Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision

Vincenzo Innocente and Paul Zimmermann

August 29, 2022

> *Computer users, most of whom assume they are working with reliable routines, unwittingly accept results from functions where the accuracies vary significantly from one mathematical library to another, from one library function to another, and even over different argument intervals of the same function. [...] Users are not likely to demand an improved situation because most of them, having neither the time nor the inclination to test manufacturer-supplied software, do not know the problem exists. This paper contains the results of such tests of elementary functions from several computer companies. The data (see Table I) demonstrate that the industry does not satisfy the needs of those who require accurate and efficient mathematical software.*

These lines, written in 1984 by Black, Burton and Miller [5], are unfortunately still very true today.

The IEEE 754 standard, even in its latest 2019 revision [13], does not *require* correctly rounded mathematical functions, it only *recommends* them. In turn, current mathematical libraries do not provide correct rounding, which is the best possible result. Thus, users might get different results with different libraries, or different versions of the same library. This can have dramatic consequences: for example missed collisions in the Large Hadron Collider [4] or reproducibility issues in neuroimaging [10].

This document compares the accuracy of several mathematical libraries for the evaluation of mathematical functions, in single, double and quadruple precision (respectively `binary32`, `binary64`, and `binary128` in the IEEE 754 standard), and also in the extended double format. For single precision, an exhaustive search is possible for univariate functions, thus the given values are upper bounds. For larger precisions or bivariate functions, since an exhaustive search is not possible with academic resources, we use a black-box algorithm that tries to locate the values with the largest error; the given values are only lower bounds, but comparing them can give an idea of the relative accuracy of different libraries. An interesting fact is that, for several functions, different libraries yield the same largest known error, for the exact same input value, which probably means they use the same code base. Note that some libraries document the maximal errors [6] or known maximal errors [11].

Today, at least for single precision and most double precision functions, it is known how to get correct rounding (for all rounding modes, not only for rounding to nearest) at very low cost, and reference implementations exist that outperform current libraries [22].

1 Introduction

In this document we compare the accuracy of the following mathematical libraries (in the rounding to nearest mode): GNU libc 2.36 [12], the Intel Math Library shipped with the Intel oneAPI DPC++ Compiler 2022.0.0 (IML) [14], AMD LibM 3.9 [1], RedHat Newlib 4.2.0 [18], OpenLibm 0.8.1 [20], Musl 1.2.3 [17], the Apple Math Library available under Darwin 21.2.0 [2], the LLVM libc 14.0.6 [15], the CUDA mathematical library 11.6.0 [7], and the ROCm mathematical library 5.0.2 [19]. We do not compare to the x87 instructions `fsin` and others, which are known to have bad accuracy [8].

For each function, assuming y is the value returned by the library, and z is the exact result (as with infinite precision), we denote by e the absolute difference between y and z in terms of units-in-last-place of z. The value z is approximated with the GNU MPFR library [9], using a larger precision. Our definition of ulp (unit-in-last-place) is the following: for $2^{e-1} \leq |x| < 2^e$, and precision p, we define $\mathrm{ulp}(x) = 2^{e-p}$. i.e., the distance between two consecutive p-bit floating-point numbers in the binade $[2^{e-1}, 2^e]$, see [16].

The results for GNU libc, AMD LibM, Newlib, OpenLibm, Musl and LLVM libc were obtained on an Intel Core i5-4590, with GCC 12.1.0 under Debian (note that some results might differ sligthly from one `x86_64` processor to another one, due for example to the use of fused-multiply add or not). Those for the Intel Math Library (IML in short, where the version is that of the Intel C compiler) were obtained on an AMD EPYC 7282 with the Intel compiler[1] version 2022.0.0, using `-fp-model=strict`. Those for the Apple math library were obtained on a Mac M1 (arm64) under Darwin 21.2.0, with clang 12.0.5. The CUDA library was tested on a NVidia Tesla V100-SXM2 running CUDA 11.6.0 hosted on a Intel Xeon Gold 6148. The test were also run on a GTX1060 GPU, hosted on an AMD Ryzen 7 1800X, obtaining identical results. The ROCm library was tested on an AMD Radeon Instinct MI50 running ROCm 5.0.2 hosted on a Intel Xeon Silver 4114. Test were also run on a "Radeon Pro WX 9100", hosted on a AMD Ryzen 9 5900X, obtaining identical results.

Newlib was configured with default flags (in particular, without use of hardware FMA), and with the default configuration.[2]

In all tables, values of e are given with 3 decimal digits, rounded up; thus for example $e = 2.17$ for a univariate single-precision function means that the relative error is bounded by $2.17\mathrm{ulp}(z)$ for all `binary32` inputs, and in all other cases (larger formats or bivariate functions) it means the largest *known* error is bounded by $2.17\mathrm{ulp}(z)$, with at least one case giving an error of more than $2.16\mathrm{ulp}(z)$.

2 Single Precision

2.1 Univariate Functions

The IEEE 754 single-precision (`binary32`) format has $2^{32} - 2^{24} = 4278190080$ values, not counting +Inf, -Inf, and NaN. For a function with a single input—i.e., excluding the `pow` function for example—it is possible to check all values by exhaustive search.

[1] Through the Docker image `intel/oneapi-hpckit`.

[2] For binary32, by default, the old SunPro functions are used; with `OBSOLETE_MATH_DEFAULT=0`, Newlib will use instead a new set of mathematical functions provided by Arm, that use binary64 for intermediate computations.

Table 1 summarizes the maximal value of e for each function and each library. For univariate functions, the corresponding input can easily be found by exhaustive search; Table 2 gives the corresponding inputs for bivariate functions.

In all tables, the notation NA means "Not Available" (`exp10` in OpenLibm, the Bessel functions `j0`, `j1`, `y0`, `y1` in Apple libm, and many functions in LLVM).

We see that for all libraries, the sqrt function is correctly rounded for all binary32 inputs, as required by IEEE 754. The single-precision cubic root function (`cbrt`) is also correctly rounded in OpenLibm, Musl and the Apple library.

The `j0`, `j1`, `y0`, and `y1` functions give large errors for all libraries where there are available, except for GNU libc, IML and LLVM (apart from `j0` for the latter).

2.2 Bivariate Functions

For bivariate functions, it is not possible to perform an exhaustive search with academic resources, since there are up to 2^{64} possible pairs of inputs. For example, for the power function x^y, there are about 2^{61} input pairs $x, y > 0$ that do not yield underflow nor overflow. We thus used the algorithm described in §3.1 to obtain the values of Table 2, which are *lower bounds* for the maximal error.

Notes about AMD LibM. We noticed a regression in AMD LibM 3.9: for $x =$ `0x1.62e8p+61`, expm1f yields `0x1.62e8p+61` instead of +Inf. The maximal error for `exp10f` is 1.00 since for $x =$ `-0x1.66d3eap+5`, it yields 0 instead of the smallest subnormal 2^{-149}, where 10^x is slightly smaller than the smallest subnormal (a similar issue was fixed in release 3.8 for expf and exp2f).

Notes about Newlib. In Newlib 4.2.0, for $x =$ `0x1.62e302p+6`, `expf` yields +Inf instead of `0x1.ff691ep+127`; for $x =$ `-0x1.000002p-65`, `tgammaf` yields -Inf instead of `-0x1.fffffcp+64`; for $x =$ `-0x1p-70`, `lgammaf` yields +Inf instead of `0x1.842994p+5`. We used the `lgammaf_r` function, since we were unable to compile the `lgammaf` function (`undefined reference to '_impure_ptr'`).

Notes about the Apple Math Library. The `erf`, `lgamma` and `tgamma` functions seem to call the corresponding double function, which explains the very good accuracy and the very small number of incorrectly-rounded results. The single precision `exp10` function is available as `__exp10f`.

Notes about LLVM-libc. LLVM only provides few mathematical functions so far. However, the development version of LLVM-libc contains more correctly rounded functions (as of August 2022, `atanhf`, `cosf`, `coshf`, `expf`, `exp2f`, `expm1f`, `log1pf`, `sinf`, `sinhf`, `tanf`, `tanhf`); when these functions will be included in a future release, we expect several more **0.500** entries in further updates of this article.

