UNIVERSITY OF LEEDS

Accuracy of Mathematical Functions in Julia

Mantas Mikaitis

School of Computer Science, University of Leeds, Leeds, UK
Joint work with Tejaswa Rizyal

JuliaCon Global 2026, Conference on the Julia programming language
Johannes Gutenberg University Mainz, Germany
13 August, 2026

IEEE 754: correct rounding

correct rounding (IEEE 754-2019 Clause 2.1)

"[...] method of converting an infinitely precise result to a floating-point number, as determined by the applicable rounding direction."

Correct rounding assures implementations have bit-wise equivalent outputs for each input.

IEEE 754-2019 Clause 9.2 "Additional mathematical operations":

A conforming operation shall return results correctly rounded for the applicable rounding direction for all operands in its domain.

Rounding

Rounding itself is easy—it is the approximation of a function that causes difficulties. **Correctly rounding an inaccurate approximation yields an incorrectly rounded result.**

IEEE 754: required operations—correct rounding applies

- Arith: $+$, $-$, $\times$, $\div$, $\sqrt{x}$, $ab + c$ (FMA)
- Round to integer
- Convert to/from int
- Convert formats
- Convert FP to decimal char. sequenc.

Building blocks

Expressions computed by correctly rounded operations are not correctly rounded—including the mathematical operations such as exp, log, which have, e.g. polynomial evaluation inside.

IEEE 754: mathematical functions (recommended)

$\underline{e^x}$, $e^x - 1$ (expm1), $\underline{2^x}$, $2^x - 1$ (exp2m1), $\underline{10^x}$, $10^x - 1$ (exp10m1), $\underline{\log_e(x)}$, $\underline{\log_2(x)}$, $\underline{\log_{10}(x)}$, $\underline{\log_e(1+x)}$ (logp1), $\log_2(1+x)$ (log2p1), $\log_{10}(1+x)$ (log10p1),

$\sqrt{x^2+y^2}$ (hypot), $1/\sqrt{x}$, $(1+x)^n$ (compound), $x^{1/n}$ (rootn), x^n, x^y,

$\underline{\sin(x)}$, $\underline{\cos(x)}$, $\underline{\tan(x)}$, $\underline{\sin(x\pi)}$ (sinpi), $\underline{\cos(x\pi)}$ (cospi), $\underline{\tan(x\pi)}$ (tanpi),

$\underline{\mathrm{asin}(x)}$, $\underline{\mathrm{acos}(x)}$, $\underline{\mathrm{atan}(x)}$, $\underline{\sinh(x)}$, $\underline{\cosh(x)}$, $\underline{\tanh(x)}$, $\underline{\mathrm{asinh}(x)}$, $\underline{\mathrm{acosh}(x)}$, $\underline{\mathrm{atanh}(x)}$.

Correct rounding of math functions: the challenge?

Rounding breakpoints

Inputs at which the rounding function changes values.

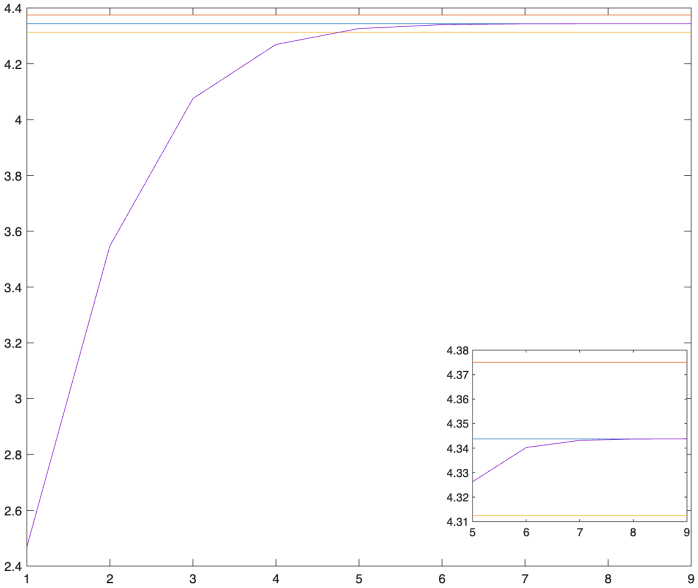
Table Maker's Dilemma

How accurate does the approximation need to be so that we are on the right side of the *rounding breakpoints*?

