Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic

David Defour, Guillaume Hanrot, Vincent Lefèvre, Jean-Michel Muller, Nathalie Revol, Paul Zimmermann

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic

David Defour — Guillaume Hanrot — Vincent Lefèvre

Jean-Michel Muller — Nathalie Revol — Paul Zimmermann

N° 5406

December 2004

Thème SYM

ISSN 0249-6399 ISRN INRIA/RR--5406--FR+ENG

Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic

David Defour , Guillaume Hanrot , Vincent Lefèvre
Jean-Michel Muller , Nathalie Revol , Paul Zimmermann

Thème SYM — Systèmes symboliques
Projets Arénaire et Spaces

Rapport de recherche n° 5406 — December 2004 — 47 pages

Abstract: 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.

Key-words: floating-point arithmetic, mathematical function, standard, implementation, rounding, exception.

This text is also available as a research report LIP 2004-54 of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP/`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 —Télécopie +33 4 76 61 52 52

Proposition de norme pour l'implantation des fonctions mathématiques en arithmétique virgule flottante

Résumé : Quelques aspects de ce que pourrait être une norme pour l'implantation des fonctions mathématiques sont présentés. Tout d'abord, le besoin d'une telle norme est motivé. Ensuite, la proposition de norme est détaillée. La question des arrondis constitue une partie importante de ce rapport : trois niveaux sont proposés, qui vont d'un niveau relativement facile à atteindre (avec une erreur relative maximale qui est fixée) à un niveau de qualité optimale, où l'arrondi est correct sur tout le domaine de définition de chaque fonction.

Nous ne prétendons pas avoir suggéré les meilleurs choix dans tous les cas, ni même avoir envisagé toutes les questions. L'objectif principal de cette proposition est de faire naître des questions et de lancer la discussion qui conduira à une norme.

Mots-clés : arithmétique virgule flottante, fonction mathématique, norme, implantation, arrondi, exception.

Proposal for a Standardization of Mathematical Function Implementation in Floating-Point Arithmetic

David Defour , Guillaume Hanrot , Vincent Lefèvre ,
Jean-Michel Muller , Nathalie Revol , Paul Zimmermann

Defour@univ-perp.fr
Jeune Équipe de Recherche en Informatique, Université de Perpignan, France
<Jean-Michel.Muller,Nathalie.Revol>@ens-lyon.fr
Project Arenaire, LIP (CNRS/ENSL/INRIA/UCBL), École Normale Supérieure de Lyon, France
<Guillaume.Hanrot,Vincent.Lefevre,Paul.Zimmermann>@loria.fr
Project Spaces, LORIA/INRIA Lorraine, 615 rue du jardin botanique, F-54602 Villers-lès-Nancy Cedex, France

1 Introductory Discussion

1.1 Need for a Standard

We take the opportunity of the current discussion on the revision of the IEEE-754 Standard for Floating-Point Arithmetic [8] 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 [2] 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 [11, 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 [20] or [19] 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 three lists, the "exceptional" values (Section 2.5), the sign of null results (Section 2.6) and the cases of exact results (Section 2.7). The considered functions are recapitulated (graphical representation and exact, infinite or exceptional values) in a first appendix; the arguments that lead to extremal values for the trigonometric functions, along with these values, are given in a second appendix: this proves that no overflow can occur for the trigonometric functions with the usual floating-point precisions.

Our suggestions are based on the reading of the IEEE-754 and IEEE-854 standards [2, 5, 9], some of the works of Kahan [10, 11, 12, 13], the draft of the current revision of the IEEE-754 standard [8], drafts of other current standardization efforts [4, 17], 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 [15, 16, 21]. In the following, text defining the proposed standard is set off from surrounding expository material and from comments by indentation from both margins and by a different font, as in [5].

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 $(2/\pi) \arctan x$ 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 (cf. Section 2.5.1), 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 [17], the second part of LIA, which was published in 2001) and language standardization, e.g. ISO/IEC 9899 for the C programming language [4].

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

$$\frac{13176795}{8388608} = 1.5707963705062866210937 5 > \frac{\pi}{2}.$$

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,

$$\tan\left(\arctan\left(2^{30}\right)\right) = -2.2877\cdots \times 10^{7}.$$

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 §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

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

must be preserved, and than the ISO/IEC 9899 standard for C, which does not address this issue;

- exceptions: most standards agree on exceptions handling, at the possible exception of some choices discussed in §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). Here is a subset of functions we are targeting (cf. Appendix 1 for more details about these functions):

> log, $\log_2$, $\log_{10}$, $\log_b x$ (even if for this function it is not yet known how to satisfy the requirements of the standard), $\log(1+x)$, exp, $\exp(x)-1$, 2^x, 10^x, sin, cos, tan, cot, sec, csc, arcsin, arccos, arctan, arctan $\frac{x}{y}$, 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.

2.3 Choice of Range, Monotonicity, Symmetry

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` for round-to-nearest and round towards zero modes and `preserve-rounding` for rounding towards $\pm\infty$ modes.

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 2.7 – the system must return that value). In the **round towards $-\infty$ mode**, denoted by $\bigtriangledown$, 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 $\bigtriangleup$, 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].

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

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 ($\triangle$ or $\triangledown$), 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 $1/2$. For the more demanding level 2, the relative error is bounded by $1/2$ ulp for rounding to nearest and by 1 ulp for the other rounding modes, as for arithmetic operations. This implies that $\tau = 0$. To be more precise: the error considered here is *relative* for normal numbers and *absolute* for subnormals.

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 1 and 2 give the hardest-to-round cases for double-precision exponentials and logarithms.

Table 1: *Worst cases for the exponential function in the full double-precision range. Exponentials of numbers less than* $\log(2^{-1074})$ *are underflows (a routine should return* 0 *or the smallest non zero positive representable number, depending on the rounding mode). Exponentials of numbers larger than* $\log(2^{1024})$ *are overflows.*

Interval	worst case (binary)
$[\log(2^{-1074}), -2^{-30}]$	$\exp(-1.11101101001100011000111011111011011000100111111101010 \times 2^{-27})$ $= 1.11111111111111111111111111000010010110011100111000100 \quad 1 \quad 1^{59}0001... \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.00000000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.11111111111111111111111111111111111111111111111111111100 \quad 0 \quad 0^{100}1010... \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.11111111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.00000000000000000000000000000000000000000000000000000000 \quad 1 \quad 1^{104}0101...$
$[2^{-30}, \log(2^{1024})]$	$\exp(1.01111111111111100111111111111111011100000000000100100 \times 2^{-32})$ $= 1.00000000000000000000000000000010111111111111101000 \quad 0 \quad 0^{57}1101...$ $\exp(1.10000000000000010111111111111111011011111111111011100 \times 2^{-32})$ $= 1.00000000000000000000000000000011000000000000010111 \quad 1 \quad 1^{57}0010...$ $\exp(1.10011110100111001011101111111110101100000100000001011 \times 2^{-31})$ $= 1.00000000000000000000000000000011001111010011100101111 \quad 1 \quad 0^{57}1010...$ $\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \quad 0 \quad 0^{57}1000...$

Table 2: *Worst cases for the natural (radix e) logarithm in the full double-precision range.*

Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.10010100011101101110001100000100110011010111111000111 \times 2^{-384})$ $= -100001001.10110110000011001010111101000111101100110101 \quad 1 \quad 0^{60}1010...$ $\log(1.11101010011100011101100001011100111011100000000100000 \times 2^{-509})$ $= -101100000.00101001011010100110011010110100001011111111 \quad 1 \quad 1^{60}0000...$ $\log(1.00100110111010011100010011010011001001111001011000000 \times 2^{-232})$ $= -10100000.10101011001011000010010111100110100001000010 \quad 0 \quad 0^{60}1001...$
$(1, 2^{1024}]$	$\log(1.01100010101010001000011000010011011000101001101101100 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{64}1110...$

These tables are extracted from [15]. 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 and thus the maximal computing precision is known, then it is possible to optimize the code for this function,

in particular the size of memory needed is known.

