Proof of Properties in Floating-Point Arithmetic

LMS Colloquium — Verification and Numerical Algorithms

Jean-Michel Muller

CNRS - Laboratoire LIP
(CNRS-INRIA-ENS Lyon-Université de Lyon)

http://perso.ens-lyon.fr/jean-michel.muller/

Floating-Point Arithmetic

- too often, viewed as a set of cooking recipes ;
- many "theorems" that hold. . . frequently ;
- simple–yet correct !–models such as the standard model

  in the absence of overflow and underflow,

  $$\circ(a \top b) = (a \top b) \cdot (1 + \delta), \quad |\delta| \leq 2^{-p},$$

  (in radix 2, rounded to nearest, arithmetic) are very useful, but do not allow to catch subtle behaviors such as those in

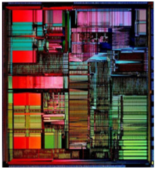
  ```
  s = a + b ; z = s - a ; r = b - z
  ```

  and many others.
- by the way, are these "subtle behaviors" robust ?

Desirable properties

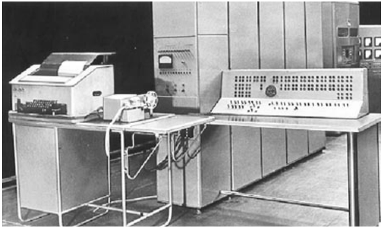
- Speed : tomorrow's weather must be computed in less than 24 hours ;
- Accuracy, Range ;
- "Size" : silicon area and/or code size ;
- Power consumption ;
- Portability : the programs we write on a given system must run on different systems without requiring huge modifications ;
- Easiness of implementation and use : If a given arithmetic is too arcane, nobody will use it.

Some can do a very poor job...



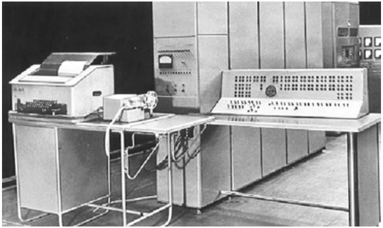
- 1994 : Pentium 1 division bug : 8391667/12582905 gave $0.666869\cdots$ instead of $0.666910\cdots$;
- Maple version 6.0. Enter 214748364810, you get 10. Notice that $214748364810 = 100 \cdot 2^{31} + 10$;
- Excel'2007 (first releases), compute $65535 - 2^{-37}$, you get 100000 ;
- November 1998, USS Yorktown warship, somebody erroneously entered a «zero» on a keyboard → division by 0 → series of errors → the propulsion system stopped.

Some strange things



- Setun Computer, Moscow University, 1958. 50 copies ;
- radix 3 and digits -1, 0 and 1 ;
- idea : radix β, n digits, "Cost" : $\beta \times n$;
- if we wish to be able to represent M numbers, minimize $\beta \times n$ knowing that $\beta^n \geq M$.

Some strange things



- Setun Computer, Moscow University, 1958. 50 copies ;
- radix 3 and digits -1, 0 and 1 ;
- idea : radix β, n digits, "Cost" : $\beta \times n$;
- if we wish to be able to represent M numbers, minimize $\beta \times n$ knowing that $\beta^n \geq M$.
- as soon as :

$$M \geq e^{\frac{5}{(2/\ln(2))-(3/\ln(3))}} \approx 1.09 \times 10^{14}$$

the best β is always 3

Floating-Point System

Parameters :

$$\begin{cases} \text{radix (or base)} & \beta \geq 2 \text{ (almost always 2 in this presentation)} \\ \text{precision} & p \geq 1 \\ \text{extremal exponents} & e_{\min}, e_{\max}, \end{cases}$$

A finite FP number x is represented by 2 integers :

- integral significand : M, $|M| \leq \beta^p - 1$;
- exponent e, $e_{\min} \leq e \leq e_{\max}$.

such that

$$x = M \times \beta^{e+1-p}$$

with $|M|$ largest under these constraints ($\rightarrow |M| \geq \beta^{p-1}$, unless $e = e_{\min}$).
(Real) significand of x : the number $m = M \times \beta^{1-p}$, so that $x = m \times \beta^e$.

Normal and subnormal numbers

- normal number : $|x| \geq \beta^{e_{\min}}$. The absolute value of its integral significand is $\geq \beta^{p-1}$;
- subnormal number : $|x| < \beta^{e_{\min}}$. The absolute value of its integral significand is $< \beta^{p-1}$.