Example in bfloat16 (8-bit precision):

1. Take Maclaurin series $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$.
2. $e^{1.46875} = \boxed{100.01011}\,00000000011\cdots$ (due to Lefèvre & Muller)
3. Correct rounding (towards zero):
   $\mathrm{RZ}\left(\boxed{100.01011}\,00000000011\cdots\right) = 100.01011$
4. For $k = 8$, $\boxed{100.01010}\,1111111110010\cdots$ (would yield incorrect rounding towards zero)

Correct rounding of math functions: $e^{1.46875}$ in bfloat16



See survey paper in ACM Comput. Surv. 58 (2025)

Correctly Rounded Evaluation of a Function: Why, How, and at What Cost?

NICOLAS BRISEBARRE, Université de Lyon, CNRS, ENS de Lyon, Inria, Université Claude-Bernard Lyon 1, Laboratoire LIP (UMR 5668), Lyon, France
GUILLAUME HANROT, Cryptolab, Inc. & Université de Lyon, CNRS, ENS de Lyon, Inria, Université Claude-Bernard Lyon 1, Laboratoire LIP (UMR 5668), Lyon, France
JEAN-MICHEL MULLER, Université de Lyon, CNRS, ENS de Lyon, Inria, Université Claude-Bernard Lyon 1, Laboratoire LIP (UMR 5668), Lyon, France
PAUL ZIMMERMANN, Université de Lorraine, CNRS, Inria, France

The goal of this article is to give a survey on the various computational and mathematical issues and progress related to the problem of providing efficient correctly rounded elementary functions in floating-point arithmetic. We also aim at convincing the reader that a future standard for floating-point arithmetic should require the availability of a correctly rounded version of a well-chosen core set of elementary functions. We discuss the interest and feasibility of this requirement.

CCS Concepts: • **Mathematics of computing → Mathematical software**; **Numerical analysis**; • **Computer systems organization → Reliability;**

Additional Key Words and Phrases: computer arithmetic, floating-point arithmetic, elementary functions, standardization, correct rounding, table maker's dilemma

ACM Reference Format:
Nicolas Brisebarre, Guillaume Hanrot, Jean-Michel Muller, and Paul Zimmermann. 2025. Correctly Rounded Evaluation of a Function: Why, How, and at What Cost?. *ACM Comput. Surv.* 58, 1, Article 27 (September 2025), 34 pages. https://doi.org/10.1145/3747840

Floating-point formats in Julia

Type	$e_{\min}$	$e_{\max}$	p	$s_{\min}$	$f_{\min}$	$f_{\max}$	Finite numbers
binary16	-14	15	11	2^{-24}	2^{-14}	65504	$2^{16} - 2^{11} = 63488$
binary32	-126	127	24	2^{-149}	2^{-126}	$\sim 3.4 \times 10^{38}$	$2^{32} - 2^{24} = 4278190080$
binary64	-1022	1023	53	2^{-1074}	2^{-1022}	$\sim 1.79 \times 10^{308}$	$2^{64} - 2^{53} \approx 1.8 \times 10^{19}$

ULP errors

Denote a floating-point format with $\mathbb{F}\langle e_{\min}, e_{\max}, p\rangle$ and take $x \in \mathbb{R}$, then

$$\mathrm{ulp}(x) = \begin{cases} 2^{\max(e_{\min}, \lfloor \log_2 |x| \rfloor) - p + 1} & \text{if } x \neq 0, \\ 2^{e_{\min} - p + 1} & \text{otherwise.} \end{cases}$$

Worst-case ulp errors

We report the *worst-case errors* or the *known worst-case errors* in units of ULP. For example, 2.5ULPs of error tells us that the approximation of a function f is not farther away from f than $2.5\mathrm{ulp}(f(x))$ for any x in the input domain of the approximation of f, where $f(x)$ is a reference solution.

Error measure

$E(x) := \frac{|\widehat{f}(x) - f(x)|}{\mathrm{ulp}(\mathrm{RZ}(f(x)))}$ (Use RZ to avoid hitting next power of two where ulp doubles). $f(x)$ computed to 63 or 127 bits using MPFR.

