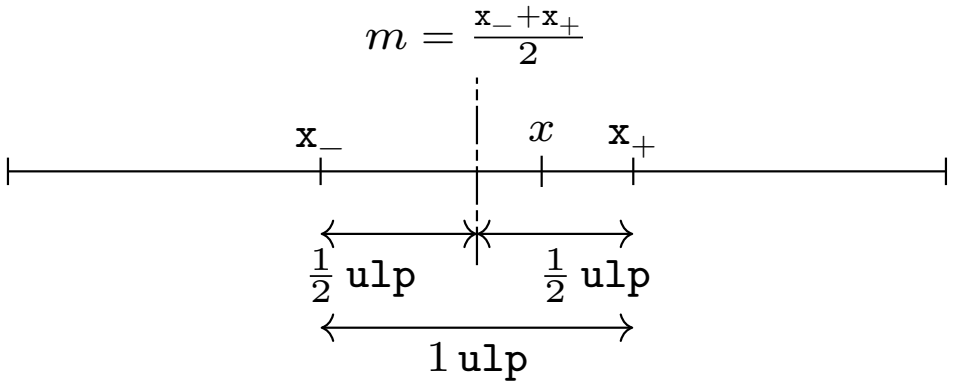
- clôture :
  $\forall (x, y) \in \mathbb{B}^2, \quad x + y \in \mathbb{B}$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y \in \mathbb{B}$
- commutativité $\forall (x, y) \in \mathbb{B}^2, \quad x + y = y + x$,
  $\forall (x, y) \in \mathbb{B}^2, \quad x \times y = y \times x$
- associativité :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x + (y + z) = (x + y) + z$,
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y \times z) = (x \times y) \times z$
- distributivité :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \times (y + z) = x \times y + x \times z$
- ordre total :
  $\forall (x, y, z) \in \mathbb{B}^3, \quad x \leq y$ and $y \leq z \Rightarrow x \leq z$ (transitivité) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ and $y \leq x \Rightarrow x = y$ (antisymétrie) ;
  $\forall x \in \mathbb{B}, \quad x \leq x$ (réflexivité) ;
  $\forall (x, y) \in \mathbb{B}^2, \quad x \leq y$ or $y \leq x$ (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 :
  $\forall (x,y) \in \mathbb{F}^2, \quad x+y \notin \mathbb{F}$,
  $\forall (x,y) \in \mathbb{F}^2, \quad x \times y \notin \mathbb{F}$
  $\Rightarrow$ arrondi et extension $\overline{\mathbb{F}} = \mathbb{F} \cup \{\pm\texttt{Inf}\} \cup \{\texttt{NaN}\} \cup \{0\_\}$ overflow, underflow, inexact
- 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 :
  $\forall (x,y,z) \in \mathbb{F}^3, \quad x+(y+z) \neq (x+y)+z$,
  $\forall (x,y,z) \in \mathbb{F}^3, \quad x \times (y \times z) \neq (x \times y) \times z$
- distributivité :
  $\forall (x,y,z) \in \mathbb{F}^3, \quad x \times (y+z) \neq x \times y + x \times z$
- ordre total :
  $\forall (x,y,z) \in \mathbb{F}^3, \quad x \leq y \wedge y \leq z \Rightarrow x \leq z$ (transitivité) ;
  $\forall (x,y) \in \mathbb{F}^2, \quad x \leq y \wedge y \leq x \Rightarrow x = y$ (antisymétrie) ;
  $\forall x \in \mathbb{F}, \quad x \leq x$ (réflexivité) ;
  $\exists (x,y) \in \overline{\mathbb{F}}^2, \quad x \leq y \wedge y \leq x$ ($\texttt{NaN}$).
- 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
  $\Rightarrow$ 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`	$25.4Q0 = 25.399999999999999999999999999999877\cdots$
2 byte	`half`	$25.4\_2 = 25.406\cdots$

- $\mathbb{B} \subset \mathbb{D}$: 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 plutôt que flottante

pi $\neq \pi$?

	exact	fp32	fp64	fp16*
$\sin \pi$	0	-8.7422777e-8	1.2246467991473532e-16	9.6750e-4
$\cos \pi$	-1	-1.000000	-1.0000000000000000	-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

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

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

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 : `sind`, `cosd`,...(tous les bons compilateurs `Fortran`...+ `F23`)

Addition : absorption

$$\sum_{n=1}^{N} 1/n \sim \ln N + \gamma + \frac{1}{2N} - \frac{1}{12N^2} + \frac{1}{120N^4} + \cdots$$

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

Addition : absorption

```
  program harmonique_fp16
  use, intrinsic :: iso_fortran_env, only: sp => REAL32, dp => REAL64
  implicit none
  integer (8), parameter :: pr = sp   , nbmax = 3200000
  integer (8) :: idx
  real (pr) :: somme_croissante = 0, somme_decroissante = 0
  real (pr), parameter :: euler = 0.57721566, &
       somme_theorique = euler + log (real (nbmax, sp))

  do idx = 1, nbmax
    somme_croissante = somme_croissante + 1.0_pr / real (idx, pr)
  end do

  do idx = nbmax, 1, -1
    somme_decroissante = somme_decroissante + 1.0_pr / real (idx, pr)
  end do

  write (*, *) nbmax, somme_croissante, somme_decroissante, somme_theorique
end program harmonique_fp16
```

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`

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

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

Double arrondi (arrondi de la haute précision à la précision intermédiaire, puis à la basse précision) peuvent donner un moins bon arrondi final qu'attendu.