3 Double Precision

For double precision it is not possible to perform an exhaustive search with academic resources. We thus designed a black-box algorithm that tries to find large errors. (We did not want to analyze the code of each library, since this approach would need more human work, and requires to start again from scratch for each new version of the library.) Therefore, the values in the double-precision tables are only lower bounds of the maximal error.

library	GNU libc	IML	AMD	Newlib	OpenLibm	Musl	Apple	LLVM	CUDA	ROCm
version	2.36	2022.0.0	3.9	4.2.0	0.8.1	1.2.3	12.1	14.0.6	11.6.0	5.0.2
acos	0.899	**0.528**	0.897	0.899	0.918	0.918	0.634	NA	1.69	1.47
acosh	2.01	**0.501**	0.504	2.01	2.01	2.01	0.502	NA	2.18	0.564
asin	0.898	**0.528**	0.781	0.926	0.743	0.743	0.634	NA	2.05	2.54
asinh	1.78	0.527	0.518	1.78	1.78	1.78	**0.515**	NA	1.78	0.573
atan	0.853	0.541	**0.501**	0.853	0.853	0.853	0.722	NA	2.06	2.10
atanh	1.73	**0.507**	0.547	1.73	1.73	1.73	0.511	NA	3.16	0.574
cbrt	0.969	0.520	0.548	3.56	**0.500**	**0.500**	**0.500**	NA	1.17	1.14
cos	0.561	0.548	0.729	2.91	**0.501**	**0.501**	0.862	0.776	1.52	1.61
cosh	1.89	**0.506**	1.03	2.51	1.36	1.03	0.589	NA	2.34	0.567
erf	0.968	0.507	0.968	0.968	0.943	0.968	**0.501**	NA	1.14	1.51
erfc	3.13	**0.502**	3.13	63.9	3.17	3.13	0.750	NA	4.49	3.33
exp	0.502	0.506	**0.501**	1.94e4	0.911	0.502	0.576	0.502	1.94	1.00
exp10	**0.502**	0.507	1.00	1.06	NA	3.88	0.580	NA	2.07	1.00
exp2	0.502	0.519	**0.501**	1.02	**0.501**	0.502	0.570	0.502	2.39	0.871
expm1	0.813	**0.544**	Inf	0.813	0.813	0.813	0.687	1.50	1.45	1.45
j0	9.00	**0.678**	9.01e5	6.18e6	3.66e6	3.66e6	NA	NA	3.78e10	7.60e7
j1	9.00	**1.69**	9.00	1.68e7	2.25e6	2.25e6	NA	NA	7.48e9	7.53e7
lgamma	6.78	0.510	6.78	8.93e43	7.50e6	7.50e6	**0.501**	NA	1.35e7	7.50e6
log	0.818	0.519	0.577	0.888	0.888	0.818	0.511	**0.500**	0.865	1.89
log10	2.07	0.516	1.40	2.10	0.832	0.832	0.502	**0.500**	2.09	1.71
log1p	1.30	0.525	**0.501**	1.30	0.839	0.835	0.513	NA	0.887	0.579
log2	0.752	0.508	0.766	1.65	0.865	0.752	0.502	**0.500**	0.919	1.00
sin	0.561	0.546	0.530	1.37	**0.501**	**0.501**	0.846	0.561	1.50	1.61
sinh	1.89	0.538	**0.500**	2.51	1.83	1.83	0.601	NA	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**
tan	1.48	0.520	**0.509**	3.48	0.800	0.800	0.746	NA	3.10	2.33
tanh	2.19	**0.514**	1.56	2.19	2.19	2.19	0.817	NA	1.82	1.41
tgamma	7.91	0.510	7.91	1.55e26	**0.501**	**0.501**	**0.501**	NA	11.5	1.68e7
y0	8.98	**3.40**	8.98	4.84e6	4.84e6	4.84e6	NA	NA	2.36e10	7.53e7
y1	9.00	**2.07**	9.00	6.18e6	4.17e6	3.66e6	NA	NA	4.96e10	9.35e7
atan2	1.52	**0.550**	0.584	1.52	1.55	1.55	0.722	NA	2.18	2.01
hypot	0.501	0.501	0.501	1.21	1.21	0.927	0.501	**0.500**	1.03	1.57
pow	0.817	**0.515**	1.56	169.	0.970	0.817	**0.515**	NA	10.3	1.40

Table 1: Single precision: maximal value of e (for univariate functions), and largest *known* value of e (for bivariate functions).

	GNU libc 2.36			IML 2022.0.0		
	x	y	max e	x	y	max e
atan2	-0x1.f9cf48p+49	0x1.f60598p+51	1.52	-0x1.58a7ecp-118	0x1.58a7bep-123	0.550
hypot	-0x1.0554acp+44	-0x1.6dc9e6p+32	0.501	-0x1.003222p-20	-0x1.6a2d58p-32	0.501
pow	0x1.025736p+0	0x1.309f94p+13	0.817	0x1.fe7782p-1	-0x1.c361cap+14	0.515

	AMD LibM 3.9			RedHat Newlib 4.2.0		
	x	y	max e	x	y	max e
atan2	0x1.fffe24p+59	0x1.000adcp+73	0.584	-0x1.f9cf48p+49	0x1.f60598p+51	1.52
hypot	-0x1.0554acp+44	-0x1.6dc9e6p+32	0.501	-0x1.6b05c4p-127	0x1.6b3146p-126	1.21
pow	0x1.10fff4p+0	0x1.58fd76p+10	1.56	0x1.d55902p-1	-0x1.fe037ep+9	169.

	OpenLibm 0.8.1			Musl 1.2.3		
	x	y	max e	x	y	max e
atan2	0x1.a10104p+123	0x1.99f182p+125	1.55	0x1.a10104p+123	0x1.99f182p+125	1.55
hypot	-0x1.6b05c4p-127	0x1.6b3146p-126	1.21	0x1.26b188p-127	-0x1.a4f2fp-128	0.927
pow	0x1.343e4ep+0	0x1.af3c4p+8	0.970	1.025736p+0	1.309f94p+13	0.817

	Apple 12.1			LLVM 14.0.6		
	x	y	max e	x	y	max e
atan2	-0x1.ce62cep-116	0x1.cbf9bp-113	0.722	NA	NA	NA
hypot	-0x1.0554acp+44	-0x1.6dc9e6p+32	0.501	0x1.5804ccp-40	-0x1.a3bp-52	0.500
pow	0x1.034016p+0	0x1.b782b4p+12	0.515	NA	NA	NA

	CUDA 11.6.0			ROCm 5.0.2		
	x	y	max e	x	y	max e
atan2	0x1.a59982p-74	0x1.915c9p-74	2.18	0x1.684824p+26	0x1.db5dc8p+25	2.01
hypot	0x1.007594p+1	-0x1.003512p+1	1.03	-0x1.ad2d0ap+111	-0x1.7f456ap+118	1.57
pow	0x1.714882p-1	-0x1.0f68b4p+8	10.3	0x1.6e3446p+18	-0x1.b40d2p+2	1.40

Table 2: Single precision bivariate functions.

3.1 Search Algorithm

The idea of the algorithm is to subdivide recursively the set of values to search for. We describe it for a univariate double precision function, but it works for any IEEE format, as long as there is a corresponding integer type with the same bit-width, and it also works for bivariate functions.

Assume $f(x)$ is a univariate double precision function. The number of possible inputs of f is less than 2^{64}, thus each one can be mapped to a 64-bit integer. Assume we have a conversion function `to_uint64` from `uint64_t` to `double`. The algorithm takes as input a range $[a, b]$ of `uint64_t` values, and a threshold t. If $b - a < t$, it checks exhaustively all double precision values `x=to_uint64(i)` for $a \leq i < b$. This means for each x, we compute the ulp-error e between the value $y \approx f(x)$ returned by the corresponding library, and the exact result z (as with infinite precision), as described in §1.

If $b - a \geq t$, we subdivide the interval $[a, b]$ into two equal intervals, in each interval we generate t random values and compute the corresponding errors. We then recurse in the interval where we found the largest error.

For example with $t = 10^6$, the initial interval has 2^{64} values, thus we compute $f(x)$ on $2t$ random inputs x (t in each sub-range of 2^{63} values), and so on... The recursion stops when the recursive algorithm would perform more function evaluations than trying all the values in the current interval.

In practice we used a variant of this algorithm suggested by Eric Schneider: instead of recursing only in the sub-interval giving the largest error on the random sample, we keep at each level of the search tree a list of say 20 intervals with the largest sample errors. Then we subdivide each of those intervals, which yields 40 smaller intervals (or 80 for bivariate functions), and keep again the 20 better ones.