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, $\arctan\frac{x}{y}$ and $\log_b x$. Moreover, it is expected that arguments outside the prescribed range are expensive to deal with in terms of computing time, size of tables (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. Let us notice that the introduction of a restricted range on which full accuracy is required is already to be found in the IEEE-754 standard [2, Section 5.6], for the conversion between binary and decimal formats.

2.5 Exceptions

A standard must also specify the results of calls with "special" arguments (infinities and NaN) and with arguments at the boundary of the function's domain. This latter case is dealt with first.

For clarity purpose, in the following, the sign value will be denoted by $(-1)^s$ with $s \in \{0, 1\}$.

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 [12] suggests to choose $0^0 = 1$. A consequence of that (also mentioned by Kahan) is that it implies that $\mathrm{NaN}^0 = 1$ whereas $0^{\mathrm{NaN}} = \mathrm{NaN}$, since x^0 is 1 for *any* x, whereas 0^y is 1 if $y = 0$ and 0 otherwise. 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 [7], -0 may be thought as a small negative number that has underflowed to zero. This would favor the choice $\log(-0) = \mathrm{NaN}$. 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*; thus, $\log(-0)$ should be $-\infty$. For these reasons of consistency (and only for these reasons), we would prefer the choice $\log(-0) = -\infty$, but we of course recognize this choice has drawbacks.

$1^{\pm\infty}$ 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 [12] suggests $1^{\pm\infty} = \mathrm{NaN}$, which implies (for reasons of consistency) $1^{\mathrm{NaN}} = \mathrm{NaN}$.

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 $\mathrm{NaN}^0 = 1$ (if 0^0 is defined as 1, cf. discussion above).

- $\log$, $\log_2$, $\log_{10}$, $\log_b$ of a negative number;
- $\log_b$ with $b < 0$;
- $\log(1 + x)$ with $x < -1$;
- x^y with $x < 0$ and $y \notin \mathbb{N}$;
- 1^{NaN} must be a NaN if 1^∞ is a NaN;
- $\sin, \cos, \tan, \cot, \sec, \csc$ of $\pm\infty$;
- $\arcsin x$, $\arccos x$, $\operatorname{arctanh} x$ for $x \notin [-1, +1]$;
- $\operatorname{arccosh} x$ for $x < 1$, $\operatorname{arccoth} x$ for $|x| \in (-1, +1)$;
- $\operatorname{arcsech} x$ for $x \notin [0, 1]$;

- $\operatorname{arccot} 0$ is either a NaN since arccot is not defined[3] in 0, or $\operatorname{arccot}(-0) = -\pi/2$ and $\operatorname{arccot}(+0) = \pi/2$;
- $\operatorname{arcsec} x$ and $\operatorname{arccsc} x$ for $x \in (-1, 1)$.

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 the rounded result is larger (or smaller) than the largest (or smallest) representable real number.

"exact infinities": they can be produced by one of the following operations:

- $\log_\beta(+0) = -\infty$ with $\beta \in \{e, 2, 10, b\ (b > 0)\}$ and $\operatorname{arcsech}(+0) = +\infty$, with flag "zero divide" being raised. We suggest the same for -0, according to the choice discussed in 2^{nd} point of Section 2.5.1.
- $f((-1)^s 0) = (-1)^s \infty$ for $f = \cot, \csc, \coth, \operatorname{csch}, \operatorname{arccsch}$, *e.g.* $\cot(+0) = +\infty$ and $\cot(-0) = -\infty$;
- $((-1)^s 0)^x = \infty$ with x negative value that is not an odd integer, and $((-1)^s 0)^x = (-1)^s \infty$ with x negative odd integer, as recommended in [12];
- $\operatorname{arctanh}((-1)^s) = (-1)^s \infty$, $\operatorname{arccoth}((-1)^s) = (-1)^s \infty$;
- $\log_\beta(+\infty) = +\infty$ with $\beta \in \{e, 2, 10, b\ (b > 0)\}$;
- $\log(1 + x) = +\infty$ for $x = +\infty$;
- $\log(1 + x) = -\infty$ for $x = -1$;
- $\beta^{+\infty} = +\infty$ with $\beta \in \{e, 2, 10, b\ (b > 1)\}$;

[3] At least two definitions of arccot exist: for Maple, arccot is a continuous decreasing function with $\mathbb{R}$ as domain and $(0, \pi)$ as range; for Abramowitz and Stegun [1], arccot is a continuous decreasing function from $(-\infty, 0)$ to $[-\pi/2, 0)$ and from $(0, +\infty)$ to $(0, \pi/2]$. We chose to follow [1], since this definition seems to be the most commonly accepted one.

- $\beta^{-\infty} = +\infty$ with $0 < \beta < 1$;
- $\exp(+\infty) = +\infty$;
- $\exp(x) - 1 = +\infty$ for $x = +\infty$;
- $f((-1)^s\infty) = (-1)^s\infty$ for $f = \sinh, \operatorname{arcsinh}$;
- $\cosh(\pm\infty) = +\infty$;
- $\operatorname{arccosh}(+\infty) = +\infty$.

It is worth being noticed that tangents, cotangents, secants and cosecants never return infinities for current precisions (no single, double, double extended nor quadruple precision floating-point number is close enough to a multiple of $\pi/2$: this has been checked using the program given pp 152-153 of [19]).

Table 3 gives, for each floating-point format, the features of this format: number of bits of the mantissa and of the exponent, then the floating-point number which is closest to an integer multiple of π or $\pi/2$ and the values of the trigonometric functions tan, cot, sec and csc for this floating-point number. It shows that no overflow can occur for these trigonometric functions, since no floating-point argument is close enough to a singularity of these functions. (The complete details are to be found in Appendix 2).

2.6 Sign of Null Results

Since there are two zeroes in the IEEE-754 standard for floating-point arithmetic, the sign of a null result must be specified. We suggest the following choice when $f(x)$ gives a zero, but let us give a short explanation first.

The signed zeroes can represent either an exact zero, in which case the sign does not have much meaning as mathematically zero is unsigned, or an arbitrary small (in absolute value) real number z, because of an underflow or an approximated input, in which case the zero must be positive if z is known to be positive and the zero must be negative if z is known to be negative.

Table 3: *Floating-point numbers that are the closest to a non-zero singularity of the* tan, cot, sec *and* csc *functions and values of these functions for these arguments.*

format: name nb bits: mantissa, exp.	C	f: FP number closest to a non-zero multiple of C	$\tan f$	$\cot f$	$\sec f$	$\csc f$
single 24, 8	$\pi/2$	16367173.2^{72}	$-6.19\,10^{8}$	$-1.61\,10^{-9}$	$-6.19\,10^{8}$	1.00
single 24, 8	π	$16367173\,2^{73}$	$3.23\,10^{-9}$	$3.10\,10^{8}$	-1.00	$-3.10\,10^{8}$
double 53, 11	$\pi/2$	$6381956970095103\,2^{797}$	$-2.13\,10^{18}$	$-4.69\,10^{-19}$	$-2.13\,10^{18}$	1.00
double 53, 11	π	$6381956970095103\,2^{798}$	$9.37\,10^{-19}$	$1.07\,10^{18}$	-1.00	$-1.07\,10^{18}$
ext. double 64, 15	$\pi/2$	$17476981849448541921\,2^{10531}$	$5.84\,10^{22}$	$1.82\,10^{-23}$	$-5.84\,10^{22}$	-1.00
ext. double 64, 15	π	$17476981849448541921\,2^{10532}$	$-3.65\,10^{-23}$	$-2.74\,10^{22}$	-1.00	$2.74\,10^{22}$
quadruple 113,15	$\pi/2$	$87948731350338293497\backslash$ $02184924722639\,2^{1852}$	$1.27\,10^{37}$	$7.88\,10^{-38}$	$1.27\,10^{37}$	1.0
quadruple 113, 15	π	$87948731350338293497\backslash$ $02184924722639\,2^{1853}$	$-1.58\,10^{-37}$	$-6.34\,10^{36}$	-1.0	$6.34\,10^{36}$

When $f(x) \neq 0$ but the result is 0 due to an underflow, the sign of the 0 must be the sign of $f(x)$.

