Computing Hard-To-Round Cases of Sin, Cos, Tan in Double Precision

Vincent Lefèvre[1], Tue Ly[2], Paul Zimmermann[3]

ARITH 2026 - Fulda, Germany

[1] Inria, ENS de Lyon, CNRS
[2] Google LLC
[3] Université de Lorraine, CNRS, Inria, LORIA

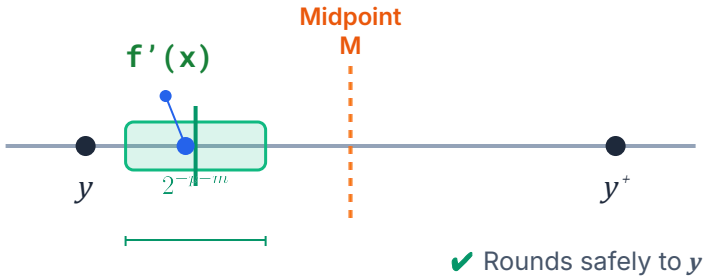
What is a hard-to-round case of a function (w.r.t. RN)

Given a target precision `p`, suppose we can compute an approximation `f'(x)` of `f(x)` up to precision `p + m`:

$$|f(x) - f'(x)| \leq \mathrm{ulp}(RN(x)) \cdot 2^{-m-1}$$

Easy-to-Round Case

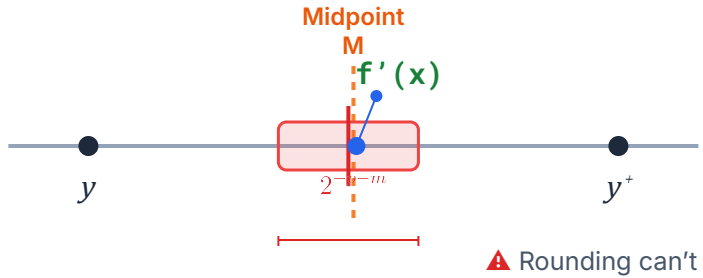
Interval does not cross the midpoint; rounding is certain.



✔ Rounds safely to y

Hard-to-Round Case

Interval straddles the midpoint; rounding is ambiguous.



⚠ Rounding can't be decided.

Hard-to-round cases and correct rounding

Lemma 1 (trivial)

Let $TMD(m)$ be the set of all $(p + m)$-bit hard-to-round cases of f. Let f' be an approximation of f such that:

- It can be proved (analytically, formally) that for all x:

$$|f(x) - f'(x)| \leq ulp(RN(x)) \cdot 2^{-m-1}$$

- For all $x \in TMD(m)$,

$$RN(f'(x)) = RN(f(x))$$

Then $RN(f'(x))$ is a correctly rounded for all x.

Hard-to-round cases for double precision unary functions

- V. Lefèvre, J.-M. Muller (2001):
  - e^x, 2^x, log, log2, arcsin, arccos, arctan, sinh, arcsinh
  - sin(x) : $|x| \leq 21'059/8'192$
  - cos(x) : $|x| \leq 12'867/8'192$
  - tan(x) : $|x| \leq \arctan(2)$
  - cosh(x) : $|x| \leq 32$
- Prior to our work:
  - 10^x, log10
  - expm1, exp2m1, exp10m1, log1p, log2p1, log10p1,
  - cosh, tanh, arccosh, arctanh
  - All the *pi variants
  - sin(x), cos(x) : $|x| \leq 2^{11}$
  - tan(x) : $|x| \leq 10\pi$

How hard is it to find worst-cases?

Lemma 2

Let x be a double precision number. If $\mathrm{RN}_{53+m}(\sin(x)) = \mathrm{RN}_{54}(\sin(x))$,

then $x \in \mathrm{TMD}(m)$.

- Using MPFR to compute the condition in Lemma 2, it takes ~ 12’000 clock cycles for each input.
- For the current best AMD desktop CPU, Ryzen 9950X 16 cores/32 threads @5GHz, it will takes ~ 96 million hours (or ~ 11k years).

How can we make it faster?

Generally speaking, we could employ the following strategies individually or together:

- Perform one or more faster checks of hard-to-round necessary conditions with low probability of passing through:

  `x → Check1(x) → Check2(x) → Lemma 2 check → hard-to-round cases.`

- Group multiple input `x` together to share some computational cost, and share intermediate results.
- Parallelism: multi-threading + SIMD (+ more machines).