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

⇒ 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

⇒ 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

⇒ 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

⇒ 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 ?

Évaluation de polynôme : Horner

Horner-Ruffini

- coût computationnel de toutes les exponentiations,
- perte de précision.

$$\begin{aligned} p(x) &= a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n \\ &= a_n (x - x_1) \times \cdots \times (x - x_n) \\ &= a_0 + x\Big(a_1 + x(\cdots + x(a_{n-1} + x\, a_n)\cdots)\Big) \end{aligned} \tag{1}$$

- gain en vitesse, (économie d'opérations)
- également en précision, en partie pour la même raison,
- garantie de stabilité du résultat et sûreté vis-à-vis des dépassements intermédiaires

$\Rightarrow$ instructions machine "multiply-accumulate" (`fma`).

$+$ techniques de sommes compensées, comme l'algorithme de somme de W. Kahan

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 *compensations calamiteuses* (« *catastrophic cancelation* ») possibles

- $-b$ & $\sqrt{\Delta}$

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

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

- 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

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

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

- 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"*,

`https://doi.org/10.1109/FIE.1992.683365`

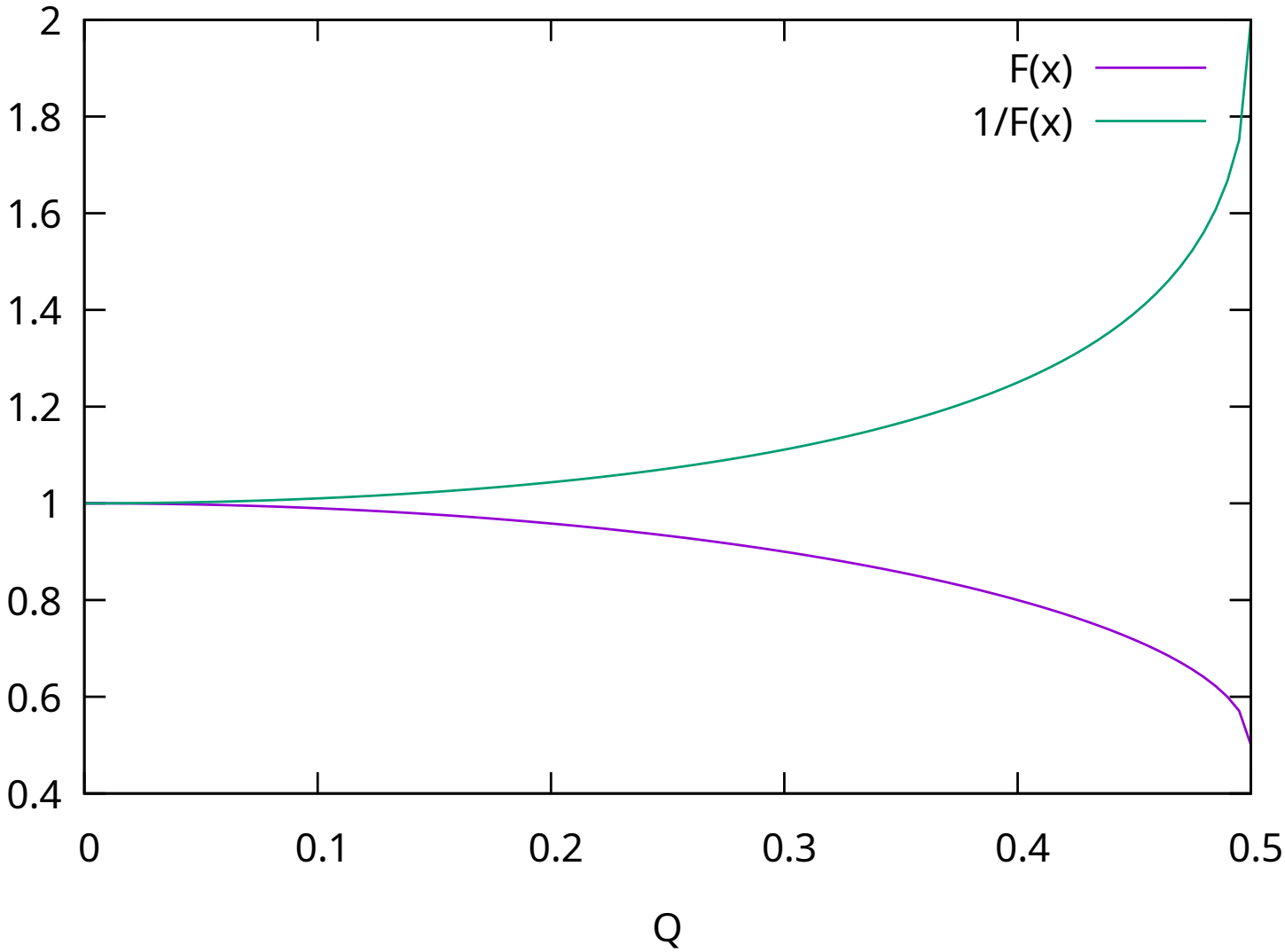


Figure – $F = \frac{1}{2}\left(1+\sqrt{1-4Q^2}\right)\ \forall Q \in [0;1/4[$.

