Worst Cases for Correct Rounding of the Elementary Functions in Double Precision

Vincent Lefèvre
INRIA, Projet Spaces, LORIA, Campus Scientifique
B.P. 239, 54506 Vandoeuvre-lès-Nancy Cedex, FRANCE
Vincent.Lefevre@loria.fr

Jean-Michel Muller
CNRS, Projet CNRS/ENS Lyon/INRIA Arnaire
LIP, Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, FRANCE
Jean-Michel.Muller@ens-lyon.fr

Abstract

We give the results of a four-year search for the worst cases for correct rounding of the major elementary functions in double precision. These results allow the design of reasonably fast routines that will compute these functions with correct rounding, at least in some interval, for any of the four rounding modes specified by the IEEE-754 standard. They will also allow one to easily test libraries that are claimed to provide correctly rounded functions.

1 Introduction

In general, the result of an arithmetic operation on two floating-point (FP) numbers is not exactly representable in the same FP format: it must be *rounded*. In a FP system that follows the IEEE 754 standard [2, 5], the user can choose an *active rounding mode* from: rounding towards $-\infty$, $+\infty$, 0 and to the nearest. The standard requires that the system should behave as if the results of the operations $+$, $-$, $\div$, $\times$ and $\sqrt{x}$ were first computed with "infinite precision", and then rounded accordingly to the active rounding mode. Operations that satisfy this property are called correctly (or exactly) rounded.

Unfortunately, there is no such requirement for the elementary functions[1], probably because it has been believed that correct rounding of these functions would be too expensive for double precision (for single precision, since checking 2^{32} input numbers is quickly done, there already exist libraries that provide correct rounding. See for instance [11]).

Requiring correctly rounded results would not only improve the accuracy of computations: it would help to make numerical software more portable. Moreover, as noticed by Agarwal et al. [1], correct rounding facilitates the preservation of useful properties such as monotonicity, symmetry and important identities. See [10] for more details.

Before going further, let us start with definitions. We call **Infinite mantissa** of a nonzero real number x the number

$$\mathcal{M}_\infty(x) = x/2^{\lfloor \log_2 |x| \rfloor}.$$

$\mathcal{M}_\infty(x)$ is the real number x' such that $1 \leq x' < 2$ and $x = x' \times 2^k$, where k is an integer. If x is a FP number, then $\mathcal{M}_\infty(x)$ is the mantissa of its FP representation. If a and b belong to the same "binade" (they have the same sign and satisfy $2^p \leq |a|, |b| < 2^{p+1}$, where p is an integer), we call their **Mantissa distance** the distance between their infinite mantissas, that is, $|a - b|/2^p$.

Let f be an elementary function and x a FP number. Unless x is a very special case – e.g., $\log(1)$ or $\sin(0)$ –, $y = f(x)$ cannot be exactly computed. The only thing we can do is to compute an *approximation* y^* to y. If we wish to provide correctly rounded functions, we have to know what the accuracy of this approximation should be to make sure that rounding y^* is equivalent to rounding y. In other words, from y^* and the known bounds on the approximation, the only information we have is that y belongs to some interval Y. Let us call $\diamond$ the rounding function. Let us call a **breakpoint** a value z where the rounding changes, that is, if t_1 and t_2 are real numbers satisfying $t_1 < z < t_2$ then $\diamond(t_1) < \diamond(t_2)$. For "directed" rounding modes (i.e., towards

[1] By *elementary functions* we mean the radix 2, e and 10 logarithms and exponentials, and the trigonometric and hyperbolic functions.

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

$+\infty$, $-\infty$ or 0), the breakpoints are the FP numbers. For rounding to the nearest mode, they are the exact middle of two consecutive FP numbers.

If Y contains a breakpoint, then we cannot provide $\diamond(y)$: the computation must be carried again with a larger accuracy. There are two ways of solving that problem:

- iteratively increase the accuracy of the approximation, until interval Y no longer contains a breakpoint[2]. The problem is that it is difficult to predict how many iterations will be necessary;
- compute, only once and in advance, the smallest nonzero mantissa distance between the image[3] of a FP number and a breakpoint. This makes it possible to deduce the accuracy with which $f(x)$ must be approximated to make sure that rounding the approximation is equivalent to rounding the exact result.

The first solution was suggested by Ziv [12]. It has been implemented in a library available through the internet[4]. The last iteration uses 768 bits of precision. There is no proof that this suffices (the results presented in this paper actually give the proof for the functions and domains considered here), but probabilistic arguments[3, 4, 10] show that requiring a larger precision is extremely unlikely.