We tried three variants of this algorithm, depending on how we choose the "best" sub-interval. The first strategy—described above—keeps the sub-interval with the maximal ulp-error. A second strategy keeps the sub-interval with the maximal *average* ulp-error (considering only inputs which yield a non-zero ulp-error, i.e., discarding those giving NaN, zero or $\pm\infty$). A third strategy keeps the sub-interval with the largest expected ulp-error; for this, we estimate the mean and standard deviation of the ulp-error on each sub-interval, from which we deduce an estimate of the largest ulp-error for the number of points in the sub-interval [21]. In practice we found the first strategy to be more effective, with the second and third strategies finding sometimes larger maximal errors. Thus when the search program is run on a machine with n cores, we assign one core to the second and third strategies, and $n - 2$ cores to the first one.

The program also keeps track of the worst cases found for each library, and tries those input values for the other libraries. This helps determining the libraries using the same code base. The search programs (`check_sample.c` for univariate functions, and `check_sample2.c` for bivariate functions), the exhaustive search program for `binary32` univariate functions (`check_exhaustive.c`) and the source code of this article (containing in comment the x-values yielding the largest errors for `binary32`) are available from `https://gitlab.inria.fr/zimmerma/math_accuracy`.

We have also used the worst cases found by Vincent Lefèvre, publicly available at `https://www.vinc17.net/research/testlibm/`.

3.2 Results

We used a threshold of at least $t = 10^6$ for all libraries, often on processors with at least 32 cores, and the search program was run multiple times, cycling over all libraries, to detect common large

errors.

Table 3 summarizes the maximal known errors found using the above algorithm, for example the 0.531 entry for `acos` and IML means that for all inputs tried by the above algorithm, the ulp-error e for the arc-cosine function with the Intel Math Library was bounded by 0.531 ulp. On each line, bold-face entries correspond to the smallest maximal known error. Detailed tables (Tables 4, 5, 6, 7 and 8) give the input values (in hexadecimal) yielding the corresponding ulp-error e, which enables the reader to reproduce our results.

In double precision, the Intel Math Library gives the best results in most cases (for 19 of the 30 univariate functions). However, it was observed that the Intel Math Library gives better results on AMD hardware than on Intel hardware for `acosh`, `asin`, `asinh` and `atan2`; a possible explanation is that those functions use the `rsqrt` instructions, which is known to be more accurate on AMD hardware [3]. The square root function seems to be correctly rounded for all libraries, as required by IEEE 754. Large errors occur for the AMD `exp10`, `expm1`, `sinh`, `atan2` and `hypot` functions, for the `j0`, `j1`, `y0` and `y1` functions for all libraries except the Intel Math Library, for the `lgamma` function from Newlib, OpenLibm, Musl, Apple, CUDA and ROCm libraries, for the `tgamma` function from Newlib, OpenLibm and the Apple library, for the power function from Newlib and OpenLibm, and for the `cos`, `sin`, and `tan` functions from LLVM libc.

Notes about AMD LibM. We noticed several regressions in AMD LibM 3.9: the first one is for $x =$ `0x1.facf4856ce3c8p+491`, where exp10 yields x instead of +Inf (same issue with expm1); for $x =$ `0x1.fffffffffffep-26`, sinh yields `0x1.fffffffffffep-25` instead of half ot it which is the correct result; for subnormal numbers, `atan2` gives huge errors; finally for $x =$ `-0x0.fffffffffffep-1022` and $y =$ `0x0.0000000000001p-1022`, the `hypot` function yields `0x0.0000000000001p-1022` instead of `0x0.fffffffffffep-1022`.

Notes about LLVM-libc. For $x =$ `-0x1.13a5ccd87c9bbp+1008`, the `cos` and `sin` functions return x instead of `0x1.a1fa1068d0b59p-1` and `-0x1.27b3964185d8dp-1` respectively, and for $x =$ `0x1.f967bd3017f2bp+62`, `tan` yields `-0x1.a4ee5870d415fp+20` instead of `0x1.554235622076p+6`.

4 Double Extended Precision

This format corresponds to the C type `long double` on `x86_64` processors. The results are summarized in Table 9, and detailed in Tables 10 and 11. We see that in this format, the Intel Math library is better than all other libraries for all functions, both univariate and bivariate, except for the hypot function.

For the Intel Math Library, the j0, j1, y0, and y1 functions call the corresponding quadruple precision function, which explains why the maximal error is 0.5 ulp in our experiments[3] (assuming the quadruple precision functions are correctly rounded, an incorrect rounding can only occur when the last $113 - 64 = 49$ bits are exactly 100...000, which occurs with probability 2^{-49}). AMD Libm does not provide long double functions. Newlib only provides long double functions for platforms where `long double` is the same as `double` (which is not the case of the `x86_64` processor) with two exceptions: `sqrt` and `hypot`. However, in Newlib 4.2.0, the `hypotl` function does not work properly: for $x \geq 2^{8192}$, the call `hypotl(x,0)` gives infinity. OpenLibm does not provide the following long

[3]Except for j0 where we found an input that is not correctly rounded.

library	GNU libc	IML	AMD	Newlib	OpenLibm	Musl	Apple	LLVM	CUDA	ROCm
version	2.36	2022.0.0	3.9	4.2.0	0.8.1	1.2.3	12.1	13.0.0	11.6.0	5.0.2
acos	**0.523**	0.531	1.36	0.930	0.930	0.930	1.06	NA	1.53	0.772
acosh	2.25	**0.509**	1.30	2.25	2.25	2.25	2.25	NA	2.48	0.658
asin	**0.516**	0.531	1.05	0.981	0.981	0.981	0.746	NA	1.99	0.710
asinh	1.92	**0.502**	1.62	1.92	1.92	1.92	1.58	NA	2.52	0.656
atan	**0.523**	0.528	2.79	0.861	0.861	0.861	0.870	NA	1.76	1.73
atanh	1.81	**0.507**	1.02	1.81	1.81	1.80	2.01	NA	2.49	0.663
cbrt	3.67	0.523	0.502	0.670	0.668	0.668	0.729	NA	**0.501**	**0.501**
cos	**0.516**	0.518	1.28	0.887	0.834	0.834	0.947	Inf	1.52	0.797
cosh	1.93	**0.516**	1.81	2.67	1.47	1.04	0.523	NA	1.40	0.563
erf	1.43	**0.507**	1.43	1.02	1.02	1.02	6.22	NA	1.48	1.11
erfc	5.19	**0.505**	5.19	4.08	4.08	3.72	10.7	NA	4.42	4.06
exp	**0.511**	0.530	1.01	0.949	0.949	**0.511**	0.521	NA	0.904	0.927
exp10	2.01	0.538	Inf	0.896	NA	4.14	**0.521**	NA	1.10	1.11
exp2	**0.511**	0.535	1.01	0.895	0.751	**0.511**	0.521	NA	0.946	0.946
expm1	0.914	**0.512**	Inf	0.909	0.909	0.909	0.706	NA	1.18	1.91
j0	4.51e14	**0.600**	4.51e14	9.01e15	4.51e14	4.51e14	4.51e14	NA	3.36e19	1.25e13
j1	4.47e14	**0.615**	4.47e14	9.01e15	1.10e15	1.10e15	1.10e15	NA	4.26e19	4.80e13
lgamma	11.1	**0.515**	11.1	4.45e15	4.45e15	4.45e15	2.33e16	NA	5.11e15	4.45e15
log	0.520	0.518	0.562	0.946	0.946	0.520	**0.508**	NA	0.563	0.660
log10	1.62	0.532	1.09	2.08	0.814	0.814	**0.514**	NA	1.42	0.784
log1p	0.903	**0.521**	0.636	0.896	0.896	0.900	0.667	NA	1.50	1.00
log2	0.555	**0.504**	1.72	2.06	0.921	0.555	0.515	NA	1.30	0.734
sin	**0.516**	0.518	0.892	0.888	0.831	0.831	0.944	Inf	1.51	0.800
sinh	1.93	**0.521**	9.01e15	2.67	1.88	1.88	0.539	NA	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**
tan	0.619	**0.550**	1.36	1.02	1.02	1.02	3.53	2.43e19	2.07	1.30
tanh	2.22	**0.556**	1.40	2.22	2.22	2.22	0.613	NA	1.48	0.866
tgamma	8.47	**0.519**	8.47	2.27e3	1.03e3	16.0	1.03e3	NA	9.62	13.7
y0	5.93e15	**1.14**	5.93e15	1.42e15	1.42e15	1.42e15	1.42e15	NA	2.99e19	1.95e13
y1	5.56e15	**1.25**	5.56e15	5.56e15	5.56e15	5.56e15	5.56e15	NA	5.85e19	6.14e13
atan2	**0.524**	0.548	2.13e9	1.55	1.55	1.55	0.747	NA	1.76	1.80
hypot	0.792	0.751	4.51e15	1.21	1.21	1.04	1.21	**0.500**	1.88	1.21
pow	**0.523**	1.73	0.754	636.	636.	0.525	0.757	NA	1.40	1.40