Hard-to-round cases for trigonometric functions

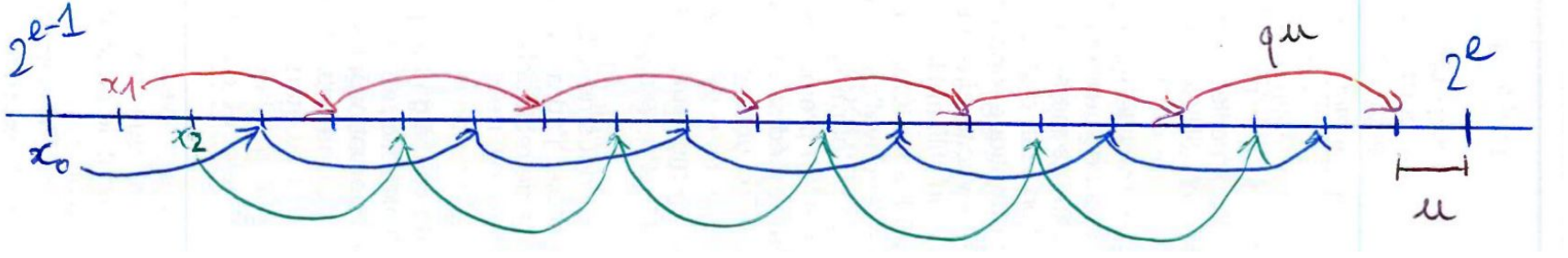
- Lefèvre, SLZ algorithms:
  - group the inputs into subintervals
  - translate the problem into finding small integer vectors, reducing the search space
  - require that `f(x)` and `f(nextabove(x))` are close.
- It is **NOT** the case for trigonometric functions with large inputs. For example, with $x = 2^{1023}$,

  $\sin(x) \approx 0.563127779850884$

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

State of the art

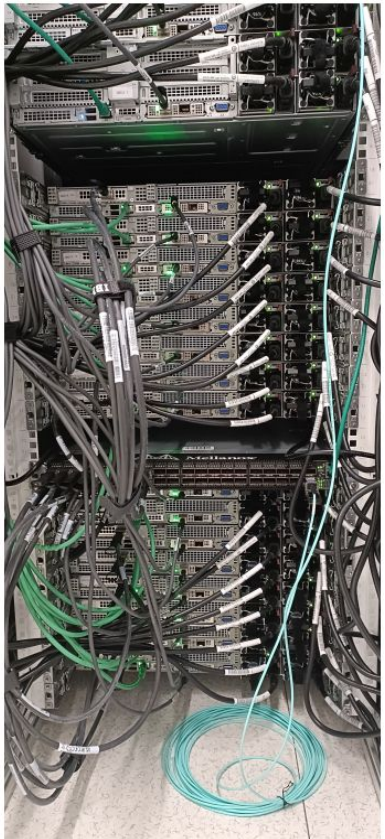
- G. Hanrot, V. Lefèvre, D. Stehlé, P. Zimmermann, "*Worst Cases of a Periodic Function For Large Arguments*", ARITH'18, 2007.
- Idea: group the inputs in the same binade into arithmetic progressions `x, x + qu, x + 2qu, …`, where `u := ulp(x)` and `qu mod 2π` is small
- Apply SLZ algorithm to sub-progressions to filter the candidates for Lemma 2 checks.
- Implemented in BaCSeL - reduced to ~ 131 clock cycles per input.
- Full search will require ~ 120 years on a Ryzen 9950X 16C/32T @5GHz.



The brute-force attack

Credit https://www.letournepage.com/

Our menhir



- The grdix cluster
- 16 x AMD EPYC 9754 (256C/512T @2.7GHz)
- Will take BaCSeL ~ 320 days.
- We will miss ARITH 2026 😱😱😱

Our brute-force algorithm

- Deriving a fast necessary check with low probability of passing to Lemma 2 check.
- Assume that we have already done an expensive computation for `sin(x)`. If `u := ulp(x)` and `qu mod 2π` is small enough, many consecutive elements of the arithmetic progression `x, x + qu, x + 2qu, …,` will be close enough to `x` that `sin(x + nqu)` can be approximated by Taylor polynomial: for `t = qu mod 2π`,

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

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

- Necessary Check1:

$$x + it \in TMD(m) \implies |a + ib + i^2c + i^3d|_{\mathbb{F}(54)} < 2^{-m}u + i^4 e.$$

Make things more expensive to share a lot more.

