A problem left unsolved by Jean-Michel

Paul Zimmermann, INRIA
(joint work with Tue Ly, Google)

RAIM
Lyon, November 6, 2025

A small problem resists

A problem left unsolved by Jean-Michel

Credit `https://www.celticyourte.fr`

"Correctly Rounded Evaluation of a Function: Why, How, and at What Cost?", N. Brisebarre, G. Hanrot, **J.-M. Muller**, P. Zimmermann, ACM Computing Surveys, 2025:

> *[...] in binary64 arithmetic, the bad cases for the sine and cosine functions are not known for input arguments larger than* 2^{11}.
>
> *[...] the table maker's dilemma can be considered as solved for the binary64 format for univariate functions with the possible exception of trigonometric functions of large arguments*

Hard-to-round cases

$$\sin(\texttt{0x1.fe767739d0f6dp-2}) = 0.0\underbrace{111...101}_{53}1\underbrace{111...111}_{65}0000101...$$

The known algorithms (Lefèvre, SLZ) for searching hard-to-round cases assume that $f(x)$ and $f(\text{nextabove}(x))$ are close.

For the largest binade $[2^{1023}, 2^{1024})$, this is not the case. For $x = 2^{1023}$,

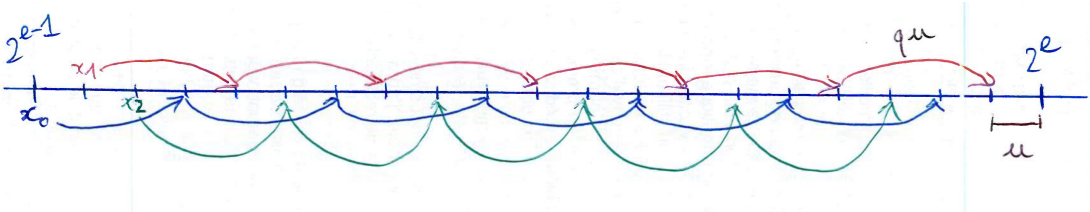
$$\sin(x) \approx 0.563127779850884$$

$$\sin(\text{nextabove}(x)) \approx -0.976171771886648$$

State of the art

Worst Cases of a Periodic Function For Large Arguments. Guillaume Hanrot, Vincent Lefèvre, Damien Stehlé, Paul Zimmermann, ARITH'18, 2007.

Idea: in a given binade, where $u := \mathrm{ulp}(x)$, consider arithmetic progressions $x, x + qu, x + 2qu, \ldots$, where qu mod 2π is small.



Example: for the largest binade $[2^{1023}, 2^{1024})$, $u = 2^{971}$, we can take $q = 15106909301$, where $\tau := qu \bmod 2\pi \approx 4.41 \cdot 10^{-13}$. For $x = 2^{1023}$,

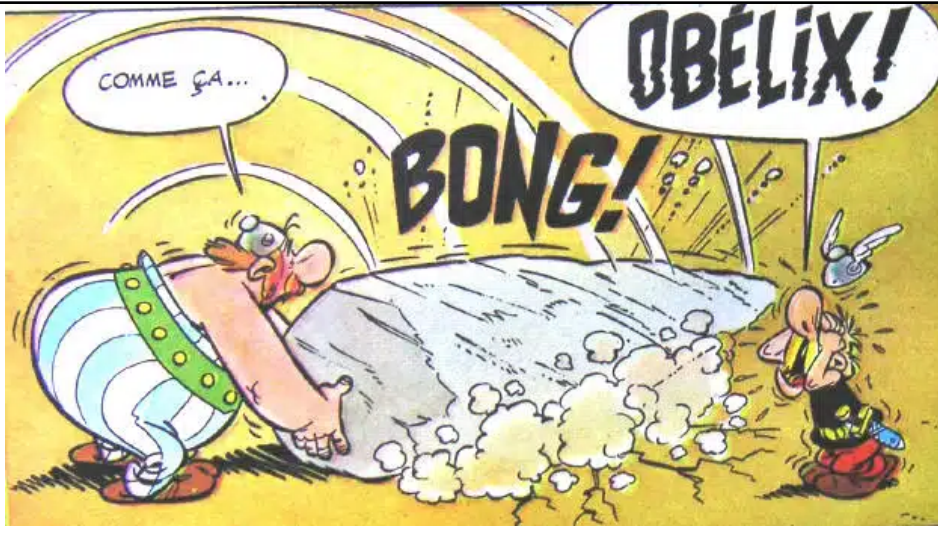
$$\sin(x) \approx 0.563127779850884$$

$$\sin(x + qu) \approx 0.563127779850519$$

We can then use classical algorithms on the q arithmetic progressions of size about $300,000$. This algorithm was implemented in the BaCSeL software tool. However, the BaCSeL code has a large overhead.

