An Accurate Elementary Mathematical Library for the IEEE Floating Point Standard

SHMUEL GAL and BORIS BACHELIS
IBM Israel Science and Technology and Scientific Center

The algorithms used by the IBM Israel Scientıfic Center for the elementary mathematical library using the IEEE standard for binary floating point arithmetic are described. The algorithms are based on the "accurate tables method." Thıs methodology achieves high performance and produces very accurate results. It overcomes one of the main problems encountered in elementary mathematical functions computations: achieving last bit accuracy. The results obtained are correctly rounded for almost all argument values.

Our main idea in the accurate tables method is to use "nonstandard tables," which are different from the natural tables of equally spaced points in which the rounding error prevents obtaining last bit accuracy. In order to achieve a small error we use the following idea: Perturb the original, equally spaced, points in such a way that the table value (or tables values in case we need several tables) will be very close to numbers which can be exactly represented by the computer (much closer than the usual double precision representation). Thus we were able to control the error introduced by the computer representation of real numbers and extended the accuracy without actually using extended precision arithmetic.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*computer arıthmetıc, error analysıs, numerıcal algorıthms*; G.1.2 [**Numerical Analysis**]: Approximation—*elementary function approximation*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*

General Terms: Algorithms

Additional Key Words and Phrases: Accurate tables method, IEEE arithmetic, last bit accuracy

1. INTRODUCTION

The algorithms used by the IBM Israel Scientific Center for the elementary mathematical library using the IEEE standard for binary floating point arithmetic [7] are described. The algorithms are based on methodology developed by Gal [5], called the *accurate tables method*, which achieves high performance and produces very accurate results. It overcomes one of the main problems encountered in elementary mathematical functions computations achieving last bit accuracy. The results that we obtain are correctly rounded for almost all argument values. This methodology appears as a

Authors' address: IBM Israel Scientific Center, Technion City, Nuclear Engineering Building, Haifa 32000, Israel.

remedy to the situation expressed by Black et al. [3]. "The data (results of tests of elementary function from several computer companies) demonstrate that the industry does not satisfy the needs of those who require accurate and efficient mathematical software."

Our method is based on three steps: Range reduction for the argument, a table lookup, and a minimax polynomial approximation of the function near the table value. The range reduction is somewhat similar to the standard one, but the second step is quite different. A priori, we determine a set S of L points: $X_1 < X_2 < \dots X_L$ in the interval $[A, B]$, where $X_{i+1} - X_i \leq d$, d being a predetermined constant. In order to calculate $f(x)$ for x in $[A, B]$ we find a point X_i in S which is "close enough" to x and then calculate $f(x)$ using a table value (or some table values) associated with $f(X_i)$, and a polynomial in $x - X_i$. To clarify the idea we present two such examples.

Example 1. In order to calculate $sin(y)$ (or $cos(y)$), we first make a range reduction (discussed in Section 2.3.1) so that the reduced argument y lies in $[0, \pi/4)$. Then one can find an X_i with $-d/2 \leq y - X_i < d/2$ and use the identity

$$sin(y) = sin(X_i) \times cos(h) + cos(X_i) \times sin(h), \tag{1}$$

where $h = y - X_i$. The values of $sin(X_i)$ and $cos(X_i)$ can be extracted from a table, prepared in advance, while $cos(h)$ and $sin(h)$ can be approximated by minimax polynomials in $[-d/2, d/2]$.

Example 2. In order to calculate the exponent function $exp(y)$, first make a (standard) range reduction into the range $0 \leq y < 1$ and then use the identity

$$exp(y) = exp(X_i) \times exp(h), \tag{2}$$

where $h = |y - X_i| \leq d/2$. The value of $exp(X_i)$ can be extracted from a table, while $exp(h)$ can be approximated by a minimax polynomial $p(x)$ in $[-d/2, d/2]$.

The distance d is determined as a compromise between the two contradictory goals: (1) As d gets smaller, so does the degree of the minimax polynomial $p(x)$ needed to achieve the required accuracy. Thus the subroutine performance (speed) is improved. Working with a small d has the additional advantage from the accuracy point of view, as is seen later in the accuracy analysis. (2) On the other hand, the overall size of the tables is limited by the space allocated to them in the computer memory and thus d cannot be too small. A practical value for d used is usually 1/256.

The main idea in the accurate tables method is to use "nonstandard tables," different from the natural tables of equally spaced points, in which the rounding error prevents obtaining last bit accuracy. In order to achieve a small error we use the following idea: Perturb the original, equally spaced, points in such a way that the table value (or tables values in case we need several tables) will be very close to numbers which can be exactly represented by the computer (much closer than the usual double precision representation). Thus, we are able to control the error introduced by the computer representation of real numbers and to extend the accuracy without actually using extended precision arithmetic. The detailed analysis of the method appears in Gal [5, Sec. 4].

