Numerical Algorithms **37:** 367–375, 2004.

Proposal for a standardization of mathematical function implementation in floating-point arithmetic

David Defour [a], Guillaume Hanrot [b], Vincent Lefèvre [b], Jean-Michel Muller [a], Nathalie Revol [a] and Paul Zimmermann [b]

[a] *Equipe DALI, LP2A, Université de Perpignan, France*
E-mail: David.Defour@univ-perp.fr, {Jean-Michel.Muller;Nathalie.Revol}@ens-lyon.fr
[b] *Project Spaces, LORIA/INRIA, 615 rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex, France*
E-mail: {Guillaume.Hanrot;Vincent.Lefevre;Paul.Zimmermann}@loria.fr

Some aspects of what a standard for the implementation of the mathematical functions could be are presented. Firstly, the need for such a standard is motivated. Then the proposed standard is given. The question of roundings constitutes an important part of this paper: three levels are proposed, ranging from a level relatively easy to attain (with fixed maximal relative error) up to the best quality one, with correct rounding on the whole domain of every function. We do not claim that we always suggest the right choices, or that we have thought about all relevant issues. The mere goal of this paper is to raise questions and to launch the discussion towards a standard.

Keywords: floating-point arithmetic, IEEE-754 standard, mathematical function, rounding, exception, implementation

AMS subject classification: 65G30, 65G50

1. Introductory discussion

We take the opportunity of the current discussion on the revision of the IEEE-754 Standard for floating-point arithmetic to discuss the possibility of standardizing (some of) the elementary functions. "Elementary" or "mathematical" functions are the functions usually available in a mathematical library. The IEEE-754 Standard [1] does not deal with these functions. This is due to several reasons, the most serious one being the problem of providing correctly rounded transcendentals, known as the Table Maker's Dilemma problem. Indeed, "ultimate" accuracy is expected from standardized functions, and seems hard to obtain at a reasonable cost in practice, at least in hardware. To quote Valerio [8, guest lecture No. 13],

> Committing an approximation function to hardware is unlikely to be an unqualified success. Delivering any result other than the mathematically correct result rounded to the destination format is open to criticism. Delivering a result more slowly than a software implementation can raise questions of why the function is in hardware. Dedicating significant amounts of chip area to support transcendental functions is usually better spent improving the speed of vector multiplication. In short, the silicon implementation should be fast, accurate, and cost nothing.

An unfortunate consequence of that current lack of standardization is that extremely poor libraries are still in use (see [17] or [16] for some examples). Nevertheless, the quality of most elementary function libraries has greatly improved during the last decade.

This paper proposes a standard for elementary functions, detailed in section 2. After a list of concerned functions (section 2.1) and features (relative error bound in section 2.2, options in section 2.3), the question of roundings is addressed (section 2.4): it constitutes the core of this proposal. We suggest three levels of quality, the lowest one (level 0) being regarded as the minimum acceptable level for a library, and the highest one representing the best quality that can be reached (on the whole, correct rounding in all the input domain). This hierarchy of levels is suggested because what is currently achievable is still far from the best. The end of this proposal contains two lists, the "exceptional" values (section 2.5) and the cases of exact results (section 2.6).

Our suggestions are based on the reading of some of the works of Kahan [7–10], the draft of the current revision of the IEEE-754 standard [6], drafts of other current standardization efforts [2,14], our own experience on studying elementary function implementation and recent progress of some of the authors of this paper concerning the Table Maker's Dilemma [12,13,18]. In the following, text defining the proposed standard is set off from comments by indentation from both margins and by a different font, as in [3].

Desirable properties and incompatibilities

High expectations are put on standardized functions and several properties would be desirable. Among them:

1. Correct rounding (for all rounding modes).

2. Preservation of the output range: e.g., one would like a sine to be between -1 and 1, an arctangent to be between $-\pi/2$ and $+\pi/2$, etc. Not satisfying this preservation could have nonnegligible consequences. If the output range of function f is $[a, b]$, a programmer might successively compute: $y = f(x)$, $z = g(y)$, with g defined on $[a, b]$ only. Hence, if the implementation for f returns a value of y out of that interval (even slightly), important errors might occur. For instance, let $f(x)$ be equal to $2 \arctan x/\pi$ and $g(x) = \arcsin x$. For any value of x, the sequence $y = f(x)$, $z = g(y)$ is mathematically well defined. If a value out of $[-1, 1]$ is returned for f evaluated on some x (see the end of this section), then an error will occur during computation.

3. Bounded error on every result, with a known bound. This bound should be explicitly given by the constructor for every function of a mathematical library. Except in the case of a subnormal result, a relative error, in ulps, should be given.