Determining input ranges of functions

Reducing input space of functions

Given some $\mathbb{F}\langle e_{\min}, e_{\max}, p\rangle$, find $[a, b]$, $a, b \in \mathbb{F}\langle e_{\min}, e_{\max}, p\rangle$ for each function f such that

- the function is defined
- does not overflow ($|f(x)| \leq f_{\max}$) if correctly rounded to any of the standard rounding modes
- does not converge to a constant output value, such as zero when raising e to large negative power

Input space reduction

Solving simple equations by substituting $f_{\max}$ or a convergence value for each function/format, we found the ranges. These are hardcoded exactly in the source code: `src/functions.jl`.

Example: function input ranges for binary32

Function	Input domain I_f
acos	$[-1.0, 1.0]$
acosh	$[1.0, f_{\max}]$
asin	$[-1.0, 1.0]$
asinh	$[-f_{\max}, f_{\max}]$
atan	$[-f_{\max}, f_{\max}]$
atanh	$[\mathrm{nextfloat}(-1.0), \mathrm{prevfloat}(1.0)]$
cbrt	$[-f_{\max}, f_{\max}]$
cos	$[-f_{\max}, f_{\max}]$
cosh	$[-89.415985107421875,\ 89.415985107421875]$
exp	$[-103.27892303466796875,\ 88.72283172607421875]$
exp10	$[-44.85346603393554687 5,\ 38.5318412780761718 75]$
exp2	$[-149,\ 127.99999237060546875]$
log	$[s_{\min}, f_{\max}]$
log10	$[s_{\min}, f_{\max}]$
log1p	$[\mathrm{nextfloat}(-1)), f_{\max}]$
log2	$[s_{\min}, f_{\max}]$
sin	$[-f_{\max}, f_{\max}]$
sinh	$[-89.415985107421875,\ 89.415985107421875]$
sqrt	$[0.0, f_{\max}]$
tan	$[-f_{\max}, f_{\max}]$
tanh	$[-9.010912895202636718 75,\ 9.010912895202636718 75]$
cospi	$[-f_{\max}, f_{\max}]$
sinpi	$[-f_{\max}, f_{\max}]$
tanpi	$[-f_{\max}, f_{\max}]$

Prev. work (Gladman, Innocente, Mather, Zimmermann)

Binary32 (exhaustive)