Table 3: Double precision: Maximal known error.

function	GNU libc 2.36 x	max e	IML 2022.0.0 x	max e
acos	0x1.dffffb3488a4p-1	0.523	0x1.6c05eb219ec46p-1	0.531
acosh	0x1.0001ff6afc4bap+0	2.25	0x1.018dfa697553p+0	0.509
asin	-0x1.0000045b2c904p-3	0.516	0x1.6c042a6378102p-1	0.531
asinh	-0x1.02657ff36d5f3p-2	1.92	0x1.00038856b259ep-4	0.504
atan	0x1.f9004c4fef9eap-4	0.523	-0x1.ffff8020d3d1dp-7	0.528
atanh	0x1.f5805b28679f4p-4	1.81	-0x1.e2cfb2667f17ep-9	0.507
cbrt	0x1.7a337e1ba1ec2p-257	3.67	-0x1.f7af4893d1d51p-616	0.523
cos	-0x1.7120161c92674p+0	0.516	-0x1.d19ebc5567dcdp+311	0.518
cosh	-0x1.633c654fee2bap+9	1.93	-0x1.5a364e6b98134p+9	0.516
erf	0x1.c332bde7ca515p-5	1.43	0x1.00b4cd58903b2p+2	0.507
erfc	0x1.3ff2d63705b29p+0	5.19	0x1.5d164509e8235p-1	0.505
exp	-0x1.49f33ad2c1c58p+9	0.511	0x1.fce66609f7428p+5	0.530
exp10	0x1.334ab33a9aaep-2	2.01	-0x1.5cd9d94d49a85p+1	0.538
exp2	-0x1.1a4ce073ea908p-5	0.511	0x1.f3ffd85f33423p-1	0.535
expm1	0x1.62f69d171fa65p-2	0.914	-0x1.62fe464c64f65p-8	0.512
j0	0x1.33d152e971b4p+1	4.51e14	0x1.aff859518c846p+7	0.600
j1	-0x1.ea75575af6f09p+1	4.47e14	-0x1.67b5541c7d8b7p+7	0.615
lgamma	-0x1.f613ab0969f81p+1	11.1	-0x1.3f62c60e23b31p+2	0.515
log	0x1.1211bef8f68e9p+0	0.520	0x1.008000db2e8bep+0	0.518
log10	0x1.de02157073b31p-1	1.62	0x1.feda7b62c1033p-1	0.532
log1p	-0x1.2c10396268852p-2	0.903	0x1.000aee2a2757fp-9	0.521
log2	0x1.0b53197bd66c8p+0	0.555	0x1.00b0d7b252144p+0	0.504
sin	-0x1.f8b791cafcdefp+4	0.516	-0x1.0e16eb809a35dp+944	0.518
sinh	-0x1.633c654fee2bap+9	1.93	-0x1.adc135eb544c1p-2	0.521
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.317cd745dd37cp+9	0.619	0x1.49adfd996a81dp+18	0.550
tanh	-0x1.e134557098e37p-3	2.22	0x1.002629fd74484p+0	0.556
tgamma	-0x1.202e0f30fe00cp+3	8.47	-0x1.3e0001ad3bee3p+6	0.519
y0	0x1.c982eb8d417eap-1	5.93e15	0x1.4cdee58a47eddp-31	1.14
y1	0x1.193bed4dff243p+1	5.56e15	0x1.c513c569fe78ep+0	1.25
atan2	0x1.ed6060626eecfp-429 0x1.f42ebb62994dcp-426	0.524	0x1.b77ade79a36d5p-326 0x1.ff6a37b72b52bp-319	0.548
hypot	0x0.603e52daf0bfdp-1022 -0x0.a622d0a9a433bp-1022	0.792	0x0.19deaac345ffap-1022 0x0.92c8727c389b6p-1022	0.751
pow	0x1.010e2e7ee71aep+0 0x1.44bf0047427f6p+17	0.523	0x1.fffff9c61ce4p-1 0x1.c4e304ed4c734p+31	1.73

Table 4: Double precision: GNU libc and Intel Math Library.

function	AMD LibM 3.9 x	max e	RedHat Newlib 4.2.0 x	max e
acos	0x1.35d84799c86cap-1	1.36	-0x1.0068b067c6feep-1	0.930
acosh	0x1.20703c454bba3p+0	1.30	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.01bddfd228a2dp-1	1.05	-0x1.004d1c5a9400bp-1	0.981
asinh	0x1.001c939e14315p+0	1.62	-0x1.02657ff36d5f3p-2	1.92
atan	-0x1.05deacb86c0dbp+0	2.79	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.d7a5c53da7132p-2	1.02	-0x1.f97fabc0650c4p-4	1.81
cbrt	0x1.09806cdccbfa1p-748	0.502	-0x1.00ddafe7d9deep-885	0.670
cos	0x1.91b2e22984b85p-1	1.28	-0x1.4ae182c1ab422p+21	0.886
cosh	-0x1.ff1ecc8c7ea4fp+0	1.81	0x1.633cc2ae1c934p+9	2.67
erf	0x1.c332bde7ca515p-5	1.43	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.3ff2d63705b29p+0	5.19	0x1.5182d8799b84bp+0	4.08
exp	0x1.b97dbeb2777ccp+5	1.01	0x1.2e8f20cf3cbe7p+8	0.949
exp10	0x1.facf4856ce3c8p+491	Inf	0x1.ce7ef793d4b0ap-2	0.896
exp2	0x1.3bedfcb8aba8dp-1	1.01	-0x1.ff95ecb4e6331p-2	0.896
expm1	0x1.facf4856ce3c8p+491	Inf	0x1.62ff47a01658fp-2	0.909
j0	0x1.33d152e971b4p+1	4.51e14	0x1.45f3067a0f4b2p+847	9.01e15
j1	-0x1.ea75575af6f09p+1	4.47e14	0x1.45f3066f80258p+325	9.01e15
lgamma	-0x1.f613ab0969f81p+1	11.1	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.0ffead33dce6fp+0	0.562	0x1.48ae5a67204f5p+0	0.946
log10	0x1.10e857d0cb59dp+0	1.09	0x1.55535a0140a21p+0	2.08
log1p	0x1.e004312b997bp-4	0.636	-0x1.2bf1de6b04a8ap-2	0.896
log2	0x1.0b51de9e20b25p+0	1.72	0x1.68d778f076021p+0	2.06
sin	0x1.86485d1a8b19dp-1	0.892	-0x1.842d8ec8f752fp+21	0.888
sinh	0x1.fffffffffffep-26	9.01e15	-0x1.633cae1335f26p+9	2.67
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.b5825e79ac164p+12	1.37	0x1.3f9605aaeb51bp+21	1.02
tanh	0x1.fd619059e2342p-1	1.40	-0x1.e134557098e37p-3	2.22
tgamma	-0x1.202e0f30fe00cp+3	8.47	-0x1.535175475cc8dp+7	2.27e3
y0	0x1.c982eb8d417eap-1	5.93e15	0x1.c982eb8d417eap-1	1.42e15
y1	0x1.193bed4dff243p+1	5.56e15	0x1.193bed4dff243p+1	5.56e15
atan2	0x0.00000003ad06cp-1022 0x0.102efd000411bp-1022	2.13e9	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	-0x0.fffffffffffffp-1022 0x0.0000000000001p-1022	4.51e15	0x1.6a0a41410b1abp-1004 -0x0.a24afe71b539fp-1022	1.21
pow	0x1.0320265433e91p+4 -0x1.fcc785238238fp+7	0.754	0x1.000002c5e2e99p+0 0x1.c9eee35374af6p+31	636.