4. Preservation of monotonicity.

5. Preservation of symmetries (e.g., $\sin(-x) = -\sin x$).

6. Preservation of the direction of rounding when a directed rounding mode is selected, even if correct rounding cannot be satisfied.

7. Correct handling of exceptions and subnormal numbers.

8. Each time a function cannot be uniquely defined using continuity, a NaN should be returned: examples are $1^{\pm\infty}$ or $\sin\infty$. A possible exception is 0^0: it is not uniquely defined, but the convention "$0^0 = 1$" has the advantage of preserving some mathematical formulas, hence many authors suggest to keep it.

9. Compatibility with other standardization efforts, such as ISO/IEC 10967 (LIA and in particular LIA-2, the second part of LIA, which seems to still be under discussion [14]) and language standardization, e.g., ISO/IEC 9899 for the C programming language [2].

It is worth being noticed that these desirable properties are sometimes not compatible. For instance, in single precision and round-to-nearest mode, a correctly rounded routine for arctangent would return values larger than $\pi/2$ for input values large enough. The single precision number which is closest to $\arctan(2^{30})$ is

$$\frac{13176795}{8388608} = 1.57079637050628662109375 > \frac{\pi}{2}.$$

Therefore, if the arctangent function is implemented in round-to-nearest mode, we get an arctangent larger[1] than $\pi/2$. A consequence of this is that in such a system,

$$\tan\bigl(\arctan\bigl(2^{30}\bigr)\bigr) = -2.2877\ldots \times 10^7.$$

The same incompatibility exists between the range requirement and a directed rounding, should that be correct or not.

A more obvious example is the fact that with rounding modes towards $\pm\infty$, correct rounding and preservation of symmetries are not compatible; the same incompatibility occurs with the weaker requirement of preservation of the direction of rounding and of symmetries.

As to the compatibility with other norms, our proposal may differ in the following points:

- floating-point values: our proposal considers only floating-point arguments, whereas the LIA-2 standard also discusses irrational inputs;
- bounded relative errors: our first level (level 0 in section 2.4) is close to the requirements of the LIA-2 standard; the ISO/IEC 9899 standard for the C programming language does not mention this point;
- mathematical properties (monotonicity, symmetry): our proposal is more demanding than the LIA-2 standard, for which only the monotonicity must be preserved, and than the ISO/IEC 9899 standard for C, which does not address this issue;

[1] But *equal* to the machine representation of $\pi/2$.

- exceptions: most standards agree on exceptions handling, at the possible exception of some choices discussed in section 2.5. Our choices may be significantly different from those of other standards, such as ISO/IEC 9899, where pow(-1, ∞) returns 1 (instead of NaN).

2. What could be included in a standard

A first point which deserves to be noticed is that a standard should apply to any available precision, even if for the time being we are mostly interested in the single, double, extended double and quadruple floating-point precisions.

2.1. *Functions being considered here*

The functions concerned by this standardization proposal are functions for which the proposed standard (or at least some levels) are reachable, or functions listed in the LIA-2 standard (for compatibility with other standards):

$\log$, $\log_2$, $\log_{10}$, $\log_b x$ (even if for this function it is not yet known how to satisfy the standard's requirements), $\log(1+x)$, exp, $\exp(x)-1$, 2^x, 10^x, sin, cos, tan, cot, sec, csc, arcsin, arccos, arctan, $\arctan(x/y)$, arccot, arcsec, arccsc, x^y, sinh, cosh, tanh, coth, sech, csch, arcsinh, arccosh, arctanh, arccoth, arcsech, arccsch.

But most of what is said here could apply to special functions (gamma, erf, erfc, Bessel functions, etc.) and some algebraic functions such as reciprocal square root $x^{-1/2}$, cube root or hypotenuse $\sqrt{x^2+y^2}$.

2.2. *Bounded relative error*

Every computed value must have a fixed maximal relative error, i.e. every result has a guaranteed quality (see section 2.4 for more details). This error is an absolute one in the case of subnormal results.

This would be extremely useful for automatic forward error computation [11] and for interval arithmetic [5,15].

2.3. *Choice of range, monotonicity, symmetry: optional modes*

Since the various desirable properties of rounding (correct or not), preservation of the output range and symmetry can be mutually incompatible, the preferred property can be chosen as an option. The possible options are denoted in the following by `preserve-rounding`, `preserve-range` or `preserve-symmetry`, the default choice being `preserve-range.`

2.4. *Roundings*

We suggest to allow three levels of quality. Which level is actually provided should appear clearly in the documentation of the elementary function library (or hardware). It is of course allowable to provide all levels, the programmer being then able to select a tradeoff between quality and speed. In such a case, the default must be the highest available level.