library version	GNU libc 2.43	IML 2025.3.2	AMD 5.2	Newlib 4.6.0	OpenLibm 0.8.7	Musl 1.2.5	Apple 26.1	LLVM 21.1.8	MSVC 2026	FreeBSD 15.0	ArmPL 25.07	CUDA 12.8	ROCm 6.2.4
acos	**0.500**	0.501	0.897	0.899	0.918	0.918	0.634	**0.500**	0.653	0.895	1.32	1.34	1.47
acosh	**0.500**	0.501	0.504	2.01	2.01	2.01	0.502	**0.500**	2.89	2.01	2.79	2.18	0.564
asin	**0.500**	0.501	0.781	0.926	0.743	0.743	0.634	**0.500**	0.846	0.672	2.41	1.36	2.54
asinh	**0.500**	0.527	0.518	1.78	1.78	1.78	0.515	**0.500**	1.99	1.78	3.57	1.78	0.573
atan	**0.500**	0.541	0.501	0.853	0.853	0.853	0.722	**0.500**	0.501	0.853	2.88	1.21	2.10
atanh	**0.500**	0.507	0.547	1.73	1.73	1.73	0.511	**0.500**	2.35	1.73	3.09	3.16	0.574
cbrt	**0.500**	0.520	0.548	3.56	**0.500**	**0.500**	**0.500**	**0.500**	1.83	**0.500**	1.53	1.17	1.14
cos	0.561	0.548	0.729	2.91	0.501	0.501	0.846	**0.500**	0.530	0.501	0.561	1.52	1.61
cosh	**0.500**	0.506	1.03	2.51	1.36	1.03	0.579	**0.500**	**0.500**	1.36	1.89	2.34	0.567
erf	**0.500**	0.780	0.531	0.968	0.943	0.968	0.631	**0.500**	3.98	0.890	1.93	1.04	1.51
erfc	**0.500**	0.934	3.00	63.9	3.17	3.13	0.750		5.79	3.18	1.64	4.49	3.33
exp	0.502	0.506	0.501	0.911	0.911	0.502	0.514	**0.500**	0.501	0.911	0.502	1.94	1.00
exp10	0.502	0.507	0.501	1.00		3.88	0.514	**0.500**				2.07	1.00
exp2	0.502	0.519	0.501	1.00	0.501	0.502	0.514	**0.500**	0.502	0.501	0.502	2.39	0.871
expm1	**0.500**	0.544	0.508	0.813	0.813	0.813	0.687	**0.500**	3.02	0.813	1.51	1.45	1.45
j0	9.00	**0.678**		6.18e6	3.66e6	3.66e6				3.66e6		3.78e10	7.60e7
j1	9.00	**1.69**		1.68e7	2.25e6	2.25e6				2.25e6		7.48e9	7.53e7
lgamma	**0.500**	0.510		7.50e6	7.50e6	7.50e6	0.501		2.92e5	7.50e6		1.35e7	7.50e6
log	0.818	0.509	0.577	0.888	0.888	0.818	0.563	**0.500**	0.562	0.888	0.818	0.865	1.89
log10	**0.500**	0.508	1.40	2.10	0.832	0.832	0.502	**0.500**	0.626	0.832	0.82	2.09	1.71
log1p	**0.500**	0.525	0.501	1.30	0.839	0.835	0.563	**0.500**	1.44	0.839	2.02	0.887	0.579
log2	0.752	0.507	0.766	1.65	0.865	0.752	0.502	**0.500**	0.752	0.865	0.752	0.919	1.00
sin	0.561	0.546	0.530	1.37	0.501	0.501	0.846	**0.500**	0.530	0.501	0.561	1.50	1.61
sinh	**0.500**	0.538	**0.500**	2.51	1.83	1.83	0.579	**0.500**	0.501	1.83	2.26	2.94	0.922
sqrt	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**		**0.500**	**0.500**
tan	**0.500**	0.520	0.509	3.48	0.800	0.800	0.746	**0.500**	0.502	0.800	3.30	3.10	2.33
tanh	**0.500**	0.514	**0.500**	2.19	2.19	2.19	0.817	**0.500**	1.24	2.19	2.59	1.82	1.41

Prev. work (Gladman, Innocente, Mather, Zimmermann)

Binary64 (known worst cases)

