A few words on floating-point arithmetic

Jean-Michel Muller

November 2000

www.ens-lyon.fr/~jmmuller

Microsoft Excel (Kahan)

Versions 3.0 to 7.0. Enter :

1.40737488355328

You will get:

0.64

« free » software too

- Linux on DEC alpha (at least some versions)
- 16777213×8388609 with rounding to nearest: 1.4073747158e+14 instead of 1.4073747997e+14
- some wrong additions when one of the operands is a denormal number

The Pentium bug

Operation	Pentium	Good result
8391667 / 12582905	0.666869455…	0.666910145…
12845015 / 11010020	1.166619406…	1.16666591…
4195835.0 / 3145727.0	1.333739068…	1.33382044…

Rounding errors

- Compute, for a = 77617.0 and b = 33096.0,

$$f(a,b) = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b}$$

- on an IBM 370, we get:
  - single precision: 1.172603
  - double precision: 1.17260394005 31
  - double extended: 1.172603940053178
- exact result: – 0.8273960599…

Not only in scientific applications

- 1982: the Vancouver stock exchange introduced an index with nominal value 1000.000
- after each transaction, recomputed and truncated to the 3rd fractional digit
- after 22 months, index = 524.881
- correct value 1098.811

Explanation: bias. The rounding errors are all of the same sign.

Wrong handling of exceptions

- First launching of the Ariane 5 rocket (1996): overflow;
- guided-missile cruiser USS Yorktown (1998): A crew member mistakenly entered a zero for a data value. Resulted in a division by 0. The error cascaded and eventually shut down the ship's propulsion system (Scientific American, November 1998)

Straightforward things

$$(-1)^s \, x_0 . x_1 x_2 \cdots x_{n-1} \times B^e$$

- In general: B=2 (although 16 and 10 still exist)
- e : exponent
- *normalized* mantissa: $x_0 \neq 0$, gives mantissa between 1 and B
- radix 2, with normalized mantissas: $x_0 = 1$, so no need to store it (Implicit 1 convention. Need of special representation for zero)

	Base B	*# mantissa digits.*	*minimum exponent*	*maximum exponent*
CRAY 1	2	48	-8192	8191
	2	96	-8192	8191
DEC VAX	2	24	-127	127
	2	56	-127	127
IBM 370	16	6 (24 bits)	-64	63
IEEE 754	2	23 (+1)	-126	127
	2	52 (+1)	-1022	1023

Rounding modes (IEEE 754 Std)

- In general, result of an operation must be rounded
- « Exact », or « correct » rounding: deterministic, choice among 4 modes
  - To the nearest (special convention if exactly halfway. Does not occur for division or sqrt)
  - Towards $+\infty$
  - Towards $-\infty$
  - Towards zero

The system must behave as if the computation was first carried out with « infinite » accuracy, then rounded.

Properties of correct rounding

- The behaviour of the system is reproductible and previsible. One can build *algorithms* and *proofs*.
- Better portability of numerical software.
- Interval arithmetic.
- Monotonicity.
- Seems difficult to satisfy for transcendental functions.

Some examples

- Algorithms for division/sqrt that are proved correct (Intel/HP, Arith 14);
- Proofs of respect of output ranges;
- Algorithms for accurate summation, expansions...

B = radix of the internal FP representation

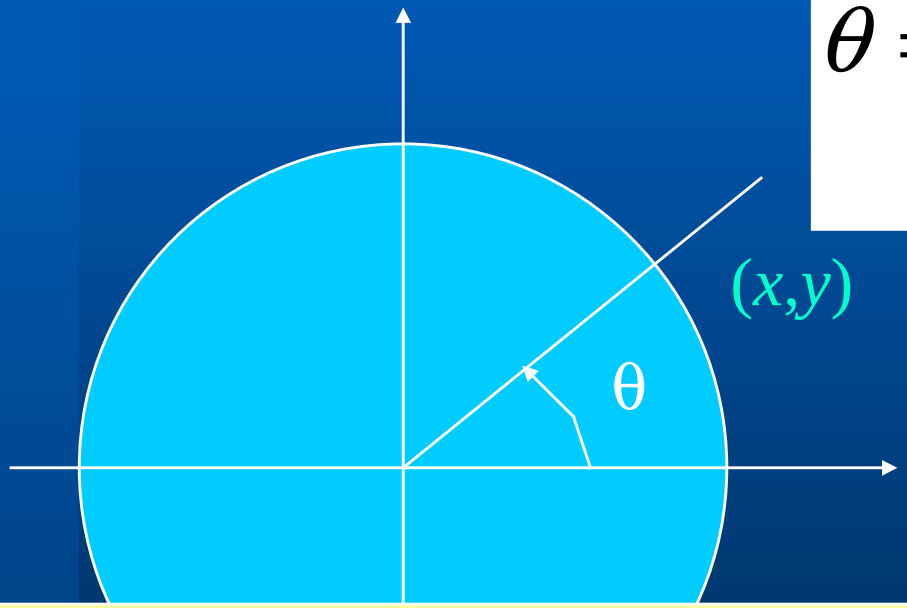
A := 1; B := 1;