Subnormal numbers (believed to be) difficult to implement efficiently, but their availability allows for nice properties, e.g.,

> *the relative error of a rounded-to-nearest FP addition is always bounded by* $(1/2) \cdot \beta^{-p+1}$

IEEE-754 Standard for FP Arithmetic (1985 and 2008)

- leader : Kahan (father of the arithmetics of the HP35 and Intel 8087) ;
  - formats ;
  - specification of operations and conversions ;
  - exception handling (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.) ;
- put an end to a mess (no portability, variable quality) ;

$$a \times 1 \rightarrow \text{overflow on some machines}$$

- new version of the standard : August 2008.

Correct rounding

Definition 1 (Correct rounding)

The user chooses a *rounding function* among :

- toward $-\infty$: $\mathrm{RD}(x)$ is the largest FP number $\leq x$;
- toward $+\infty$: $\mathrm{RU}(x)$ is the smallest FP number $\geq x$;
- toward 0 : $\mathrm{RZ}(x)$ is equal to $\mathrm{RD}(x)$ if $x \geq 0$, and to $\mathrm{RU}(x)$ if $x \leq 0$;
- to nearest : $\mathrm{RN}(x) =$ FPN closest to x. If halfway between two consecutive FPN : the one whose integral significand is even (default).

For a function $f : \mathbb{R}^n \mapsto \mathbb{R}$, correctly rounded implementation with rounding function $\circ$: we get $\circ[f(x_1, \ldots, x_n)]$ for all input FP numbers $x_1, x_2, \ldots, x_n$.

Correct rounding

IEEE-754 (1985) : Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{}$ and some conversions. Advantages :

- if the result of an operation is exactly representable, we get it ;
- if we just use the 4 arith. operations and $\sqrt{}$[1], deterministic arithmetic : one can elaborate algorithms and proofs that use the specifications ;
- accuracy and portability are improved ;
- playing with rounding towards $+\infty$ and $-\infty$ $\rightarrow$ certain lower and/or upper bounds : interval arithmetic.

1. and if the compiler is kind enough...

First example : Sterbenz Lemma

Lemma 2 (Sterbenz)

Radix β,with subnormal numbers available. Let a and b be positive FPNs. If

$$\frac{a}{2} \leq b \leq 2a$$

then $a - b$ is a FPN ($\rightarrow$ computed exactly, with any rounding function).

Proof : straightforward using the notation $x = M \times \beta^{e+1-p}$.

Error of FP addition (Møller, Knuth, Dekker, ...)

First result : representability. RN (x) is x rounded to nearest.

Lemma 3

Let a and b be two FP numbers. Let

$$s = RN(a + b)$$

and

$$r = (a + b) - s.$$

If no overflow when computing s, then r is a FP number.

Error of FP addition (Møller, Knuth, Dekker)

Proof : Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \longrightarrow$ it is closest to $a+b$ than a is. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore
$$|r| \leq |b|.$$

2. denote $a = M_a \times \beta^{e_a-p+1}$ and $b = M_b \times \beta^{e_b-p+1}$, with $|M_a|, |M_b| \leq \beta^p - 1$, and $e_a \geq e_b$.
$a+b$ is multiple of $\beta^{e_b-p+1} \Rightarrow s$ and r are multiple of β^{e_b-p+1} too $\Rightarrow \exists R \in \mathbb{Z}$ s.t.
$$r = R \times \beta^{e_b-p+1}$$
but, $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq \beta^p - 1 \Rightarrow r$ is a FP number.

Get r : the fast2sum algorithm (Dekker)

Theorem 4 (Fast2Sum (Dekker))

$\beta \leq 3$, subnormal numbers available. Let a and b be FP numbers, s.t. $|a| \geq |b|$. Following algorithm : s and r such that

- *$s + r = a + b$ exactly ;*
- *s is "the" FP number that is closest to $a + b$.*

Algorithm 1 (FastTwoSum)

$$s \leftarrow RN(a + b)$$
$$z \leftarrow RN(s - a)$$
$$r \leftarrow RN(b - z)$$

C Program 1

```
s = a+b;
z = s-a;
r = b-z;
```

Important remark : Proving the behavior of such algorithms requires use of the correct rounding property. . . beware of "optimizing" compilers.