We decided to implement the second solution, since the only way to implement the first one safely is to overestimate the accuracy that is needed in the worst cases. The basic principle of our algorithm for searching the worst cases was outlined in [9]. We now present properties that have allowed us to hasten the search, as well as the results obtained after having run our algorithms for 4 years on several workstations, and consequences of our results. The results we have obtained are worst cases for the Table Maker's Dilemma, that is, FP numbers whose image is closest to a breakpoint. For instance, the worst case for the natural logarithm in the full double precision range is attained for

$$x = 1.01100010101010001000011000010011011000101 0\,0110110110 \times 2^{678}$$

whose logarithm is

$$\log x = \overbrace{111010110.01000111100111101\cdots110001}^{53\text{ bits}} \underbrace{000000000000000000\cdots000000000000000}_{65\text{ zeroes}}1110...$$

[2] This is not possible if $f(x)$ is *equal* to a breakpoint. And yet, $x = 0$ is the only FP input value for which $\sin(x)$, $\cos(x)$, $\tan(x)$, $\arctan(x)$ and e^x have a finite radix-2 representation – and the breakpoints *do* have finite representations –, and $x = 1$ is the only FP input value for which $\ln(x)$ has a finite representation. Concerning 2^x and 10^x, they have a finite representation if and only if x is an integer. Also, $\log_2(x)$ (resp. $\log_{10}(x)$) has a finite representation if and only if x is an integer power of 2 (resp. 10). All these cases are straightforwardly handled separately, so we do not discuss them in the rest of the paper.

[3] We call *image* of x the number $f(x)$, where f is the elementary function being considered.

[4] http://www.alphaWorks.ibm.com/tech/mathlibrary4java.

This is a "difficult case" in a directed rounding mode, since it is very near a FP number. One of the two worst cases for radix-2 exponentials in the full double precision range is

$$1.11100100010110010110010100100110101111111 00101001101 \times 2^{-10}$$

whose radix-2 exponential is

$$\overbrace{1.00000000010100111111110000 10111\cdots0011}^{53\text{ bits}} 0\underbrace{11111111111111111\cdots11111111111111111}_{59\text{ ones}}0100...$$

It is a difficult case for rounding to the nearest, since it is very close to the middle of two consecutive FP numbers.

2 Our algorithms for finding the worst cases

2.1 Basic principles

The basic principles are given in [9], so we only quickly describe them and focus on new aspects. Assume we wish to find the worst cases for function f in double precision. Let us call **test number** (TN) a number that is representable with **54** bits of mantissa (it is either a FP number or the exact middle of two consecutive FP numbers). The TNs are the values that are breakpoints for one of the rounding modes. Finding worst cases now reduces to the problem of finding FP numbers x such that $f(x)$ is closest (for the mantissa distance) to a TN. We proceed in two steps: we first use a fast "filtering" method that eliminates all points whose distance to the closest breakpoint is above a given threshold. The value of the threshold is chosen so that this filtering method does not require highly accurate computations, and so that the number of values that remain to be checked after the filtering is so small that an accurate computation of the value of the function at each remaining value is possible. Details on the choice of parameters are given in [8].

In [9], we suggested to perform the filtering as follows:

- first, the domain where we look for worst cases is split into "large subdomains" where all input values have the same exponent;
- each large subdomain is split into "small subdomains" that are small enough so that in each of them, within the accuracy of the filtering, the function can be approximated by a linear function. Hence in each small subdomain, our problem is to find a point on a grid that is closest to a straight line. We solve a slightly different problem: given a threshold ϵ we just try to know if there can be a point of the grid at distance less than ϵ from the straight line. ϵ is chosen so that for one given small subdomain this event is very unlikely.

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

- using a variant to the Euclidean algorithm suggested by V. Lefèvre [7], we solve that problem. If we find that there can be a point of the grid at distance less than ϵ from the straight line, we check all points of the small subdomain.

2.2 Optimization: f and f^{-1} simultaneously

Let us improve that method. Instead of finding **floating-point numbers** x such that $f(x)$ is closest to a test number, we look for **test numbers** x such that $f(x)$ is closest to a TN. This makes it possible to compute worst cases for f and for its inverse f^{-1} in one pass only (the image $f(a)$ of a breakpoint a is near a breakpoint b if and only if $f^{-1}(b)$ is near a). One could object that by checking the images of TNs instead of checking the double precision FP numbers only, we double the number of points that are examined. So getting in one pass the results for two functions (f and f^{-1}) seems to be a no-win no-loss operation. This is not quite true, since there are sometimes *much fewer* values to check for one of the two functions than for the other one.

