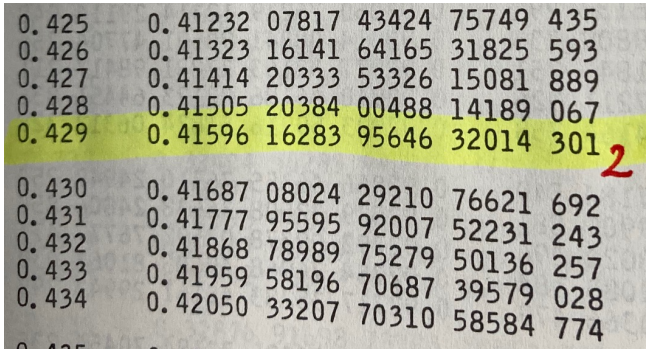
The Table Maker's Dilemma and Correctly-Rounded Functions: A Quick Review of 25 Years of Work

Jean-Michel Muller

CNRS – LIP Laboratory
ARITH 2026, June 2026

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

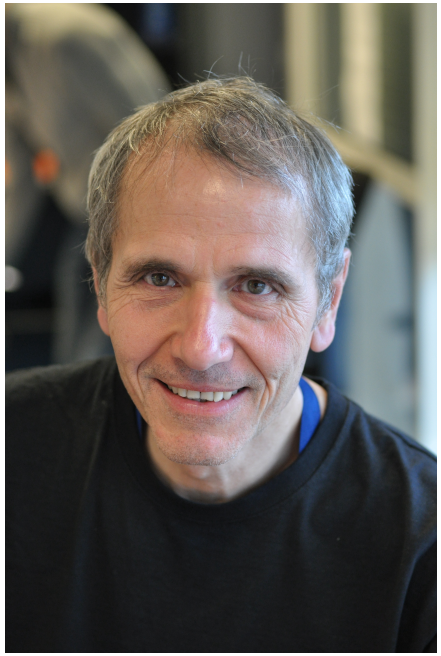


$$2^{43}\cdot\log\left(3119607361246427\cdot 2^{627}\right) = 4136624566035640.5000000000000000000122\cdots$$

This talk is dedicated to the memory of...

Peter Kornerup

Tomas Lang



Serge Torres

... good friends, with whom it was always a pleasure to work.

Purpose?...

FP arithmetic is not a mere "fuzzy" approximation of the reals. Use and possibly enhance the structure in the arithmetic of our computers....

- The result of `c = a + b` or `c = a / b` is unambiguously determined. So should be the result of `y = sin(x);`
- Allows proving subtle and useful properties of programs;
- Facilitates validation, testing, debugging, portability, and reusability of programs.

Linked to "coincidences" such as:

$$2^{53} \cdot \cos\left(\frac{5307249884967}{2^{41}}\right) = -6723126034308461.4999999999999999898\cdots$$

$$2^{43} \cdot \log\left(3119607361246427 \cdot 2^{627}\right) = 4136624566035640.5000000000000000000122\cdots$$

Binary Floating-Point (FP) Arithmetic

- Base-2 FP system of width w:

$$\begin{cases} \text{precision} & p \geq 1 \\ \text{extreme exponents} & e_{\min}, e_{\max}, \end{cases}$$

  with $w = \lceil \log_2 (e_{\max} - e_{\min} + 1) \rceil + p$.

- FP number x: integral significand M and exponent e, $(M, e) \in \mathbb{Z}$, $|M| \leq 2^p - 1$, and $e_{\min} \leq e \leq e_{\max}$, such that

$$x = \left(\frac{M}{2^{p-1}} \right) \times 2^e.$$

  If multiple choices, we take minimal e under these constraints.

Official name	Common name	w	p	$e_{\min}$	$e_{\max}$
binary16		16	11	−14	15
binary32	single precision	32	24	−126	127
binary64	double precision	64	53	−1022	1023
binary128	quad. precision	128	113	−16382	16383

The world of floating-point computing until the early 80s...

Machine	Underflow	Overflow
DEC PDP-11, VAX, F and D formats	$2^{-128} \approx 2.9 \times 10^{-39}$	$2^{127} \approx 1.7 \times 10^{38}$
DEC PDP-10; Honeywell 600, 6000; Univac 110x single; IBM 709X, 704X	$2^{-129} \approx 1.5 \times 10^{-39}$	$2^{127} \approx 1.7 \times 10^{38}$
Burroughs 6X00 single	$8^{-51} \approx 8.8 \times 10^{-47}$	$8^{76} \approx 4.3 \times 10^{68}$
H-P 3000	$2^{-256} \approx 8.6 \times 10^{-78}$	$2^{256} \approx 1.2 \times 10^{77}$
IBM 360, 370; Amdahl;	$16^{-65} \approx 5.4 \times 10^{-79}$	$16^{63} \approx 7.2 \times 10^{75}$
Most handheld calculators	10^{-99}	10^{100}
CDC 6X00, 7X00, Cyber	$2^{-976} \approx 1.5 \times 10^{-294}$	$2^{1070} \approx 1.3 \times 10^{322}$
DEC VAX G format	$2^{-1024} \approx 5.6 \times 10^{-309}$	$2^{1023} \approx 9 \times 10^{307}$

Source: W. Kahan, *Why do we need a Floating-Point Standard*, 1981.
Available at `https://people.eecs.berkeley.edu/~wkahan/ieee754status/why-ieee.pdf`

A must-read!

The world of floating-point computing until the early 80s...

- vastly different environments (precision, range, exception handling ...);
- some good, many quite poor;
- almost no portability of numerical software;
- impossible to prove anything (what can you prove if you don't even know what $c \leftarrow a + b$ gives?);
- need to know "wizard tricks":
  - `(0.5 - x) + 0.5` often better than `(1 - x);`
  - multiplying by 1 sometimes helps
    $\cdots$ on the other hand, one could get an overflow when doing that;

$\rightarrow$ In short... cooking recipes! (and rarely *good* recipes)

See *An Interview with the Old Man of Floating-Point*, IEEE Computer, March 1998.

The world of floating-point computing until the early 80s...

In a tech report by K.C. Ng, *Argument reduction for huge arguments*, 1992:

Environment	$\sin\left(10^{22}\right)$
Exact result	$-0.852200849767188801772\cdots$
HP 700	0.0
HP 375, Sun3	−0.653652882
IBM RS/6000 AIX 3005	−0.85220084···
IBM 3090/600S-VF AIX 370	0.0
IBM PC with Borland Turbo C	4.67734e-240
IBM PC with Borland Turbo Pascal	0.0
Casio fx-8100	Error

Concerning math functions, huge improvements by the late 80's/early 90's:

- Cody & Waite's book,
- Peter Tang's *table-driven* algorithms (e.g., ARITH'1991 paper);
- Gal & Bachelis;
- Payne & Hanek range reduction algorithm...

The IEEE-754 Floating-Point standard (1985)

Specifies formats, exception handling, and requires correct rounding for certain operations.

Definition 1 (Correct Rounding)

Given a *rounding function* $\circ$ from $\mathbb{R}$ to FP numbers among:

- $\circ_n(x)$: to nearest (default), with well-specified tie-breaking rule(s) if x is the midpoint of two consecutive FP numbers;
- $\circ_u(x)$: towards $+\infty$;
- $\circ_d(x)$: towards $-\infty$;
- $\circ_z(x)$: towards zero.

An operation whose inputs are FP numbers is correctly rounded if the returned result is what would be obtained by applying the rounding function to the exact result.

The IEEE-754 Floating-Point standard (1985)

Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$.

Assuming the chosen rounding function is $\circ_n$:

$$\text{program } \texttt{c = a+b} \rightarrow \text{computation } c = \circ_n(a+b)$$

Advantages:

- If only $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$ are used, and the order of operations does not change, arithmetic is deterministic $\rightarrow$ algorithms and proofs;
- Improved accuracy and portability;
- $\forall t$ in normal domain, $|\circ_n(t) - t| \leq \frac{1}{2^p+1} \cdot |t| \rightarrow$ upper bound $v = \frac{1}{2^p+1}$ of the relative error of $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$;

$\rightarrow$ rounding error analysis *à la* Wilkinson.

2008 update: correct rounding recommended but not required for some math functions.