library version	GNU libc 2.43	IML 2025.3.2	AMD 5.2	Newlib 4.6.0	OpenLibm 0.8.7	Musl 1.2.5	Apple 26.1	LLVM 21.1.8	MSVC 2026	FreeBSD 15.0	ArmPL 25.07	CUDA 12.8	ROCm 6.2.4
acos	0.523	0.531	1.36	0.930	0.930	0.930	1.06	**0.500**	0.932	0.930	1.52	1.53	0.772
acosh	**0.500**	0.509	1.32	2.25	2.25	2.25	2.25		3.22	2.25	2.66	2.52	0.662
asin	0.516	0.531	1.06	0.981	0.981	0.981	0.711	**0.500**	0.957	0.981	2.69	1.99	0.710
asinh	**0.500**	0.507	1.65	1.92	1.92	1.92	1.58		2.05	1.92	2.04	2.57	0.661
atan	**0.523**	0.528	0.863	0.861	0.861	0.861	0.876	1.00	0.863	0.861	2.24	1.77	1.73
atanh	**0.500**	0.507	1.04	1.81	1.81	1.80	2.01		2.50	1.81	3.00	2.50	0.664
cbrt	3.67	0.523	0.503	0.670	0.668	0.668	0.729	**0.500**	1.86	0.668	1.79	0.501	0.501
cos	0.516	0.518	0.919	0.887	0.834	0.834	0.948	**0.500**	0.897	0.834		1.52	0.798
cosh	1.93	**0.516**	1.85	2.67	1.47	1.04	0.523		1.42	1.47	1.93	1.40	0.563
erf	**0.500**	0.773	1.00	1.02	1.02	1.02	6.41		3.32	1.02	2.29	1.50	1.12
erfc	**0.500**	0.827	4.31e6	4.08	4.08	3.72	10.7		5.85	4.08	1.71	4.51	4.08
exp	0.511	0.530	1.01	0.949	0.949	0.511	0.521	**0.500**	1.50	0.949	0.511	0.928	0.929
exp10	0.513	0.538	1.02	0.896		4.14	0.521	**0.500**			0.510	1.11	1.11
exp2	0.511	0.535	1.05	0.896	0.751	0.511	0.521	**0.500**	0.505	0.751	0.509	0.948	0.947
expm1	0.913	0.512	0.670	0.909	0.909	0.909	0.706	**0.500**	3.06	0.909	2.18	1.18	1.91
j0	4.51e14	**0.600**		9.01e15	4.51e14	4.51e14	3.83e14		2.42e33	4.51e14		3.08e20	1.25e13
j1	4.47e14	**0.615**		9.01e15	1.10e15	1.10e15	1.10e15		8.19e32	1.10e15		1.73e21	4.80e13
lgamma	**0.500**	0.515		4.45e15	4.45e15	4.45e15	2.33e16		5.10e13	4.45e15		5.11e15	4.45e15
log	0.520	0.518	0.611	0.946	0.946	0.520	0.508	**0.500**	0.577	0.946	0.520	0.564	0.663
log10	1.62	0.532	1.09	2.08	0.814	0.814	0.514	**0.500**	0.633	0.814	1.62	1.43	0.785
log1p	0.899	0.521	0.636	0.896	0.896	0.900	0.667	**0.500**	1.44	0.896	1.74	1.50	1.00
log2	0.548	0.509	1.72	2.06	0.921	0.555	0.515	**0.500**	0.812	0.921	0.554	1.31	0.734
sin	0.516	0.518	0.895	0.888	0.831	0.831	0.944	**0.500**	0.799	0.831		1.52	0.800
sinh	1.93	**0.521**	1.49	2.67	1.88	1.88	0.539		1.41	1.88	2.59	1.51	0.868
sqrt	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**		**0.500**	**0.500**
tan	0.619	0.550	1.38	1.02	1.02	1.02	3.53	**0.500**	1.32	1.02		2.09	1.30
tanh	2.21	**0.556**	1.40	2.22	2.22	2.22	0.613		1.33	2.22	2.76	1.48	0.866

Covering input ranges of functions

How many inputs to test

We can cover all inputs for all functions easily for binary16 and binary32 (univariate). For the cases where it is not feasible, we need a *search strategy*.

Determining N the number of tests (per thread)

Given an input domain in $\mathbb{F}\langle e_{\min}, e_{\max}, p\rangle$ of a function f for a particular floating-point format, I_f, form $I \subset I_f$ according to one of the strategies below. Take N members of I_f to compile $I = \{x_1, \ldots, x_N\}$.

- Seconds: $N = \lfloor \frac{10^9}{t} \rfloor$. Here t is an approximate latency of a function.
- Minutes: $N = \lfloor \frac{60 \times 10^9}{t} \rfloor$.
- Hours: $N = \lfloor \frac{3600 \times 10^9}{t} \rfloor$.
- Days: $N = \lfloor \frac{24 \times 3600 \times 10^9}{t} \rfloor$.
- Exhaustive: $I = I_f$, $N = \mathrm{length}_{\mathbb{F}\langle e_{\min}, e_{\max}, p\rangle}(I_f)$.

Testbench configuration: JSON interface

```json
{
  "test-binary16RN-exhaustive-nofastmath" : {
    "format" : "binary16",
    "rounding" : "RN",
    "fastmath" : 0,
    "search" : "exhaustive"
  },
  "test-binary32RN-exhaustive-nofastmath" : {
    "format" : "binary32",
    "rounding" : "RN",
    "fastmath" : 0,
    "search" : "exhaustive"
  },
  "test-binary64RN-hours-nofastmath" : {
    "format" : "binary64",
    "rounding" : "RN",
    "fastmath" : 0,
    "search" : "hours"
  }
}
```

Run: `julia -t <T> run.jl`

Results: system specification

Binary16 and binary32: Macbook Pro M4 12x cores. Julia 1.12.6.