Level 0. Faithful rounding and guaranteed relative error. In **round-to-nearest** mode denoted by $\circ$, the returned result must always be one of the two floating-point numbers that surround the exact result (if the exact result is a floating-point value – which is rare with the transcendental functions: see section 2.6 – the system must return that value). In the **round towards** $-\infty$ mode, denoted by ∇, the returned result must always be less than or equal to the exact result. No error greater than 1.5 ulps is allowed. In the **round towards** $+\infty$ mode denoted by Δ, the returned result must always be larger than or equal to the exact result. No error more than 1.5 ulps is allowed. The **round towards zero** mode, denoted by $\mathcal{Z}$, behaves as the round towards $-\infty$ mode for positive values and the round towards $+\infty$ mode for negative values. In all cases where the exact function is monotonic, *the implementation must be monotonic too.* In **round-to-nearest** and **round towards zero** modes, the symmetries of the function around 0 (properties of the kind $f(-x) = \pm f(x)$) must be preserved.[2]

Level 1. Correct rounding on a restricted range. There is a domain (usually around 0) where the implemented function is correctly rounded. Outside this domain, the implementation must satisfy the criteria of level 0. We suggest the domain should at least contain $[-2\pi, +2\pi]$ for sin, cos and tan; and $[-1, 1]$ for exp, cosh, sinh, 2^x and 10^x (other functions: to be discussed, to reach a compromise involving the facility of implementation and the usefulness of requirement).

Level 2. Correct rounding. Correct rounding in the whole domain where the function is mathematically defined. We might suggest the use of the `preserve-range` mode when output range has priority, which is not to be considered higher or lower in quality. In this case, correct rounding is provided unless this prevents preservation of output range: the closest floating-point number belonging to the output range is returned. Correct rounding cannot be incompatible with monotonicity. In case of correct *directed* rounding (Δ or ∇), it is assumed that the user is well aware of the incompatibility with symmetry, thus the `preserve-symmetry` mode is not available.

We consider level 0 as the minimum acceptable level.

The error thresholds given for level 0 can be replaced in a more general framework: the relative error must be upper bounded and the bound is $(1/2 + \tau)$ ulp for the round to nearest mode and $(1 + \tau)$ ulp for the other rounding modes. For level 0, the "tolerance" τ cannot exceed $1/2$ whereas for the more demanding level 2, τ is fixed to 0.

Level 2 is attainable at reasonable cost for at least some functions, in single and double precisions. To be correctly rounded, a result has to be computed using a higher precision than the precision of the returned value. The arguments for which this intermediate computing precision is maximal among all possible arguments are called "hardest-to-round cases". Tables giving the hardest-to-round cases for double precision exponentials and logarithms can be found in [12]. The hardest-to-round cases for the full double precision range are also already known for 2^x and $\log_2 x$. When the hardest-to-round cases are known, then it is possible to optimize the code for this function, in particular the size of memory needed is known.

[2] In practice this requirement is not a problem: function implementers will use these symmetries for simplifying their programs.

Finding the hardest-to-round cases for the trigonometric functions in double, extended and quadruple precisions might be difficult. In particular, it might be tricky to determine the hardest-to-round cases for functions with two arguments, typically x^y and $\log_b x$. Moreover, it is expected that arguments outside the prescribed range are expensive to deal with in terms of computing time, table's size (for table-based methods)..., because a range reduction is involved; this price may be considered as too high to pay. This is the reason for level 1.

Indeed, level 1 is proposed in order to provide a better level than the basic level 0, as long as level 2 remains out of reach (at an acceptable time overhead). However, it is not very satisfactory since it requires the highest quality only on an arbitrarily restricted range.

2.5. *Exceptions*

2.5.1. *Values that cannot be defined using continuity*

Sometimes, different choices are legitimate. What is important is consistency. Consider three examples.

The case of 0^0 is important. On the one hand, as said above, there is no way of defining 0^0 using continuity, but on the other hand, many important properties remain satisfied if we choose $0^0 = 1$ (which is frequently adopted, as a convention, by mathematicians). Kahan [9] suggests to choose $0^0 = 1$. A consequence of that (also mentioned by Kahan) is that it implies that $\text{NaN}^0 = 1$ whereas $0^{\text{NaN}} = \text{NaN}$, since x^0 is 1 for *any* x, whereas 0^y is 1 if $y = 0$ and 0 if $y > 0$. If we happen to choose that 0^0 is NaN (which is perfectly legitimate), then NaN^0 is NaN.