A few well-known small Examples

Program	What is actually computed	Provable property

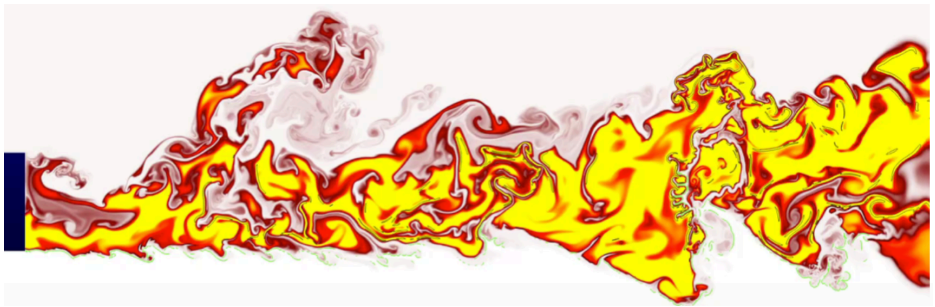
`s = a+b;`
`z = s-a;`
`t = b-z;`
	$s = \circ_n(a+b)$
	$z = \circ_n(s-a)$
	$t = \circ_n(b-z)$
		$t = (a+b) - s$
		(t = error of FP addition of a and b)
		(Kahan-Møller-Dekker)

`z = x / sqrt(x*x+y*y)`	$z = \circ_n\left(\frac{x}{\circ_n\left(\sqrt{\circ_n\left(\circ_n(x^2) + \circ_n(y^2)\right)}\right)}\right)$	$\lvert z\rvert \leq 1$ (Kahan)

`w = b*c;`
`e = fma(-b,c,w);`
`f = fma(a,d,-w);`
`x = f+e`
(Kahan's alg. for $ad - bc$)
	$w = \circ_n(bc)$
	$e = \circ_n(w - bc)$
	$f = \circ_n(ad - w)$
	$x = \circ_n(f + e)$
		$\left\lvert \frac{x-(ad-bc)}{ad-bc} \right\rvert \leq 2^{-p+1}$
		(Jeannerod, Louvet, M.)

Note: `fma(a,b,c)` is $\circ_n(ab + c)$.

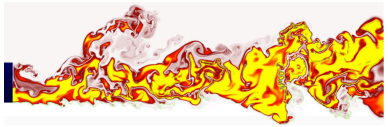
In the 80's, not that many application domains



Numerical simulation

Financial calculations

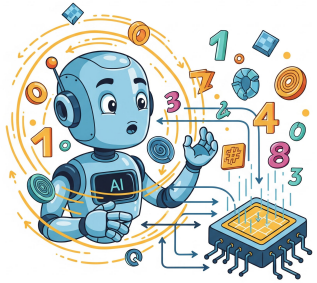
Now the situation is quite different



Numerical simulation

Financial calculation

Games, entertainment



Machine learning

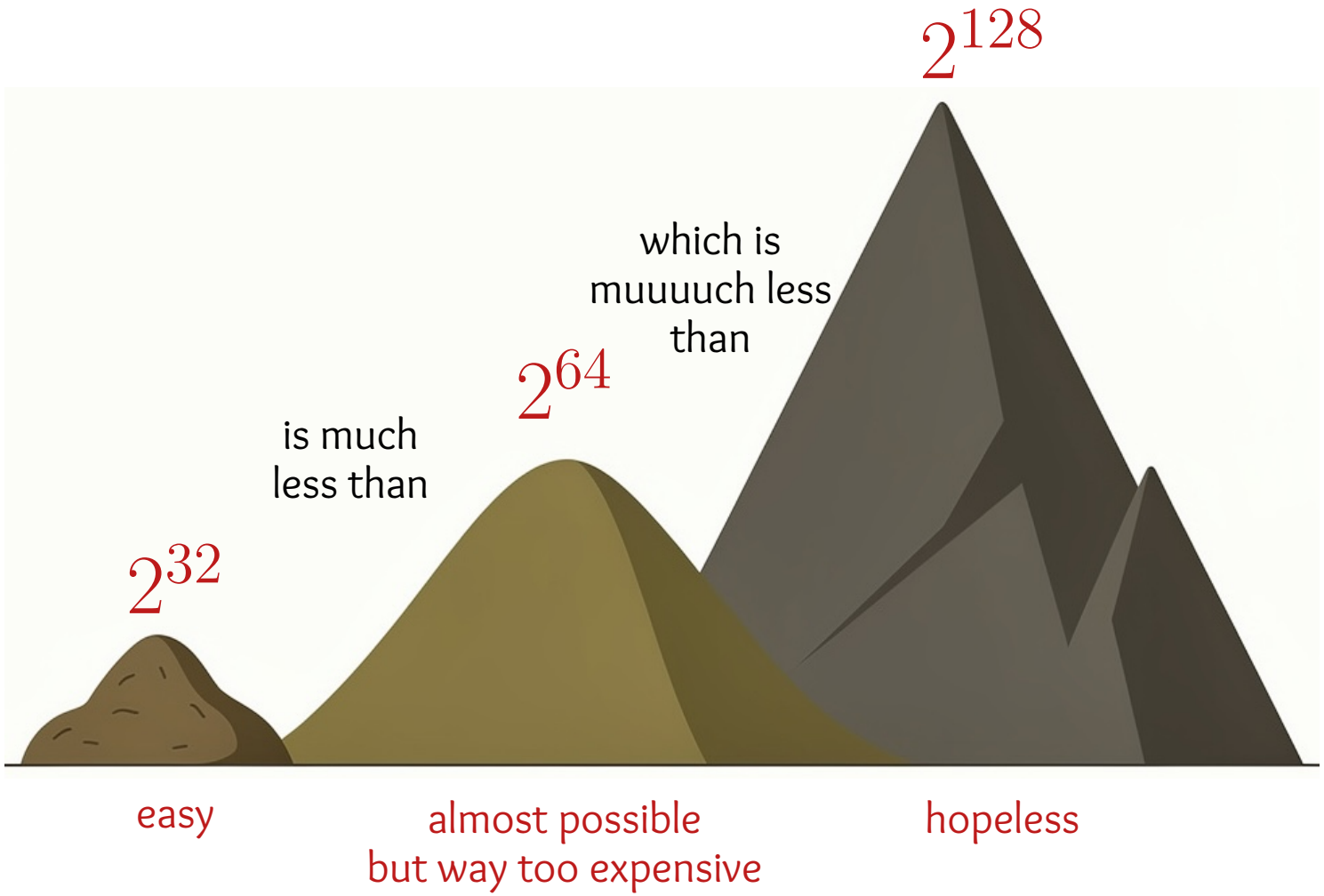
Embedded computing

• • •

Reproducibility and possibility of (formal) proof have become much more important.

Building a core set of correctly rounded functions

- Goals: Portability, accuracy, reproducibility, same upper bound $\frac{1}{2^p+1}$ on the relative error as for operations;
- But don't forget that if we need exhaustive testing:



Building a core set of correctly rounded functions

Several issues:

- Which functions?
- Is it possible to compute them efficiently *and* with correct rounding?
- In large formats (64 or 128 bits), how to guarantee that our programs provide correct rounding?