- If our initial cost raises 100 times but it is reused 10000 times, then we actually reduced the cost by ~100 times.
- Start with computing `t = qu mod 2π` (per binade).
- Compute `sin(x), cos(x)` to a some higher precision (TBD).
- Approximate

  `a = sin(x)`
  `b = t·cos(x)`
  `c = -t^2/2·sin(x)`
  `d = -t^3/6·cos(x)`
  `e = |t^4/24·sin(ξ)|`

  Scale + Truncate

  `A = 2^64 frac(2^k·a) mod 2^64`
  `B = 2^64 frac(2^k·b) mod 2^64`
  `C = 2^64 frac(2^k·c) mod 2^64`
  `D = 2^64 frac(2^k·d) mod 2^64`
  `G = 2^64 frac(2^k·e) + g mod 2^64`

The table of differences method

- Initialize

$$\alpha = A + G \mod 2^{64}$$
$$\beta = B + C + D \mod 2^{64}$$
$$\gamma = 2C + 6D \mod 2^{64}$$
$$\delta = 6D \mod 2^{64}$$

- For i from 0 to n do

  if $\alpha \leq 2G$ then Check$(x + it)$

$$\alpha \leftarrow \alpha + \beta \mod 2^{64}$$
$$\beta \leftarrow \beta + \gamma \mod 2^{64}$$
$$\gamma \leftarrow \gamma + \delta \mod 2^{64}$$

Some more sharing

- Write `x = j·u`. Assuming computing high precision `sin(x)`, `cos(x)` is a lot more expensive than multiplications and additions in the same precision, we can share the cost of computing `sin(x)`, `cos(x)` by only do it for `sin(u)`, `cos(u)` per binade using “binary exponentiation”:

Algorithm 3 (ComputeAB)

Input: an integer $j > 0$, two numbers $s \approx \sin u$ and $c \approx \cos u$
Output: approximations $S \approx \sin(ju)$ and $C \approx \cos(ju)$

1: write $j = 2^k + b_1 2^{k-1} + \cdots + b_{k-1} 2^1 + b_k 2^0$ ▷ binary decomposition
2: $(S, C) \leftarrow (s, c)$
3: for i from 1 to k do
4: $\quad (S, C) \leftarrow (2SC, C^2 - S^2)$
5: $\quad$ if $b_i = 1$ then
6: $\quad\quad (S, C) \leftarrow (sC + cS, cC - sS)$
7: return (S, C)

Even more sharing - Batching

- Binary exponentiation is still quite expensive. We can actually have a simple iteration:

$$\sin((j+1)\cdot u) = \sin(j\cdot u)\cdot\cos(u) + \cos(j\cdot u)\cdot\sin(u)$$

$$\cos((j+1)\cdot u) = \cos(j\cdot u)\cdot\cos(u) - \sin(j\cdot u)\cdot\sin(u)$$

- The rounding errors ~ $O(8^j)$ that needs to be accounted for.
- By using 6 x 64-bit fixed point precision for the initial $\sin(u), \cos(u)$ computations, we were able to perform batching for up to 128 iterations before another binary exponentiation is needed again.

Other optimizations

- Optimize the choices of q
- Maximize parallelism:
  - avoiding imbalance working threads as much as possible
  - vectorize both inter-progressions and intra-progression
- Reduce to degree-2 Taylor approximation

Final results - hard-to-round cases for `sin`

- For sin/cos, we averaged ~2.4 clock cycles per input improved ~ 55x compared to BaCSeL, finished all binades ≥ 2^{11} in less than a week.
- We found 1,048,756 hard-to-round cases for `sin` with `m = 43`.

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

Final results - hard-to-round cases for `cos`

- We found 1,049,705 hard-to-round cases for `cos` with `m = 43`.

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

Final results - hard-to-round cases for `tan`

- We average ~ 5 clock cycles per input.
- We found 1,045,244 hard-to-round cases with `m = 43`

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

Final remarks - Correctly rounded math libraries

- We use the complete set of hard-to-round cases to check math libraries claiming to be correctly rounded for trigonometric double precision functions.

Math library	Rounding modes	sin	cos	tan
MathLib/Libultim (glibc 2.27)	RN	✅	✅	✅
Libmcr	RN	❌	❌	❌
CRLIBM	RN, RU, RD, RZ	✅	✅	✅
CORE-MATH	RN, RU, RD, RZ	✅	✅	✅
LLVM libc	RN, RU, RD, RZ	✅	✅	✅