The accurate tables method was implemented several years ago for 370-type machines. (See [1, 2, 6].) The purpose of the present work has been to adopt this method to the IEEE Standard for Binary Floating Point Arithmetic and to implement the algorithms for the *RT-PC*. (The details of the IEEE standard are described in [7].)

The algorithms described in this report were devised for the following elementary functions: *sin*, *cos*, *tan*, *cotan*, *arctan*, *arctan* 2, *exp*, *ln*, log_{10}, *sinh*, and *cosh*. They were chosen because most of them are frequently used in engineering and scientific computations. (The functions *sinh* and *cosh* are used less frequently than the others, but they can be obtained with little investment once the *exp* routine is written, so we decided to write it as an "extra".)

Accuracy of the Results. Using the accurate tables method, the result of the subroutines is an expression of the following type:

$$r = f_i + b, \tag{3}$$

which is rounded according to the required rounding mode; where f_i is an accurate table value and b is usually smaller than $f_i \times 2^{-9}$. Thus f_i contains at least κ (implicit) zeros after bit 53 of the mantissa of f_i (where κ is an arbitrary number determined by the table builder). In our routines $\kappa \geq 11$, so the relative error induced by the tables is smaller than 2^{-63}. The relative error induced by the polynomial approximation in our routines is usually smaller than 2^{-62}. Since the relative error of the term b in (3), induced by calculating b and by its 53-bit representation, is usually about $2^{-53-9} = 2^{-62}$, it follows that usually the dominant error of the result is the error induced by the arithmetic, that is, about 2^{-62}. The implication of such an error on the proportion (Pr) of incorrectly rounded results is analyzed in Gal [5]. In this work it was demonstrated that distant bits of any elementary function, computed for a floating point argument, are approximately uniformly distributed. Using this fact, it was shown that [5, pp. 9, 14–16]

$$Pr < 2^{-(62-53)} = 2^{-9} < 0.003,$$

so the percent of arguments with correctly rounded results is greater than 99.7 percent. Our experiments with random choice of x have demonstrated that, on the average, the last bit accuracy is 99.8–99.9^{+} percent. The detailed results are presented in Appendix *A*. Obviously, the result never deviates from the correct result by more than 1 unit in the last place (*Ulp*).

Last Bit Accuracy. An inaccurate result may be expected only if the computed value of b from (3) has one of the following two configurations for the last 9 bits: 100000000 or 011111111. Thus it is possible to know for sure whether r has the last bit correctly rounded or the last bit is uncertain. The proportion of uncertain cases is 2^{-8} with half of them resulting in an incorrect last bit. We may thus consider the following procedure. If the last

9 bits of b do not have a "dangerous" configuration, then the computation of r gives the correctly rounded last bit. If b has a "dangerous" configuration, then the result obtained from (3) is uncertain and we apply an independent routine which makes computations with much higher precision (say, in 137 bits). The same analysis will show whether the result from this procedure is certain or uncertain. Based on our probability estimates we conjecture that this procedure will assure last bit accuracy of the computed function for all argument values. This follows from the following facts:

(1) Working with 137-bit accuracy we are not able to determine the correct double precision rounding point only when the configuration of the 85 bits in the places 53–137 of the mantissa of $f(x)$ is 01111... or 1000... (denoted as the dangerous configurations).

(2) The set of the fractional parts $[f(x) \times 2^{53}]$ where x varies over all the double precision numbers, is approximately uniformly distributed in the unit interval. (See [5, p. 16].)

Since the number of double precision arguments does not exceed 2^{64}, the likelihood that a dangerous configuration will appear in computing $f(x)$ for any machine argument x is roughly $2^{64} \times 2^{-84} = 2^{-20}$, which is less than 1 in a million. (Incidentally, the exponent for the short precision exponent of the IBM elementary mathematical library [6, p. 17] produces last bit accuracy by calling the double precision exponent in the uncertain case. An exhaustive testing of all the short precision argument values has shown that double precision is sufficient to guarantee correct rounding for single precision. This evidence strengthens our conjecture.)

Probably the above approach will always produce correctly rounded results, while the overhead of applying extended precision procedures is very small since the probability of applying them is very small (less than 0.3 percent). However, the task of actually proving that the above method will really produce last bit accuracy for all argument values seems formidable.