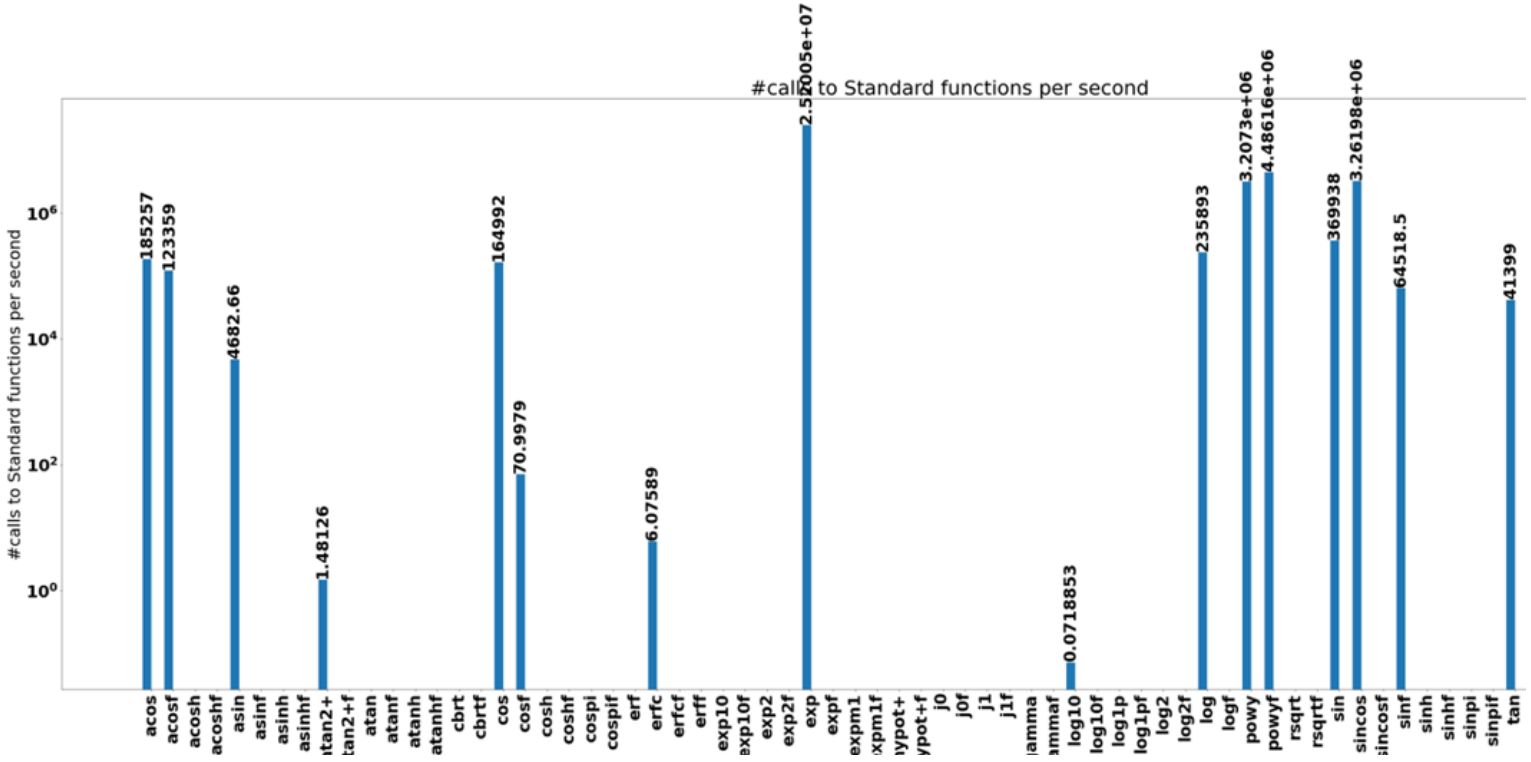
  (Small formats: enumerate all possible inputs)

Reminder: the programs computing exp, sin, ... will be called billions of times:

→ Their design can take time;

→ Their execution must be fast.

Which Functions? Frequency of calls in CERN applications



Piparo and Innocente, *The CptnHook Profiler - A tool to investigate usage patterns of mathematical functions,* J. Phys.: Conf. Ser., 2016.

Winners: exp, log, sin, cos (and x^y, maybe often for wrong reasons: 2^x, $x^{1/2}$, x^2, ...).

Building a core set of correctly rounded functions

Work program (spread over 25 years):

- make sure we "measure" the errors correctly: tight and certain bounds on
  - the errors of the polynomial approximations,
  - the (rounding) errors of evaluating these approximations;
- "Almost best" polynomial approximations with FP coefficients;
- With what accuracy approximate the function to be certain that rounding the approximation is equivalent to rounding the exact value? This question: Table Maker's Dilemma (TMD).
- Design function evaluation algorithms
  - two steps: a fast step, followed if necessary by an accurate step (Ziv's strategy)
  - Average time $\approx$ time of the fast step.

Done with Nicolas Brisebarre; Sylvain Chevillard; David Defour; Florent de Dinechin; Guillaume Hanrot; Mioara Joldes; Christoph Lauter; Vincent Lefèvre; Erik Martin-Dorel; Guillaume Melquiond; Damien Stehlé; Arnaud Tisserand; Serge Torres; Paul Zimmermann.

Near best approximations – Just an example: function $\log(1+x)$ in $[0,1]$

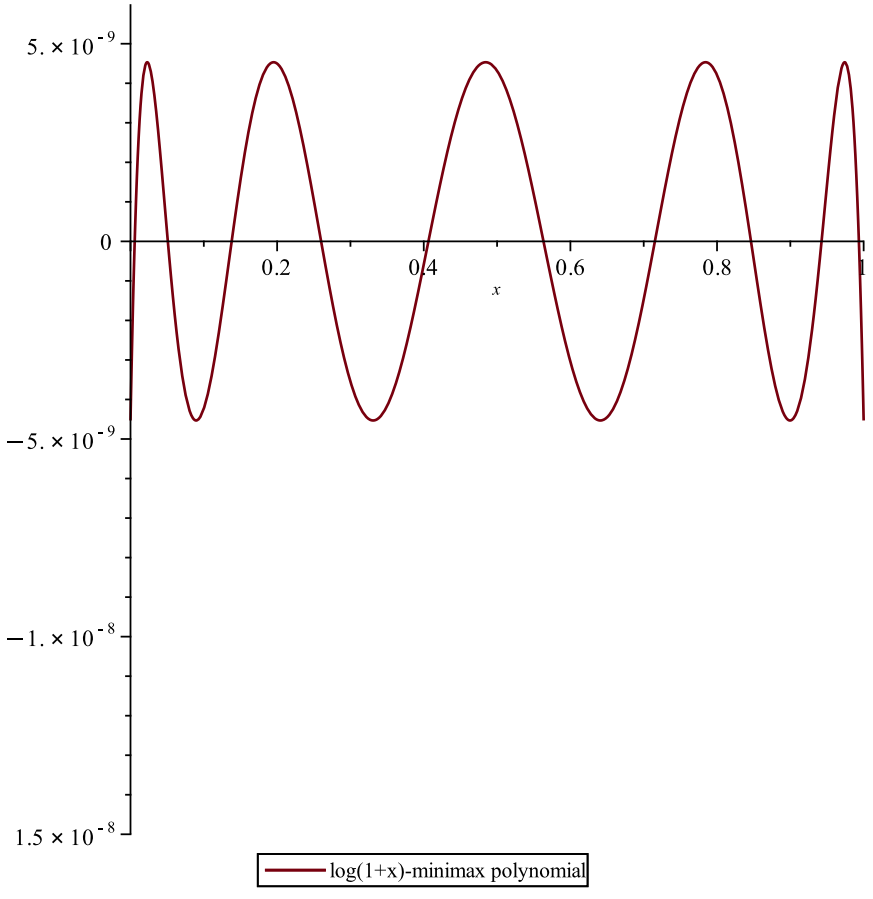
- Goal: binary32/single precision arithmetic ($\beta = 2$, $p = 24$);
- here: degree-9 polynomial;
- the minimax polynomial returned by Remez' algorithm is:

$$
\begin{aligned}
p^*(x) = {} & 4.531262676417885304508397507377211944683978417446495645952964327098704181 87 \cdot 10^{-9} \\
& +0.999990025258539069369331280889443153540235373687738534675639041879165305071668 8 \cdot x \\
& -0.499965301733306862646350263286494969422243084300307456583308608494202815567973 3 \cdot x^2 \\
& +0.332849806189948672280944634938400232746236110583281184032142726147068820095047 5 \cdot x^3 \\
& -0.246517965788047515383441886060448202153937598635915727063060602574084424542720 9 \cdot x^4 \\
& +0.185154569966668361477975368769654445113328710403271118930768041265705015304244 4 \cdot x^5 \\
& -0.126057455452415889884958065630540564789570519172717287039473166654700319951009 6 \cdot x^6 \\
& +0.067294552999244180561977536465001023978793925662150291856930540995123804269168 22 \cdot x^7 \\
& -0.023455350690510775380743240497604729489911455890469902806116859364202076167174 5 \cdot x^8 \\
& +0.003845299809826069022496755870766178552568672023224498117158896293738444907987 099 \cdot x^9 .
\end{aligned}
$$

- error $\approx 4.53 \times 10^{-9}$.