Binary64: Two AMD EPYC 9634 84-core CPUs running at 2.2GHz, overall providing 168 cores. We have used Julia 1.11.6 (latest results running).

2 days of runtime, mainly testing binary64.

Results: binary16 (exhaustive)

Function	ULPs	Input	Output	MPFR	Tests
acos	0.50002	−0.556640625	2.16015625	2.1611328533	30,721
acosh	0.50005	1.22265625	0.6552734375	0.6555176014	16,384
asin	0.49993	−0.1364746094	−0.1368408203	−0.1369018472	30,721
asinh	0.50007	−258.5	−6.25	−6.2480466158	63,487
atan	0.50003	−0.0283966064	−0.0283813477	−0.0283889774	63,487
atanh	0.49986	−0.0357666016	−0.0357971191	−0.0357818648	30,719
cbrt	0.49999	−10,888	−22.171875	−22.1640627339	63,487
cos	0.50001	−0.0584716797	0.998046875	0.9982910183	63,487
cosh	0.50004	−0.03125	1	1.000488321	37,833
cospi	0.49994	−0.0070343018	0.9995117188	0.999755829	63,487
exp	0.50003	0.0226898193	1.0234375	1.0229491913	38,324
exp10	0.50024	−3.763671875	0.0001723766	0.000172317	35,850
exp2	0.50001	0.0007042885	1	1.0004882948	39,424
log	0.50004	0.0053405762	−5.234375	−5.2324217345	31,743
log10	0.50006	2,976	3.47265625	3.4736329269	31,743
log1p	0.50001	−0.0058479309	−0.0058670044	−0.005865097	47,103
log2	0.49991	0.2834472656	−1.8193359375	−1.8188477431	31,743
sin	0.50004	−300	1	0.9997558399	63,487
sinh	0.49998	−7.34375	−773	−773.2499920787	37,833
sinpi	0.49989	−0.0000471473	−0.0001480579	−0.0001481175	63,487
sqrt	0.49994	0.0002440214	0.0156173706	0.0156211848	31,744
tan	0.50002	−0.0357666016	−0.0357666016	−0.0357818608	63,487
tanh	0.49994	−0.0283966064	−0.0283813477	−0.0283889762	35,075
tanpi	0.50006	−0.0001212358	−0.0003809929	−0.0003808737	63,487

Results: binary32 (exhaustive)

Function	ULPs	Input	Output	Tests
acos	0.9179250677	-0.50042	2.09488	2,130,706,435
acosh	1.9944941297	1.00775	0.1244	1,073,741,826
asin	0.7298913713	-0.50073	-0.52445	2,130,706,435
asinh	1.5277680587	-0.12024	-0.11996	4,278,190,081
atan	0.8521076359	-0.6922	-0.60547	4,278,190,081
atanh	1.9411387117	$-7.79411 \cdot 10^{-3}$	$-7.79427 \cdot 10^{-3}$	2,130,706,433
cbrt	0.499999982	$-1.74563 \cdot 10^{38}$	$-5.58879 \cdot 10^{12}$	4,278,190,081
cos	0.5009001762	$-15{,}705.27148$	-0.90053	4,278,190,081
cosh	2.414086836	-89.37019	$3.25051 \cdot 10^{38}$	2,238,032,379
cospi	0.5000704434	-0.42245	0.24123	4,278,190,080
exp	0.8934687937	26.68997	$3.90217 \cdot 10^{11}$	2,239,758,569
exp10	1.0477928032	4.08913	12,278.02344	2,219,674,257
exp2	0.8747607674	-126.42215	$8.77285 \cdot 10^{-39}$	2,249,523,202
log	0.5507246331	1.06556	0.0635	2,139,095,041
log10	0.586694695	1.06589	0.02771	2,139,095,041
log1p	0.5635540398	0.082	0.07881	3,204,448,257
log2	0.5707293911	1.08145	0.11297	2,139,095,041
sin	0.5009001286	$-2.57601 \cdot 10^{31}$	-0.80778	4,278,190,081
sinh	2.414086836	-89.37019	$-3.25051 \cdot 10^{38}$	2,238,032,378
sinpi	0.5000706535	$-3.81455 \cdot 10^{-5}$	$-1.19838 \cdot 10^{-4}$	4,278,190,080
sqrt	0.4999999925	$4.70198 \cdot 10^{-38}$	$2.1684 \cdot 10^{-19}$	2,139,095,040
tan	0.7998569047	$-6.46722 \cdot 10^{10}$	-7.99478	4,278,190,081
tanh	1.3663689677	-0.12551	-0.12486	2,183,158,121
tanpi	0.500082014	-0.24998	-0.99986	4,278,190,080