$$F = \frac{1}{2}\left(1 + \sqrt{1 - 4Q^2}\right) \qquad \forall Q \in [0; 1/4[$$

$$\kappa = \frac{-8Q^2}{\sqrt{1 - 4Q^2}\left(1 + \sqrt{1 - 4Q^2}\right)} \qquad \forall Q \in [0; 1/4[$$

Table – *Racines du trinôme*

A	$-B/2$	C	true Δ	true roots	computed Δ	computed roots
10.27	29.61	85.37	0.0022	2.88772… 2.87859…	0.1000	2.914 2.852 `dec4`
10.28	29.62	85.34	0.0492	2.90290… 2.86075…	0	2.881 2.881 `dec4`
10.27	29.61	85.37	0.0022	2.88772… 2.87859…	0.1000	2.883 `fp16`
10.28	29.62	85.34	0.0492	2.90290… 2.86075…	0	2.881 `fp16`
94906265.625	94906267.000	94906268.375	1.89…	1.000000028975958… 1.0	0.0	1.000000014487979 1.000000014487979
94906266.375	94906267.375	94906268.375	1.0	1.000000021073424… 1.0	2.0	1.000000025437873 0.999999995635551

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 :

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

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

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

- ...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 :

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

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

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

- ...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 : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\operatorname{versin}\theta}{2} = \operatorname{haversin}\theta$

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

- ...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 : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\operatorname{versin}\theta}{2} = \operatorname{haversin}\theta$

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

- ...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 : $\cos 2\alpha = 1 - 2\sin^2 \alpha \Leftrightarrow \sin^2 \frac{\theta}{2} = \frac{1-\cos\theta}{2} = \frac{\operatorname{versin}\theta}{2} = \operatorname{haversin}\theta$

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

- ...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

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

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

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

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

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

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

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

Intérêts composés

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

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

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

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

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

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

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

Intérêts composés

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

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

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

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

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

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

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

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

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

⇒ *exercice : coder et tester avec les données de* `https://people.eecs.berkeley.edu/~wkahan/Triangle.pdf`

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

Testing precision with BASIC'85

```
for (unsigned nbTot = NBITERMIN; nbTot < NBITERMAX; nbTot++) {
  float x = X0;
  for (unsigned nbIter = 0; nbIter < nbTot; nbIter++) x = sqrt (x);
  float bottomRadix = x;
  for (unsigned nbIter = 0; nbIter < nbTot; nbIter++) x = x * x;
  printf ("%d %f %f (%+e) %f (%+e)\n", nbTot, X0, x, x-X0, bottomRadix, bottomRadix-1.0);
}
```

iter	X0	x	x − X0	btmRdx	btmRdx − 1
10	2.000000	1.999958	(−4.184246e−05)	1.000677	(+6.771088e−04)
11	2.000000	2.000196	(+1.962185e−04)	1.000339	(+3.385544e−04)
12	2.000000	2.000196	(+1.962185e−04)	1.000169	(+1.692772e−04)
13	2.000000	2.000196	(+1.962185e−04)	1.000085	(+8.463860e−05)
14	2.000000	2.000196	(+1.962185e−04)	1.000042	(+4.231930e−05)
15	2.000000	1.996286	(−3.713965e−03)	1.000021	(+2.110004e−05)
16	2.000000	1.988545	(−1.145530e−02)	1.000010	(+1.049042e−05)
17	2.000000	1.988545	(−1.145530e−02)	1.000005	(+5.245209e−06)
18	2.000000	1.988545	(−1.145530e−02)	1.000003	(+2.622604e−06)
19	2.000000	1.988545	(−1.145530e−02)	1.000001	(+1.311302e−06)
20	2.000000	1.868132	(−1.318680e−01)	1.000001	(+5.960464e−07)
21	2.000000	1.648514	(−3.514862e−01)	1.000000	(+2.384186e−07)
22	2.000000	1.648514	(−3.514862e−01)	1.000000	(+1.192093e−07)
23	2.000000	1.000000	(−1.000000e+00)	1.000000	(+0.000000e+00)

Quelle est la sensibilité relative d'une fonction à une fluctuation en entrée ?
$\Rightarrow$ « conditionnement » (*condition number*) ou valeur absolue de l'*élasticité*

$$\kappa(x) = \frac{\left|\frac{f(x_a)-f(x)}{f(x)}\right|}{\left|\frac{x_a-x}{x}\right|} = \frac{\left|\frac{f(x_a)-f(x)}{(x_a-x)}\right|}{\left|\frac{f(x)}{x}\right|} \sim \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{\mathrm{d}\ (\ln|f(x)|)}{\mathrm{d}\ln|x|}\right| \tag{2}$$