Table 5: Double precision: AMD LibM and RedHat Newlib.

function	OpenLibm 0.8.1 x	max e	Musl 1.2.3 x	max e
acos	-0x1.0068b067c6feep-1	0.930	-0x1.0068b067c6feep-1	0.930
acosh	0x1.0001ff6afc4bap+0	2.25	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.004d1c5a9400bp-1	0.981	-0x1.004d1c5a9400bp-1	0.981
asinh	-0x1.02657ff36d5f3p-2	1.92	-0x1.0240f2bdb3f25p-2	1.92
atan	0x1.62ff6a1682c25p-1	0.861	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.f97fabc0650c4p-4	1.81	-0x1.f8a404597baf4p-4	1.80
cbrt	-0x1.13a5ccd87c9bbp+1008	0.668	-0x1.13a5ccd87c9bbp+1008	0.668
cos	-0x1.34e729fd08086p+21	0.834	-0x1.34e729fd08086p+21	0.834
cosh	-0x1.6310ab92794a8p+9	1.47	-0x1.502bf5ad80729p+0	1.04
erf	-0x1.c57541b55c8ebp-16	1.02	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.5182d8799b84bp+0	4.08	0x1.527f4fb0d9331p+0	3.72
exp	0x1.2e8f20cf3cbe7p+8	0.949	-0x1.18209ecd19a8cp+6	0.511
exp10	NA	NA	-0x1.fe8c27141c94ap+3	4.14
exp2	-0x1.ff1eb5acee46bp+9	0.751	-0x1.1a4ce073ea908p-5	0.511
expm1	0x1.62ff47a01658fp-2	0.909	0x1.62ff47a01658fp-2	0.909
j0	0x1.33d152e971b4p+1	4.51e14	-0x1.33d152e971b4p+1	4.51e14
j1	-0x1.ea75575af6f09p+1	1.10e15	0x1.ea75575af6f09p+1	1.10e15
lgamma	-0x1.3a7fc9600f86cp+1	4.45e15	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.48ae5a67204f5p+0	0.944	0x1.dc0b586f2b26p-1	0.520
log10	0x1.553e1cb579ee9p+0	0.814	0x1.553e1cb579ee9p+0	0.814
log1p	-0x1.2bf1de6b04a8ap-2	0.896	-0x1.2bf32aaf122e2p-2	0.900
log2	0x1.67eaf07ce24d1p+0	0.921	0x1.0b53197bd66c8p+0	0.555
sin	0x1.4d84db080b9fdp+21	0.831	0x1.4d84db080b9fdp+21	0.831
sinh	-0x1.63324af2fb5b7p-1	1.88	-0x1.63324af2fb5b7p-1	1.88
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	0x1.3f9605aaeb51bp+21	1.02	0x1.3f9605aaeb51bp+21	1.02
tanh	-0x1.e134557098e37p-3	2.22	-0x1.e134557098e37p-3	2.22
tgamma	-0x1.540b170c4e65ep+7	1.03e3	-0x1.fc4b534c8eccp+2	16.0
y0	0x1.c982eb8d417eap-1	1.42e15	0x1.c982eb8d417eap-1	1.42e15
y1	0x1.193bed4dff243p+1	5.56e15	0x1.193bed4dff243p+1	5.56e15
atan2	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	0x1.6a0a41410b1abp-1004 -0x0.a24afe71b539fp-1022	1.21	0x1.00014d4b1c6b9p-1015 -0x1.000105ba9bf4p-1015	1.04
pow	0x1.000002c5e2e99p+0 0x1.c9eee35374af6p+31	636.	0x1.010e2e7ec0c83p+0 0x1.44bf00479249dp+17	0.525

Table 6: Double precision: OpenLibm and Musl.

	Apple 12.1		LLVM 14.0.6	
function	x	max e		
acos	-0x1.8d313198a2e03p-53	1.06	NA	NA
acosh	0x1.00007fb3703ddp+0	2.25	NA	NA
asin	0x1.eaeb8b58c0655p-2	0.746	NA	NA
asinh	-0x1.fdefd03df4cd7p-3	1.58	NA	NA
atan	-0x1.13ff259eb9ca8p+1	0.870	NA	NA
atanh	0x1.ffd834a270fp-10	2.01	NA	NA
cbrt	0x1.facf4856ce3c8p+491	0.729	NA	NA
cos	0x1.2f29eb4e99fa2p+7	0.948	-0x1.13a5ccd87c9bbp+1008	Inf
cosh	-0x1.62dabd4848dc4p-2	0.523	NA	NA
erf	-0x1.e057e7a0e494cp-2	6.41	NA	NA
erfc	0x1.bba14dc3507ccp+1	10.7	NA	NA
exp	-0x1.4133f4fd79c1cp-13	0.521	NA	NA
exp10	-0x1.c37443e446523p-16	0.521	NA	NA
exp2	-0x1.b3d9b47ad1b2fp-13	0.521	NA	NA
expm1	0x1.e7f93188565ecp-5	0.706	NA	NA
j0	0x1.33d152e971b4p+1	4.51e14	NA	NA
j1	-0x1.ea75575af6f09p+1	1.10e15	NA	NA
lgamma	-0x1.bffcbf76b86fp+2	2.33e16	NA	NA
log	0x1.490af72a25a81p-1	0.508	NA	NA
log10	0x1.2501ee5628b08p-1	0.514	NA	NA
log1p	-0x1.ffffff3fffffdp-28	0.667	NA	NA
log2	0x1.6b015f8d9a784p-1	0.515	NA	NA
sin	-0x1.07e4c92b5349dp+4	0.944	-0x1.13a5ccd87c9bbp+1008	Inf
sinh	0x1.d7131e11fc6b3p-2	0.539	NA	NA
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.a81d98fc58537p+6	3.53	-0x1.bab7afea4e624p+62	2.43e19
tanh	0x1.00cf9f273d84p+1	0.613	NA	NA
tgamma	-0x1.540b170c4e65ep+7	1.03e3	NA	NA
y0	0x1.c982eb8d417eap-1	1.42e15	NA	NA
y1	0x1.193bed4dff243p+1	5.56e15	NA	NA
atan2	-0x1.6a539153430d8p-416 0x1.d2b5b9dc716d8p-415	0.747	NA	NA
hypot	0x1.6a0a41410b1abp-1004 -0x1.4495fce36a73ep-1023	1.21	0x1.a308e1455f447p+0 0x1.9d931a83ef879p+0	0.500
pow	0x1.111616f835fb1p-72 0x1.c6cfa07925d49p+3	0.757	NA	NA

Table 7: Double precision: Apple and LLVM.

function	CUDA 11.6.0 x	max e	ROCm 5.0.2 x	max e
acos	0x1.2666588897b44p-1	1.53	-0x1.36b1482765f6dp-1	0.772
acosh	0x1.1d54c596b4841p+0	2.48	0x1.0aaabb1c24e51p+0	0.658
asin	0x1.2f0ac0bf00139p-1	1.99	0x1.df27e1c764802p-2	0.710
asinh	-0x1.0a1c1f63bd51ap-1	2.52	0x1.2ab2623eb028bp-2	0.656
atan	-0x1.5255c29696368p+0	1.76	-0x1.0684fa9fa7481p+0	1.73
atanh	0x1.f55ef33b30c76p-3	2.49	-0x1.24947129273cap-3	0.663
cbrt	-0x1.408204d3c2ea4p+238	0.501	0x1.1e0ef6faa076p+175	0.501
cos	0x1.9c0d0ef9b225ap+7	1.52	0x1.2a33ae49ab15dp+1	0.797
cosh	-0x1.e7ff7e05eb336p+1	1.40	-0x1.e7fa36b6eb43p+1	0.563
erf	-0x1.3449179197e69p-2	1.48	0x1.11bc66429fd1ep+0	1.11
erfc	0x1.1523c9af3d2bbp-6	4.42	0x1.ee237d05a9d5p-19	4.06
exp	0x1.e7fe6bd658aa2p+1	0.904	-0x1.625f224158408p+9	0.927
exp10	0x1.5c219f5a59907p+0	1.10	0x1.5c1ecc70739p+0	1.11
exp2	-0x1.ff3dab124ed1cp+9	0.946	-0x1.ff3e56ca91531p+9	0.946
expm1	0x1.29728a0720ca1p+5	1.18	0x1.632cfb1033275p-2	1.91
j0	-0x1.277684658e8f8p+25	3.36e19	0x1.ddca13ef271d2p+3	1.25e13
j1	0x1.06744d0d5c3aap+26	4.26e19	0x1.aa5baf310e5a2p+3	4.80e13
lgamma	-0x1.fa471547c2fe5p+1	5.11e15	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.69f3bc7a473cdp-1	0.563	0x1.5556123e9d684p-1	0.660
log10	0x1.803d81f3aac43p-1	1.42	0x1.5555808b2d4p+0	0.784
log1p	-0x1.fffffbaefe27p-2	1.50	-0x1.5efad5491a79bp-1022	1.00
log2	0x1.677b4718e3c41p+0	1.30	0x1.5556d5fbb94cbp+0	0.734
sin	0x1.cfa11e400ebefp+16	1.51	-0x1.f05e952d81b89p+5	0.800
sinh	0x1.be8da306a2538p+0	1.51	-0x1.ff9faf1f76d29p-5	0.868
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	0x1.002fe47823e82p+12	2.07	-0x1.66af736e8555p+18	1.30
tanh	-0x1.1931761a1bbe8p-1	1.48	-1 0x1.003d20e8473fdp-4	0.866
tgamma	-0x1.2ba89f2297cf1p+7	9.62	-0x1.200766d03438dp+2	13.7
y0	0x1.3f5cf973a65c9p+25	2.99e19	0x1.ab8e1c4a1e74ap+3	1.95e13
y1	0x1.512c1dcdad492p+26	5.85e19	0x1.e9e480605283cp+4	6.14e13
atan2	0x1.9cde4ff190e45p+931 0x1.37d91467e558bp+931	1.76	0x1.a7493caaab39dp-690 0x1.a2d221ea1ef97p-690	1.80
hypot	-0x1.587b2ce3e8f17p+33 0x1.7a8b3312418c8p+33	1.88	-0x1.de07b719ee7p-423 -0x1.6a11905a3d941p-407	1.21
pow	0x1.cf2685420aa04p+839 -0x1.fa0a63dc66fb7p-8	1.40	0x1.9e6221042f2cbp-828 0x1.00d1798b91fd2p-7	1.40