Results: binary64 (1-2 hours per function, 168 cores)

Function	ULPs	Input	Output	Tests
acos	0.9006986618	-0.52243	2.1205	26,590,058,380
acosh	2.0152511597	1.00699	0.11819	21,778,279,324
asin	0.8815574646	-0.5023	-0.52625	28,884,011,020
asinh	1.552775383	0.52074	0.49969	15,313,993,634
atan	0.8525094986	0.69962	0.61047	18,443,002,540
atanh	1.9880361557	$1.9499 \cdot 10^{-3}$	$1.9499 \cdot 10^{-3}$	31,533,156,148
cbrt	0.6670742035	$-1.50513 \cdot 10^{-160}$	$-5.31934 \cdot 10^{-54}$	35,003,831,524
cos	0.8252658844	$-1.1852 \cdot 10^{280}$	0.72064	26,920,755,796
cosh	1.9222135544	-710.47184	$1.79048 \cdot 10^{308}$	34,457,676,796
cospi	0.9097290039	$-1.2756 \cdot 10^{7}$	0.72012	27,455,885,860
exp	0.7575774193	-708.6358	$1.75138 \cdot 10^{-308}$	32,446,565,428
exp10	0.7585949898	-307.70809	$1.95843 \cdot 10^{-308}$	24,004,809,868
exp2	0.7587532997	$-1{,}022.03188$	$2.17645 \cdot 10^{-308}$	13,775,094,148
log	0.5551567078	1.08195	0.07877	22,568,962,084
log10	0.6261129379	1.06618	0.02783	17,515,452,364
log1p	0.5656337738	0.07422	0.0716	24,852,795,412
log2	0.5981225967	1.08182	0.11346	18,707,470,156
sin	0.8293628693	$6.86426 \cdot 10^{30}$	0.71468	24,285,228,748
sinh	1.9222135544	-710.47184	$-1.79048 \cdot 10^{308}$	23,702,916,724
sinpi	0.9081516266	$-8.78737 \cdot 10^{6}$	0.71028	23,496,667,156
sqrt	0.5	$2.91022 \cdot 10^{-309}$	$5.39465 \cdot 10^{-155}$	23,709,682,418
tan	1.041472435	$3.02146 \cdot 10^{88}$	-0.79934	18,693,623,932
tanh	1.5872955322	0.50202	0.4637	35,662,764,484
tanpi	2.4517974854	$-1.55527 \cdot 10^{7}$	-0.98508	26,232,941,620

Conclusions

- We have developed an easy-to-run testsuite for Julia.
- We find that most functions are close to being correctly rounded.
- Future work: bivariate functions, different search strategy for binary64, bigger proportion of input space.
- What should Julia do? Utilise @fastmath switch for correctly rounded mode (see CORE-MATH by Zimmermann et al).

Preprint available

M. Mikaitis and T. Rizyal. *Accuracy of Mathematical Functions in Julia*. **Preprint. arXiv:2509.05666 [cs.MS]** (revision coming next week!). `http://bit.ly/3WpMbxn`.

Slides at `http://mmikaitis.github.io/talks`

References I

B. Gladman, V. Innocente, J. Mather, P. Zimmermann
Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision
Preprint. hal-03141101. Aug. 2025

K. Ozaki
Accuracy of Low-Precision Elementary and Special Functions Using Verified Numerical Computations
J. Adv . Simulat. Sci. Eng., 12:1. Apr. 2025.

N. Brisebarre, G. Hanrot, J.-M. Muller, P. Zimmermann
Correctly-Rounded Evaluation of a Function: Why, How, and at What Cost?
ACM Comput. Surv., 58:1. Sep. 2025.