Proof in the case $\beta = 2$

$$\begin{aligned} s &= \mathrm{RN}\,(a+b) \\ z &= \mathrm{RN}\,(s-a) \\ t &= \mathrm{RN}\,(b-z) \end{aligned}$$

- if a and b have same sign, then $|a| \leq |a+b| \leq |2a|$ hence (radix $2 \rightarrow 2a$ is a FP number, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz Lemma) $z = s - a$. Since $r = (a+b) - s$ is a FPN and $b - z = r$, we find $\mathrm{RN}\,(b-z) = r$.
- if a and b have opposite signs then
  1. either $|b| \geq \frac{1}{2}|a|$, which implies (Sterbenz Lemma) $a+b$ is a FPN, thus $s = a+b$, $z = b$ and $t = 0$;
  2. or $|b| < \frac{1}{2}|a|$, which implies $|a+b| > \frac{1}{2}|a|$, hence $s \geq \frac{1}{2}|a|$ (radix $2 \rightarrow \frac{1}{2}a$ is a FPN, and rounding is increasing), thus (Sterbenz Lemma) $z = \mathrm{RN}\,(s-a) = s - a = b - r$. Since $r = (a+b) - s$ is a FPN and $b - z = r$,we get $\mathrm{RN}\,(b-z) = r$.

The TwoSum Algorithm (Møller-Knuth)

- no need to compare a and b;
- 6 operations instead of 3 yet, on many architectures, very cheap in front of wrong branch prediction penalty when comparing a and b.

Algorithm 2 (TwoSum)

$$
\begin{aligned}
s &\leftarrow RN(a+b)\\
a' &\leftarrow RN(s-b)\\
b' &\leftarrow RN(s-a')\\
\delta_a &\leftarrow RN(a-a')\\
\delta_b &\leftarrow RN(b-b')\\
r &\leftarrow RN(\delta_a+\delta_b)
\end{aligned}
$$

Knuth : if no underflow nor overflow occurs then $a+b=s+r$, and s is nearest $a+b$.

Boldo et al : formal proof + underflow does not hinder the result (overflow does).

TwoSum is optimal, in a way we are going to explain.

TwoSum is optimal

Assume an algorithm satisfies :

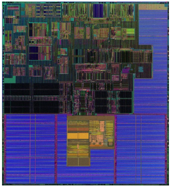
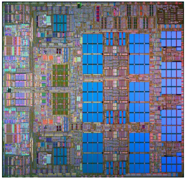
- it is without tests or min/max instructions ;
- it only uses rounded to nearest additions/subtractions : at step i we compute $\mathrm{RN}(u+v)$ or $\mathrm{RN}(u-v)$ where u and v are input variables or previously computed variables.

If that algorithm algorithm always computes the same results as 2Sum, then it uses at least 6 *additions/subtractions (i.e., as much as 2Sum).*

- proof : most inelegant proof award ;
  - 480756 algorithms with 5 operations (after suppressing the most obvious symmetries) ;
  - each of them tried with 2 well-chosen pairs of input values.

What about products ?

- *fused multiply-add* (fma), computes RN $(ab + c)$. RS6000, Itanium, PowerPC. AMD Bulldozer, Intel Haswell. Specified in IEEE 754-2008.
- if a and b are FP numbers, then $r = ab -$ RN (ab) is a FP number ;
- obtained with algorithm TwoMultFMA $\left\{ \begin{array}{lcl} p & = & \text{RN}\,(ab) \\ r & = & \text{RN}\,(ab - p) \end{array} \right. \rightarrow 2$ operations only. $p + r = ab$.
- without fma, Dekker's algorithm : 17 operations ($7\ \times$, $10\ \pm$).

Itanium 2   PowerPC 5

Relative error – unit roundoff

If $z = \circ(a \top b)$, where $\circ \in \{ \text{RU}, \text{RD}, \text{RZ}, \text{RN} \}$, and if no overflow occurs, then

$$z = (a \top b)(1 + \epsilon) + \epsilon',$$

with

- $|\epsilon| \leq \frac{1}{2}\beta^{1-p}$ and $|\epsilon'| \leq \frac{1}{2}\beta^{e_{\min}-p+1}$ if $\circ = \text{RN}$, and
- $|\epsilon| < \beta^{1-p}$ and $|\epsilon'| < \beta^{e_{\min}-p+1}$ otherwise.

Moreover, ϵ and ϵ' cannot both be nonzero. Notice that

- if $|z| \geq \beta^{e_{\min}}$ then $\epsilon' = 0$;
- if $|z| < \beta^{e_{\min}}$ then $\epsilon = 0$. Moreover, if $\top$ is $+$ or $-$, then the result is *exact*, so that $z = a \top b$ (i.e., $\epsilon' = 0$ too).