κ est sans dimension, un nombre pur (*dérivée doublement logarithmique*)
Les lois de puissance $x \to C \times x^n$ (avec C et n constantes réelles) sont les fonctions de conditionnement constant $\forall x,\ \kappa(x) = n$.
$\log_2 \kappa$: nombre de bits de précision perdus *dans le meilleur des cas, l'arrondi correct*
$f : x \to x^2 \Rightarrow \kappa = \frac{2x \cdot x}{x^2} = 2$: pas de singularité, l'erreur relative double à chaque itération
$f : x \to \sqrt{x} \Rightarrow \kappa = \frac{1}{2}$: pas de singularité, l'erreur relative est divisée par deux à chaque itération (mais elle ne peut pas tomber en dessous de $\frac{1}{2}$ `ulp`)
Très peu d'erreur induite par les itérations de $\sqrt{}$, mais la dernière demie `ulp` tue 100% de la précision

Puis les itérations de $x \to x^2$ amplifient cette erreur généralement négligeable en une erreur macroscopique.

$$\kappa_{f\circ g} = \kappa_f \times \kappa_g$$

$$\kappa_{f\times g} = \kappa_f + \kappa_g$$

$$\kappa_{f^n} = n\kappa_f$$

- $f : x \to x - c \Rightarrow \kappa = \frac{x}{x-c}$: singularité $x = c$ *(compensation calamiteuse)*
- $f : x \to \ln x \Rightarrow \kappa(x) = \frac{1}{\ln x}$: singularité $x = 1$, $f(x = 1 + h) = \ln(1 + h)$

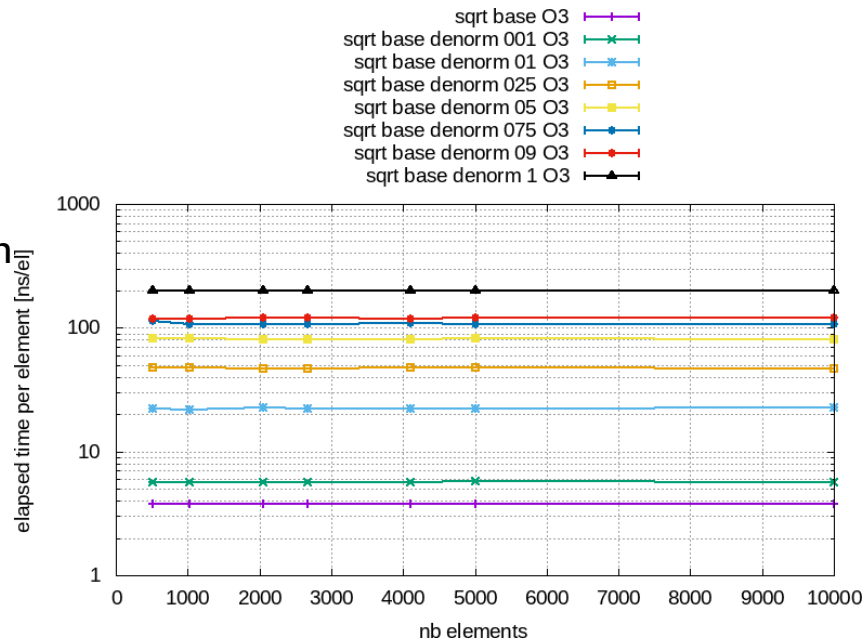
$$\kappa(h) = \frac{h}{(1+h)\ln(1+h)} \underset{h\to 0}{\sim} \frac{1}{(1+h)}$$ d'où l'importance de `log1p`

- $f : x \to \exp x - 1 \Rightarrow \kappa(x) = \frac{x \exp x}{\exp x - 1}$: forme indeterminée $x = 0$, $\kappa(h) \underset{h\to 0}{\sim} 1$
  d'où l'importance de `expm1`
- $f : x \to \cos x - 1 \Rightarrow \kappa(x) = \frac{-x \sin x}{\cos x - 1}$: forme indeterminée $x = 0$, $\kappa(h) = \frac{h \cos \frac{h}{2}}{\sin \frac{h}{2}} \underset{h\to 0}{\sim} 2$
  d'où l'importance de la trigonométrie

Si on veut résoudre le problème de la tour de racines proprement, même en simple précision, il faut changer l'approche naïve du calcul, et utiliser `log1p` et `expm1` ⇒ *exercise : do it !*

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 !)