while ((A+1)-A-1) = 0 do A := 2*A;

while ((A+B)-A-B) ≠0 do B := B+1;

Respect of output ranges (Kahan)

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



No problem with correct rounding to nearest

Exception handling

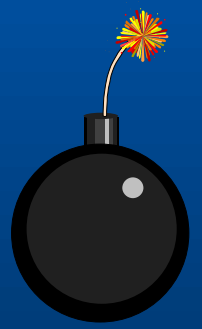
- Two infinities ⟺ two zeros, with « intuitive » behaviour (well… $\sqrt{(-0)} = -0$);
- « Denormal » numbers: smallest possible exponent;

$$x \neq y \Leftrightarrow x-y \neq 0$$

- Not A Number (NaN):
  - $(\pm\infty) \pm (\pm\infty)$ if not determined ;
  - $0 * \pm\infty$, $0 / 0$ and $\pm\infty / \pm\infty$
  - Remainder of divisions: $x/0$ and ∞/x;
  - Square root of negative number.

NaN (cont.)

- Basic idea: the computation always continue and ends-up safely;
- propagation of NaNs (best strategy for simulation, *maybe dangerous for critical systems*);
- if *x* is a NaN, then *x* == *x* returns « false », whereas *x* != *x* returns « true »



- if *x* or *y* is a NaN, then *x* < *y*, *x* <= *y*, *x* == *y*, *x* >= *y* and *x* > *y* must return “ false ” ⇒ must take that into account in compiler optimization and proofs.

Binary coding

number	*sign bit*	*Exponent*	*Implicit 1*	*Fraction*
NaN	?	111…11	No	1xxxx…xx
SnaN	?	111…11	No	0xxxx…xx
Infinities	±	111…11	No	0
Normalized	±	E+biais	1	= 0
Denorms	±	0	No	> 0
Zeros	±	0	No	0

IEEE Std and conversions

No errors

- Radix-10 number, at most 6 digits converted to SP then converted back to radix 10 with same # of digits;
- SP number converted to radix-10 string with at least 9 digits, then converted back to SP;
- Radix-10 number, at most 15 digits converted to DP then converted back to radix 10 with same # of digits;
- DP number converted to radix-10 string with at least 17 digits, then converted back to SP;

Goal: no error when storing temporary results in files

Side effect: one may feel that the precision is larger than what it is

Influence of the Compiler

- problems with constants: $0.1 \times x$
- detection of dependence between change of rounding mode and arithmetic operation (DEC Alpha)
- tests to make sure that dangerous simplifications are not performed

Shame on them (Ng)

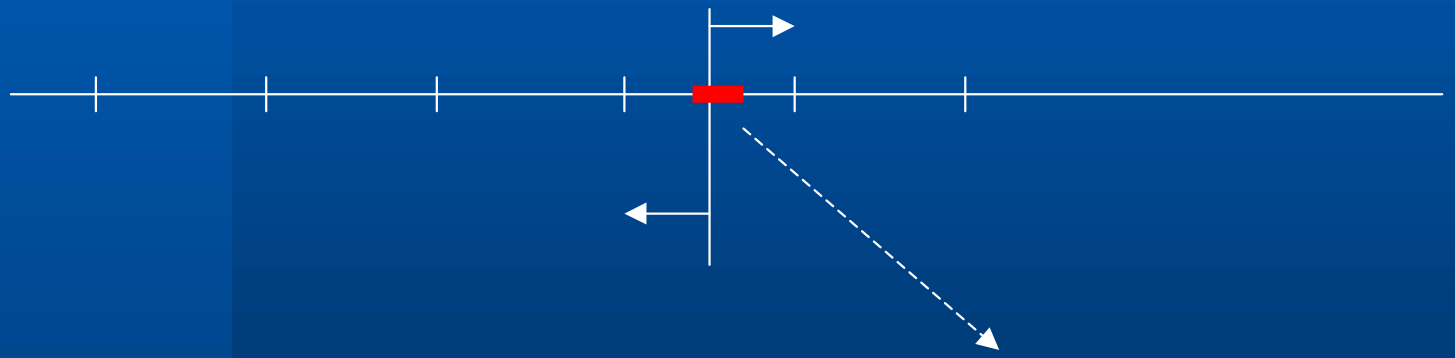
System	sin (10^{22})
exact	-0.852200849767…
HP 48 GX	-0.852200849767
HP 700	0.0
IBM 3090/600S-VF AIX 370	0.0
Matlab V.4.2c.1 for SPARC	-0.8522
Matlab V.4.2c.1 macintosh	0.8740
Silicon graphics Indy	0.87402806
Sharp EL5806	-0.090748172
DEC Station 3100	NaN