Now, let us assume that $f(x) = 0$. Then the sign is determined by supposing that x approximates an exact value belonging to an interval I_ε defined as a function of $\varepsilon > 0$, as follows. If x is $+\infty$, then $I_\varepsilon = (\varepsilon^{-1}, +\infty)$. If x is $-\infty$, then $I_\varepsilon = (-\infty, -\varepsilon^{-1})$. If x is $+0$, then $I_\varepsilon = [0, \varepsilon)$. If x is -0, then $I_\varepsilon = (-\varepsilon, 0]$. If x is a nonzero real value, then $I_\varepsilon = (x - \varepsilon, x + \varepsilon)$.

Let $J_\varepsilon = f(I_\varepsilon \cap D)$, where D is the domain of f. If there exists $\varepsilon > 0$ such that $J_\varepsilon = \{0\}$, then there is no general rule; for instance, the IEEE-754 standard requires that $\sqrt{-0}$ gives -0. Otherwise, if $J_\varepsilon \geq 0$ for small enough ε, then the result should be $+0$, and if $J_\varepsilon \leq 0$ for small enough ε, then the result should be -0. If J_ε contains both positive and negative numbers for any $\varepsilon > 0$, then the sign of the result is determined by the rounding mode: in the round towards $-\infty$ mode, the result should be -0, and in the other rounding modes, the result should be $+0$.

2.7 "Inexact Result" Flag

The following is extracted from [19] 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 [3], 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.

With the most common functions, the only exact operations are given below. For the radix β logarithm and the exponential functions, such that $\beta \in \{e, 2, 10, b\ (b > 1)\}$:

1. $\log_\beta(\pm 0) = -\infty$ or NaN (cf. Section 2.5.1) and $\log_\beta(+\infty) = +\infty$;
2. $\log_\beta 1 = +0$ except for $\triangledown(\log_\beta 1) = -0$;
3. $\beta^{-\infty} = +0$;
4. $x^{-\infty} = +\infty$ if $0 < x < 1$;
5. $\beta^{+\infty} = +\infty$;
6. $x^{+\infty} = 0$ if $|x| < 1$;
7. for any integer p such that β^p is exactly representable, $\log_\beta \beta^p = p$. In particular:
   - $\beta^0 = 1$;
   - when $\beta = e$ then this can occur only for $p = 0$;
   - when $\beta = 2$ then $\log_2 2^p = p$ and 2^p is also exact for any integer p in the exponent range;
   - when $\beta = 10$ then $\log_{10} 10^p = p$ and 10^p is also exact for any integer $p \geq 0$ such that 5^p is exactly representable;
   - when $\beta = b$ then $\log_b b^p = p$;

8. $\log(1+x) = -\infty$ for $x = -1$, $(-1)^s 0$ for $x = (-1)^s 0$, and $+\infty$ for $x = +\infty$;
9. $\exp(x) - 1 = -1$ for $x = -\infty$, $(-1)^s 0$ for $x = (-1)^s 0$, and $+\infty$ for $x = +\infty$.

For the trigonometric functions and their reciprocals:

1. $f((-1)^s 0) = (-1)^s 0$ for $f = \sin, \tan, \arcsin, \arctan$;
2. $\cos 0 = 1$;
3. $\cot((-1)^s 0) = (-1)^s \infty$;
4. $\sec 0 = 1$;
5. $\csc((-1)^s 0) = (-1)^s \infty$;
6. $\arccos 1 = +0$;
7. $\arctan\left(\frac{(-1)^s 0}{y}\right) = (-1)^s \operatorname{sign}(y) 0$ for $y \neq 0$;
8. $\operatorname{arccot}((-1)^s \infty) = (-1)^s 0$;
9. $\operatorname{arcsec} 1 = +0$;
10. $\operatorname{arccsc}((-1)^s \infty) = (-1)^s 0$.

For the hyperbolic functions and their reciprocals:

1. $f((-1)^s 0) = (-1)^s 0$ for $f = \sinh, \tanh, \operatorname{arcsinh}, \operatorname{arctanh}$;
2. $f((-1)^s \infty) = (-1)^s \infty$ for $f = \sinh, \operatorname{arcsinh}$;
3. $\cosh 0 = 1$ and $\cosh(\pm\infty) = +\infty$;
4. $\tanh((-1)^s \infty) = (-1)^s$;
5. $\coth((-1)^s 0) = (-1)^s \infty$ and $\coth((-1)^s \infty) = (-1)^s 0$;
6. $\operatorname{sech} 0 = 1$ and $\operatorname{sech}(\pm\infty) = +0$;
7. $\operatorname{csch}((-1)^s 0) = (-1)^s \infty$ and $\operatorname{csch}((-1)^s \infty) = (-1)^s 0$;
8. $\operatorname{arccosh} 1 = +0$, and $\operatorname{arccosh}(+\infty) = +\infty$;
9. $\operatorname{arctanh}((-1)^s) = (-1)^s \infty$;
10. $\operatorname{arccoth}((-1)^s \infty) = (-1)^s 0$, and $\operatorname{arccoth}((-1)^s 1) = (-1)^s \infty$;
11. $\operatorname{arcsech} 1 = +0$, and $\operatorname{arcsech}(+0) = +\infty$, $\operatorname{arcsech}(-0) = +\infty$ or NaN, depending on the choice discussed in 2.5.1;
12. $\operatorname{arccsch}((-1)^s \infty) = (-1)^s 0$, and $\operatorname{arccsch}((-1)^s 0) = (-1)^s \infty$.

2.8 Bindings with Others Standards

When preparing this proposal, we tried to keep compatibility with other standards that include specifications about mathematical functions. These standards are composed of, but only, current discussions on LIA-2 [17] and C99 [4]. It should be noticed that each language such as Ada, Pascal, Lisp etc. has its own standard (see `http://www.open-std.org/JTC1/SC22/` for a set of current discussions for programming languages) and its own collection of criteria to satisfy.

However our proposal differs in some points.

Precision. Most standards related to programming languages, such as C99, do not have requirements about the final accuracy. The LIA-2 standard on the contrary, has requirements about accuracy for mathematical functions. The accuracy is required to lie in $[0.5, 2 \times C]$ ulp where the constant C depends on the representation number system. This interval is not unique and is specific to a group of mathematical functions. For trigonometric functions, the accuracy condition is required only for a sub-interval centered in zero.

Properties. The C99 standard does not have more requirements on properties that have to be satisfied by an evaluation implementation than it has on the accuracy. Within the whole set of properties given in Section 1.1, the LIA-2 standard includes only the monotonicity. It is required for trigonometric functions on an interval centered in 0; this interval depends on the representation number system (for instance, $[-2^{27}, +2^{27}]$ for the double-precision floating-point numbers of IEEE-754 standard). For other functions, the monotonicity property is required on the whole interval where the function is mathematically monotonic.

Exceptions. Apart from a few differences mainly covered in the open questions (see the discussion in 2.5.1), every standard has an identical answer to the question of exceptions.

Some remarks. The LIA-2 standard required some properties that seem hard to be satisfied in a floating-point number system. Indeed, it imposes

conditions on the result of a function with irrational number as input argument, which is a non-machine representable number (*i.e.* e or π).

One can noticed that the third part of the LIA standard is dedicated to complex mathematical functions. This is another difference with our proposal that does not cover these functions.

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.