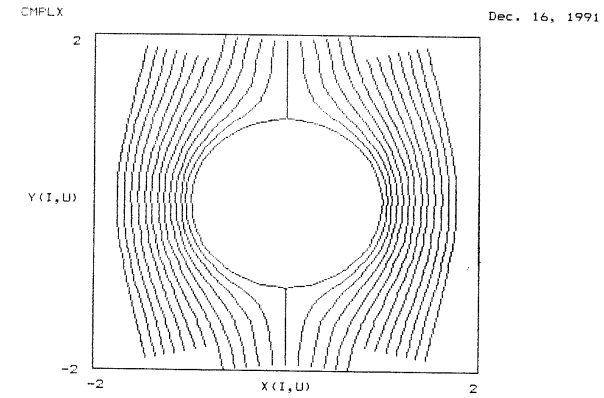


Figure 1 : Eluding Flow Past the Unit Disk

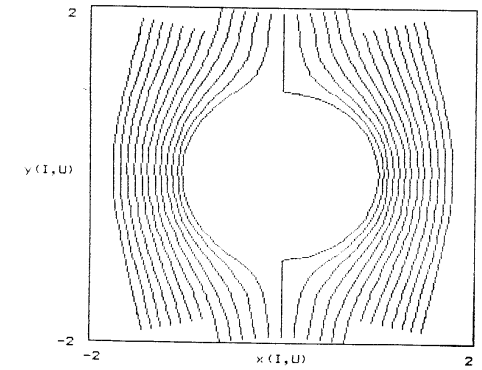


Figure 2 : Eluding Flow Past the Unit Disk, Almost

7

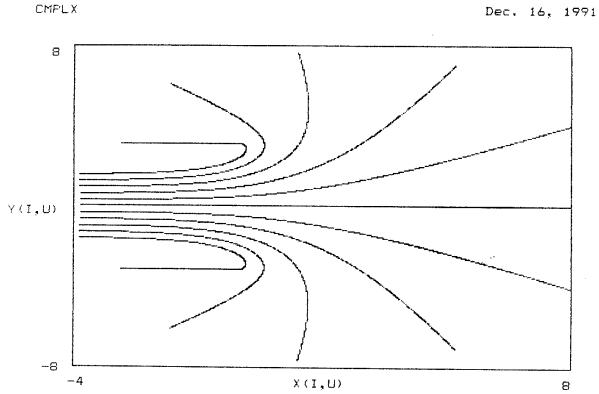


Figure 3 : Borda's Mouthpiece

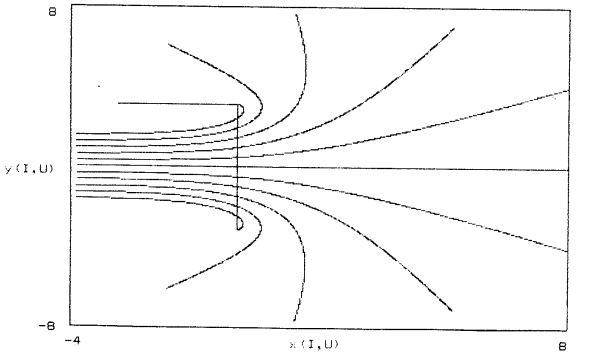


Figure 4 : Borda's Mouthpiece, Almost

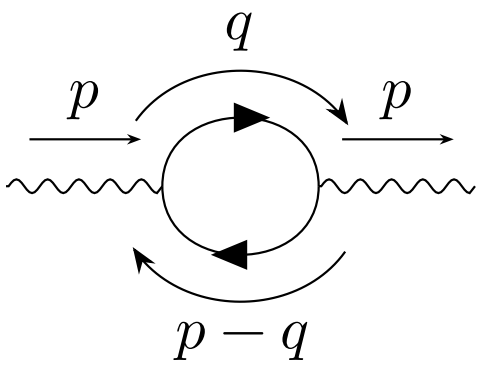
8

Eluding Flow past a Disk : $f : Z \mapsto (Z - 1/Z)/2$ and $g : W \mapsto W - i\sqrt{iW-1}\sqrt{iW+1}$

Do not "simplify" $g(W)$ to $W - i\sqrt{-W^2-1}$ nor to $W - \sqrt{W^2+1}$ since they behave differently. Though $\forall W, f(g(W)) = W$, $\forall |Z| > 1, g(f(Z)) = Z$ only, and some $|Z| = 1$; otherwise $g(f(Z)) = -1/Z$.
Deducing where these identities hold is tricky.

Borda's Mouthpiece : $W \mapsto 1 + W^2 + W\sqrt{W^2+1} + \ln(W^2 + W\sqrt{W^2+1})$

as W runs on radial straight lines through 0 in the right half-plane, including the imaginary axis.



$$
\begin{aligned}
B_0(p,m_1,m_2) &= 16\pi^2 Q^{4-n} \int \frac{d^n q}{i\,(2\pi)^n} \frac{1}{\left[q^2-m_1^2+i\varepsilon\right]\left[(q-p)^2-m_2^2+i\varepsilon\right]} \\
&= \frac{1}{\hat{\epsilon}} - \int_0^1 dx\ \ln \frac{(1-x)\ m_1^2 + x\ m_2^2 - x(1-x)\ p^2 - i\varepsilon}{Q^2} \\
&= \frac{1}{\hat{\epsilon}} - \ln\left(\frac{p^2}{Q^2}\right) - f_B(x_+) - f_B(x_-)
\end{aligned}
$$

$$
s = p^2 - m_2^2 + m_1^2,\ x_\pm = \frac{s \pm \sqrt{s^2 - 4p^2(m_1^2 - i\varepsilon)}}{2p^2}\ , \quad f_B(x) = \ln(1-x) - x \ln\left(1 - x^{-1}\right) - 1
$$

$\Rightarrow$ the (microscopic) difference of ε induces a (macroscopic) difference of 2π on the imaginary part

$\Rightarrow$ the analytic functions [1] of complex analysis are sharply discontinuous at the crossing of their *branch cut*

Discrete Stochastic Arithmetic (DSA) [Vignes'04]