Why ?

- No specification of elementary functions;
- Origin of the problem: Table Maker ’s Dilemma

To make sure that we get correct rounding with *n* bits of mantissa, what is the required precision for intermediate calculations ?

Example: rounding to nearest



f(x) is somewhere in this red area

Radix 2

- « Directed » rounding modes:

n bits

x.xxxxxxxxxxx 0000000000000 1zzzzzz

x.xxxxxxxxxxx 111111111111110 zzzzzz

m bits

- Rounding to nearest

x.xxxxxxxxxxx 1000000000000 1zzzzzz

x.xxxxxxxxxxx 0111111111110 zzzzzz

Example: 24 bits mantissas

- The single prec. number 1.11001100111000110011001 equals :

$$x = \frac{15102361}{8388608}$$

- its exponential is:

110.00001101001111010011 0 11111111111111111111111
01100100001…

Results from number theory

- Baker (1975 - Special case)
- $\alpha = p/q$, $\beta = r/s$, $p,q,r,s < 2^n$,
- $C = 16^{200}$

$$|\alpha - \log \beta| > \left(n 2^n\right)^{-Cn \log n}$$

Application: to compute ln and exp in double precision (n=53), it suffices to get an intermediate approximation good to approximately

10^{244} bits

We have partially solved the problem*

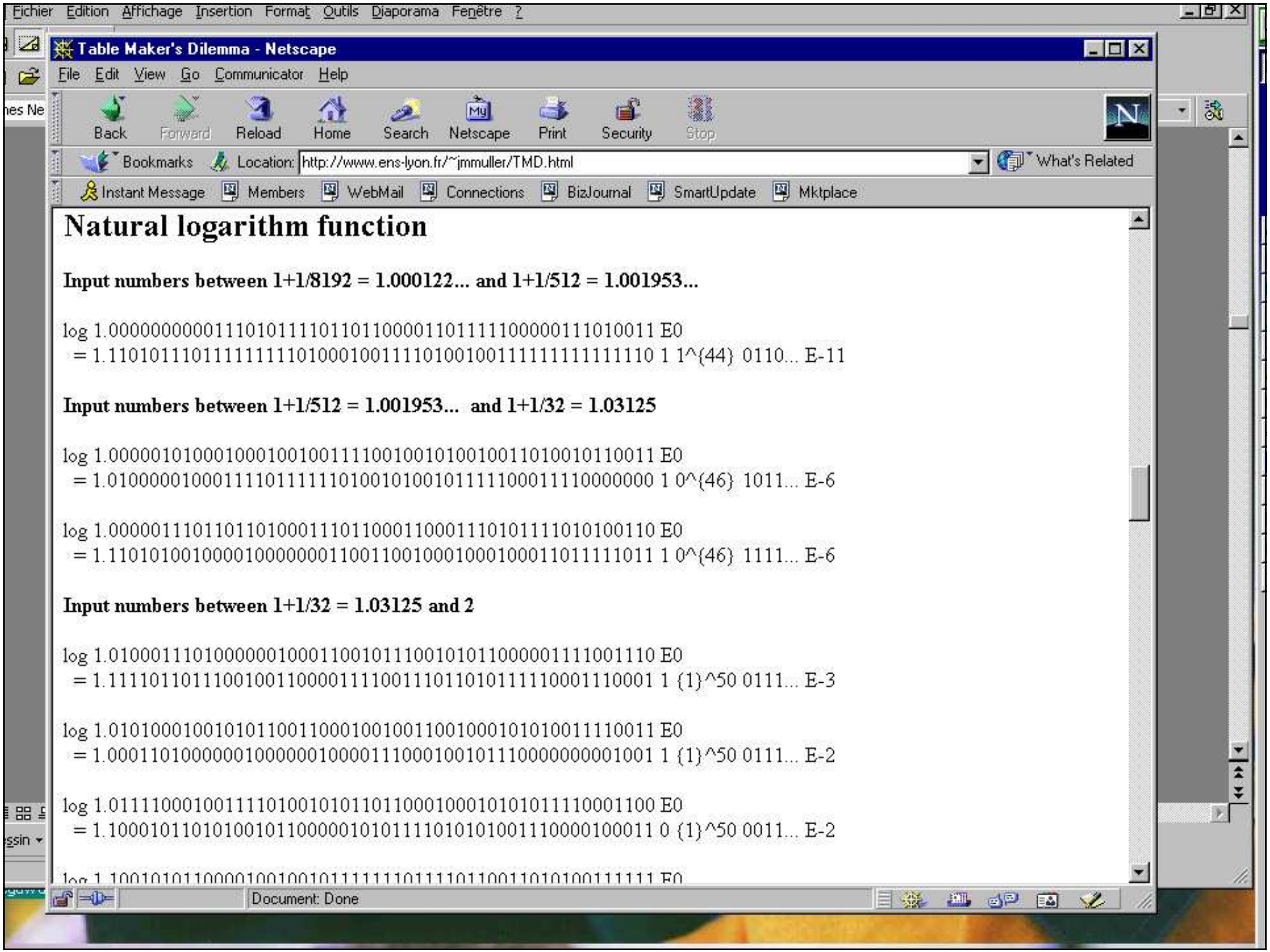
** for double precision & some functions!*