The bound on ϵ (namely $\frac{1}{2}\beta^{1-p}$ of β^{1-p}) is frequently called the unit roundoff, denoted **u**.

Adding n numbers : $x_1 + x_2 + x_3 + \cdots + x_n$

- large literature, some recent and smart algorithm ;
- here : Pichat, Ogita, Rump, and Oishi's algorithm

RN : rounding to nearest

Algorithm 3

$s \leftarrow x_1$
$e \leftarrow 0$
for $i = 2$ *to* n ***do***
  $(s, e_i) \leftarrow 2Sum(s, x_i)$
  $e \leftarrow RN(e + e_i)$
end for
return $\sigma = RN(s + e)$

Adding n numbers : $x_1 + x_2 + x_3 + \cdots + x_n$

Theorem 5 (Ogita, Rump and Oishi)

Let

$$\mathbf{u} = \frac{1}{2}\beta^{-p+1}$$

and

$$\gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}}.$$

If $n\mathbf{u} < 1$*, even in case of underflow (but without overflow), the computed result* σ *satisfies*

$$\left|\sigma - \sum_{i=1}^{n} x_i\right| \leq \mathbf{u}\left|\sum_{i=1}^{n} x_i\right| + \gamma_{n-1}^2 \sum_{i=1}^{n} |x_i|.$$

$ad - bc$ with fused multiply-add (radix 2)

Assume an fma instruction is available. Kahan's algorithm for $x = ad - bc$:

$\hat{w} \leftarrow \mathrm{RN}(bc)$
$e \leftarrow \mathrm{RN}(\hat{w} - bc)$
$\hat{f} \leftarrow \mathrm{RN}(ad - \hat{w})$
$\hat{x} \leftarrow \mathrm{RN}(\hat{f} + e)$
Return $\hat{x}$

$u = 2^{-p}$

- using relative error bound $\mathbf{u}$ for operations :

$$|\hat{x} - x| \leq J|x|$$

with $J = 2\mathbf{u} + \mathbf{u}^2 + (\mathbf{u} + \mathbf{u}^2)\mathbf{u}\frac{|bc|}{|x|} \rightarrow$ high accuracy as long as $\mathbf{u}|bc| \not\gg |x|$

- using properties of RN (Jeannerod, Louvet, M., 2011)

$$|\hat{x} - x| \leq 2\mathbf{u}|x|$$

asymptotically optimal error bound.

- Complex division.

Mistakes do not need to be subtle

- The Mars Climate Orbiter probe crashed on Mars in 1999 ;

Mistakes do not need to be subtle

- The Mars Climate Orbiter probe crashed on Mars in 1999 ;
- one of the software teams assumed the unit of length was the meter ;

Mistakes do not need to be subtle

- The Mars Climate Orbiter probe crashed on Mars in 1999 ;
- one of the software teams assumed the unit of length was the meter ;
- the other team assumed it was the foot.

So we do live in the best of all possible worlds. . .

- correct rounding $\rightarrow$ "deterministic arithmetic" ;
- we easily compute the error of a FP addition or multiplication ;
- we can re-inject that error later on in a calculation, to compute accurate sums, dot-products, norms. . .
- already many such compensated algorithms, maybe more to come.

So we do live in the best of all possible worlds. . .

- correct rounding → "deterministic arithmetic" ;
- we easily compute the error of a FP addition or multiplication ;
- we can re-inject that error later on in a calculation, to compute accurate sums, dot-products, norms. . .
- already many such compensated algorithms, maybe more to come.

. . . except life is not that simple !

Deterministic arithmetic ?

C program :

```
double a = 1848874847.0;
double b = 19954562207.0;
double c;
c = a * b;
printf("c = %20.19e\n", c);
return 0;
```

Depending on the environment, `3.6893488147419103232e+19` or `3.6893488147419111424e+19` (`double` number closest to exact product).

Double roundings

- several FP formats supported in a given environment → difficult to know in which format some operations are performed ;
- may make the result of a sequence of operations difficult to predict ;

Assume the various declared variables of a program are of the same format. Two phenomenons may occur when a wider format is available :

- for implicit variables such as the result of "`a+b`" in "`d = (a+b)*c`") : not clear in which format they are computed ;
- explicit variables may be first computed in the wider format, and then rounded to their destination format → sometimes leads to a problem called double rounding.

What happened in the example ?