Classic arithmetic

$A \oplus B \longrightarrow R$

R =3.14237654356891

DSA

Random rounding

$A_1 \oplus B_1 \longrightarrow R_1$

$A_2 \oplus B_2 \longrightarrow R_2$

$A_3 \oplus B_3 \longrightarrow R_3$

R_1 =**3.14**1354786390989
R_2 =**3.14**3689456834534
R_3 =**3.14**2579087356598

- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the confidence level 95%
- operations executed synchronously
  - ⇒ detection of numerical instabilities
    Ex: `if (A>B)` with `A-B` numerical noise
  - ⇒ optimization of stopping criteria

The CADNA library

- met en œuvre l'arithmétique stochastique pour des codes C/C++ ou Fortran
- peu de réécriture
- tous les opérateurs et les fonctions mathématiques sont surchargées
- support pour des codes MPI, OpenMP, GPU, vectorisés
- support pour la demie précision émulée ou native
- en une exécution CADNA : la précision de tous les résultats, liste complète des instabilités numériques

coût de CADNA

- memory : 4
- run time ≈ 10

Exécuter CADNA

Before modifying the precisions used, we want to explore the current accuracy.

Exécuter CADNA

Before modifying the precisions used, we want to explore the current accuracy.

To execute CADNA, we essentially change the types.

Exécuter CADNA

Before modifying the precisions used, we want to explore the current accuracy.

To execute CADNA, we essentially change the types.

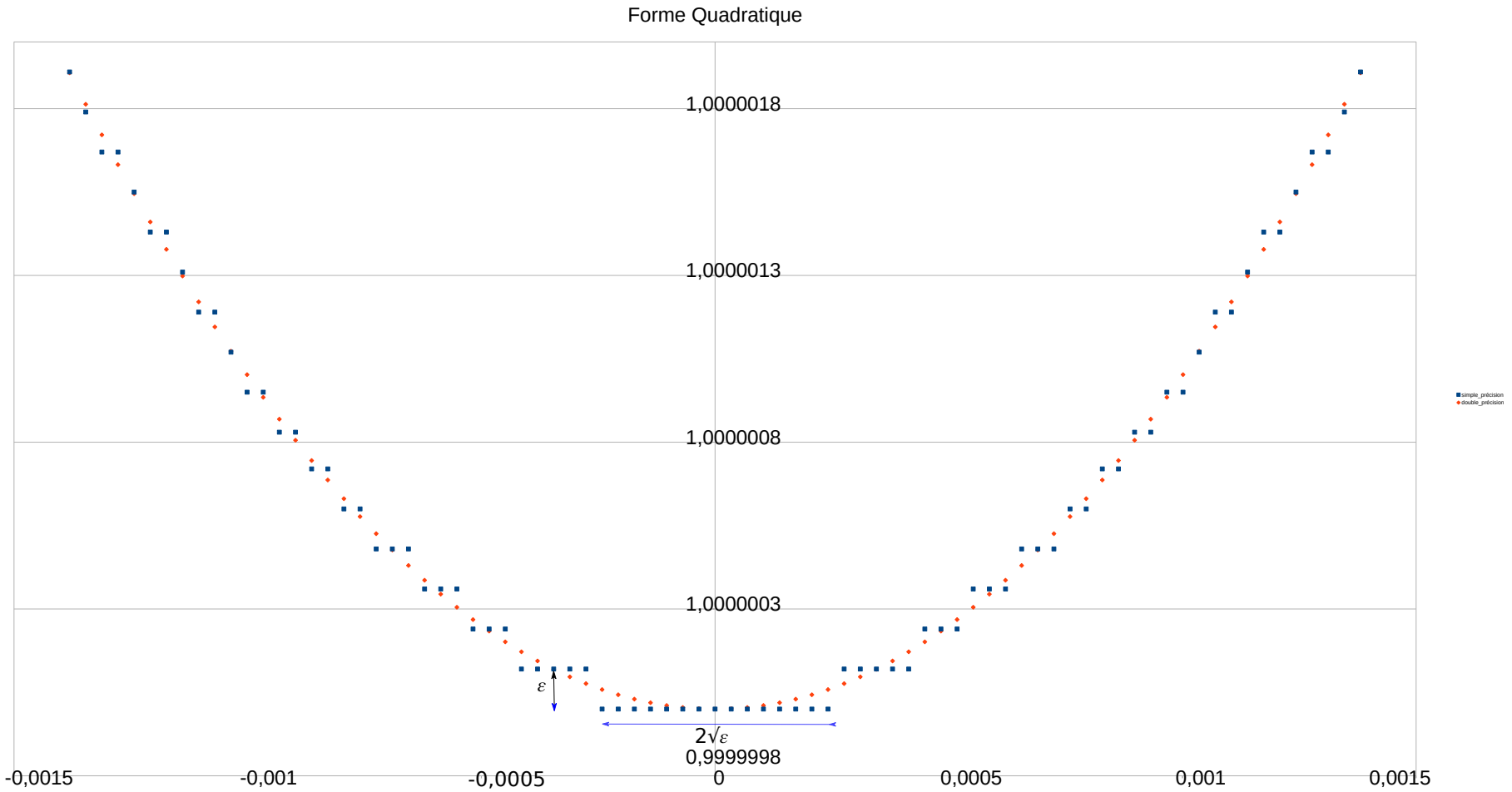
This execution exposed multiple numerical instabilities that hide potential massive loss of accuracy.

```
-----------------------------------------------------------------
CADNA_C 3.1.11 software

CRITICAL WARNING: the self-validation detects major problem(s).
The results are NOT guaranteed.

There are 538393974 numerical instabilities
10409 UNSTABLE DIVISION(S)
40122229 UNSTABLE MULTIPLICATION(S)
267297 UNSTABLE BRANCHING(S)
448561143 UNSTABLE INTRINSIC FUNCTION(S)
266 UNSTABLE MATHEMATICAL FUNCTION(S)
49432630 LOSS(ES) OF ACCURACY DUE TO CANCELLATION(S)
-----------------------------------------------------------------
```