Mathematical functions were not addressed by previous standards because it was feared that providing correctly rounded results was out of reach for these times. To correctly round mathematical functions, the main difficulty is to efficiently evaluate 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 in the determination of hardest-to-round cases [15, 16, 19, 20, 21] and in the implementation of mathematical libraries [6] give hints that the quality required by a standard is becoming reality.

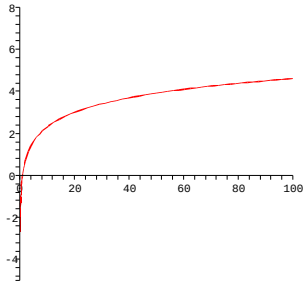
We are eagerly looking for comments from the computer arithmetic and numerical analysis communities.

Appendix 1

Below, the main characteristics of the considered functions are recalled: exact, infinite and exceptional values, and a graphical representation is given for each of them.

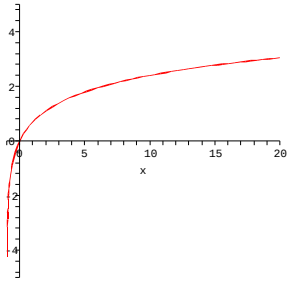
Logarithms and Exponentials

Logarithm



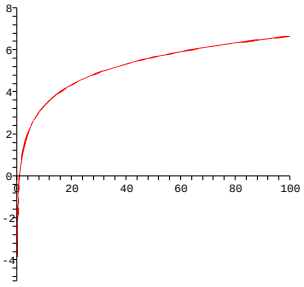
- exact values: $\log 1 = 0$;
- infinities: $\log 0 = -\infty$, $\log(+\infty) = +\infty$;
- NaN: $\log x$ is NaN if $x < 0$.

Logarithm (1+x)



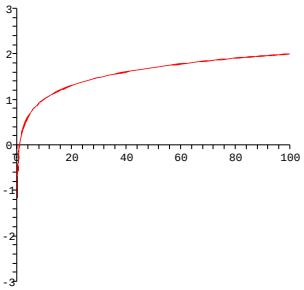
- exact values: $f(\pm 0) = \log(1 + \pm 0) = \pm 0$;
- infinities: $f(-1) = \log(-1 + 0) = -\infty$, $f(+\infty) = \log 1(+\infty) = +\infty$;
- NaN: $f(x) = \log(1 + x)$ is NaN if $x < -1$.

Radix-2 Logarithm



- definition: $\log_2 x = \frac{\log x}{\log 2}$;
- exact values: $\log_2 1 = 0$, $\log_2 2^p = p$;
- infinities: $\log_2 0 = -\infty$, $\log_2(+\infty) = +\infty$;
- NaN: $\log_2 x$ is NaN if $x < 0$.

Radix-10 Logarithm

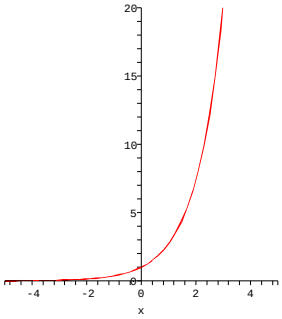


- definition: $\log_{10} x = \frac{\log x}{\log 10}$;
- exact values: $\log_{10} 1 = 0$, $\log_{10} 10^p = p$;
- infinities: $\log_{10} 0 = -\infty$, $\log_{10}(+\infty) = +\infty$;
- NaN: $\log_{10} x$ is NaN if $x < 0$.

Radix-b Logarithm

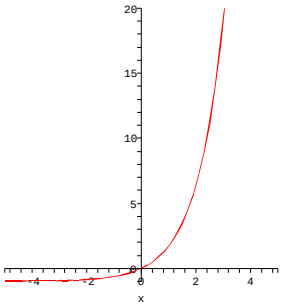
- definition: $\log_b x = \frac{\log x}{\log b}$;
- exact values: $\log_b 1 = 0$, $\log_b b^p = p$ if b^p is representable;
- infinities: $\log_b 0 = -\infty$, $\log_b(+\infty) = +\infty$;
- NaN: $\log_b x$ is NaN if $x < 0$ or $b < 0$.

Exponential



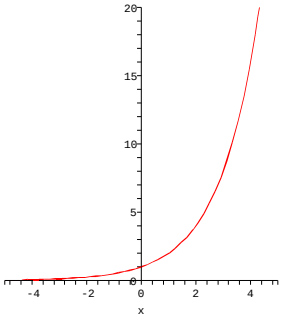
- exact values: $\exp(-\infty) = +0$, $\exp 0 = 1$;
- infinities: $\exp(+\infty) = +\infty$;
- NaN: the only value of x such that $\exp x$ is NaN is x is NaN.

Exponential minus 1



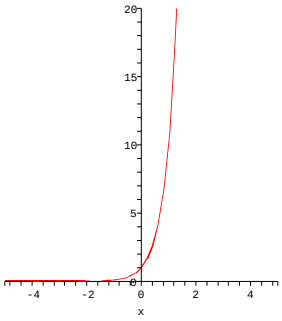
- exact values: $f(-\infty) = \exp(-\infty)-1 = -1$, $f(0) = \exp(0) - 1 = 0$;
- infinities: $f(+\infty) = \exp(+\infty) - 1 = +\infty$;
- NaN: the only value of x such that $f(x) = \exp(x) - 1$ is NaN is x is NaN.

Powers of 2



- exact values: $2^{-\infty} = +0$, $2^0 = 1$ and more generally 2^p is exact if p is an integer and 2^p is representable;
- infinities: $2^{+\infty} = +\infty$;
- NaN: the only value of x such that 2^x is NaN is x is NaN.

Powers of 10



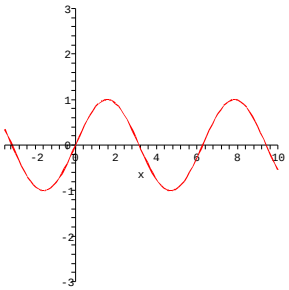
- exact values: $10^{-\infty} = 0$, $10^0 = 1$ and more generally 10^p is exact if p is a nonnegative integer and 10^p is representable;
- infinities: $10^{+\infty} = +\infty$;
- NaN: the only value of x such that 10^x is NaN is x is NaN.

Powers of b

- exact values: $b^{-\infty} = 0$ if $|b| > 1$, $b^{+\infty} = 0$ if $|b| < 1$, $b^0 = 1$ (if $b \neq 0$, cf. discussion Section 2.5.1) and more generally b^p is exact if it is representable;
- infinities: $b^{+\infty} = +\infty$ if $b > 1$, $b^{-\infty} = +\infty$ if $0 < b < 1$, $b^x = \pm\infty$ if $b = \pm 0$ and x is a negative odd integeer, $b^x = +\infty$ if $b = \pm 0$ and x is a negative value that is not a odd integer;
- NaN: b^x is NaN if $b \leq 0$ and $x \notin \mathbb{N}$.

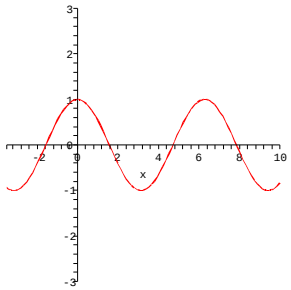
Trigonometric Functions and Their Reciprocals

Sine



- exact values: $\sin 0 = 0$;
- infinities: none;
- NaN: $\sin(\pm\infty)$ is NaN.

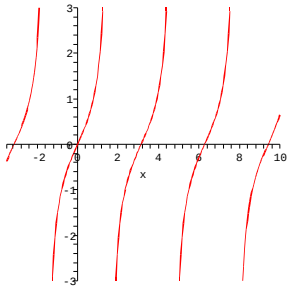
Cosine



- exact values: $\cos 0 = 1$;
- infinities: none;
- NaN: $\cos(\pm\infty)$ is NaN.