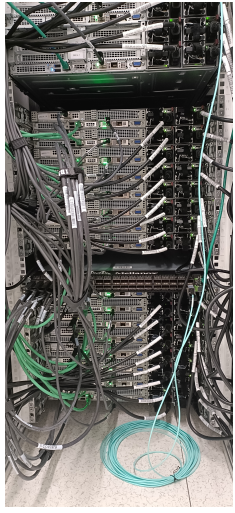
In 2023, a full search was done with BaCSeL for $\sin x$ in the largest binade $[2^{1023}, 2^{1024})$. It took a total of 102 days on one node (Intel Xeon Gold 6130 at 2.1Ghz). A full search would require about 300 years on one such node.

The brute force approach



Credit https://www.letournepage.com/

Our menhir



The grdix cluster (cf Antoine Jégo's talk)

16 nodes

each node is an AMD EPYC 9754
running at 2.7Ghz with 256 cores
(512 with hyper-threading)

if we can check one value in one cycle,
we can process a binade (2^{52} values)
in less than one hour on one node

Our proposed brute-force algorithm

Use a degree-3 Taylor approximation with remainder ($\tau = qu \bmod 2\pi$):

$$\sin(x + i\tau) = \sin x + i\tau \cos x - \frac{1}{2}(i\tau)^2 \sin x - \frac{1}{6}(i\tau)^3 \cos x + \frac{1}{24}(i\tau)^4 \sin \xi$$

Can we make the initialization step fast?

Can we make the search step fast?

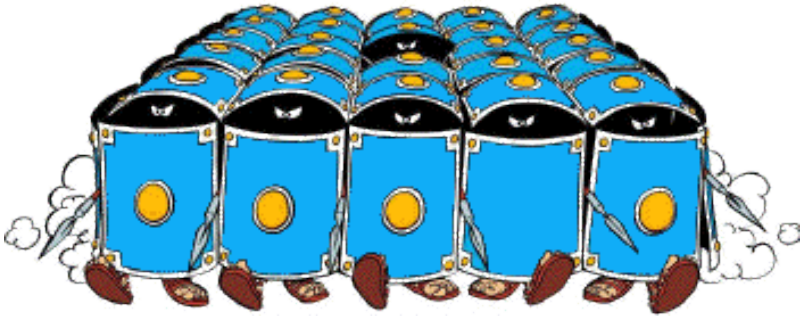
New algorithm in image

Let $x_i = 2^{e-1} + iu$, where $u = 2^{e-53}$.

Process the arithmetic progression $x_0, x_0 + qu, x_0 + 2qu, ...$

Then $x_1, x_1 + qu, x_1 + 2qu, ...$

Finally $x_{q-1}, x_{q-1} + qu, x_{q-1} + 2qu, ...$



Credit `https://histoire-geo-ensemble.overblog.com`

Taylor approximation (1/2)

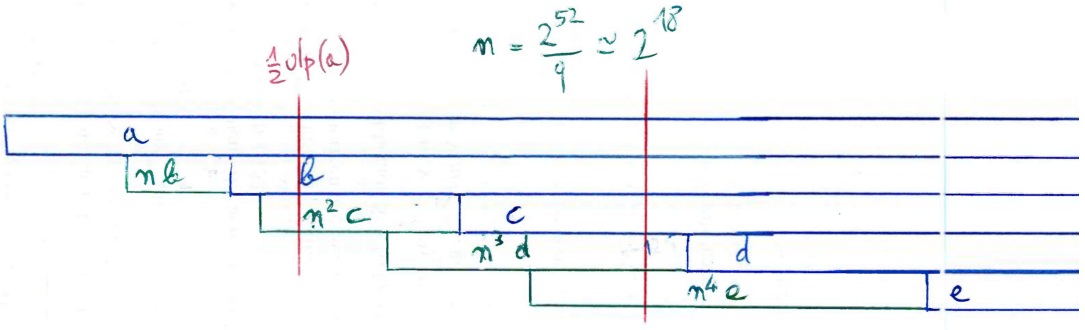
In the arithmetic progression $x, x+qu, x+2qu, \ldots$, we have the Taylor expansion, where $\tau = qu \bmod 2\pi$:

$$\sin(x+i\tau) = \underbrace{\sin x}_{a} + \underbrace{i\tau\cos x}_{ib} - \underbrace{\frac{1}{2}(i\tau)^2 \sin x}_{i^2 c} - \underbrace{\frac{1}{6}(i\tau)^3\cos x}_{i^3 d} + \underbrace{\frac{1}{24}(i\tau)^4 \sin\xi}_{i^4 e}$$

$$|\sin(x+i\tau) - (a+ib+i^2c+i^3d)| \le i^4 e$$

For the largest binade: $q = 15106909301$, $\tau \approx 4.41 \cdot 10^{-13}$. For $x = 2^{1023}$, we get $a \approx 2^{-1}$, $|b| \approx 2^{-42}$, $c \approx 2^{-84}$, $|d| \approx 2^{-126}$, $e \approx 2^{-170}$.