Another example is $\log(-0)$. On the one hand, as suggested by Goldberg [4], -0 may be thought as a small negative number that has underflowed to zero. This would favor the choice $\log(-0) = \text{NaN}$. On the other hand, such a choice would imply that we can have $x = y$ and $\log x \neq \log y$, with $x = +0$ and $y = -0$, since the IEEE-754 Standard requires that the comparison $-0 = +0$ returns *true*.

$1^{\pm\infty}$ is similar to 0^0. One can build $u_n \to 1$ and $v_n \to +\infty$ such that $u_n^{v_n}$ goes to anything you desire (or nothing at all). Kahan [9] suggests $1^{\pm\infty} = \text{NaN}$, which implies (for reasons of consistency) $1^{\text{NaN}} = \text{NaN}$.

2.5.2. *NaNs (as input or output values)*

All functions having at least one NaN as input value must return a NaN, with the possible exception $\text{NaN}^0 = 1$ (if 0^0 is defined as 1, cf. discussion above).

$\sin(\pm\infty)$, $\cos(\pm\infty)$, $\tan(\pm\infty)$ are NaNs. log, $\log_2$, $\log_{10}$ of a negative number, $\log_b x$ with x or b negative, or $\text{negative}^{\text{noninteger}}$ are NaNs. arcsin, arccos, arctanh of a number outside $[-1, +1]$ are NaNs. The choice for 1^{NaN} must be NaN if 1^∞ is NaN. Other cases where a NaN is returned are the following: $\log(1 + x)$ for $x < -1$, $\cot(\pm\infty)$, $\sec(\pm\infty)$, $\csc(\pm\infty)$, $\operatorname{arcsec} x$ and $\operatorname{arccsc} x$ for $x \in (-1, 1)$, $\operatorname{arctanh} x$ for $|x| > 1$, $\operatorname{arccosh} x$ for $x < 1$, $\operatorname{arccoth} x$ for $x \in (-1, 1)$, $\operatorname{arcsech} x$ for $x < 0$ or $x > 1$.

It is *not allowed* to return a NaN when the exact result is mathematically defined (e.g., sine of a huge number).

2.5.3. *Infinities*

Two kinds of infinities can be produced:

Infinities as result of overflow: they occur when the result is mathematically defined but larger or smaller than ± the largest representable number. They must be handled as any overflow that is produced by an arithmetic operation.

"Exact infinities": $\log(+0)$, $\log_2(+0)$, $\log_{10}(+0)$ and $\log_b(+0)$ with $b > 0$ are $-\infty$ and $\mathrm{arcsech}(+0) = +\infty$, with flag "zero divide" being raised. We suggest the same for -0 (see 2nd point of section 2.5.1).

However, the sign of 0 is relevant in the following cases where "exact infinities" are returned: $\cot(\pm 0) = \pm\infty$ with the sign of 0, i.e. $\cot(+0) = +\infty$ and $\cot(-0) = -\infty$, $\csc(\pm 0) = \pm\infty$, $\coth(\pm 0) = \pm\infty$, $\mathrm{csch}(\pm 0) = \pm\infty$, $\mathrm{arccsch}(\pm 0) = \pm\infty$ and also $(\pm 0)^x = \pm\infty$ with $x < 0$, as recommended in [9], in every case with the sign of 0. The following must also hold: $\mathrm{arctanh}(-1) = -\infty$, $\mathrm{arctanh}(+1) = +\infty$, $\mathrm{arccoth}(-1) = -\infty$ and $\mathrm{arccoth}(+1) = +\infty$.

It is worth being noticed that tangents, cotangents, secants and cosecants never return infinities for current precision (no single precision nor double precision floating-point number is close enough to a multiple of $\pi/2$ [16]).

2.6. *"Inexact result" flag*

The following is extracted from [16] and completed for the additional functions.

It is difficult to know when functions such as x^y or $\log_b x$ give an exact result. However, using a theorem due to Lindemann, one can show that the sine, cosine, tangent, exponential, or arctangent of a nonzero finite machine number, or the logarithm of a finite machine number different from 1 is not a machine number, so that its computation is always inexact.

Here are some examples of exact operations (for a complete list, see the INRIA Research Report corresponding to this paper). For the radix-2 logarithm and exponential functions:

1. $\log_2(\pm 0) = -\infty$ and $\log_2(+\infty) = +\infty$;
2. $\log_2 1 = +0$ except that $\nabla(\log_2 1) = -0$;
3. for any integer p such that 2^p is exactly representable, $\log_2 2^p = p$;
4. $2^{-\infty} = +0$ and $2^{+\infty} = +\infty$;
5. for any integer p such that 2^p is exactly representable, $\exp_2 p = 2^p$. In particular, $\exp_2 0 = 2^0 = 1$.