Consider as an example the radix-2 exponential and logarithm, with input domain $I = [-1, 1]$ for 2^x, which corresponds to input domain $J = [1/2, 2]$ for $\log_2(y)$. The two following strategies would lead to the same final result: the worst cases for 2^x in I and for $\log_2(y)$ in J.

1. check 2^x for every test number x in I;
2. check $\log_2(y)$ for every test number y in J.

If we use the first strategy, we need to check all TNs of exponent between[5] -53 and -1. There are 106×2^{53} such numbers. With the second strategy, we need to check all positive TNs of exponent equal to -1 or 0, that is, 2×2^{53} numbers. The second strategy is approximately 53 times as fast as the first one.

If we separately check all FP numbers in I and all FP numbers in J, we check $106 \times 2^{52} + 2 \times 2^{52}$. The second strategy is 27 times as fast as this last method.

Hence, in the considered domain, it is much better to check $\log_2(y)$ for every TN y in $[1/2, 2]$. In other domains, the converse holds: when we want to check both functions in the domain defined by $x > 1$ (for 2^x) or $y > 2$ (for $\log_2(y)$), we only have to consider 10 values of the exponent if we check 2^x for every TN in the domain, whereas we would have to consider 1022 values of the exponent if we decided to check $\log_2(x)$ for the TNs in the corresponding domain.

The decision whether it is better to base our search for worst cases on the examination of f in a given domain I or f^{-1} in $J = f(I)$ can be helped by examining $T_f(x) = |x \times f'(x)/f(x)|$ in I. If $T_f(x) \gg 1$, then I contains fewer test numbers than J, so it is preferable to check f in I. If $T_f(x) \ll 1$, it is preferable to check f^{-1} in J. When $T_f(x) \approx 1$, a more thorough examination is necessary. In all cases, another important point is which of the two functions is simpler to approximate.

[5] For numbers of smaller absolute value, there is no longer any problem of implementation: their radix-2 exponential is 1 or $1^- = 1 - ulp(1/2)$ or $1^+ = 1 + ulp(1)$ depending on their sign and the rounding mode.

2.3 Optimization: special input values

For most functions, it is not necessary to perform tests for the input arguments that are extremely close to 0. For example, consider the exponential of a very small positive number ϵ, on a FP format with p-bit mantissas, assuming rounding to nearest. If $\epsilon < 2^{-p}$ then (since ϵ is a p-bit number), $\epsilon \leq 2^{-p} - 2^{-2p}$. Hence,

$$e^{\epsilon} \leq 1 + (2^{-p} - 2^{-2p}) + \frac{1}{2}(2^{-p} - 2^{-2p})^2 \ldots < 1 + 2^{-p}.$$

therefore $\exp(\epsilon) < 1 + (1/2)ulp(1)$. Thus, the correctly rounded value of $\exp(\epsilon)$ is 1. A similar reasoning can be done for other functions and rounding modes. Some results are given in Table 1.

2.4 Normal and denormal numbers

Our algorithms assume that input and output numbers are normalized FP values. Hence, we have to check whether there exist normalized FP numbers x such that $f(x)$ is so small that we should return a denormal number. To do that, we use a method based on the continued fraction theory, suggested by Kahan [6], and originally designed for finding the worst cases for range reduction. It gives the normalized FP number that is closest to an integer nonzero multiple of $\pi/2$. This number is $\alpha = 16367173 \times 2^{72}$ in single precision, and $\beta = 6381956970095103 \times 2^{797}$ in double precision. Therefore, $A = |\cos(\alpha)| \approx 1.6 \times 10^{-9}$ and

Table 1. *Some results for small values in double precision, assuming rounding to the nearest. These results make finding worst cases useless for negative exponents of large absolute value.*

This function	can be replaced by	when
$\exp(\epsilon), \epsilon \geq 0$	1	$\epsilon < 2^{-53}$
$\exp(\epsilon), \epsilon \leq 0$	1	$\lvert\epsilon\rvert \leq 2^{-54}$
$\sin(\epsilon), \arcsin(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq 2^{-26}$
$\cos(\epsilon)$	1	$\lvert\epsilon\rvert \leq \sqrt{2} \times 2^{-27}$
$\tan(\epsilon), \arctan(\epsilon)$	ϵ	$\lvert\epsilon\rvert \leq 2^{-27}$

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

$B = |\cos(\beta)| \approx 4.7 \times 10^{-19}$ are lower bounds on the absolute value of the sine, cosine and tangent of normalized single precision (for A) and double precision (for B) FP numbers. These values are larger than the smallest normalized FP numbers. Hence the sine, cosine and tangent of a normalized FP number can always be rounded to normalized FP numbers.