Table 8: Double precision: CUDA and ROCm.

double functions: `exp10`, `j0`, `j1`, `y0` and `y1`, its `powl` does not seem to be thread-safe, its `tgammal` function yields `+Inf` for `x=-0x6.db747ae147ae148p+8l` instead of `0xe.deaa8ed2a29cp-16396l`. Musl does not provide `j0`, `j1`, `y0`, and `y1` either.

The Apple Darwin ABI for ARM processors maps the C long double type to double, thus there is no real "double extended" format.

The LLVM-libc library only implements the square root function in double-extended precision, and for this function we could not find any error larger than 0.5 ulp (for rounding to nearest). Since a single function is implemented, we don't mention LLVM-libc in Tables 9 to 11.

5 Quadruple Precision

Only the GNU libc and the Intel Math Library support quadruple precision, through the `_Float128` type in GNU libc, and `_Quad` in the Intel Math Library (using the option of the Intel C compiler `-Qoption,cpp,--extended_float_types`). The results are summarized in Table 12, and detailed in Table 13. Only the square root function is correctly rounded (or at least seems to be). The Intel Math Library gives better results than the GNU libc for all functions, except for `lgamma` and `tgamma`. Apart from those two functions, and from the Bessel functions `j0`, `j1`, `y0`, `y1`, the observed error for the Intel Math Library is at most 1.4 ulps. The GNU libc has large errors for `j0`, `j1`, `y0` and `y1`.

Acknowledgements. The authors thank Claude-Pierre Jeannerod and Vincent Lefèvre who helped to improve that article, Alexei Sibidanov who helped to compile Newlib, Eric Schneider and Nick Timmons for interesting discussions. Joseph Myers suggested to included the double extended format. Experiments presented in this article were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). This work was also supported by the French "Ministère de l'Enseignement Supérieur et de la Recherche", by the "Conseil Régional de Lorraine", and by the European Union, through the "Cyber-Entreprises" project. Access to the Apple Math library was possible thanks to the GCC Compile Farm. Experiments on GPU were performed on hardware made available by CERN.

library	GNU libc	Intel Math Library	OpenLibm	Musl
version	2.36	IML 2022.0.0	0.8.1	1.2.3
acos	1.75	**0.505**	0.938	1.75
acosh	2.99	**0.502**	3.14	2.99
asin	1.15	**0.506**	1.03	2.00
asinh	2.96	**0.506**	3.19	2.96
atan	0.640	**0.501**	1.10	0.640
atanh	2.88	**0.501**	85.4	3.19
cbrt	0.824	**0.503**	0.890	0.890
cos	1.51	**0.502**	0.799	0.799
cosh	3.40	**0.502**	4.86	3.73
erf	1.17	**0.518**	1.17	1.17
erfc	4.73	**0.527**	5.77	5.12
exp	1.27	**0.501**	2.00	1.54
exp10	1.50	**0.501**	NA	40.1
exp2	0.788	**0.501**	2.18	0.788
expm1	3.08	**0.502**	1.94	9.71e3
j0	9.79e17	**0.501**	NA	NA
j1	3.38e18	**0.500**	NA	NA
lgamma	12.2	**0.549**	9.08e19	9.08e19
log	0.998	**0.501**	1.22	0.998
log10	1.36	**0.502**	1.22	1.36
log1p	2.49	**0.501**	2.60	2.49
log2	0.995	**0.502**	1.64	0.995
sin	1.51	**0.502**	0.798	0.798
sinh	3.40	**0.503**	4.85	9.71e3
sqrt	**0.500**	**0.500**	**0.500**	**0.500**
tan	1.75	**0.504**	1.02	1.02
tanh	3.22	**0.506**	2.56	2.95
tgamma	9.77	**0.554**	Inf	3.69e19
y0	1.38e18	**0.500**	NA	NA
y1	4.61e18	**0.500**	NA	NA
atan2	0.751	**0.501**	1.69	0.751
hypot	**0.584**	0.751	0.981	1.08
pow	0.914	**0.501**	533.	533.

Table 9: Double extended precision: Maximal known error.

function	GNU libc 2.36 x	max e	IML 2022.0.0 x	max e
acos	0xf.fe002cabd608585p-4l	1.75	0x8.af256cd27462348p-4l	0.505
acosh	0x1.1ecdb5b8f0c5d79p+0l	2.99	0x1.1f9c4feedfe4f2cp+0l	0.502
asin	0x8.171fd358c4cb27bp-4l	1.15	-0x8.018aef8787e5a6bp-4l	0.506
asinh	-0x8.0bb656992eac437p-4l	2.96	0x7.ff15da44c3651abp-4l	0.506
atan	-0x1.0411ae010d4c5b1ep+0l	0.640	-0x8.00f60592e42d79p+8l	0.501
atanh	-0x3.337ceaccc9025258p-4l	2.88	0x3.e7be418257523408p-4l	0.501
cbrt	-0xc.f4fd71a450e6a0bp-14732l	0.824	-0x2.320375fd33ed311cp-13376l	0.503
cos	-0x3.d067a048093bdf94p+9160l	1.51	-0x4.b0df0d7d55044918p+8l	0.502
cosh	0x2.c5d375f827733ac4p+12l	3.40	-0x7.f6a09874512cf768p-4l	0.502
erf	0xd.7fe64ab05cf75e8p-4l	1.17	-0x1.c55160e785ee1cbap-4l	0.518
erfc	0x1.59723d7ee47e3034p+0l	4.73	0x3.03c7b9f943690558p-4l	0.527
exp	0x5.8b9111182b4467ep-4l	1.27	0x2.c590e6ab0d71c77p+12l	0.501
exp10	0x1.2da9675e95849c3ep+12l	1.50	-0x1.2ab76ac25255a1aap+12l	0.501
exp2	-0x7.3f819acf048f1678p-4l	0.788	-0x3.fe9a346527a75d98p-16l	0.501
expm1	0x5.8b910bbe3c26818p-4l	3.08	-0x1.0040016b56008656p-8l	0.502
j0	-0x2.67a2a5d2e367f784p+0l	9.79e17	-0x1.6a09e667f3bd238cp-32l	0.501
j1	0x3.d4eaaeb5ede115p+0l	3.38e18	-0x1.8p-16444l	0.500
lgamma	-0x3.ec9403f23a1f21cp+0l	12.2	-0x4.07fe15510b6a28p+0l	0.549
log	0x1.20dad075f537ae56p+0l	0.998	0x1.1001246349edf00cp+0l	0.501
log10	0x1.272b7c3bbb08ae12p+0l	1.36	0x1.010141e1049fce68p+0l	0.502
log1p	-0x6.451f6c3fd0d4a218p-4l	2.49	-0xe.fefa23913fa3eb7p-8l	0.501
log2	0x1.058f12b8b3ac44bep+0l	0.995	0x1.01004bfffe4316bep+0l	0.502
sin	-0x6.e2368c0ed74e5698p+16l	1.51	-0xc.141cf155623856bp+8l	0.502
sinh	0x2.c5d375f827733ac4p+12l	3.40	0x7.b0af44fc25df3efp-4l	0.503
sqrt	0xf.ffffffffffffffp-4l	0.500	0xf.ffffffffffffffp-4l	0.500
tan	0x1.974ccdb290851e7cp+8l	1.75	0xc.845cb771b06f4c5p+0l	0.504
tanh	0x3.b9979a543d0fbfa8p-4l	3.22	0x7.fb808a1ef99076ep-4l	0.506
tgamma	-0x1.70a55b2628a7cb68p+4l	9.77	-0x6.cc7ff7f0fb0649ap+8l	0.554
y0	0xe.4c175c6a0bf51e8p-4l	1.38e18	0xe.fddf6a6afc1efccp-12440l	0.500
y1	0xb.bfc89c6a1903022p+0l	4.61e18	0xd.749961e354cf884p-4284l	0.500
atan2	-0x7.9301460b8463cbp+15368l 0xf.25cd5eb1280b4d1p+15372l	0.751	-0x5.c0c9cc5a59632f88p+16340l 0x5.db7810fba1ce4908p+16348l	0.501
hypot	-0x2.97b86706043d619p+7240l 0x1.8256bdd12d2e163ep+7240l	0.584	-0x3.00bad8a56d87a0cp-16384l -0xe.6d794db04791398p-16388l	0.751
pow	0x2.21dda4bcec55b158p-3616l 0x7.ef1ef5fbe3df50dp-16l	0.914	0xc.b80572af668bb57p+152l -0x6.8a6d3d7b442f3c18p+4l	0.501