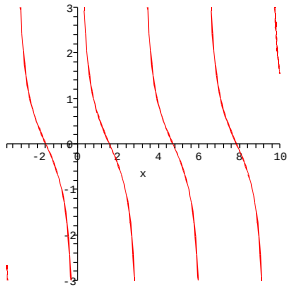
Tangent

- definition: $\tan x = \frac{\sin x}{\cos x}$;



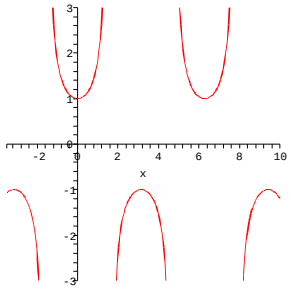
- exact values: $\tan 0 = 0$;
- infinities: attained for $\pi/2 + k\pi$ with $k \in \mathbb{Z}$: no floating-point number, in usual floating-point precision, is close enough to such a number to yield an overflow;
- NaN: $\tan(\pm\infty)$ is NaN.

Cotangent



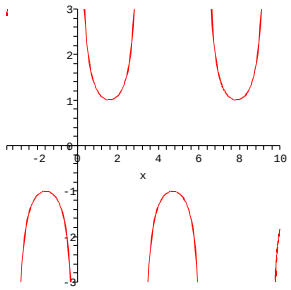
- definition: $\cot x = \frac{\cos x}{\sin x}$;
- exact values: none;
- infinities: $\cot(\pm 0) = \pm\infty$ (∞ with the sign of 0);
- NaN: $\cot(\pm\infty)$ is NaN.

Secant



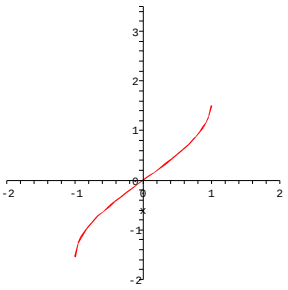
- definition: $\sec x = \frac{1}{\cos x}$;
- exact values: $\sec 0 = 1$;
- infinities: attained for $\pi/2 + k\pi$ with $k \in \mathbb{Z}$: no floating-point number, in usual floating-point precision, is close enough to such a number to yield an overflow;
- NaN: $\sec(\pm\infty)$ is NaN.

Cosecant



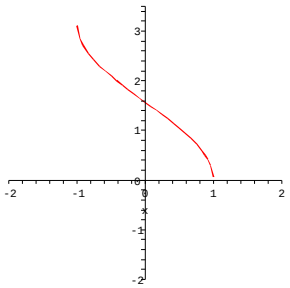
- definition: $\csc x = \frac{1}{\sin x}$;
- exact values: none;
- infinities: $\csc(\pm 0) = \pm\infty$ (∞ with the sign of 0);
- NaN: $\csc(\pm\infty)$ is NaN.

Inverse Sine



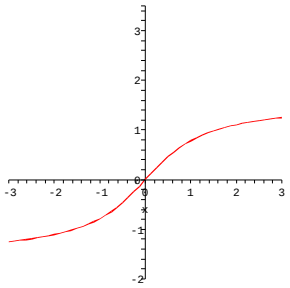
- exact values: $\arcsin 0 = 0$;
- infinities: none;
- NaN: $\arcsin x$ is NaN if $|x| > 1$.

Inverse Cosine



- exact values: $\arccos 1 = +0$;
- infinities: none;
- NaN: $\arccos x$ is NaN if $|x| > 1$.

Inverse Tangent

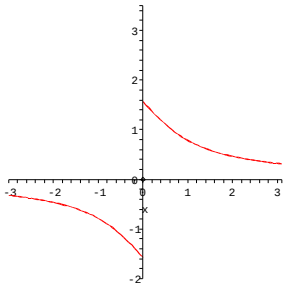


- exact values: $\arctan 0 = 0$, $\arctan(\pm\infty) = \pm\pi/2$ but it is not an exactly representable value in any floating-point format;
- infinities: none;
- NaN: the only value of x such that $\arctan x$ is NaN is x is NaN.

Inverse Tangent (x/y)

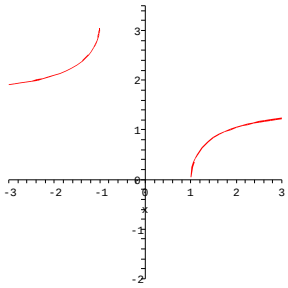
- exact values: $\arctan \frac{(-1)^s 0}{y} = (-1)^s . \operatorname{sign} y . 0$ if $y \neq 0$;
- infinities: none;
- NaN: $\arctan(x/y)$ is NaN when x is NaN or y is NaN.

Inverse Cotangent



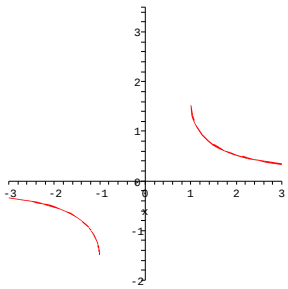
- exact values: $\operatorname{arccot}(\pm\infty) = \pm 0$, $\operatorname{arccot} \pm 0 = \pm\pi/2$ but it is not an exactly representable value in any floating-point format;
- infinities: none;
- NaN: the only value of x such that $\operatorname{arccot} x$ is NaN is x is NaN with this definition of arccot, cf. [1]. With Maple definition of arccot, $\operatorname{arccot} 0$ is NaN.

Inverse Secant



- exact values: $\operatorname{arcsec} 1 = +0$, $\operatorname{arcsec}(-1) = \pi$ and $\operatorname{arcsec}(\pm\infty) = \pi/2$ but the last two cases are not exactly representable values in any floating-point format;
- infinities: none;
- NaN: $\operatorname{arcsec} x$ is NaN if $|x| < 1$.

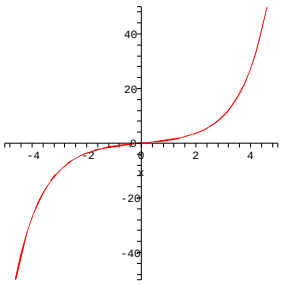
Inverse Cosecant



- exact values: $\operatorname{arccsc}(\pm\infty) = \pm 0$; it also holds that $\operatorname{arccsc}(\pm 1) = \pm\pi/2$ but these values are not exactly representable values in any floating-point format;
- infinities: none;
- NaN: $\operatorname{arccsc} x$ is NaN if $|x| < 1$.

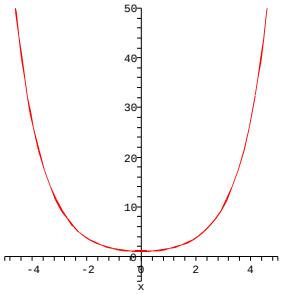
Hyperbolic Functions and Their Reciprocals

Hyperbolic Sine



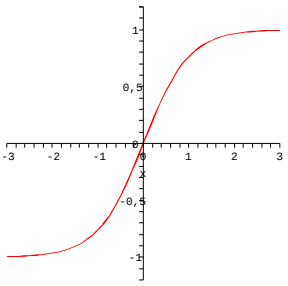
- definition: $\sinh x = \frac{\exp x - \exp(-x)}{2}$;
- exact values: $\sinh 0 = 0$;
- infinities: $\sinh(\pm\infty) = \pm\infty$;
- NaN: the only value of x such that $\sinh x$ is NaN is x is NaN.

Hyperbolic Cosine



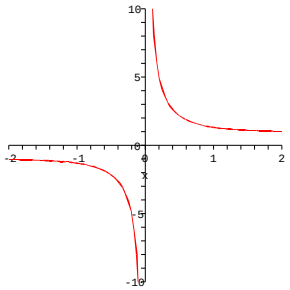
- definition: $\cosh x = \frac{\exp x + \exp(-x)}{2}$;
- exact values: $\cosh 0 = 1$;
- infinities: $\cosh(\pm\infty) = +\infty$;
- NaN: the only value of x such that $\cosh x$ is NaN is x is NaN.