3 Implementation of the method

3.1 Overview of the implementation

The tests are implemented in three steps:

1. As said above, the first step is a *filter*. It amounts to testing if 32 (in general) consecutive bits are all zeroes[6] thus keeping one argument out of 2^{32}, in average. This step is very slow and needs to be parallelized.

2. The 2nd step consists of reducing the number of worst cases obtained from the first step and grouping all the results together in the same file. This is done with a slower but more accurate test than in the 1st step. As the number of arguments has been drastically reduced, this step is performed on a single machine.

3. The 3rd step is run by the user to restrict the number of worst cases. Results on the inverse function are also obtained. This step is very fast.

Most programs are written in Perl (text data handling, process control...). The tests of the first step have been written in Sparc assembly language, as they need to be as fast as possible. For the other calculations, we have used Maple with an interval arithmetic package.

3.2 Details on the first step

Let us give more details about the first step. The user chooses a function f, an exponent, a mantissa size (usually 53), and the first step starts as follows.[7]

- First, the tested interval is split into 2^{13} subintervals J_i containing 2^{40} TNs and f is approximated by polynomials P_i of degree d_i (~ 4 to 20) on J_i. For each i, we start with $d_i = 1$, and increase d_i until the approximation is accurate enough. P_i is expressed modulo the distance between two consecutive TNs, as we only need to estimate the bits following the mantissa.
- Then, each J_i is split into subintervals $K_{i,j}$ containing 2^{15} arguments and P_i is approximated by degree-2 polynomials $Q_{i,j}$ on $K_{i,j}$, with 64-bit precision.
- On $K_{i,j}$: $Q_{i,j}$ is approximated by a degree-1 polynomial (by ignoring the degree-2 coefficient) and the variant of the Euclidean algorithm is used. If it fails, that is, if the obtained distance is too small, then:
  - $K_{i,j}$ is split into 4 subintervals $L_{i,j,k}$.
  - For each k: the Euclidean algorithm is used on $L_{i,j,k}$, and if it fails, the arguments are tested the one after the other, using two 64-bit additions for each argument.

The program performs the first point (using Maple), generates a C/assembly source for the following points, then compiles and executes it.

The first step requires much more time than the other ones, thus it is parallelized (we have used around 100 workstations, in background). As the calculations in different intervals are totally independent, there is no need for communications between the different machines. We only have a server that distributes intervals to each client (the program that performs the tests). The workstations have primary users. We must not disturb them. So, the programs were written so that they can run with a low priority, automatically stop after a given time, and automatically detect when a machine is used and stop if this is the case.

4 Results: e^x and $\ln(x)$

For these functions, there is no known way of deducing the worst cases in a domain from the worst cases in another domain. And yet, we have obtained the worst cases for all possible double precision FP inputs. They are given in Tables 2 and 3. From these results we can deduce the following properties.

Property 1 (Computation of exponentials) *Let y be the exponential of a double-precision number x. Let y^* be an approximation to y such that the mantissa distance[8] between y and y^* is bounded by ϵ.*

- *for $|x| \geq 2^{-30}$, if $\epsilon \leq 2^{-53-59} = 2^{-112}$ then for any of the 4 rounding modes, rounding y^* is equivalent to rounding y;*
- *for $|x| < 2^{-30}$, if $\epsilon \leq 2^{-53-104} = 2^{-157}$ then rounding y^* is equivalent to rounding y.*

[6] These are the bits following the first 54 bits of the mantissa, unless the exponent of the output values changes in the tested domain.

[7] The numbers given here are just those that are generally chosen; other values may be chosen for particular cases.

[8] If one prefers to think in terms of relative error, one can use the following well-known properties: if the mantissa distance between y and y^* is less than ϵ then their relative distance $|y - y^*|/|y|$ is less than ϵ. If the relative distance between y and y^* is less than ϵ_r then their mantissa distance is less than $2\epsilon_r$.