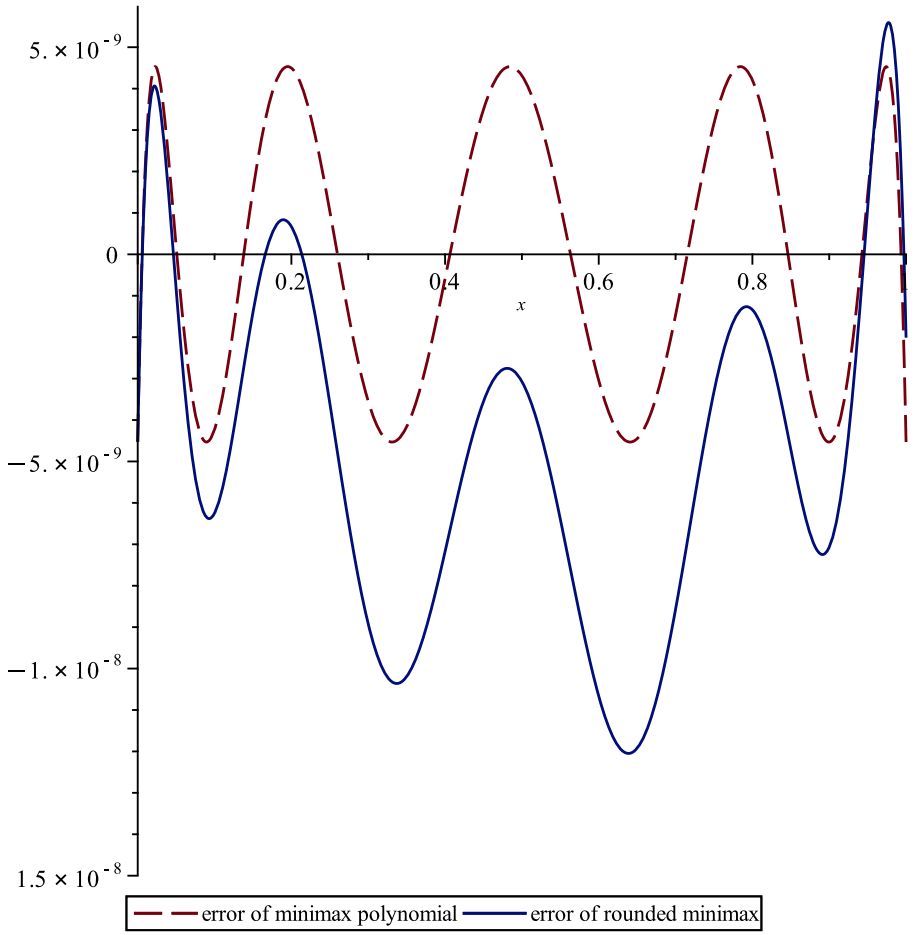
Near best polynomial approximations with FP coefficients – Just an example: function $\log(1+x)$ in $[0,1]$

- error curve $\log(1+x) - P^*(x)$:



- Problem: P^* has real coefficients. I want to implement the function in binary32 arithmetic. What happens if I round each coefficient of P^* to the nearest FP number?

The disaster...



Error multiplied by ≈ 2.66

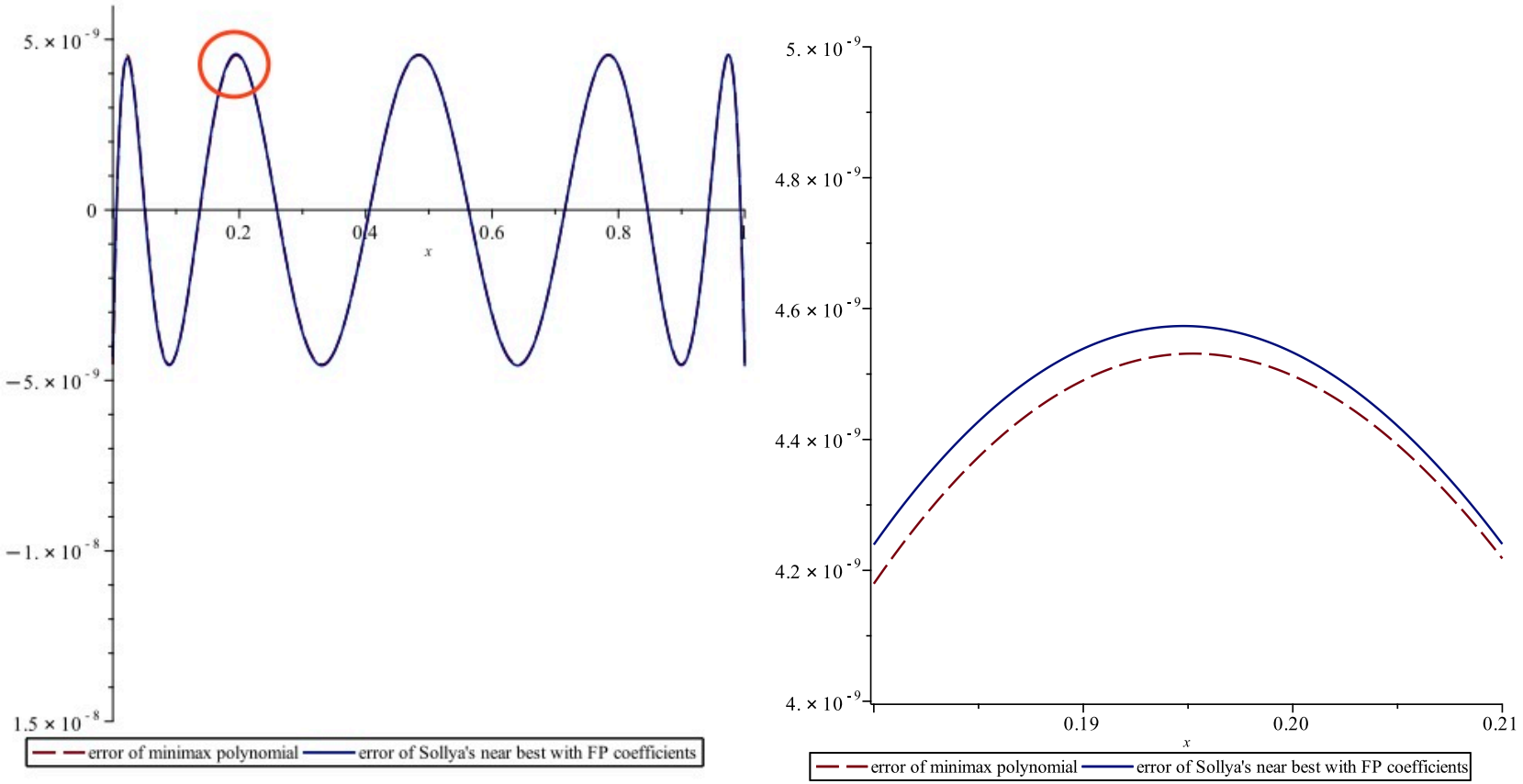
Near best polynomial approximations with FP coefficients

- algorithms that give near-best polynomials approximations under the constraint that the coefficients be FP numbers;
- more complex than Remez yet reasonable cost (and done once for all):

  Brisebarre and Chevillard, Efficient Polynomial L^∞-Approximations, 18th IEEE Symposium on Computer Arithmetic, 2007.
- still very active domain: various other constraints on coefficients, norms other than L^∞, rational approximations, taking into account the evaluation error when choosing the polynomial...
- Sollya software (Chevillard, Joldes, Lauter). Widely used for designing function libraries:

`https://www.sollya.org`

With our example of function $\log(1+x)$ in $[0,1]$



Sollya prompt: `P3 = fpminimax(log(1+x),9,[|24...|],[0;1],absolute);`

Error Sollya / Error minimax ≈ 1.012

$\rightarrow$ we recover almost all the loss due to the discretization of the polynomial.

Obtaining tight yet certain bounds on approximation errors

The algorithm in Sollya: paper by Chevillard, Harrison, Joldes, and Lauter, Theoretical Comp. Sci, April 2011.