Hyperbolic Tangent



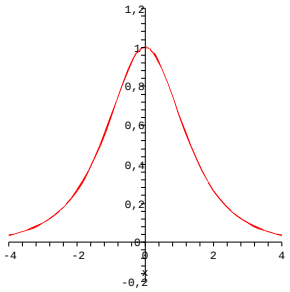
- definition: $\tanh x = \frac{\sinh x}{\cosh x} = \frac{\exp x - \exp(-x)}{\exp x + \exp(-x)}$;
- exact values: $\tanh 0 = 0$, $\tanh(\pm\infty) = \pm 1$;
- infinities: none;
- NaN: the only value of x such that $\tanh x$ is NaN is x is NaN.

Hyperbolic Cotangent



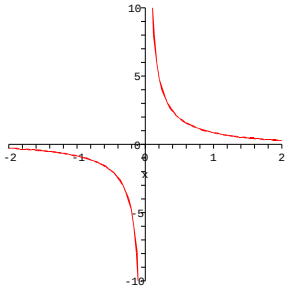
- definition: $\coth x = \frac{\cosh x}{\sinh x} = \frac{\exp x + \exp(-x)}{\exp x - \exp(-x)}$;
- exact values: $\coth(\pm\infty) = \pm 0$;
- infinities: $\coth(\pm 0) = \pm\infty$;
- NaN: $\coth 0$ may be defined as NaN, cf. discussion in Section 2.5.1.

Hyperbolic Secant



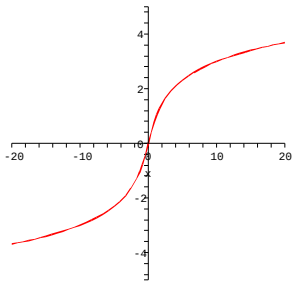
- definition: $\operatorname{sech} x = \frac{1}{\cosh x} = \frac{2}{\exp x + \exp(-x)}$;
- exact values: $\operatorname{sech} 0 = 1$, $\operatorname{sech}(\pm\infty) = \pm 0$;
- infinities: none;
- NaN: the only value of x such that $\operatorname{sech} x$ is NaN is x is NaN.

Hyperbolic Cosecant



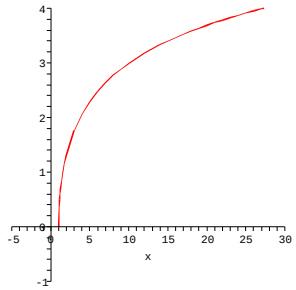
- definition: $\operatorname{csch} x = \frac{1}{\sinh x} = \frac{2}{\exp x - \exp(-x)}$;
- exact values: $\operatorname{csch}(\pm\infty) = \pm 0$;
- infinities: $\operatorname{csch}(\pm 0) = \pm\infty$;
- NaN: $\operatorname{csch} 0$ may be defined as NaN, cf. discussion in Section 2.5.1.

Inverse Hyperbolic Sine



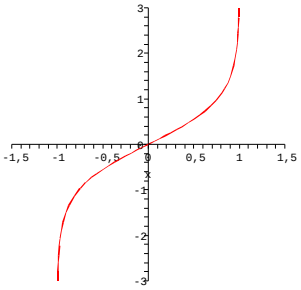
- exact values: $\operatorname{arcsinh} 0 = 0$;
- infinities: $\operatorname{arcsinh}(\pm\infty) = \pm\infty$;
- NaN: the only value of x such that $\operatorname{arcsinh} x$ is NaN is x is NaN.

Inverse Hyperbolic Cosine



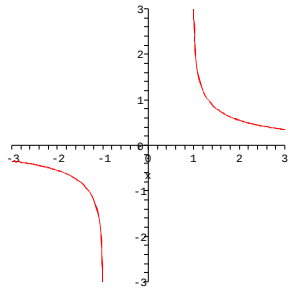
- exact values: $\operatorname{arccosh} 1 = +0$;
- infinities: $\operatorname{arccosh}(+\infty) = +\infty$;
- NaN: $\operatorname{arccosh} x$ is NaN for $x < 1$.

Inverse Hyperbolic Tangent



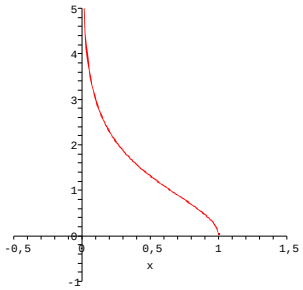
- exact values: $\operatorname{arctanh} 0 = 0$;
- infinities: $\operatorname{arctanh}(\pm 1) = \pm\infty$;
- NaN: $\operatorname{arctanh} x$ is NaN for $|x| > 1$.

Inverse Hyperbolic Cotangent



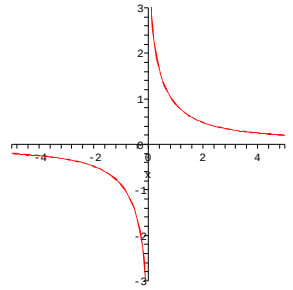
- exact values: $\operatorname{arccoth}(\pm\infty) = \pm 0$;
- infinities: $\operatorname{arccoth}(\pm 1) = \pm\infty$;
- NaN: $\operatorname{arccoth} x$ is NaN for $|x| < 1$.

Inverse Hyperbolic Secant



- exact values: $\operatorname{arcsech} 1 = +0$;
- infinities: $\operatorname{arcsech}(+0) = +\infty$, we suggest $\operatorname{arcsech}(-0) = +\infty$;
- NaN: $\operatorname{arcsech}(-0)$ can be defined as NaN, cf. discussion in Section 2.5.1, $\operatorname{arcsech} x$ is NaN if $x \notin [0, 1]$.

Inverse Hyperbolic Cosecant



- exact values: $\operatorname{arccsch}(\pm\infty) = \pm 0$;
- infinities: $\operatorname{arccsch}(\pm 0) = \pm\infty$;
- NaN: $\operatorname{arccsch} 0$ can be defined as NaN, cf. discussion in Section 2.5.1.

Appendix 2: No Overflow for Trigonometric Functions

The procedure that determines the floating-point number (for a fixed format) which is closest to a multiple of π or $\pi/2$ has been proposed by Kahan in [10] and translated in Maple by Muller in [19].