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

Property 2 (Computation of logarithms) *Let y be the natural (radix-e) logarithm of a double-precision number x. Let y^* be an approximation to y such that the mantissa distance between y and y^* is bounded by ϵ. If $\epsilon \leq 2^{-53-64} = 2^{-117}$ then for any of the 4 rounding modes, rounding y^* is equivalent to rounding y.*

5 Results: 2^x and $\log_2(x)$

5.1 Radix-2 exponentials

Using the identity $2^{n+x} = 2^n 2^x$ allows one to efficiently speed the search. First, getting the worst cases for $x \in [1, 2)$ makes it possible to derive all worst cases for $x < -1$ and $x > 1$. The worst cases for $|x| < 1$ were obtained through the radix-2 logarithm in $(1/2, 2)$. These results, given in Table 4, make it possible to deduce the following property.

Property 3 (Computation of radix-2 exponentials) *Let y be the radix-2 exponential 2^x of a double-precision number x. Let y^* be an approximation to y such that the mantissa distance between y and y^* is bounded by ϵ. If $\epsilon \leq 2^{-53-59} = 2^{-112}$ then for any of the 4 rounding modes, rounding y^* is equivalent to rounding y.*

5.2 Radix-2 logarithms

Concerning radix-2 logarithms, let us show that it suffices to test the input numbers greater than 1, and whose exponent is a power of 2.

First, it suffices to test the input numbers whose exponent is a positive power of 2 to get the worst cases for all input values greater than 1. Consider $x = m \times 2^p$, with $p \geq 1$ and $1 \leq m < 2$. Define $y = \log_2(x) = p + \log_2(m)$. $\mathcal{M}_\infty(y)$ begins with $\ell_p = \lfloor \log_2(p) \rfloor$ bits that represent p, followed by the representation of $\log_2(m)$. Let $p' = 2^{\ell_p}$. Since $\lfloor \log_2(p') \rfloor = \lfloor \log_2(p) \rfloor = \ell_p$, the infinite mantissa of the radix-2 logarithm y' of $x' = m \times 2^{p'}$ has the same bits as $\mathcal{M}_\infty(y)$ after position ℓ_p. Hence, there is a chain of k consecutive 1s (or 0s) after bit 54 of $\mathcal{M}_\infty(y)$ if and only if there is a chain of k consecutive 1s (or 0s) after bit 54 of $\mathcal{M}_\infty(y')$. Hence, from the worst cases for an exponent equal to 2^ℓ we deduce the worst cases for exponents between $2^\ell + 1$ and $2^{\ell+1} - 1$. In Table 5, we only give one of the worst cases: the input value has exponent 512. The other ones have the same mantissa, and exponents between 512 and 1023.

Now, let us show how to deduce the worst cases for numbers less than 1 from the worst cases for numbers greater than 1. Consider a FP number $x = m \times 2^{-p}$, with $1 < m < 2$, and $p \geq 1$. Define $y = \log_2(x) = -p + \log_2(m)$. The integer part of $|y|$ is $p - 1$ and its fractional part is $1 - \log_2(m)$. So $\mathcal{M}_\infty(y)$ begins with the bits that represent $p - 1$, followed by the bits that represent $1 - \log_2(m)$. Now, consider the FP number $x' = m \times 2^{p-1}$. Define $y' = \log_2(x') = (p - 1) + \log_2(m)$. $\mathcal{M}_\infty(y')$ begins with the bits that represent $p - 1$ (the same as for y), followed by the bits that represent $\log_2(m)$. But the bits that represent $1 - \log_2(m)$ are obtained by complementation[9] of the bits that represent $\log_2(m)$. Hence, there is a chain of k consecutive 1s (or 0s) after bit 54 of $\mathcal{M}_\infty(y)$ if and only if there is a chain of k consecutive 0s (or 1s) after bit 54 of $\mathcal{M}_\infty(y')$. Therefore, x is the worst case for input values < 1 if and only if x' is the worst case for input values > 1. This is illustrated in Table 5: the infinite mantissa of the worst case for $x > 1$ starts with the same bit chain (1000000000) as the mantissa of the worst case for $x < 1$, then the bits that follow are complemented ($100010001111110\ldots000110\quad 0\quad 0^{55}1100\ldots$ for the case $x < 1$ and $011101110000001\ldots111001\quad 1\quad 1^{55}0011\ldots$ for $x > 1$).