For the hyperbolic functions and their reciprocals:

1. $f(\pm 0) = \pm 0$ for $f = \sinh, \tanh, \mathrm{arcsinh}, \mathrm{arctanh}$;
2. $f(-\infty) = -\infty$ and $f(+\infty) = +\infty$ for $f = \sinh, \mathrm{arcsinh}$;

3. $\cosh 0 = 1$, $\cosh(-\infty) = +\infty$ and $\cosh(+\infty) = +\infty$;
4. $\tanh(-\infty) = -1$ and $\tanh(+\infty) = 1$.

3. Now, it's up to you

The benefits expected from this standardization are the same as those provided by the IEEE-754 standard for floating-point arithmetic: better portability of codes, reproducibility of numerical results, along with a sound definition of floating-point mathematical functions which can be used to study and prove results on algorithms using these functions.

To round correctly mathematical functions, the main difficulty is to evaluate efficiently a mathematical function with enough intermediate precision to be able to correctly round the result. Two directions are explored to solve this problem: on the one hand, hardest-to-round cases are sought after for every function and every computing precision; on the other hand, practical implementations of mathematical functions, in hardware or in software, are getting more and more efficient. Recent advances cases [12,13,16–18] give hints that the quality required by a standard is becoming reality.

We are eagerly looking for comments from the computer arithmetic and numerical analysis communities.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers, IEEE standard for binary floating-point arithmetic, ANSI/IEEE Standard, Std 754-1985, New York (1985).

[2] C: ISO/IEC 9899, Revision of the standardization of the C language, known as C99. `http://std.dkuug.dk/jtc1/sc22/wg14/` (2002).

[3] W.J. Cody, J.T. Coonen, D.M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F.N. Ris and D. Stevenson, A proposed radix-and-word-length-independent standard for floating-point arithmetic, IEEE MICRO 4(4) (1984) 86–100.

[4] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Comput. Surveys 23(1) (1991) 5–47.

[5] T.J. Hickey, Q. Ju and M.H. van Emden, Interval arithmetic: From principles to implementation, J. ACM 48(5) (2001) 1038–1068.

[6] IEEE Standard for binary floating-point arithmetic (Draft of the 17th December 2002), `http://www.validlab.com/754R/` (2002).

[7] W. Kahan, Minimizing $qm - n$, `http://http.cs.berkeley.edu/~wkahan/testpi/nearpi.c` (1983).

[8] W. Kahan, Computer system support for scientific and engineering computation, Lecture notes available at `http://www.validlab.com/fp-1988/lectures/` (1988).

[9] W. Kahan, Lecture notes on the status of IEEE-754, `http://http.cs.berkeley.edu/~wkahan/ieee754status/ieee754.ps` (1996).

[10] W. Kahan, What can you learn about floating-point arithmetic in one hour? `http://http.cs.berkeley.edu/~wkahan/ieee754status/cs267fp.ps` (1996).

[11] W. Kraemer and A. Bantle, Automatic forward error analysis for floating point algorithms, Reliable Comput. 7 (2001) 321–340.

[12] V. Lefèvre and J.M. Muller, Worst cases for correct rounding of the elementary functions in double precision, in: *Proc. of the 15th IEEE Symposium on Computer Arithmetic*, Vail, USA, 2001 (IEEE Computer Soc. Press) pp. 111–118.

[13] V. Lefèvre, J.-M. Muller and A. Tisserand, Toward correctly rounded transcendentals, IEEE Trans. Comput. 47(11) (1998) 1235–1243.

[14] LIA-ISO/IEC 10967: Language Independent Arithmetic; LIA-2: Elementary numerical functions, Binding techniques, `http://std.dkuug.dk/jtc1/sc22/wg11/` (2002).

[15] R.E. Moore, *Interval Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1966).

[16] J.M. Muller, *Elementary Functions, Algorithms and Implementation* (Birkhäuser, Boston, 1997).

[17] K.C. Ng, Argument reduction for huge arguments: Good to the last bit, Technical Report, SunPro (1992); available by e-mail: `kwok.ng@eng.sun.com`.

[18] D. Stehlé, V. Lefèvre and P. Zimmermann, Worst cases and lattice reduction, in: *Proc. of the 16th IEEE Symposium on Computer Arithmetic*, Santiago de Compostela, Spain (IEEE Computer Soc. Press, 2003) pp. 142–147.