- we wish to show $\|P - f\|_\infty < \alpha$;
- reduce the problem to

$$\|P - Q\|_\infty < \underbrace{\alpha - \delta}_{\beta}$$

  Q: very high-degree polynomial approximation to f (e.g. Taylor or Chebyshev) whose error bound δ is known analytically and such that $\delta \ll \alpha$;
- equivalent to showing positivity of the polynomials $P - Q + \beta$ and $Q - P + \beta$;
- positivity of a polynomial R?
  - in $\mathbb{R}$: express R as a sum of squares;
  - in $[a, b]$: change of variables

$$x = \frac{a + by}{1 + y}$$

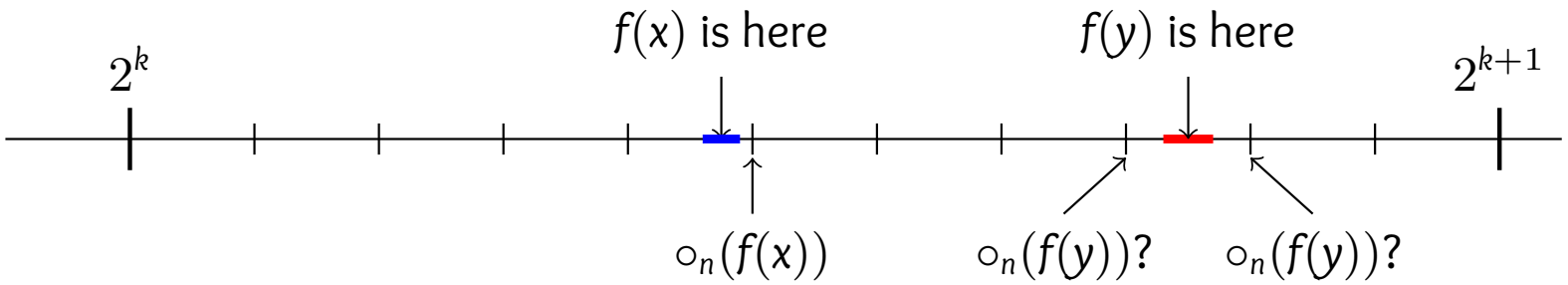
  - $R(x) > 0$ for $x \in [a, b] \Leftrightarrow$ the polynomial

$$S(y) := (1 + y)^n R\left(\frac{a + by}{1 + y}\right)$$

    is > 0 for $y \in \mathbb{R}$.

The Table Maker's Dilemma

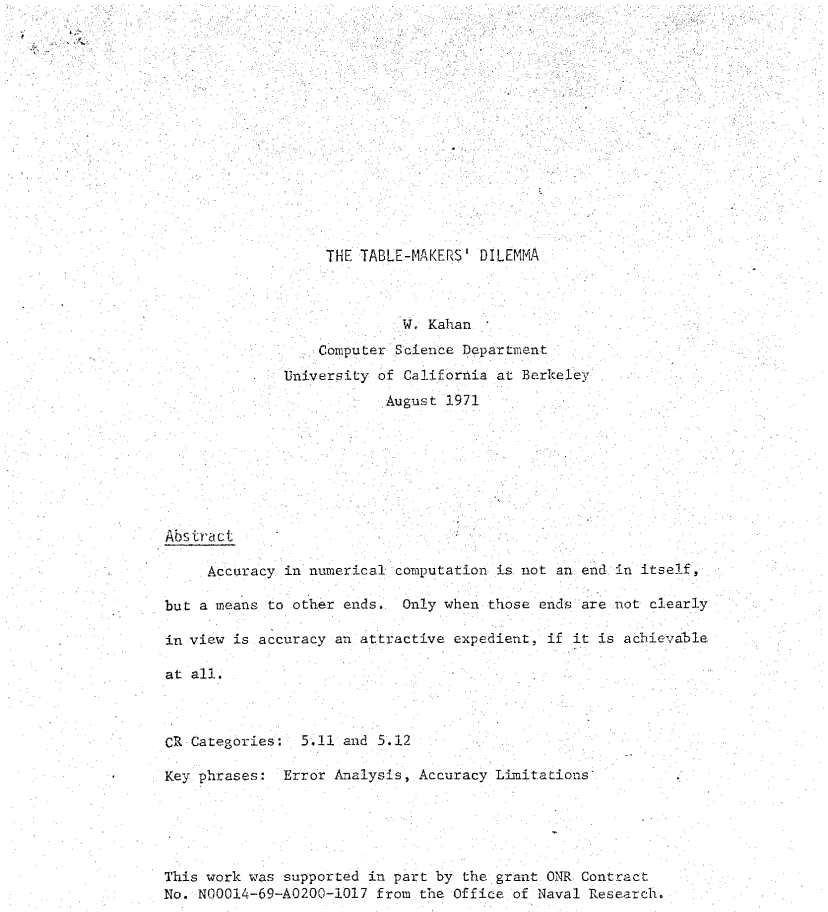
- Breakpoints: discontinuities of the rounding function. For $\circ_n$: midpoints of two consecutive FP numbers;
- Approximation of $f(x)$ to a certain accuracy $\rightarrow$ $f(x)$ is in an interval;
- Does this interval contain a breakpoint?



- No: we are done;
- Yes: the approximation is not sufficient to know $\circ_n(f(x))$. We can decide to continue with a more accurate approximation, but...
  - Does this process always stop?
  - And when?
- In 1971, W. Kahan coined the name Table Maker's Dilemma for this problem.

Early Steps

In 1971, W. Kahan coined the name Table Maker's Dilemma for a problem he deemed extremely difficult to solve.



THE TABLE-MAKERS' DILEMMA

W. Kahan
Computer Science Department
University of California at Berkeley
August 1971

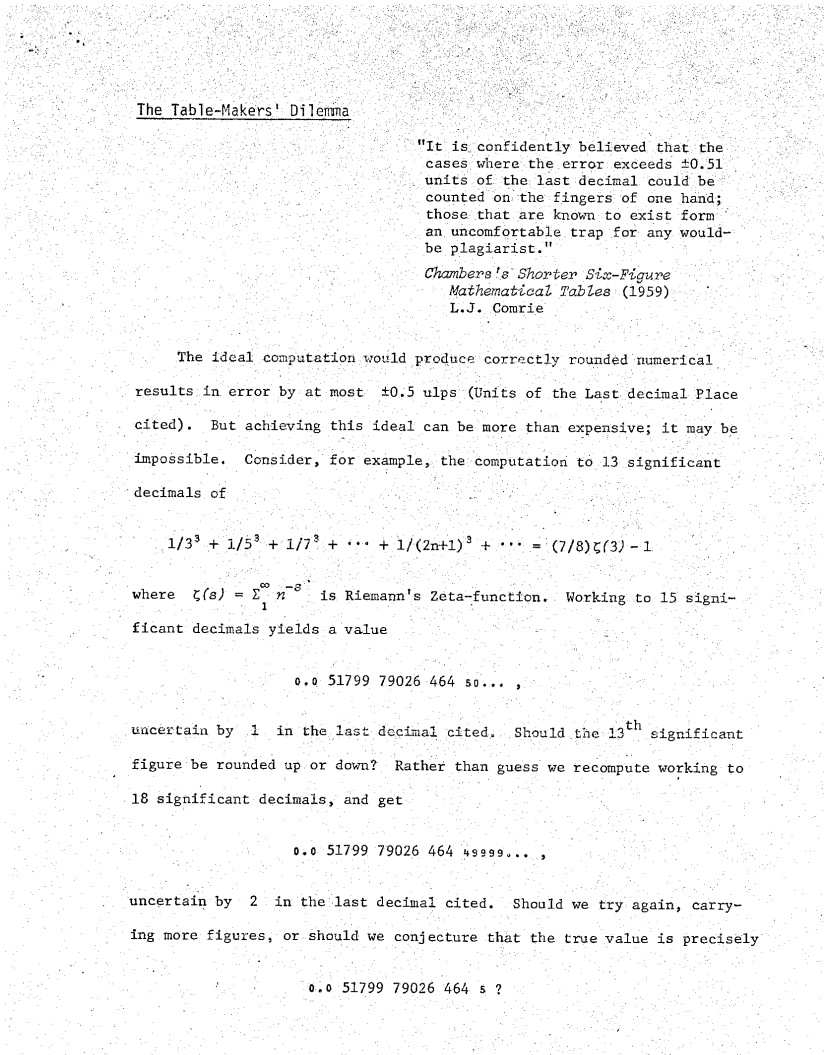
Abstract

Accuracy in numerical computation is not an end in itself, but a means to other ends. Only when those ends are not clearly in view is accuracy an attractive expedient, if it is achievable at all.

CR Categories: 5.11 and 5.12

Key phrases: Error Analysis, Accuracy Limitations

This work was supported in part by the grant ONR Contract No. N00014-69-A0200-1017 from the Office of Naval Research.



The Table-Makers' Dilemma

"It is confidently believed that the cases where the error exceeds ±0.51 units of the last decimal could be counted on the fingers of one hand; those that are known to exist form an uncomfortable trap for any would-be plagiarist."