Using these properties, we rather quickly obtained the worst cases for the radix-2 logarithm of all possible double precision input values: it sufficed to run our algorithm for the input numbers of exponents 0, 1, 2, 4, 8, 16, ... 512.

These results, given in Table 5, make it possible to deduce the following property.

Property 4 (Computation of radix-2 logarithms) *Let y be the radix-2 logarithm $\log_2(x)$ of a double-precision number x. Let y^* be an approximation to y such that the mantissa distance between y and y^* is bounded by ϵ. If $\epsilon \leq 2^{-53-55} = 2^{-108}$ then for any of the 4 rounding modes, rounding y^* is equivalent to rounding y.*

6 Results: trigonometric functions

The results given in Tables 6 to 11 give the worst cases for functions sin, arcsin, cos, arccos, tan and arctan. For these functions, we have worst cases in some bounded domain only, because they are more difficult to handle than the other functions. And yet, it is sometimes possible to prune the search. Consider the arc-tangent of large values. The double precision number that is closest to $\pi/2$ is

$$\alpha = 884279719003555/2^{49}.$$

Assuming rounding to the nearest, the breakpoint that is immediately below α is $\beta = 141484755040568\,79/2^{53}$. For any FP number x, if $\arctan(x) \geq \beta$ then the correctly rounded (to the nearest) value that should be returned when evaluating $\arctan(x)$ in double precision is α. Hence, for $x \geq 5805358775541311$, we should return α. Similarly, for $2536144836019042 \leq x \leq 5805358775541310$, we should return $\alpha - ulp(\alpha)$.

[9] 1 is replaced by 0 and 0 is replaced by 1.

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 IEEE

Table 2. *Worst cases for the exponential function in the full range. Exponentials of numbers less than* $\ln(2^{-1074})$ *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* $\ln(2^{1024})$ *are overflows.*

Interval	worst case (binary)
$[\ln(2^{-1074}), -2^{-30}]$	$\exp(-1.11101101001100011000111011111011011000100111111101010 \times 2^{-27})$ $= 1.11111111111111111111111111000010010110011100111000100 \quad 1 \quad 1^{59}0001... \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.00000000000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.11111111111111111111111111111111111111111111111111111100 \quad 0 \quad 0^{100}1010... \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.11111111111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.00000000000000000000000000000000000000000000000000000000 \quad 1 \quad 1^{104}0101...$
$[2^{-30}, \ln(2^{1024})]$	$\exp(1.01111111111111100111111111111111011100000000000100100 \times 2^{-32})$ $= 1.00000000000000000000000000000001011111111111111101000 \quad 0 \quad 0^{57}1101...$ $\exp(1.10000000000000010111111111111111011011111111111011100 \times 2^{-32})$ $= 1.00000000000000000000000000000001100000000000000010111 \quad 1 \quad 1^{57}0010...$ $\exp(1.10011110100111001011101111111101011000001000000001011 \times 2^{-31})$ $= 1.00000000000000000000000000000011001111010011100101111 \quad 1 \quad 0^{57}1010...$ $\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \quad 0 \quad 0^{57}1000...$

Table 3. *Worst cases for the natural (radix* e*) logarithm in the full range.*

Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.10010100011101101110001100000100110011010111111000111 \times 2^{-384})$ $= -100001001.10110110000011001010111101000111101100110101 \quad 1 \quad 0^{60}1010...$ $\log(1.11101010011100011101100001011100111011100000000100000 \times 2^{-509})$ $= -101100000.00101001011010100110011010110100001011111111 \quad 1 \quad 1^{60}0000...$ $\log(1.00100110111010011100010011010011001001111001011000000 \times 2^{-232})$ $= -10100000.10101011001011000010010111100110100001000000100 \quad 0 \quad 0^{60}1001...$
$(1, 2^{1024}]$	$\log(1.01100010101010001000011000010011011000101001101101110 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{64}1110...$

For rounded to nearest arc-tangent, the worst case for input numbers larger than 2.25×10^{12} is

$$4621447055448553/2^{11} = 2256565945043.2387\,69\,53\,12\,5$$

whose arc-tangent is

$$\overbrace{1.100100100001111110\ldots100}^{53 \text{ bits}} \quad 1 \quad 0^{45}111011\ldots$$

From the result given in Table 6 we deduce:

Property 5 (Computation of sines) *Let* y *be the sine of a double-precision number* x *satisfying* $1/32 \leq |x| \leq 2$*. Let* y^* *be an approximation to* y *such that the mantissa distance between* y *and* y^* *is bounded by* ϵ*. If* $\epsilon \leq 2^{-53-65} = 2^{-118}$ *then for any of the* 4 *rounding modes, rounding* y^* *is equivalent to rounding* y*.*

Using Tables 7 to 11, similar properties are deduced for the other trigonometric functions: $\epsilon \leq 2^{-117}$ for arc-sine between $\sin(1/32)$ and 1; $\epsilon \leq 2^{-108}$ for cosine between $1/64$ and $12867/8192$; $\epsilon \leq 2^{-115}$ for arc-cosine between $\cos(12867/8192)$ and $\cos(1/64)$; $\epsilon \leq 2^{-110}$ for tangent between $1/32$ and $\arctan(2)$; and $\epsilon \leq 2^{-108}$ for arc-tangent between $\tan(1/32)$ and 2.

Conclusion and future work

The worst cases we have obtained will make possible the design of efficient routines for evaluating most common functions with correct rounding (at least in some intervals) in the four rounding modes specified by the IEEE-754 standard. We are extending the domains for the functions for which we have not yet obtained the worst cases in the full range. These worst cases will also be good test cases for checking whether a library provides correct rounding or not. Since the machines are getting faster and faster, what we have done for double precision will probably be feasible for double extended precision within a few years. Concerning

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 **IEEE**

Table 4. *Worst cases for the radix-2 exponential function 2^x in the full range. Integer values of x are omitted.*

Interval	worst case (binary)
$[-1074, 0)$	$2**(-1.00101000011000111010101110101110101011110111110110010 \times 2^{-15})$ $= 0.11111111111111100110010100011111010001100000111101111 \quad 0 \quad 0^{57}1110...$
	$2**(-1.01000001011011110110110001100100010001011010110011111 \times 2^{-20})$ $= 0.11111111111111111111001000010011001010111010011001110 \quad 1 \quad 1^{57}0000...$
	$2**(-1.00000101010101100000000011100100010101011001111110001 \times 2^{-32})$ $= 0.11111111111111111111111111111111101001010110110110001 \quad 1 \quad 1^{57}0000...$
	$2**(-1.00011000010110111000110110110110110101010110000000111101 \times 2^{-33})$ $= 0.11111111111111111111111111111111100111101101010111100 \quad 0 \quad 0^{57}1100...$
$(0, 1024]$	$2**(1.10111111101110111101110010001001110110111111000101 \times 2^{-25})$ $= 1.00000000000000000000000010011011001011000011100000101 \quad 0 \quad 0^{59}1011...$
	$2**(1.11100100010110010110010100100110101111111001010011101 \times 2^{-10})$ $= 1.00000000010100111111110000101110110000101011010100011 \quad 0 \quad 1^{59}0100...$

Table 5. *Worst cases for $\log_2(x)$ in the full range. Values of x that are integer powers of 2 are omitted.*

Interval	worst case (binary)
$(0, 1/2)$	$\log_2(1.01100001010101010101111110111010110001000010110110100 \times 2^{-513})$ $= -1000000000.10001000111111010010111111000010110010001100 \quad 0 \quad 0^{55}1100...$
$(1/2, 2^{1024})$	$\log_2(1.01100001010101010101111110111010110001000010110110100 \times 2^{512})$ $= 1000000000.01110111000000101101000000111101001101110011 \quad 1 \quad 1^{55}0011...$

quad precision, the only hope of getting similar results in the near future is a possible algorithmic breakthrough. The path to such a breakthrough could be a reasonably fast algorithm for getting the point of a regular grid that is closest to a polynomial curve of degree 2.

References

[1] R. C. Agarwal, J. C. Cooley, F. G. Gustavson, J. B. Shearer, G. Slishman, and B. Tuckerman. New scalar and vector elementary functions for the IBM system/370. *IBM J. of Research and Development*, 30(2):126–144, March 1986.

[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] C. B. Dunham. Feasibility of "perfect" function evaluation. *SIGNUM Newsletter*, 25(4):25–26, October 1990.

[4] S. Gal and B. Bachelis. An accurate elementary mathematical library for the IEEE floating point standard. *ACM Trans. on Math. Software*, 17(1):26–45, March 1991.

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

[6] W. Kahan. Minimizing q*m-n, text accessible electronically at http://http.cs.berkeley.edu/∼wkahan/. At the beginning of the file "nearpi.c", 1983.