Minimisation

Numerical evaluation of derivatives / gradients / Jacobian / Hessian

$$x \mapsto 1 + (x-1)^2 \quad \Rightarrow \quad f(x = x_0 + h) = f(x_0) + \underbrace{h \cdot \frac{\partial f}{\partial}}_{=0 \text{ at extremum}} + {}^t h \cdot \frac{\partial^2 f}{\partial \partial} \cdot h + o(h^2) \cdots \text{Taylor}$$



Neural Network

Exploration of Machine learning for Polynomial Root Finding

Vitaliy Gyrya, Mikhail Shashkov, Alexei Skurikhin
(T-5) Applied Mathematics & Plasma Physics, (XCP-4) Methods & Algorithms, (ISR-3) Space Data Science & Systems

Machine Learning for Computational Fluid and Solid Dynamics
February 19-21, 2019

Los Alamos NATIONAL LABORATORY EST. 1943

NNSA National Nuclear Security Administration

Motivation

We are interested in application of Machine Learning (ML) for improving numerical methods for solving partial differential equations (PDEs). One example of such an improvement is the optimization of the parameters of artificial viscosity for Lagrangian and arbitrary-Lagrangian-Eulerian methods. Another example is solving the Riemann problem, which is at the core of many numerical methods for computational gas and solid dynamics. To build confidence in ML methods and understand their strengths and weaknesses we decided to start by applying ML to solve simple quadratic equations of one variable.

Problem

Consider a quadratic equation, $ax^2 + bx + c = 0$, whose roots are r_L and r_R. We would like to learn the function

$$(a, b, c) \to (r_L, r_R)$$

without relying on our knowledge of the underlying processes. Instead we will consider a number of observations observations (training set)

$$(a^i, b^i, c^i) \to (r_L^i, r_R^i), \qquad i = 1, \ldots, N.$$

From which we will try to predict

$$(a^j, b^j, c^j) \to (\tilde{r}_L^j, \tilde{r}_R^j) \approx (r_L^j, r_R^j), \qquad j = N+1, \ldots, N+K.$$

The goal is to minimize

$$\text{COST} = \sum_j (r_L^j - \tilde{r}_L^j)^2 + \sum_j (r_R^j - \tilde{r}_R^j)^2.$$

Challenges

The quadratic equation was selected as a proxy for the following reasons that are relevant to many complex practical problems:

- There are several branches in the solution: if $a = 0$, the quadratic equation becomes a linear equation, which has one root – this is a qualitative change from one regime to a different one; depending on the discriminant the number of roots as well as the nature of the roots changes (real vs. complex).
- Finding solution involves different arithmetic operations some of which can be difficult to model by machine learning techniques. For example, division and square root are a challenge for neural networks to represent as activation functions.
- Probably, the most significant challenge is that for a small range of input parameters for which output values are increasingly large.

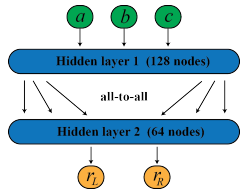
Feed-forward Neural Network

NN Architecture:

Input Layer:	3 nodes
Hidden Layer 1:	128 ReLU
Hidden Layer 2:	64 ReLU
Output Layer:	2 Linear
Connectivitys:	full.

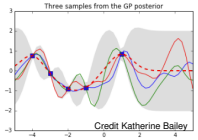
NN Training:

Batch size:	200
Training epochs:	under 500
Oprimizer:	Adam (https://arxiv.org/abs/1412.6980v8)



Gauss Process Regression (GPR)

- Probabilistic Bayesian generalization of linear regression approach.
- Built in model of uncertainty estimator.
- Need to specify a covariance kernel.
  Our choice of kernel:
  ConstantKernel()+
  Matern(*length_scale* = 2, *nu* = 3/2)+
  WhiteKernel(*noise_level* = 1)



Test & training sets

We considered a number of distributions for the coefficients (a, b, c). In all these cased we assumed that

$$a \in [\epsilon, 1], \quad b \in [-1, 1], \quad c \in [-1, 1], \quad \epsilon = 1/20$$

and the roots (r_L, r_R) are real, i.e. $D = b^2 - 4ac \geq 0$.
We considered the following distributions for (a, b, c)

- Uniform random distribution.
- Regular distribution for (a, b, c), i.e. distribution on a grid.
- Regular distribution for $(1/a, b, c)$, i.e. distribution on a grid.

The sizes of the training and test sets were approximately equal and were on the order of 40K to 50K data points.