Chambers's Shorter Six-Figure Mathematical Tables (1959)
L.J. Comrie

The ideal computation would produce correctly rounded numerical results in error by at most ±0.5 ulps (Units of the Last decimal Place cited). But achieving this ideal can be more than expensive; it may be impossible. Consider, for example, the computation to 13 significant decimals of

$$1/3^3 + 1/5^3 + 1/7^3 + \cdots + 1/(2n+1)^3 + \cdots = (7/8)\zeta(3) - 1$$

where $\zeta(s) = \sum_1^\infty n^{-s}$ is Riemann's Zeta-function. Working to 15 significant decimals yields a value

0.0 51799 79026 464 50... ,

uncertain by 1 in the last decimal cited. Should the 13th significant figure be rounded up or down? Rather than guess we recompute working to 18 significant decimals, and get

0.0 51799 79026 464 49999... ,

uncertain by 2 in the last decimal cited. Should we try again, carrying more figures, or should we conjecture that the true value is precisely

0.0 51799 79026 464 5 ?

That paper and others on the very useful website `https://www.arithmazium.org` (thanks to Jerome Coonen)

Early Steps

- Schulte & Swartzlander, *Exact rounding of Certain Elementary Functions*, ARITH'1993. Functions $1/x$, $\sqrt{x}$, 2^x, and $\log_2(x)$;
- Experimentally observed that a 51-bit approximation always allows providing a correctly rounded result in single precision.

Exact Rounding of Certain Elementary Functions

Michael Schulte and Earl Swartzlander
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Abstract

An algorithm is described which produces exactly rounded results for the functions of reciprocal, square root, 2^x, and $\log_2(x)$. Hardware designs based on this algorithm are presented for floating point numbers with 16 and 24 bit significands. These designs use a polynomial approximation in which coefficients are originally selected based on the Chebyshev series approximation and are then adjusted to ensure exactly rounded results for all inputs. To reduce the number of terms in the approximation, the input interval is divided into subintervals of equal size and different coefficients are used for each subinterval. For floating point numbers with 16 bit significands, the exactly rounded value of the function can be computed in 51 ns on a 20 mm^2 chip. For floating point numbers with 24 bit significands, the functions can be computed in 80 ns on a 98 mm^2 chip.

does not require exact for the elementary functions. This is largely due to a problem known as the Table Maker's Dilemma [1] [3]. This dilemma and a solution to it are discussed in Section 4.

Evaluating elementary functions with a method that produces exactly rounded results has several advantages. In addition to minimizing the error between the computed result and the exact value of the function, exact rounding also preserves several desirable properties of the functions such as symmetry and monotonicity [4]. Another advantage is that machines which have the same floating point format and ensure exact rounding will always produce the same results. This improves software portability and allows the correctness of floating point algorithms to be verified for a standardized system. Other advantages of having an industry standard for elementary functions are discussed in [5].

Table 1: Necessary Accuracy

	16 bit significand		24 bit significand	
Function	$Y(x)_{min}$	Bits	$Y(x)_{min}$	Bits
reciprocal	$2.33 \cdot 10^{-10}$	32	$3.56 \cdot 10^{-15}$	48
sqr. root	$2.33 \cdot 10^{-10}$	32	$3.56 \cdot 10^{-15}$	48
$\log_2(x)$	$3.69 \cdot 10^{-11}$	35	$6.11 \cdot 10^{-16}$	51
2^x $(x \geq 0)$	$2.37 \cdot 10^{-9}$	29	$6.21 \cdot 10^{-15}$	48

Early Steps

- MathLib/libultim (Ziv, IBM, 1991), functions with correct rounding. Idea of multi-step computation, with a rounding test at each step. The "ultimate" step is (much) too slow;
- 1990s: probabilistic heuristic (Gal-Bachelis; Dunham; Lefèvre-M.-Tisserand) $\rightarrow$ at that time, we don't know how to prove it, but efficient correct rounding seems possible.

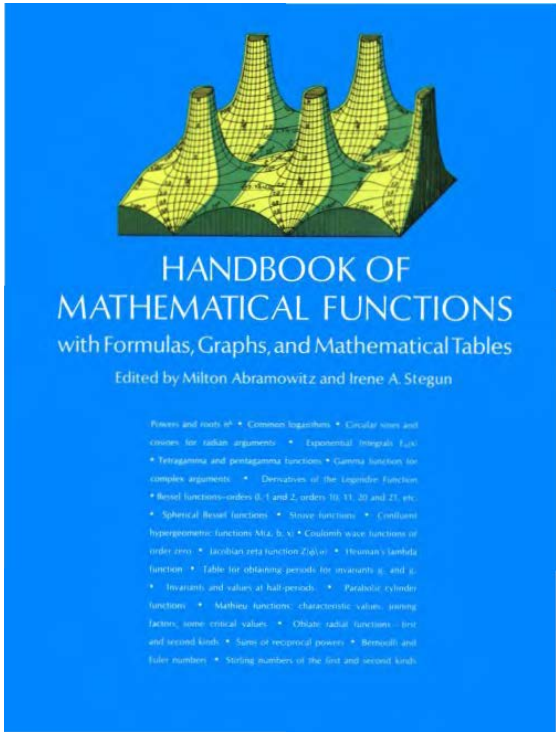
RLIBM Approach

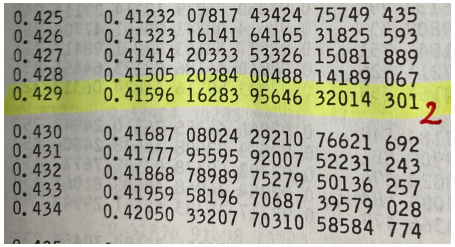
- Lim and Nagarakatte (Rutgers Univ.), paper in POPL 2022;
- correctly-rounded functions in precision p from rounded-to-odd precision $p+2$ functions;
- generation of polynomials minimizing (approximation + evaluation) error formalized as linear programming problem.

$\rightarrow$ RLIBM library of binary32/single precision functions.

Excellent approach for $w \leq 32$. I don't think that it scales to wider formats.

The Table Maker's Dilemma faced by... Table Makers





0.425	0.41232 07817 43424 75749	435
0.426	0.41323 16141 64165 31825	593
0.427	0.41414 20333 53326 15081	889
0.428	0.41505 20384 00488 14189	067
0.429	0.41596 16283 95646 32014	301 2
0.430	0.41687 08024 29210 76621	692
0.431	0.41777 95595 92007 52231	243
0.432	0.41868 78989 75279 50136	257
0.433	0.41959 58196 70687 39579	028
0.434	0.42050 33207 70310 58584	774

Abramowitz & Stegun, Handbook of Math. Functions (1964):

$\sin(0.429)$:

- Tabulated value:
  `0. 41596 16283 95646 32014 301`
- Exact value:

$$0.41596162839564632014301\ 500372652\cdots$$

$\sin(1.108)$:

- Tabulated value:
  `0. 89480 75718 42671 89157 146`
- Exact value:

$$0.89480757184267189157146\ 50062808\cdots$$

Plagiarism detection!

In Round-to-Nearest Mode (Default Mode)

Assuming $f(x) \in [2^k, 2^{k+1}[$, and approximating $f(x)$ with an error up to $2^{1+k-p-m}$, we will not know how to round $f(x)$ when

$$f(x) = 2^k \times \underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}} \overbrace{1000000\cdots000000}^{m \text{ bits}} xxx\cdots$$

or

$$f(x) = 2^k \times \underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}} \overbrace{0111111\cdots111111}^{m \text{ bits}} xxx\cdots ;$$

Largest value of m?

"Worst Cases" in Single Precision (32 bits, $p = 24$)

f	x (decimal)	significand of $f(x)$ (in binary)
exp	11615567×2^{-33}	$1.00000000010110001010111\underbrace{00000000000000000000000000}_{29}10\cdots$
sin	1258291×2^{-7}	$-1.01100011111101001011101\underbrace{11111111111111111111111111}_{30}01\cdots$
cos	12860426×2^{28}	$1.10000100101111101100010\underbrace{11111111111111111111111111}_{31}01\cdots$
arccos	4272401×2^{-34}	$1.10010010000011110110100\underbrace{00000000000000000000000000}_{33}10\cdots$
log	14192851×2^{53}	$1.10101001101000111111000\underbrace{00000000000000000000000000}_{34}11\cdots$
Γ	14583465×2^{-71}	$1.00100110100000100110011\underbrace{11111111111111111111111111}_{32}01\cdots$

$$\log\left(14192851 \times 2^{53}\right) = 13947384.5000000000564\cdots$$

Largest observed values of m: all close to 32... is this a coincidence?