The following has been output by Maple v.9.0 on a Intel Xeon: for each precision (single, double, extended double and quadruple), the floating-point numbers which are closest to an integer multiple of $\pi/2$ or π are determined (using the Maple procedure mentioned above and called `fclosestC`) and the trigonometric functions which are singular at an integer multiple of π or $\pi/2$, *i.e.* tan, cot, sec and csc, are evaluated on these arguments. The logarithms of these values are given: since they are never greater than the maximal possible exponent in the given format, this means that no overflow occurs.

```
> fclosestC := proc(B,n,emin,emax,C,ndigits)
>   # floating-point precision format: radix B,
>   # nb of digits of the mantissa n
>   # minimal exponent emin, maximal exponent emax;
>   # ndigits: number of digits used for the computations
>   # result: floating-point number closest to a multiple of C
>
>   # local variables
>   local Cf, epsilonmin, powerofBoverC,e,a,
>         Plast,r,Qlast,Q,P,NewQ,NewP,epsilon,numbermin,expmin,l;
>
>   # to enforce floating-point computations
>   Cf := evalf(C, ndigits);
>   epsilonmin := 123456.0; # any large number
>   Digits:=ndigits;
>   powerofBoverC := B^(emin-n)/Cf;
>
>   # for each possible exponent e, determination
>   # of the floating-point number with this exponent e
>   # which is closest to a multiple of C
>   for e from emin-n+1 to emax-n+1 do
>          powerofBoverC := B *powerofBoverC;
>     a := floor(powerofBoverC);
>     Plast := a;
>     r := 1/(powerofBoverC-a);
>     a := floor(r);
>     Qlast := 1;
>     Q := a;
>     P := Plast*a+1;
>
>     # computation of the best rational approximation of B^2/C
>     while Q <B^n-1 do
>       r := 1/(r-a);
>       a := floor(r);
>       NewQ := Q*a+Qlast;
```

```
>       NewP := P*a+Plast;
>       Qlast := Q;
>       Plast := P;
>       Q := NewQ;
>       P := NewP;
>     od;
>     epsilon := evalf(Cf*abs(Plast-Qlast*powerofBoverC));
>
>     # comparison with floating-point numbers with other exponents
>     if epsilon < epsilonmin then
>       epsilonmin := epsilon;
>       numbermin := Qlast;
>       expmin := e;
>     fi;
>   od;
>
>   # result: floating-point number with mantissa numbermin
>   # and exponent expmin
>   # distance to a multiple of C: epsilonmin
>   [numbermin, expmin, evalf(epsilonmin,20)];
> end:
```

```
> affiche := proc(B, C, mantissa, exponent, epsilon, ndig)
>   local auxf, auxi, auxd, auxl;
>
>   print('"mantissa"', evalf(mantissa, ndig));
>   print('"exponent"', exponent);
>   print('"epsilon"', evalf(epsilon, ndig));
>   auxl := evalf(log(evalf(epsilon))/evalf(log(evalf(B))));
>   print('log_epsilon', evalf(auxl, ndig));
>
>   # distance between an integer multiple of C
>   # and the closest floating-point number
>   auxf := evalf(mantissa* (B^exponent));
>   # floating-point number closest to an integer multiple of C
>   print('closest_fp_nb', evalf(auxf, ndig));
>   auxl := evalf(log(abs(auxf))/evalf(log(evalf(B))));
>   print('log_closest_fp_nb', evalf(auxl, ndig));
>   auxi := round(auxf/C);
>   # the integer by which C must be multiplied to be close to auxf
>   auxd := evalf(abs(auxf - evalf(auxi * evalf(C))));
>   # smallest distance between a floating-point number
>   # and an integer multiple of C
>   auxl := evalf(log(auxd)/log(evalf(B)));
>   print('distance', evalf(auxd, ndig));
>   print('log_distance', evalf(auxl, ndig));
>
>
```

```
>   # evaluation of the trigonometric functions
>   # in the given worst cases
>   # and in twice these values
>   # to check if an overflow occurs
>
>   # tangent
>   auxd:= evalf(tan(auxf));
>   print('tangent', evalf(auxd, ndig));
>   auxd := abs(auxd);
>   print('log_tangent',
>         evalf( evalf(evalf(log(auxd))/evalf(log(evalf(B)))), ndig));
>
>   # cotangent
>   auxd:= evalf(cot(auxf));
>   print('cotangent', evalf(auxd, ndig));
>   auxd := abs(auxd);
>   print('log_cotangent',
>         evalf( evalf(evalf(log(auxd))/evalf(log(evalf(B)))), ndig));
>
>   # secant
>   auxd:= evalf(sec(auxf));
>   print('secant', evalf(auxd, ndig));
>   auxd := abs(auxd);
>   print('log_secant',
>         evalf( evalf(evalf(log(auxd))/evalf(log(evalf(B)))), ndig));
>
>   # cosecant
>   auxd:= evalf(csc(auxf));
>   print('cosecant', evalf(auxd, ndig));
>   auxd := abs(auxd);
>   print('log_cosecant',
>         evalf( evalf(evalf(log(auxd))/evalf(log(evalf(B)))), ndig));
>
> end:
```

```
> # determination of the single-precision floating-point number
> # closest to a multiple of Pi/2
> l:=fclosestC(2,24,1,127,Pi/2,100);
```

$$l := [16367173, 72, 1.6147697982476211883 \times 10^{-9}]$$

```
> Digits:=100:affiche(2,Pi/2,l[1],l[2],l[3],20);
```

"*mantissa*", 16367173.0
"*exponent*", 72
"*epsilon*", 0.00000000161476979824762118 83
log_epsilon, −29.206024345165092086
closest_fp_nb, 772917891945290196610000000000.0
log_closest_fp_nb, 95.964301819799583986
distance, 0.00000000161476979824762118 83
log_distance, −29.206024345165092086
tangent, −619283318.95061386812
log_tangent, 29.206024345165092085
cotangent, −0.00000000161476979824762118 97
log_cotangent, −29.206024345165092085
secant, −619283318.95061386892
log_secant, 29.206024345165092086
cosecant, 1.0000000000000000013
log_cosecant, $1.8755035531556524284 \times 10^{-18}$

```
> # determination of the single-precision floating-point number
> # closest to a multiple of Pi
> l:=fclosestC(2,24,1,127,Pi,100);
```

$$l := [16367173, 73, 3.2295395964952423766 \times 10^{-9}]$$

```
> Digits:=100:affiche(2,Pi,l[1],l[2],l[3],20);
```

"*mantissa*", 16367173.0
"*exponent*", 73
"*epsilon*", 0.00000000322953959649524237660
log_epsilon, −28.206024345165092086
closest_fp_nb, $1.5458357838905803932 \times 10^{29}$
log_closest_fp_nb, 96.964301819799583986
distance, 0.00000000322953959649524237660
log_distance, −28.206024345165092086
tangent, 0.00000000322953959649524238780
log_tangent, −28.206024345165092081
cotangent, 309641659.47530693325
log_cotangent, 28.206024345165092081
secant, −1.0000000000000000052
log_secant, $7.5020142126226096988 \times 10^{-18}$
cosecant, −309641659.47530693487
log_cosecant, 29.206024345165092088

```
> # determination of the double-precision floating-point number
> # closest to a multiple of Pi/2
> l:=fclosestC(2,53,1,1023,Pi/2,400);
```

$$l := [6381956970095103, 797, 4.68716592425462761111 \times 10^{-19}]$$

```
> Digits:=400:affiche(2,Pi/2,l[1],l[2],l[3],20);
```

"*mantissa*", 6381956970095103.0
"*exponent*", 797
"*epsilon*", $4.68716592425462761111 \times 10^{-19}$
log_epsilon, -60.887917936171321938
closest_fp_nb, $5.3193726483265414167 \times 10^{255}$
log_closest_fp_nb, 849.50292030468859937
distance, $4.68716592425462761111 \times 10^{-19}$
log_distance, -60.887917936171321938
tangent, -2133485385753703843.7
log_tangent, 60.887917936171321937
cotangent, $-4.68716592425462761111 \times 10^{-19}$
log_cotangent, -60.887917936171321937
secant, -2133485385753703843.7
log_secant, 60.887917936171321937
cosecant, 1.0
log_cosecant, 0.0

```
> # determination of the double-precision floating-point number
> # closest to a multiple of Pi
> l:=fclosestC(2,53,1,1023,Pi,400);
```

$$l := [6381956970095103, 798, 9.3743318485092552222 \times 10^{-19}]$$

```
> Digits:=400:affiche(2,Pi,l[1],l[2],l[3],20);
```