- See V. Lefèvre 's talk
- Public part available at

`http://www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm`

- Confidential part

- exp (bases e and2), log (bases e and 2), sin, arcsin, cos, arccos, 1/sqrt,


Fichier Edition Affichage Insertion Format Outils Diaporama Fenêtre ?
Table Maker's Dilemma - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.ens-lyon.fr/~jmmuller/TMD.html
What's Related
Instant Message Members WebMail Connections BizJournal SmartUpdate Mktplace
Natural logarithm function
Input numbers between 1+1/8192 = 1.000122... and 1+1/512 = 1.001953...
log 1.0000000000111010111101101100001101111100000111010011 E0
= 1.11010111011111111101000100111101001001111111111111110 1 1^{44} 0110... E-11
Input numbers between 1+1/512 = 1.001953... and 1+1/32 = 1.03125
log 1.0000010100010001001001111001001010010011010010110011 E0
= 1.01000001000111101111110100101001011111000111100000000 1 0^{46} 1011... E-6
log 1.0000011101101101000111011000110001110101111010100110 E0
= 1.11010100100001000000011001100100010001000110111111011 1 0^{46} 1111... E-6
Input numbers between 1+1/32 = 1.03125 and 2
log 1.0100011101000000100011001011100101011000001111001110 E0
= 1.11110110111001001100001111001110110101111100011100 01 1 {1}^50 0111... E-3
log 1.0101000100101011001100010010011001000101010011110011 E0
= 1.00011010000001000000100001110001001011100000000001001 1 {1}^50 0111... E-2
log 1.0111100010011110100101011011000100010101011110001100 E0
= 1.10001011010100101100000101011110101010011100001000 11 0 {1}^50 0011... E-2
Document: Done

Some examples of worst cases

- ln(1.01100010101010001000011000010011011000101001101 10110 $\times 2^{678}$) = 111010110.0100011110011110101110100111110010010111 0001 0^{65} 11…
- 2^(1.11100100010110010110010100100110101111111001010 01101 $\times 2^{-10}$) = 1.000000000101001111111100001011011000010101101010 011 0 1^{59} 0100…
- $\log_2$(1.01100001010101010111111011101011000100001011 0110100 $\times 2^{512}$) = 1000000000.0111011100000010110100000011110100110111 001 1^{56} 0011…

Floating-point arithmetic on the Internet

- W. Kahan: **HTTP.CS.Berkeley.EDU/~wkahan/**
- D. Hough: **www.validgh.com/**
- Interval computations: **www.cs.utep.edu/interval-comp/**
- J. Harrison ’s page on FP verification: **www.cl.cam.ac.uk/users/jrh/fpv/index.html**
- our TMD page: **www.ens-lyon.fr/~jmmuller/Intro-to-TMD.htm**
- Loria ’s page on FP: **cch.loria.fr/documentation/IEEE754/**

- and many others… (apologies if I forgot you)