Approximation with an Error $\leq 2^{-p-k+1}$ on the Significand of $f(x)$

$$\frac{f(x)}{2^{e_{f(x)}}} = \boxed{1.xxxx \cdots xxxxxxx}$$

p bits, followed by:

$k=2$	$k=3$	$k=4$
00	000	0000
01	001	0001
10	010	0010
11	011	0011
	100	0100
	101	0101
	110	0110
	111	0111
		1000
		1001
		1010
		1011
		1100
		1101
		1110
		1111

Proportion of cases leading to failure to round: $1/2$ $1/4$ $1/8$

k bits $\rightarrow$ failure proportion 2^{1-k}

Probabilistic Heuristic

- w-bit format $\longrightarrow$ 2^w possible inputs (fewer if function defined on restricted domain);
- We expect the number of FP numbers x for which we have a string of m bits of the form $01111\cdots111$ or $1000\cdots000$ after the p^{th} bit in the binary representation of the significand of $f(x)$ to be of the order of

$$2^{w+1-m}.$$

$\longrightarrow$ We expect the largest value of m to be about $w+1$.

1990s: Gal-Bachelis; Dunham; Lefèvre-M.-Tisserand

Rigorous justification for $m < p/3$: Brisebarre, Hanrot & Robert, *Exponential sums and correctly-rounded functions*, IEEE Trans. Computers, Dec. 2017.

Example: $\sin(x)$ in binary32 arithmetic, for $x \in [1, 2)$

k	exact ♯ of cases	heuristic
2	4194563	4194304
3	2095419	2097152
4	1049400	1048576
5	524244	524288
6	262343	262144
7	130989	131072
8	65586	65536
9	32838	32768
10	16682	16384
11	8309	8192
12	4056	4096
13	2066	2048
14	1057	1024
15	511	512
16	273	256
17	113	128
18	76	64
19	39	32
20	19	16
21	7	8
22	9	4
23	3	2
24	3	1
25	1	1
26	1	0

Example: $\exp(x)$ in binary32 arithmetic, for $x \in [1, 2)$

k	exact ♯ of cases	heuristic
2	4193270	4194304
3	2098427	2097152
4	1048454	1048576
5	524292	524288
6	262124	262144
7	131458	131072
8	65376	65536
9	32722	32768
10	16119	16384
11	8126	8192
12	4087	4096
13	2154	2048
14	1003	1024
15	514	512
16	231	256
17	135	128
18	45	64
19	32	32
20	18	16
21	10	8
22	6	4
23	4	2
24	0	1
25	0	1
26	1	0

Unfortunately, it is only a heuristic

- binary64/double precision, we "know" that the maximum value of m will be around 65 (less for functions such as $\log_2$ and 2^x because of correlations);
- binary128/quad precision, we "know" that the maximum value of m will be around 129...

... but we have no proof of that!!!

- Solutions?
  - ask the number theorists?
  - algorithmic calculation of the max value of m?

Some Results from Number Theory

- When the function is algebraic, "Liouville-style" reasoning on its minimal polynomial $\rightarrow$ upper bound of m of the order of $p \cdot d$ if this polynomial is of degree d;
- Hermite-Lindemann Theorem (1882):

  $$z \text{ algebraic} \neq 0 \Rightarrow e^z \text{ transcendental}$$

  $\rightarrow$ for $f = \exp, \log, \sin, \cos, \tan, \arctan, \arcsin, \arccos$, etc., if x is an FP number, $f(x)$ is never a breakpoint.
- Baker (1975): If we are given $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$, and $C = 16^{200}$, we have:

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

  Double precision ($w = 64$): intermediate approximations on

  $$\approx 10^{240} \text{ bits.}$$

More Results from Number Theory

- Nesterenko-Waldschmidt (1995):
  Let $\alpha = p/q$ and $\alpha' = p'/q' \in \mathbb{Q}$. Let $\theta, A, A', E \in \mathbb{R}^*$ such that

$$E \geq e, \quad A \geq \max(p, q, e), \quad A' \geq \max(p', q').$$

  We have

$$\begin{aligned}&\left|e^\theta - \alpha\right| + |\theta - \alpha'| \geq \\ &\exp\Big\{-211 \cdot \Big(\ln A' + \ln\ln A + 2\ln(E \cdot \max\{1, |\theta|\}) + 10\Big) \\ &\cdot \Big(\ln A + 2E|\theta| + 6\ln E\Big) \cdot \Big(3.7 + \ln E\Big) \cdot \Big(\ln E\Big)^{-2}\Big\}.\end{aligned}$$

  Gives results for exp, log, cos, sin, tan, arctan, etc.
  $\rightarrow$ Intermediate approximations on a few million bits;

- Khémira-Voutier (2011): Intermediate approximations on a few tens of thousands of bits.

  (Brisebarre, Hanrot, M., and Zimmermann, *Correctly-rounded evaluation of a function: why, how, and at what cost?*, ACM Computing Surveys, 2026)

It's Still Frustrating...

- Intermediate approximations on a few tens of thousands of bits: feasible, but expensive...

Cap'tain speaking. Our landing is delayed because the computer needs to calculate a cosine.

- And yet we "know" that the largest value of m is about w...

Find a way to compute this largest value of m?

Exhaustive tests: 2^w values to test. Trivial for $w \leq 32$, accessible to humanity but at unreasonable cost for $w = 64$, unthinkable for larger values.

Strategy Adopted for $w \geq 64$

- Split the initial domain into N (small) subdomains;
- In each subdomain, polynomial approximation of f;
- Trade-off $N \leftrightarrow$ degree of polynomials;
- Degree 1: Lefèvre's PhD (2000), first "worst cases" for double precision (initially, exponential and log over the entire domain):

$$2^{43} \cdot \log\left(3119607361246427 \cdot 2^{627}\right) = 4136624566035640.5000000000000000000122\cdots$$

  - Cost $\approx \mathcal{O}\left(2^{w-p/3}\right)$;
  - Allowed the construction of the "proof of concept" CRLibm library;
  - Still very expensive (expensive in double precision, unthinkable in quad precision).

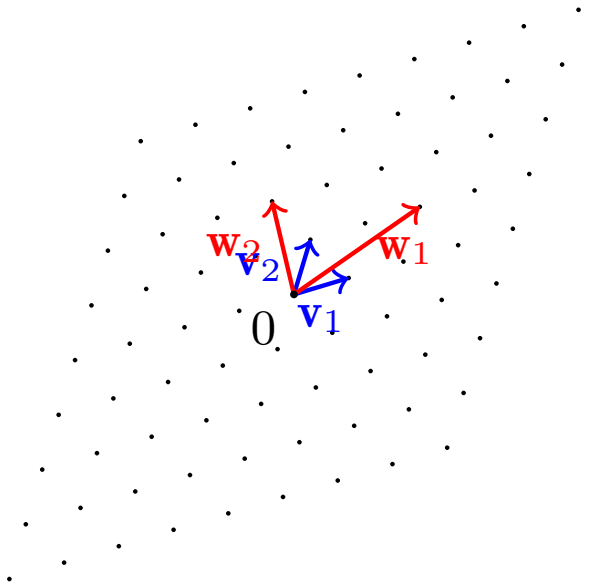
- Higher degrees: Stehlé-Lefèvre-Zimmermann (SLZ) algorithm.

Just two slides on Euclidean lattices

- $b_1, b_2, \ldots, b_d$ linearly independent vectors in $\mathbb{R}^n \rightarrow$ generated lattice

$$\mathcal{L} = \sum_{i \leq d} k_i \cdot b_i, \quad k_i \in \mathbb{Z};$$

- very useful concept (e.g. in cryptography);



- Difficult problem: find a small vector in $\mathcal{L}$.

Just two slides on Euclidean lattices

- Difficult problem: find a small vector in $\mathcal{L}$;
- LLL Algorithm (Lenstra, Lenstra and Lovász, 1982) $\rightarrow$ base $(c_1, c_2, \ldots, c_d)$ of reasonably small vectors;
- delay: linear function of the dimension d of the lattice;
- In particular, $||c_1|| \leq 2^{(d-1)/2}\lambda(L)$, where $\lambda(L)$ is the smallest norm of a nonzero element of $\mathcal{L}$.