Graphical view



Taylor approximation (2/2)

$$|\sin(x + i\tau) - (a + ib + i^2 c + i^3 d)| \leq i^4 e$$

To search hard-to-round cases with at least m identical bits after the round bit:

- [initialization] evaluate efficiently a, b, c, d, e
- [search] for each i, check whether $a + ib + i^2 c + i^3 d$ is at distance less than $2^{-m}\mathrm{ulp} + i^4 e$ from a 54-bit floating-point number

Initialization

Approximate $a = \sin x$, $b = \tau \cos x$, $c = \frac{\tau^2}{2} \sin x$, $d = \frac{\tau^3}{6} \cos x$, bound $|e| = |\frac{\tau^4}{24} \sin \xi|$.

$\tau = qu \bmod 2\pi$ is fixed for a given binade: we have to compute it only once.

However, we have to compute q different values of $\sin x, \cos x$, one for each arithmetic progression: $x = ju$, with $2^{52} \leq j < 2^{52} + q$.

Define $(s_j, c_j) = (\sin(ju), \cos(ju))$.

Compute $(s_1, c_1) = (\sin u, \cos u)$ (once for each binade).

Use then "binary exponentiation" to obtain $(s_{2n}, c_{2n}) = (2s_n c_n, c_n^2 - s_n^2)$ (doubling), and $(s_{2n+1}, c_{2n+1}) = (s_1 s_{2n} + c_1 c_{2n}, c_1 c_{2n} - s_1 s_{2n})$ (multiply).

Thus each pair (s_j, c_j) can be obtained in 52 "doublings" and on average 26 "multiplies".

Computations in fixed-point arithmetic, with 320 bits (5 words of 64 bits).

Search

We have now 320-bit approximations of a, b, c, d, e and we are looking for i, $0 \leq i \leq n = \lfloor 2^{52}/q \rfloor$, such that:

$$a + ib + i^2c + i^3d$$

is at distance less than $2^{-m}\mathrm{ulp} + i^4e$ from a 54-bit number.

- define $f = 2^{-m}\mathrm{ulp} + n^4e$
- assume there is no binade change for $a + ib + i^2c + i^3d$

If a, b, c, d, f are scaled appropriately, it suffices to look at the "fractional" bits. Let $a' = \mathrm{frac}(2^k a)$, $b' = \mathrm{frac}(2^k b)$, ...

$$|a' + ib' + i^2c' + i^3d'| \bmod 1 \leq f' \qquad \text{(centered mod)}$$

Using 64-bit integers is enough

The low-order bits of a', b', c', d' can be neglected.
Approximate $a' = A \cdot 2^{-64}$ with A a 64-bit integer, same for b', c', d', f'.
The main equation translates to:

$$|A + iB + i^2C + i^3D| \bmod 2^{64} \leq G \qquad \text{(centered mod)}$$

with G taking into account the error bound f' and the rounding error in A, B, C, D.
This translates to (cf Lefèvre's PhD thesis) with A replaced by $A + G$:

$$A + iB + i^2C + i^3D \bmod 2^{64} \leq 2G.$$

The table of differences method

How to efficiently evaluate $A + iB + i^2C + i^3D \bmod 2^{64}$ for $0 \leq i < n$?

Initialize $\alpha, \beta, \gamma, \delta = A, B + C + D, 2C + 6D, 6D \bmod 2^{64}$

for i from 0 to n do

  if $\alpha \leq 2G$ then check $x + iq$

  $\alpha \leftarrow \alpha + \beta \bmod 2^{64}$

  $\beta \leftarrow \beta + \gamma \bmod 2^{64}$

  $\gamma \leftarrow \gamma + \delta \bmod 2^{64}$

Using `int64_t` in the C language, we just write `A += B`, which performs the reduction modulo 2^{64} (wrap-around trick).

Batch computation

Assume we have computed $(s_j, c_j) = (\sin(ju), \cos(ju))$.

We have $(s_{j+1}, c_{j+1}) = (s_1 c_j + c_1 s_j, c_1 c_j - s_1 s_j)$.

For a batch of 128 consecutive values of j, we reduce the initialization cost from $128 \cdot 78$ doublings/multiplies to $78 + 127$, i.e., a speedup of about 50.

Using SIMD instructions

We can search in parallel in several progressions with parameters A_k, B_k, C_k, D_k using vector instructions.

For example, with AVX512, we can deal with 8 progressions in parallel.