Monotonicity. The elementary functions considered are monotonic, except around the extreme points of the sine and cosine. Thus it is interesting to know whether the functions computed by our algorithms preserve this property. In general, the methodology produces monotonic results. This is due to the fact that around each table point X_i the computed function is a polynomial in the argument $h = x - X_i$, where $|h|$ is small and this polynomial is always monotonic in the relevant range of h. The only place where a lack of monotonicity of at most 1 *Ulp* can arise is in the links between the adjacent intervals. Assume, for convenience, that the elementary function is monotonically increasing. Then this problem can be handled either by adding a restriction to the minimax polynomial so that it will always have a negative truncation error at the right side of the interval or by introducing a small perturbation to one of the coefficients of the minimax polynomial.

Remark. Since the IEEE standard does not deal with elementary functions, we defined our routines for "exceptional" arguments and results, that is, for infinity and *NaN's* (where *NaN* indicates not a number) in a "natural" way (*e.g.*, $exp(1000) =$ infinity, $ln(-1) = NaN$). A list of these cases is presented in Appendix C.

2. DESCRIPTION OF THE ALGORITHMS

2.1 Exponential Function

Let X_i be an "accurate table value." Thus $exp(X_i)$, written as an infinite binary fraction, contains at least κ zeros (or at least κ ones) after bit 53 of the binary fraction. We chose $\kappa = 12$ in our routine. Denote

$$X_i = i/512 - \epsilon_i, \qquad i = -177, \ldots, 177 \quad (\epsilon_i < 1/million)$$
$$f_i = exp(X_i).$$

if computed in extended precision,

$$f_i = 1.XXXX\ldots XXX000000000000000XXXX\ldots$$
$$| \longleftarrow 52 \longrightarrow || \longleftarrow \kappa = 12 \longrightarrow |$$
$$(X \text{ denoting an arbitrary binary digit})$$

Thus, although f_i contains only a 53-bit mantissa, the relative (implicit) accuracy of f_i is $2^{-(52+\kappa)} = 2^{-64}$. (The same table is also used for the hyperbolic functions. For their computation we use $1/f_i$. Thus we built the table so that $X_{-i} = -X_i$ and so that $1/f_i \cong f_{-i}$ will also be accurate.)

The exponential function $exp(X)$ is computed as follows:

Write $exp(x)$ as

$$exp(x) = 2^{x/ln(2)} = 2^n \times exp(y)$$

where $n = INT(x/ln(2))$ ($INT(x)$ is defined as the integer which is nearest to x) and $y = x - n \times ln(2)$. Note that $-1023 \le n \le 1023 < 2^{10}$, $|y| \le 0.5 \times ln(2) < 0.35$.

Denote $i = INT(y \times 512)$, $|i| \le 177$.

Let $z = y - i/512 + \epsilon_i$. Then

$$exp(x) = 2^n \times f_i \times exp(z).$$

In our computations $exp(z)$ is approximated (in the minimax sense) by a precomputed fourth-degree polynomial, with a leading coefficient 1, $1 + p(z)$, in the interval $-1/1024 \lesssim z \lesssim 1/1024$. The maximal error of this approximation is smaller than $6.5E - 19$.

The exponential function is thus calculated as

$$exp(x) = 2^n \times (f_i + f_i \times p(z)). \tag{4}$$

The computation of $y = x - n \times ln(2)$ has to be very accurate; for this purpose, $ln(2)$ is expressed as a sum of $ln(2)_0 + ln(2)_1$, where $ln(2)_0$ has 10 least significant digits in its mantissa equal to 0. This guarantees that the multiplication $n \times ln(2)_0$ will be exact for all n: $-1023 \leq n \leq 1023$. Assuming that divide is an expensive operation, we keep a constant $Iln\ 2 = 1/ln(2)_0$. Thus y is computed as follows:

$$n = INT(x \times Iln2),$$

where $INT(x)$ is defined as the integer nearest to x,

$$y = x - n \times ln(2)_0,$$

and we keep separately the correction term

$$dely = n \times ln(2)_1.$$

The term $dely$ has an order of magnitude of at most 2^{-32}. The impact of $dely$ is taken into account according to the following modification of (4):

$$\begin{aligned} exp(x) &= 2^n \times (f_i + f_i \times p(z)) \times exp(-dely) \\ &\cong 2^n \times (f_i + f_i \times (p(z) - dely - dely \times p(z))). \end{aligned} \tag{5}$$

Accuracy. Practical testing of a 300,000 argument values uniformly distributed in the interval (−170, 170) demonstrated last bit accuracy of about 99.8 percent.