Finding Worst Cases... or Not. The SLZ Algorithm

- reduction to the case $f : \left[\frac{1}{2}, 1\right] \to \left[\frac{1}{2}, 1\right]$;
- subdomain $\left[\frac{X_0 - a}{2^p}, \frac{X_0 + a}{2^p}\right]$ of $\left[\frac{1}{2}, 1\right]$, f replaced by polynomial F;
- we look for possible solutions of

$$2^p \cdot F\left(\frac{X}{2^p}\right) - \frac{1}{2} - z = 0 \mod 1;$$

  with $X \in \mathbb{Z}$, $|X| \leq a$ and $|z| \leq \frac{1}{M}$, M large integer (typically 2^p or 2^w);
- we eliminate the denominators

$$P(u, v) = 0 \mod R,$$

  $P \in \mathbb{Z}[X]$, $(u, v) \in \mathbb{Z}^2$, $|u| < U$, $|v| < V$.

Finding Worst Cases... or Not. The SLZ Algorithm

$$P(u,v) = 0 \mod R, \tag{1}$$

$P \in \mathbb{Z}[X]$, $(u,v) \in \mathbb{Z}^2$, $|u| < U$, $|v| < V$.

- Coppersmith's method: For a given parameter α, we build a family

$$P_{i,j,k}(u,v) = R^{\alpha-i} P(u,v)^i u^j v^k$$

  $(i \leq \alpha, j \leq j_{max}, k \leq k_{max})$;

- (1) implies

$$P_{i,j,k}(u,v) = 0 \mod R^\alpha.$$

  If we find a linear combination (with integer coefficients) $Q = \sum q_{m,n} u^m v^n$ of the $P_{i,j,k}$ such that

$$\sum |q_{m,n}| \cdot U^m V^n < R^\alpha,$$

  then (1) $\Rightarrow Q(u,v) = 0$ in the integers.

- Finding small vectors is precisely what LLL knows how to do!
- Two such polynomials Q_1 and Q_2: solve the system $Q_1(u,v) = 0, Q_2(u,v) = 0$.

D. Stehle, V. Lefevre & P. Zimmermann, *Searching worst cases of a one-variable function using lattice reduction*, IEEE Trans. on Computers, March 2005

Some Remarks on the SLZ Algorithm

- Cost for an input exponent value:[1]

$$\text{Poly}\,(p+\log(M))\cdot 2^{\frac{p^2}{p+\log(M)}};$$

- Worst case search: $M \approx 2^p$ (probabilistic heuristic) $\rightarrow$ cost in $2^{p/2}$.

$\rightarrow$ Worst cases now known[2] in double precision for

exp, log, exp2, log2, exp10, log10, log1p, log2p1, log10p1, expm1, exp2m1, exp10m1, rsqrt, sin, cos, tan, sinpi, cospi, tanpi, acospi, asinpi, atanpi, cosh, sinh, tanh, acosh, asinh, atanh, acos, asin, atan, cbrt, erf, erfc.

$\rightarrow$ CORE-MATH library (P. Zimmermann and colleagues): all 32 and 64-bit functions of the C standard. Very efficient. Almost as fast as Intel or GNU libc functions *not* correctly rounded;

- Still complicated for quadruple precision.

---

[1] D. Stehlé, On the randomness of bits generated by sufficiently smooth functions, Algorithmic Number Theory. ANTS 2006.

[2] `https://www.reliable-computing.org/correctrounding.html`

Hopes for Quadruple Precision?

- Cost one input exponent:

$$\text{Poly}\left(p + \log(M)\right) \cdot 2^{\frac{p^2}{p+\log(M)}};$$

- Complicated for quadruple precision... but if we accept $M \gg 2^p$: results like *with intermediate results on $6p$ (or p^2) bits, we can always round correctly*, become feasible;
- Problem:
  - Usual version ($M \approx 2^p$): "worst cases" that we can test;
  - "Large M" version: complex algorithm, highly optimized program, which after days of computation, just answers "yes" or "no"... how much trust?

→ Absolute necessity of formal proof!

- Formally proving the algorithm, the program: little hope;
- Certificate-based approach based on Q_1 and Q_2.

Hot News: November 2025

At the RAIM2025 conference, Paul Zimmermann presented the worst cases for sin/cos functions in double precision:

$$2^{56} \cdot \cos\left(7740461397073975 \times 2^{865}\right)$$
$$= -7254576615099225.4999999999999999989629326\cdots,$$

(for round-to-nearest, gives $m = 63$)

$$2^{53} \cdot \sin\left(6381956970095103 \times 2^{797}\right)$$
$$= 9007199254740991.9999999999999999999990105805\cdots$$

(for other rounding functions, gives $m = 69$)

Watch Tue Ly's talk Computing hard-to-round Cases of sin, cos, tan in Double Precision this afternoon!

Double Precision: Largest Error (in ulps), February 2023 and February 2026

For $x \in \mathbb{R}$, ulp (x) = distance between the 2 FP numbers surrounding x.
f correctly rounded: error $\leq \frac{1}{2}$ ulp $(f(t))$.
25 years ago, 5-6 ulps was considered honorable.

	GNU libc 2.37	GNU libc 2.43	IML 2023.0.0	IML 2025.0.0	LLVM 21.1.8
acos	0.523	0.523	0.531	0.531	0.500
acosh	2.25	0.500	0.509	0.509	3.22
asin	0.516	0.516	0.531	0.531	0.500
asinh	1.92	0.500	0.507	0.507	2.05
atan	0.523	0.523	0.528	0.528	1.00
atanh	1.81	0.500	0.507	0.507	2.50
cos	0.516	0.516	0.518	0.518	0.500
cosh	1.93	1.93	0.516	0.516	1.42
erfc	5.19	0.500	0.505	0.827	5.85
exp	0.511	0.511	0.530	0.530	0.500
log	0.520	0.520	0.518	0.518	0.500
sin	0.516	0.516	0.518	0.518	0.500
pow	0.523	0.523	1.73	1.73	Inf

Source: Gladman et al., *Accuracy of Mathematical Functions in Single, Double, Extended Double and Quadruple Precision*, `https://inria.hal.science/hal-03141101`

Correctly Rounded Function Libraries

- LibUltim (IBM, late 1990s), A. Ziv. Double precision. Worst cases for rounding not known $\rightarrow$ large overestimation of required precision ($m = 768$). Worst time/average time ratio: 6500;
- CRLibm (Lyon, 2000-2006). Double precision. Two steps. Worst time/average time ratio: 6.6. Knowledge of worst cases (except trigonometric functions with large arguments). *Proof of concept.* ;
- RLIBM (Rutgers University, 2020-2026), S. Nagarakatte. 16 and 32-bit formats ($\rightarrow$ exhaustive tests possible, linear programming to build approximations);
- CORE-MATH (Nancy, 2020-2026), P. Zimmermann. All 32 and 64-bit functions of the C23 standard. Almost as fast as Intel or GNU libc functions *not* correctly rounded.

$\rightarrow$ Goal: integration into other libraries (*is happening:* GNU libc, AMD, LLVM...);

- Intel is moving too: recent paper *Correctly Rounded Functions For Vector Applications: A Performance Study* by Anderson et al. (2026).

Further reading (only papers published in 2026)

- Survey paper:

  Brisebarre, Hanrot, M., and Zimmermann, *Correctly-rounded evaluation of a function: why, how, and at what cost?,* ACM Computing Surveys. Vol. 58 No 1, 2026.

- more specialized papers:

  Anderson, Cornea, Stepin, and Panu, *Correctly rounded functions for vector applications: a performance study*, `https://arxiv.org/pdf/2605.15547`,

  Brisebarre, Hubrecht, Lauter, M., and Ruiz-Rohena, *Correctly rounded vector implementation of the exponential function in binary64 arithmetic,* ARITH 2026.

  Brisebarre and Hanrot, *Integer points close to a transcendental curve: an algorithmic approach*, Annales Henri Lebesgue, to appear, `https://arxiv.org/abs/2606.04858`,

  Lefèvre, Ly, and Zimmermann, *Computing hard-to-round Cases of sin, cos, tan in Double Precision,* ARITH 2026.

  Zimmermann, *The GNU libc atanh is Correctly Rounded,* ARITH 2026.

Thank You!