Table 10: Double extended precision: GNU libc and Intel Math Library.

function	OpenLibm 0.8.1 x	max e	Musl 1.2.3 x	max e
acos	-0x8.040541d0054d89p-4l	0.938	0xf.fe002cabd608585p-4l	1.75
acosh	0x1.10384b24aec007fcp+0l	3.14	0x1.1ecdb5b8f0c5d79p+0l	2.99
asin	0x8.0519515d1e15a6bp-4l	1.03	-0x3.fff0a397b8dea17cp-8l	2.00
asinh	-0x5.c9866cb231f2c7c8p-4l	3.19	-0x8.0bb656992eac437p-4l	2.96
atan	0x6.fffde214a06fb5f8p-4l	1.10	-0x1.0411ae010d4c5b1ep+0l	0.640
atanh	-0xf.fffffffffffe78p-32l	85.4	0x3.344a915e34e5e6b8p-4l	3.19
cbrt	-0x3.fffffffa5623708p+4588l	0.890	-0x3.fffffffa5623708p+4588l	0.890
cos	0x3.e0dc8477d8e9d7acp+4l	0.799	0x3.e0dc8477d8e9d7acp+4l	0.799
cosh	0x2.c5d374f9436efd1p+12l	4.86	0x2.c5d37484e4c162bp+12l	3.73
erf	0xd.7fe64ab05cf75e8p-4l	1.17	0xd.7fe64ab05cf75e8p-4l	1.17
erfc	0x1.5cc0e1cc32a3dc98p+0l	5.77	0x1.5c9262fa4210902p+0l	5.12
exp	0x8.aa2253c0d601dedp+0l	2.00	-0x2.c5a1073a0f38b61cp+12l	1.54
exp10	NA	NA	0xd.41cfea690e121b5p+8l	40.1
exp2	-0xf.ffffd9f32ee1e06p-12l	2.18	-0x7.3f819acf048f1678p-4l	0.788
expm1	0x6.63ceda63b727c8d8p-4l	1.94	0x2.c5c85fdf170c604cp+12l	9.71e3
j0	NA	NA	NA	NA
j1	NA	NA	NA	NA
lgamma	-0x2.74ff92c01f0d82acp+0l	9.08e19	-0x2.74ff92c01f0d82acp+0l	9.08e19
log	0xb.504a14384e9b137p-4l	1.22	0x1.20dad075f537ae56p+0l	0.998
log10	0xb.ffac4b4c47e00c3p-4l	1.22	0x1.272b7c3bbb08ae12p+0l	1.36
log1p	-0x4.c669bd1813ec8bd8p-4l	2.60	-0x6.451f6c3fd0d4a218p-4l	2.49
log2	0x1.6646b082fd1065cep+0l	1.64	0x1.058f12b8b3ac44bep+0l	0.995
sin	-0x2.a2a4aca336af4538p+8l	0.798	-0x2.a2a4aca336af4538p+8l	0.798
sinh	-0x2.c5d375cbe7e4a81cp+12l	4.85	0x2.c5c85fdbc1ccc354p+12l	9.71e3
sqrt	0xf.ffffffffffffffp-4l	0.500	0xf.ffffffffffffffp-4l	0.500
tan	-0x6.fae4525c1c348edp+8l	1.02	-0x6.fae4525c1c348edp+8l	1.02
tanh	0x3.8b2602d43bdf4c28p-4l	2.56	0x4.024182351388d15p-4l	2.95
tgamma	-0x6.db747ae147ae148p+8l	Inf	-0x2.8d19fd20f3aa62cp+4l	3.69e19
y0	NA	NA	NA	NA
y1	NA	NA	NA	NA
atan2	0x3.d34c9d81dcd29354p+5568l 0xf.3afc4f6c9f5c4a2p+5568l	1.69	-0x7.9301460b8463cbp+15368l 0xf.25cd5eb1280b4d1p+15372l	0.751
hypot	0x1.73f339f61eda21dp-16384l 0x2.e45f9f9500877e2p-16384l	0.981	0x2.00007da75fd5903cp-8960l 0x2.d42207352184bff4p-8960l	1.08
pow	0xc.f620c9ea4p+16380l -0x4.0ffffcp-48l	533.	0xc.f620c9ea4p+16380l -0x4.0ffffcp-48l	533.

Table 11: Double extended precision: OpenLibm and Musl.

library version	GNU libc 2.36	Intel Math Library IML 2022.0.0
acos	1.28	**0.502**
acosh	4.00	**0.501**
asin	1.20	**0.502**
asinh	3.95	**0.501**
atan	1.41	**0.501**
atanh	3.89	**0.501**
cbrt	0.736	**0.501**
cos	1.52	**0.501**
cosh	1.92	**0.501**
erf	1.42	**0.501**
erfc	4.38	**0.504**
exp	0.751	**0.501**
exp10	2.00	**0.501**
exp2	1.08	**0.501**
expm1	1.64	**0.501**
j0	4.10e32	**2.90e28**
j1	3.57e33	**3.33e31**
lgamma	**13.0**	2.79e30
log	1.05	**0.501**
log10	2.01	**0.501**
log1p	3.51	**0.501**
log2	3.31	**0.501**
sin	1.52	**0.501**
sinh	2.07	**0.501**
sqrt	**0.500**	**0.500**
tan	1.06	**0.502**
tanh	2.39	**0.501**
tgamma	**10.7**	8.20e3
y0	1.69e33	**4.79e27**
y1	3.47e33	**1.45e30**
atan2	1.89	**0.501**
hypot	0.749	**0.501**
pow	30.3	**1.40**

Table 12: Quadruple precision: Maximal known error.