The exact value of `a*b` is 36893488147419107329. In binary :

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,{\color{red}1}0000000000}^{64\text{ bits}}\,0{\color{red}1}$$

If it is first rounded to the INTEL "double-extended" format, we get

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,{\color{red}1}0000000000}^{64\text{ bits}}\times 4$$

if that intermediate value is rounded to the binary64 destination format, this gives (round-to-nearest-even rounding mode)

$$\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\times 2^{13}$$

$$= 36893488147419103232_{10},$$

$\longrightarrow$ rounded down, whereas it should have been rounded up.

Is it a problem ?

- In most applications, these phenomenons are innocuous ;
- they make the behavior of some numerical programs difficult to predict (very interesting examples given by Monniaux) ;
- most compilers offer options that prevent this problem. However,
  - be ready to dive into huge, unreadable documentation ;
  - restricts the portability of numerical programs ;
  - may have impact on performance and accuracy

$\rightarrow$ examine which properties remain true when double roundings may occur (for instance : some summation algorithms still work, some do not).

No problem with SSE instructions, and IEEE 754-2008 improves the situation.

An example : 2Sum and double roundings

Precision-p "target" format ; precision $p + p'$ wider "internal" format.

Algorithm 4 (2Sum-with-double-roundings(a, b))

(1) $s \leftarrow RN_p(RN_{p+p'}(a+b))$ *or* $RN_p(a+b)$
(2) $a' \leftarrow RN_p(RN_{p+p'}(s-b))$ *or* $RN_p(s-b))$
(3) $b' \leftarrow \circ(s-a')$
(4) $\delta_a \leftarrow RN_p(RN_{p+p'}(a-a'))$ *or* $RN_p(a-a')$
(5) $\delta_b \leftarrow RN_p(RN_{p+p'}(b-b'))$ *or* $RN_p(b-b')$
(6) $t \leftarrow RN_p(RN_{p+p'}(\delta_a+\delta_b))$ *or* $RN_p(\delta_a+\delta_b)$

$\circ(u)$: $\mathrm{RN}_p(u)$, $\mathrm{RN}_{p+p'}(u)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(u))$, or any faithful rounding.

An example : 2Sum and double roundings

Theorem 6

$p \geq 4$ and $p' \geq 2$. If a and b are precision-p FPN, and if no overflow occurs, then Algorithm 4 satisfies :

- *if no double rounding bias occurred when computing s then $t = (a + b - s)$ exactly ;*
- *otherwise, $t = RN_p(a + b - s)$.*

$\longrightarrow$ many properties remain true, or only require slight changes ;

- watch interesting, in-progress, work of Sylvie Boldo (`http://www.lri.fr/~sboldo/`), on "hardware-independent" proofs.

The Hall of Shame (Ng)

System	$\sin(10^{22})$
exact result	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	-0.85220084976718879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Until 2008, no standard for the elementary functions.

The Hall of Shame (Ng)

System	$\sin\left(10^{22}\right)$
exact result	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	-0.8522008497671879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Until 2008, no standard for the elementary functions.

The Hall of Shame (Ng)

System	$\sin\left(10^{22}\right)$
exact result	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	-0.85220084976718879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Until 2008, no standard for the elementary functions.

The Hall of Shame (Ng)

System	$\sin\left(10^{22}\right)$
exact result	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	-0.85220084976718879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Until 2008, no standard for the elementary functions.

Correctly-rounded elementary functions ?

Evaluating $f(x)$:

- find an approximation $\hat{f}(x)$ to $f(x)$;
- round that approximation to "target" format.

To certify that we always return $\mathrm{RN}(f(x))$:

- solve the Table maker's dilemma : determine the accuracy of the approximation that guarantees (if this is possible) :
  $\forall x, \mathrm{RN}(\hat{f}(x)) = \mathrm{RN}(f(x))$;
- guarantee that your approximation is within the required accuracy.

Tools for polynomial approximations of functions

- Sollya (written by Sylvain Chevillard and Christoph Lauter) : computes nearly best approximations with constraints on the coefficients (such as requiring them to be FP numbers, or sums of 2 FP numbers, ...) ;

  `http://sollya.gforge.inria.fr/`

- Gappa (written by Guillaume Melquiond) : uses interval arithmetic to manage ranges and errors of straight-line programs (typically a polynomial evaluation), forces you to express some numerical property to prove, and outputs a proof of that property suitable for checking by Coq (or HOL light) ;

  `http://gappa.gforge.inria.fr`

- also, watch Flocq (Sylvie Boldo and Guillaume Melquiond) : floating-point formalization for the Coq system. Comprehensive library of theorems on a multi-radix multi-precision arithmetic.