We only need:

- an instruction to add two vectors of 8 words of 64 bits, each element being added modulo 2^{64} (no carry between two elements)
- an instruction to check if a vector contains a element $\leq 2G$

Hard-to-round cases of $\sin x$ for $x \geq 2^{10}$

Our SIMD program took from 2.1 hours to 2.3 hours per binade (instead of 2.6 days with BaCSeL) on a grdix node (about 2.4 cycles per value). We found 1,048,756 hard-to-round cases with at least 43 identical bits after the round bit.

x	m	$\sin x \approx$
0x1.e009c53148be1p+991	64	1.0
0x1.cfe482285f8edp+860	63	-1.0
0x1.6ac5b262ca1ffp+849	68	1.0
0x1.db41f3cb71d7bp+680	63	1.0
0x1.4c96c11134d36p+577	63	-1.0
0x1.e7e44a78ac18cp+197	63	-1.0
0x1.230280c47f5c1p+136	63	0.270
0x1.504cac51f1eafp+131	64	-1.0
0x1.b951f1572eba5p+23	65	-1.0

Largest binade: 1038 hard-to-round cases (exactly those found by BaCSeL in 2023).

Hard-to-round cases of $\cos x$ for $x \geq 2^{10}$

We found 1,049,705 hard-to-round cases with ≥ 43 identical bits after the round bit.

x	m	$\cos x \approx$
0x1.5afb7107105d9p+1006	62	0.918
0x1.e009c53148be1p+992	62	-1.0
0x1.b7fe89bf86037p+917	62	-0.101
0x1.6ac5b262ca1ffp+852	62	1.0
0x1.6ac5b262ca1ffp+851	64	1.0
0x1.6ac5b262ca1ffp+850	66	-1.0
0x1.1fa76750679fcp+285	62	0.225
0x1.504cac51f1eafp+132	62	-1.0
0x1.b951f1572eba5p+24	63	-1.0

Hard-to-round cases of $\tan x$ for $x \geq 2^{10}$

Our SIMD program took from 3.0 hours to 6.5 hours on a grdix node (mean 4.4 hours, about 5 cycles per value). We found 1,045,244 hard-to-round cases with at least 43 identical bits after the round bit.

x	m	$\tan x \approx$
0x1.20e3e80d2b617p+990	61	-1.15
0x1.94bb90326441ap+953	61	-0.32
0x1.52042b55571c6p+952	60	-1.83
0x1.fe6e530194af6p+681	62	5.00
0x1.8b4c4b528e351p+578	60	-1.59
0x1.57237795e9208p+324	61	0.68

Conclusion

- we completed a search among about 2^{62} values for each function
- maybe degree 3 would have been enough?
- does Lefèvre's algorithm combined with our initialization yield a speedup?
- the hard-to-round cases will be available from the CORE-MATH git repository
- found no failure with the LLVM and CORE-MATH sin/cos/tan functions

The TMD is now fully solved for univariate binary64 functions!

Menhirs are still useful!

Credit `https://www.decideurs-magazine.com`

Lefèvre's algorithm

New results on the distance between a segment and Z2. Application to the exact rounding, Vincent Lefèvre, ARITH 2005.

Algorithm based on the three-gap theorem to find the smallest integer $r \in [0, N-1]$ such that $(b - ra) \bmod 1 < d_0$.

Complexity $o(n)$ for an arithmetic progression of length n.

Degree-1 approximation instead of degree-3: requires larger q thus smaller length n of arithmetic progressions.

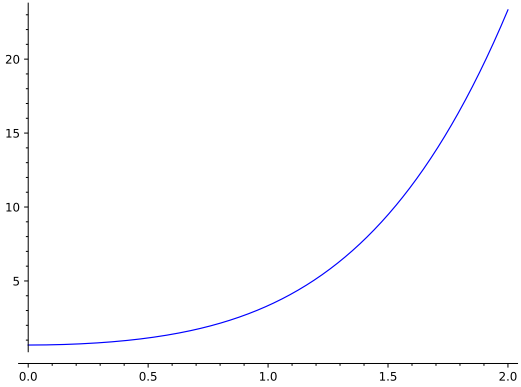
Not compatible with SIMD speedup?

Preliminary experiments for $\sin x$: 31% faster in "good" binades ($e = 996, n = 1474$), 7.2 times slower for "median" binades ($e = 920, n = 97$), 84 times slower for "bad" binades ($e = 793, n = 8$).

Much more dependent of length of arithmetic progressions.

The tangent function

$$\begin{aligned}\tan(x+h) &= \tan x + h(1+\tan^2 x) + h^2 \tan x(1+\tan^2 x) \\ &+ \frac{1}{3}h^3(1+\tan^2 x)(1+3\tan^2 x) + \frac{1}{3}h^4 \tan\xi(1+\tan^2\xi)(2+3\tan^2\xi)\end{aligned}$$



growth of the ratio (error term)/($\tan x$) with respect to $\tan x$