function	GNU libc 2.36 x	max e	IML 2022.0.0 x	max e
acos	0x9.fdbe71e81d65064f0f24b2602998p-4	1.28	0xf.f80616c2416bf63c33a739ae3a08p-4	0.502
acosh	0x1.0f97586eba090200118df0902f99p+0	4.00	0x1.004ae7a1e9d7b621b12baeda616dp+0	0.501
asin	0x7.79659a0b568bad280c8ec7eb8278p-4	1.20	0x7.ff86cc20db4e6f7fd33ce212282cp-8	0.502
asinh	0x5.a924236647ffb723576b172b52fcp-4	3.95	0x1.0000f6bea05a0cafd1e775e627d3p-4	0.501
atan	0x3.7ff864717fc99760d470d1a994cp-4	1.41	-0x1.15eb4e54ee6ca35bf8b1764f30d4p+0	0.501
atanh	0x2.c02a24f3472c7840afbd8cfb68bap-4	3.89	-0xd.9fe29c463116c87fa567e436489p-8	0.501
cbrt	-0x5.a837d1198a72e5a89695db79896cp-13792	0.736	-0x2.10d29fbb2036d1d7ffdd8bf63184p+10912	0.501
cos	-0x3.08db9df46e0cd142071fdec7eb6p+64	1.52	-0x6.081f6e15f81d27ac2a6038eed3bp+2232	0.501
cosh	-0x2.c5d376fd225ce5739bef59cb0e16p+12	1.92	-0x2.ba5adc2ddaf3f5466db2cd018394p+4	0.501
erf	0xd.f3a140b19b0e7d0fafae7eec5ebp-4	1.42	0x5.a5182e2e3fce6963a492839ebb3cp-8	0.501
erfc	0x1.517e84504890cba9f9f65ff93206p+0	4.38	0x6.0a5ca72c4efcd78f90acc0aefbbp+0	0.504
exp	-0x2.c5b323ac8f24d66ed41ee61ab6bap+12	0.751	-0x5.6622c128e27c6a8c991743947adcp-8	0.501
exp10	0x3.e9d3cc7e0cbdc5bc7fdfc1932fd6p+0	2.00	0x1.1e2a2ef09a4f66e4d3648a85045bp+12	0.501
exp2	0x1.fffe69758fd951b5213a6d47be1ap+0	1.08	-0x7.cab667376a3dd98217d7b028adccp-8	0.501
expm1	0x5.a1195b05aec378d0b236943f4a18p-4	1.64	0x8.ca3ec068eee81b45c0adcae049ap+4	0.501
j0	-0x8.a75ab6666f64eae68f8eb383dad8p+0	4.10e32	0x3.7c3f883498c0d5e0dab7e54a98b2p+4	2.90e28
j1	-0x1.7059c8d303730c6b82b12d9941b9p+8	3.57e33	-0x1.7059c8d303730c6b82b12d9941b9p+8	3.33e31
lgamma	-0x3.ec2152452b5eaf0f070d215b3418p+0	13.0	-0x3.24c1b793cb35efb8be699ad3d9bap+0	2.79e30
log	0xf.d016f49074a9c4fe793af2394278p-4	1.05	0xc.4806c5e4877bbeb4b44ed03d9f18p-5364	0.501
log10	0x1.6a291ea0aa11fb374f1df8b3ac6bp+0	2.01	0x1.9b621e77f399e4a8c1a85a964e94p-12364	0.501
log1p	0x6.a0aed5f6dad05d6ff33ecd883dc8p-4	3.51	-0x6.2611e37be5cf4388865319f859b4p-12	0.501
log2	0xb.54170d5cfa8fd72a47d6bda19068p-4	3.31	0xf.f63cee8e97ac6783532625273eap-4	0.501
sin	0x5.6a5005df151cc2274e119666a9c8p+64	1.52	0x4.246e3c1f1094e4159999f13cff24p+5604	0.501
sinh	0x6.7e79f3aada38698b910c300b19b8p-4	2.07	-0x1.6606d9c89bc66d481844a8589dcbp+0	0.501
sqrt	0xf.ffffffffffffffffffffffffffff8p-4	0.500	0xf.ffffffffffffffffffffffffffff8p-4	0.500
tan	-0x3.832b771f9462df46117b6a863fa2p+8	1.06	0xb.eb95e948d6f2a74a1d3a7694bd88p+3816	0.502
tanh	-0x3.c26abeca541298cca288adbd1e12p-4	2.39	-0x2.01d7bf6773e2b04acd388c84cd4ep-4	0.501
tgamma	-0x1.62ab0823decc5cf957d9a218cf27p+4	10.7	0x2.00003274fc8659f8ed68e96e0378p-16224	8.20e3
y0	0x6.b99c822052e965e1754eb5ffeb08p+4	1.69e33	0x3.9561432d16442ec543c74876d1c8p+4	4.79e27
y1	0x2.3277da9bfe485c85c35e5bcc806p+0	3.47e33	0x2.80bc307275f6a6a3feb2ab211838p+4	1.45e30
atan2	0x1.41df5aa214612c7e019fa6ade88p-13316 0x5.e53b26a270a29eb9f77ef8ef7af8p-13316	1.89	-0x1.fb41ff205f5ade930a9fcbba8ea8p-16384 0x2.23f098fd6b8799dbeb03219bfa08p-10520	0.501
hypot	0x2.2d5faf4036d6e68566f01054612p-8192 0x3.5738e8e2505f5d1fc2973716f05p-8192	0.749	0x8.79ec30b61f9b839fe507bbdf414p-11908 0xb.94f6832f64d0729ebd68035ed7a8p-11908	0.501
pow	0x1.364dcbbad0512d7bacaae2a8d56bp+0 -0xe.68759219434c37725fdf30d17d2p+12	30.3	0x4p-16496 0x3.ffffff39c102f0aa11bb2c8a91dp-128	1.40

Table 13: Quadruple precision: GNU libc and Intel Math Library.

References

[1] AMD LibM version 3.9. https://developer.amd.com/amd-aocl/amd-math-library-libm/, 2022.

[2] Apple Math Library (MacOS 12.1, Apple M1).

[3] Arnold, J. M. A study of the rsqrt and rcp instructions on Intel and AMD platforms. https://github.com/jeff-arnold/math_routines.git, 2016. 22 pages.

[4] Bailey, D. H. Variable precision computing: Applications and challenges. Slides presented at the ICERM workshop on Variable Precision in Mathematical and Scientific Computing, 2020. https://www.davidhbailey.com/dhbtalks/dhb-icerm-2020.pdf.

[5] Black, C. M., Burton, R. P., and Miller, T. H. The need for an industry standard of accuracy for elementary-function programs. *ACM Trans. Math. Softw. 10*, 4 (1984), 361–366.

[6] CUDA C Programming Guide v11.6.0, Section H Mathematical Functions. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#mathematical-functions-appendix, 2022.

[7] CUDA Math Library. https://developer.nvidia.com/cuda-math-library, 2022.

[8] Ferguson, W., Cornea, M., Anderson, C., and Schneider, E. The difference between x87 instructions fsin, fcos, fsincos, and fptan and mathematical functions sin, cos, sincos, and tan, 2015. https://software.intel.com/content/dam/develop/external/us/en/documents/x87trigonometricinstructionsvsmathfunctions.pdf.

[9] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.

[10] Glatard, T., Lewis, L. B., da Silva, R. F., Adalat, R., Beck, N., Lepage, C., Rioux, P., Rousseau, M., Sherif, T., Deelman, E., Khalili-Mahani, N., and Evans, A. C. Reproducibility of neuroimaging analyses across operating systems. *Frontiers Neuroinformatics 9* (2014), 12.

[11] GNU libc 2.34: Known maximum errors in math functions. http://www.gnu.org/software/libc/manual/html_node/Errors-in-Math-Functions.html, 2021.

[12] GNU libc version 2.36. https://www.gnu.org/software/libc/, 2022.

[13] IEEE standard for floating-point arithmetic, 2019. 84 pages.

[14] Intel Math Library. Distributed with the Intel oneAPI DPC++ Compiler 2022.0.0, 2022.

[15] LLVM-libc C standard library 14.0.6. https://releases.llvm.org/, 2022.

[16] Muller, J.-M. On the definition of ulp(x). Research Report RR-5504, LIP RR-2005-09, INRIA, LIP, Feb. 2005.

[17] Musl version 1.2.3. https://musl.libc.org/, 2022.

[18] Redhat Newlib version 4.2.0. https://sourceware.org/newlib/, 2021.

[19] ROCm Math Library. https://github.com/RadeonOpenCompute/ROCm, 2022.

[20] OpenLibm version 0.8.1. https://openlibm.org/, 2022.

[21] Petzold, M. A note on the first moment of extreme order statistics from the normal distribution. Tech. rep., Göteborg University. School of Business, Economics and Law, 2000. 6 pages, https://gupea.ub.gu.se/handle/2077/3092.

[22] Sibidanov, A., Zimmermann, P., and Glondu, S. The CORE-MATH Project. In *ARITH 2022 - 29th IEEE Symposium on Computer Arithmetic* (virtual, France, Sept. 2022).