The Table Maker's Dilemma

Consider the double precision FP number ($\beta = 2, p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.\underbrace{99999999999999999}_{17}8805240837303\cdots$$

So what ?

Hardest-to-round case for function 2^x and double precision FP numbers.
Joint work with Vincent Lefèvre.

Correct rounding of the elementary functions

- base 2, precision p;
- FP number x and integer m (with $m > p$) $\rightarrow$ one can compute an approximation y to $f(x)$ whose error on the significand is $\leq 2^{-m}$.
- can be done with a possible wider format, or using algorithms such as TwoSum, TwoMultFMA, Dekker product, etc.
- getting a correct rounding of $f(x)$ from y : not possible if $f(x)$ is too close to a breakpoint : a point where the rounding function changes.

Correct rounding of the elementary functions

- RN mode,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p\text{ bits}}1000000\cdots000000}^{m\text{ bits}}xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p\text{ bits}}0111111\cdots111111}^{m\text{ bits}}xxx\cdots;$$

- other modes,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p\text{ bits}}0000000\cdots000000}^{m\text{ bits}}xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p\text{ bits}}1111111\cdots111111}^{m\text{ bits}}xxx\cdots.$$

Finding m beyond which there is no problem ?

- function f : sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\rightarrow$ except for straightforward cases (e^0, $\ln(1)$, $\sin(0)$, ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\rightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;
- this reasoning does not give any hint on the order of magnitude of $m_{\max}$. Could be huge.

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application : To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application : To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

$$10^{244} \text{ bits}$$

Fortunately, in practice, much less (≈ 100).

Results

Table: *Worst cases for exponentials of double precision FP numbers.*

Interval	worst case (binary)
$[-\infty, -2^{-30}]$	$\exp(-1.1110110100110001100011101111101101100010011111101010 \times 2^{-27})$ $= 1.11111111111111111111111111100 \cdots 0111000100 \quad 1 \quad 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.0000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.111111111111111 \cdots 11111111111111100 \quad 0 \quad 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.1111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.0000000000000000000000000000000000000000000000000000 \quad 1 \quad 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.0111111111111110011111111111111011100000000000100100 \times 2^{-32})$ $= 1.0000000000000000000000000000000101111111111111101000 \quad 0 \quad 0^{57}1101\ldots$
	$\exp(1.1000000000000001011111111111111011011111111111011100 \times 2^{-32})$ $= 1.0000000000000000000000000000000110000000000000010111 \quad 1 \quad 1^{57}0010\ldots$
	$\exp(1.1001111010011100101110111111110101100000010000001011 \times 2^{-31})$ $= 1.0000000000000000000000000000001100111101001110010111 \quad 1 \quad 0^{57}1010\ldots$
	$\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \quad 0 \quad 0^{57}1000\ldots$

Results

Table: *Worst cases for logarithms of double precision FP numbers.*

Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.1110101001110001110110000101110011101110000000100000 \times 2^{-509})$ $= -101100000.0010100101101010011001101011010000101111111 \quad 1 \quad 1^{60}0000\ldots$
	$\log(1.1001010001110110111000110000010011001101011111000111 \times 2^{-384})$ $= -100001001.1011011000001100101011110100011110110011010 1 \quad 1 \quad 0^{60}1010\ldots$
	$\log(1.0010011011110100111000100110100110010011110010110000 0 \times 2^{-232})$ $= -10100000.101010110010110000100101111001101000010000100 \quad 0 \quad 0^{60}1001\ldots$
	$\log(1.0110000100111001010101011101110010000000001011111000 \times 2^{-35})$ $= -10111.111100000001011111001101110101111011000000000110101 \quad 0 \quad 1^{60}0011\ldots$
$(1, 2^{1024}]$	$\log(1.0110001010101000100001100001001101100010100110110110 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{64}1110\ldots$

Floating-point arithmetic on the web

- W. Kahan :
  http://http.cs.berkeley.edu/~wkahan/
- Goldberg's paper "What every computer scientist should know about Floating-Point arithmetic"
  http://www.validlab.com/goldberg/paper.pdf
- D. Hough :
  http://www.validlab.com/754R/
- The Arenaire team of lab. LIP (ENS Lyon)
  http://www.ens-lyon.fr/LIP/Arenaire/
- my own web page
  http://perso.ens-lyon.fr/jean-michel.muller/