"*mantissa*", 6381956970095103.0
"*exponent*", 798
"*epsilon*", $9.3743318485092552222 \times 10^{-19}$
log_epsilon, -59.887917936171321938
closest_fp_nb, $1.0638745296653082833 \times 10^{256}$
log_closest_fp_nb, 850.50292030468859937
distance, $9.3743318485092552222 \times 10^{-19}$
log_distance, -59.887917936171321938
tangent, $9.3743318485092552222 \times 10^{-19}$
log_tangent, -59.887917936171321937
cotangent, 1066742692876851921.8
log_cotangent, 59.887917936171321937
secant, -1.0
log_secant, 0.0
cosecant, -1066742692876851921.8
log_cosecant, 59.887917936171321937

```
> # The fact that sec is exactly equal to -1 is surprising...
> # but it is none of our business: we are interested only
> # in overflows.
```

```
> # determination of the double-extended precision floating-point number
> # closest to a multiple of Pi/2
> l:=fclosestC(2,64,1,16384,Pi/2,1000);
```

$$l := [13210900446005298036, 3194, 0.0]$$

```
> # surprising: epsilon is zero again...
> # let's retry this computation with some more digits

> l:=fclosestC(2,64,1,16384,Pi/2,2000);
```

$$l := [17211587951603246563, 6515, 0.0]$$

```
> l:=fclosestC(2,64,1,16384,Pi/2,4000);
```

$$l := [9693042348290821030, 13160, 0.0]$$

```
> l:=fclosestC(2,64,1,16384,Pi/2,5000);
```

$$l := [17476981849448541921, 10531, 1.8234027806337770702 \times 10^{-23}]$$

```
> Digits:=5000:affiche(2,Pi/2,l[1],l[2],l[3],20);}{}
```

"*mantissa*", 17476981849448541921.0
"*exponent*", 10531
"*epsilon*", $1.8234027806337770702 \times 10^{-23}$
log_epsilon, -75.537712901647601204
closest_fp_nb, $2.4510421094328914300 \times 10^{3189}$
log_closest_fp_nb, 10594.922089866028635
distance, $1.8234027806337770702 \times 10^{-23}$
log_distance, -75.537712901647601207
tangent, 54842518099726748007000.0
log_tangent, 75.537712901647601204
cotangent, $1.8234027806337770702 \times 10^{-23}$
log_cotangent, -75.537712901647601204
secant, $-54842518099726748007000.0$
log_secant, 75.537712901647601204
cosecant, -1.0
log_cosecant, 0.0

```
> # determination of the double-extended precision
> # floating-point number closest to a multiple of Pi
> l:=fclosestC(2,64,1,16384,Pi,5000);
```

$$l := [17476981849448541921, 10532, 3.6468055612675541404 \times 10^{-23}]$$

```
> Digits:=5000:affiche(2,Pi,l[1],l[2],l[3],20);
```

"*mantissa*", 17476981849448541921.0
"*exponent*", 10532
"*epsilon*", $3.6468055612675541404 \times 10^{-23}$
log_epsilon, -74.537712901647601204
closest_fp_nb, $4.9020842188657828600 \times 10^{3189}$
log_closest_fp_nb, 10595.922089866028635
distance, $3.6468055612675541404 \times 10^{-23}$
log_distance, -74.537712901647601206
tangent, $-3.6468055612675541404 \times 10^{-23}$
log_tangent, -74.537712901647601203
cotangent, $-27421259049863374004000.0$
log_cotangent, 74.537712901647601203
secant, -1.0
log_secant, 0.0
cosecant, 27421259049863374004000.0
log_cosecant, 74.537712901647601203

```
> # determination of the quadruple-precision
> # floating-point number closest to a multiple of Pi/2
> l:=fclosestC(2,113,1,16384,Pi/2,10000);
```

$$l := [87948731350338293497021849247226 39, 1852, 7.8813600082722437657 \times 10^{-38}]$$

```
> Digits:=7000:affiche(2,Pi/2,l[1],l[2],l[3],20);
```

$$\begin{aligned}
&\text{“}mantissa\text{”}, 8.7948731350338293497 \times 10^{33}\\
&\text{“}exponent\text{”}, 1852\\
&\text{“}epsilon\text{”}, 7.8813600082722437657 \times 10^{-38}\\
&log_epsilon, -123.25482300295536669\\
&closest_fp_nb, 2.8299681573086455207 \times 10^{591}\\
&log_closest_fp_nb, 1964.7602898984312882\\
&distance, 7.8813600082722437657 \times 10^{-38}\\
&log_distance, -123.25482300295536669\\
&tangent, 1.2688165480962727524 \times 10^{37}\\
&log_tangent, 123.25482300295536669\\
&cotangent, 7.8813600082722437657 \times 10^{-38}\\
&log_cotangent, -123.25482300295536669\\
&secant, 1.2688165480962727524 \times 10^{37}\\
&log_secant, 123.25482300295536669\\
&cosecant, 1.0\\
&log_cosecant, 0.0
\end{aligned}$$

```
> # determination of the quadruple-precision
> # floating-point number closest to a multiple of Pi
> l:=fclosestC(2,113,1,16384,Pi,10000);
```

$$l := [879487313503382934970218492472263 9, 1853, 1.5762720016544487531 \times 10^{-37}]$$

```
> Digits:=7000:affiche(2,Pi,l[1],l[2],l[3],20);
```

"*mantissa*", $8.7948731350338293497 \times 10^{33}$
"*exponent*", 1853
"*epsilon*", $1.5762720016544487531 \times 10^{-37}$
log_epsilon, -122.25482300295536669
closest_fp_nb, $5.6599363146172910415 \times 10^{591}$
log_closest_fp_nb, 1965.7602898984312882
distance, $1.5762720016544487531 \times 10^{-37}$
log_distance, -122.25482300295536669
tangent, $-1.5762720016544487531 \times 10^{-37}$
log_tangent, -122.25482300295536669
cotangent, $-6.3440827404813637622 \times 10^{36}$
cotangent, $-4.6871659242546276111 \times 10^{-19}$
log_cotangent, 122.25482300295536669
secant, -1.0
log_secant, 0.0
cosecant, $6.3440827404813637622 \times 10^{36}$
log_cosecant, 122.25482300295536669

References

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* Dover, 1965, 1972. Originally published by National Bureau of Standards in 1964.

[2] 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.

[3] G.A. Baker. *Essentials of Padé approximants.* Academic Press, New York, 1975.

[4] C: ISO/IEC 9899. Revision of the standardization of the C language, known as C99. `http://www.open-std.org/JTC1/SC22/WG14/`, 1999.

[5] 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):86–100, August 1984.

[6] CRLIBM: correctly rounded math library. `https://lipforge.ens-lyon.fr/projects/crlibm/`, 2004.

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

[8] Draft IEEE Standard for Binary Floating-Point Arithmetic, (draft of the 17th of December, 2002). `http://www.validlab.com/754R/`, 2002.

[9] American National Standards Institute, Institute of Electrical, and Electronic Engineers. IEEE standard for radix independent floating-point arithmetic. *ANSI/IEEE Standard, Std 854-1987, New York*, 1987.

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

[11] W. Kahan. Computer system support for scientific and engineering computation. `http://www.validlab.com/fp-1988/lectures/`, 1988.

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

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

[14] W. Kraemer and A. Bantle. Automatic forward error analysis for floating-point algorithms. *Reliable Computing*, 7:321–340, 2001.

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

[16] V. Lefèvre, J.-M. Muller, and A. Tisserand. Toward correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, 1998.

[17] Binding techniques. LIA-ISO/IEC 10967-2: Language Independent Arithmetic. LIA-2: Elementary Numerical Functions. `http://www.open-std.org/JTC1/SC22/WG11/`, 2001.

[18] R. Moore. *Interval analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[19] J.M. Muller. *Elementary Functions, Algorithms and Implementation.* Birkhauser, Boston, 1997.

[20] K. C. Ng. Argument reduction for huge arguments: Good to the last bit (can be obtained by sending an e-mail to the author: kwok.ng@eng.sun.com). Technical report, SunPro, 1992.

[21] D. Stehlé, V. Lefèvre, and P. Zimmermann. Worst Cases and Lattice Reduction. In Jean-Claude Bajard and Michael Schulte, editors, *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pages 142–147, Santiago de Compostela, Spain, 2003. IEEE Computer Society Press, Los Alamitos, CA. `http://www.ece.ucdavis.edu/acsel/arithmetic/arith16/papers/ARITH16_Stehle.pdf`

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399