[7] V. Lefèvre. *Developments in Reliable Computing*, chapter An Algorithm That Computes a Lower Bound on the Distance Between a Segment and $\mathbb{Z}^2$, pages 203–212. Kluwer, Dordrecht, Netherlands, 1999.

[8] V. Lefèvre. *Moyens Arithmétiques Pour un Calcul Fiable*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, 2000.

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

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

[11] M. J. Schulte and E. E. Swartzlander. Hardware designs for exactly rounded elementary functions. *IEEE Trans. Computers*, 43(8):964–973, August 1994.

[12] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Trans. on Math. Software*, 17(3):410–423, September 1991.

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 **IEEE**

Table 6. *Worst cases for the sine function in the range* $[1/32, 2]$.

Interval	worst case (binary)
$[1/32, 1]$	$\sin 1.11111110011101100111011100111001110100001111011011 01 \times 2^{-2}$ $= 1.11101001100101010000011100110000110001000110100101 01 \quad 1 \quad 1^{65}0000... \times 2^{-2}$
$[1, 2]$	$\sin(1.10010010000111111011010101000100010000101101000110 00)$ $= 0.11111111111111111111111111111111111111111111111111 11 \quad 1 \quad 1^{54}0110...$

Table 7. *Worst cases for the arc-sine function in the range* $[\sin(1/32) = 0.0312449\ldots, 1]$.

Interval	worst case (binary)
$[\sin(1/32), 1]$	$\arcsin(1.11101001100101010000011100110000110001000110100101 10 \times 2^{-2})$ $= 1.11111110011101100111011100111001110100001111011011 01 \quad 0 \quad 0^{64}1000... \times 2^{-2}$

Table 8. *Worst cases for the cosine function in the range* $[1/64, 12867/8192]$. $12867/8192$ *is slightly less than* $\pi/2$.

Interval	worst case (binary)
$[1/64, 1]$	$\cos(1.00101011110011001101001111010010110001000011100011 11 \times 2^{-6})$ $= 0.11111111111010111011001101011101010000111000010101000 \quad 1 \quad 1^{55}0111...$
$[1, \frac{12867}{8192}]$	$\cos(1.01101011100010100110001001110011111010111110000100001)$ $= 1.00110011011111111110001011011000001110010110001010010 \quad 1 \quad 0^{54}1011... \times 2^{-3}$

Table 9. *Worst cases for the arc-cosine function in the range* $[\cos(12867/8192), \cos(1/64)] \approx [0.0001176, 0.999878]$.

Interval	worst case (binary)
$[\cos\left(\frac{12867}{8192}\right), \cos(1)]$	$\arccos(1.11111101011100110111101111101001000101000101011111000 \times 2^{-11})$ $= 1.10010001111000000000011011010100000111011000110110000 \quad 1 \quad 1^{62}0010...$

Table 10. *Worst cases for the tangent function in the range* $[1/32, \arctan(2)]$, *with* $\arctan(2) \approx 1.107148$.

Interval	worst case (binary)
$[\frac{1}{32}, \arctan\left(\frac{1}{2}\right)]$	$\tan(1.01010000010010000110101100101111100001110000000010100 \times 2^{-5})$ $= 1.01010000011110001100111010111111111111001110001110010 \quad 1 \quad 0^{57}1001... \times 2^{-5}$
$[\arctan\left(\frac{1}{2}\right), \arctan(2)]$	$\tan(0.10100011010101100001101110010001001000011010100110110)$ $= 0.10111101110100100100111110111001110011000001010011110 \quad 1 \quad 1^{54}0011...$

Table 11. *Worst cases for the arc-tangent function in the range* $[\tan(1/32), 2]$, *with* $\tan(1/32) \approx 0.0312601$.

Interval	worst case (binary)
$[\tan\left(\frac{1}{32}\right), \frac{1}{2}]$	$\arctan(1.10101001001100111111111000010111011010110011011101 01 \times 2^{-3})$ $= 1.10100011001111110011001010101100010111001110101101 00 \quad 1 \quad 1^{55}0110... \times 2^{-3}$
$[\frac{1}{2}, 2]$	$\arctan(0.10111101110100100100111110111001110011000001010011111)$ $= 0.10100011010101100001101110010001001000011010100110110 \quad 0 \quad 0^{55}1111...$

Proceedings of the 15th IEEE Symposium on Computer Arithmetic (ARITH'01)
1063-6889/01 $10.00 © 2001 **IEEE**