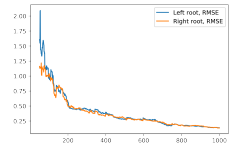
GPR for large datasets

- GPR performance degrades quickly (scaling $\sim N^3$).
- Depending on the machine the threshold of tractable training sets was between 5K and 50K sample points.
- More advanced techniques are needed for larger data sets.
- Ensambles of smaller GPR could be used

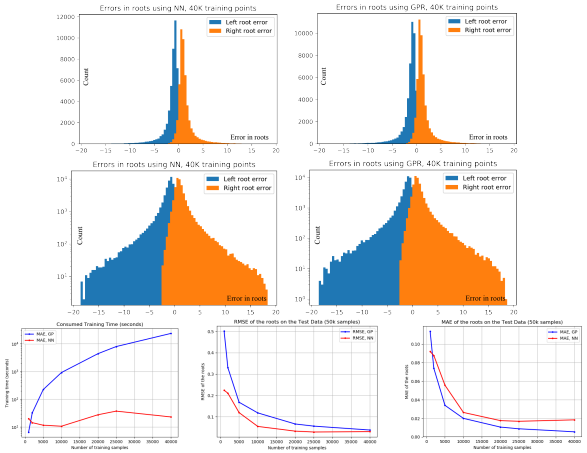
Adaptive sampling with GPR

Adaptation procedure:

- Consider the pool of uniformly distributed parameters $(a^i.b^i.c^i)$.
- Select an initial training set of points (50) at random. Generated GPR based on these points.
- For the given GPR consider the "uncertainty" σ at all of the sample points. Find the triples (a^i, b^i, c^i) with the largest uncertainty and add them to the training set.
- Generate a new GPR for the updated training set.
- Repeat steps 3-4 untill stopping criteria is statisfied, e.g. training set reached predefined size.



Results



Conclusions

- For small data sets (2K points) GPR is more accurate
- GPR can utilize adaptive sampling
- GPR does not scale well to larger data sets (∼2K points).
- NN scales well for large data sets and has better accuracy over GPR (more that 5K points) .
- NN produces fewer large errors vs. GPR for large data sets (RMSE

Floating Point Types

`float` and `double` are identified as simple or even primitive types, but they are much richer than it seems.
Object point of view : do these types fit into a hierarchy of classes ?
⇒ Violation of the LISKOV's substitution principle (LSP)

> if S subtypes T, what holds for T-objects holds for S-objects.

If S is a subtype of T, objects of type T in a program can be replaced by objects of type S without changing any of the desirable properties of that program (e.g. correct results)

A poorly encapsulated abstraction (*leaky*) : we can measure the smallest positive non-zero float, the largest one, the machine epsilon, the base : we can access the implementation details

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

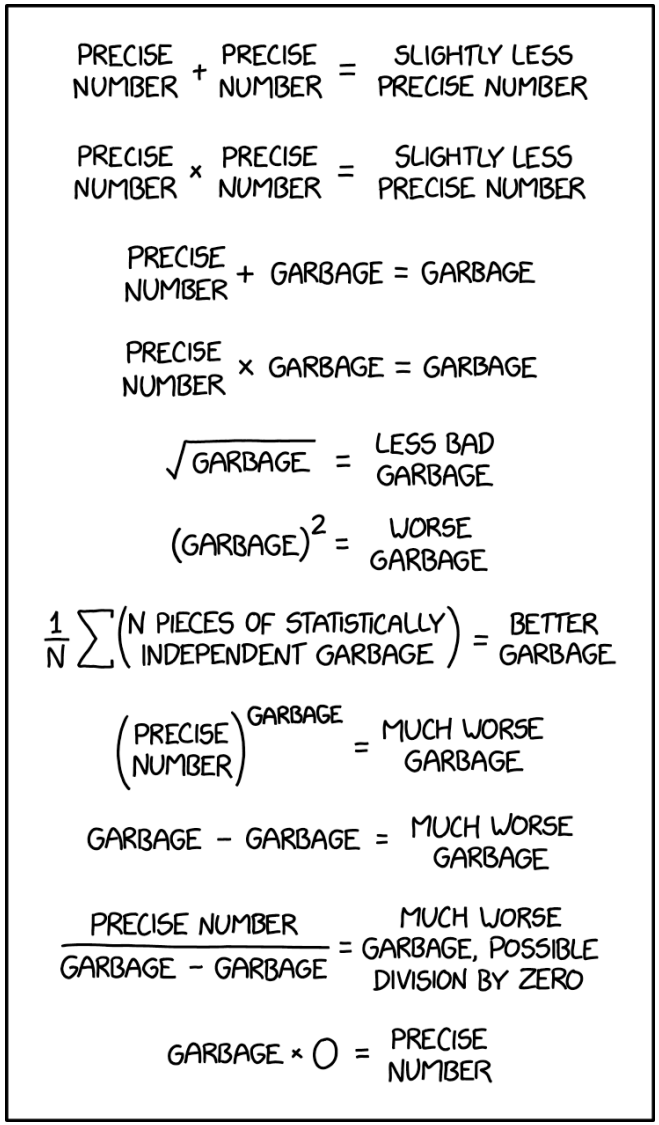
a difference of degree generates a difference of nature (catastrophe theory, bifurcation, chaos)

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



https://xkcd.com/2295/