2.2 Logarithmic Function

We use a table of triplets:

$$X_i, F_i, G_i, \qquad i = 0, \ldots, 191$$

where $X_i = 0.75 + i/256 + 1/512 + \epsilon_i$, $F_i = ln(X_i)$, and $G_i = 1/X_i$. Using the accurate tables method the numbers X_i were chosen in such a way that bits 49 to 62 of f_i and bits 53 to 62 of G_i are all 0.

The logarithmic function is calculated as follows:

Write $x = y \times 2^n$, where n is an integer and $0.75 \leq y < 1.5$. Then $ln(x) = ln(y) + n \times ln(2)$. To calculate $ln(y)$ we use the following expression:

$$\begin{aligned} ln(y) &= ln(X_i \times y/X_i) = ln(X_i) \\ &\quad + ln(1 + (y - X_i) \times 1/X_i) = ln(X_i) + ln(1 + z), \end{aligned} \tag{6}$$

where $z = (y - X_i) \times 1/X_i$. (Note that $-1/384 < z < 1/384$, and if y is close to 1, then $-1/512 \lesssim z \lesssim 1/512$).

In order to calculate $ln(1 + z)$ we use a sixth-degree polynomial approximation on $-1/384 \lesssim z \lesssim 1/384$ with maximal absolute error less than $3.1E - 21$. The usual Horner's form of this polynomial,

$$p = ((((z \times B6 + B5) \times z + B4) \times z + B3) \times z + B2) \times z^2 + z,$$

requires 6 multiplications and 5 additions. However, there exists a method, denoted by Knuth [9, p. 432] as "adaptation of coefficients," which saves multiplications. We used a variant of this method for calculating p by only 5 multiplications and 5 additions (see [5, Sec. 5B]).

The polynomial p is calculated as follows:

$$zz = z \times z$$
$$p = zz \times \big((B6 \times zz + A) \times (zz + B \times z + C) + D \big) + z,$$

where A, B, C, and D are precomputed constants. If x is near $1 (127/128 \le x \le 129/128)$, then we use an eighth-degree minimax polynomial approximating $ln(x)$, with relative error smaller than $2.3E - 20$, without using the table lookup:

$$p = \Big(\Big(\Big(\big(\big((z \times D6 + D5) \times z + D4 \big) \times z + D3 \big) \times z + D2 \Big) \times + D1 \Big) \times z + D0 \Big) \times z^2 + z.$$

Instead of using Horner's form, which requires 8 multiplications and 7 additions, we calculate the polynomial p using the following method of adaptation of coefficients (see [5, Sec. 5B]), which requires only 6 multiplications and 7 additions:

$$zz = z \times z$$
$$p = zz \times \Big(\big(\big((z + a) \times (zz + b) + c \big) \times \big((zz + d) \times z + e \big) + f \big) + g \Big) + z,$$

where a, b, c, d, e, f, and g are precomputed constants.

The final step is to calculate $n \times ln(2)$, where $ln(2)$ is represented as a sum of two numbers H and h ($H = 3FE6\ 2E42\ FEFA\ 3800$ and $h = 3D2E\ F357\ 93C7\ 6730$ (in hexadecimal representation)).

If $|n| > 25$, then the result is obtained (add from left to right) as follows:

$$(\dots) \times z^2 + n \times h + z + F_i + n \times H.$$

If $1 \le |n| \le 25$, then, in order to maintain the high accuracy of adding $n \times ln(2)$, we use the following scheme: $n \times ln(2)$ is presented as a sum of two terms, $H_n + h_n$. In order to maintain high accuracy we calculated a correction term COR, using the method described by Dekker [4], as follows:

$$R = F_i + H_n; \quad COR = (R - H_n) - F_i. \tag{7}$$

The result is obtained as follows:

$$(\dots) \times z^2 + h_n + z + COR + R.$$

The Algorithm for $log_{10}(X)$. We use the routine for $ln(X)$ until the last stage and then we obtain the result as a sum $U + u$, where U has 32 zeros at the "tail" and $|u| < 2^{-19} \times |U|$.

Let $log_{10}(e) = L + l$, where L is the 29 high order bits and 1 is the low part. Then the result is given as $u \times l + u \times L + U \times l + U \times L$.

Accuracy. Statistical testing of 300,000 argument values, logarithmically distributed in (e^{-170}, e^{170}) demonstrate last bit accuracy of about 99.9 percent for both ln and log_{10}.

2.3 Sine and Cosine

2.3.1 *Range Reduction*. Our trigonometric routines are based on calculation of functions in a reduced range $[0, \pi/4]$. The range reduction is subtraction of $n \times \pi/2$ from the argument x so that the reduced argument $y = x - n \times \pi/2$ lies in $[-\pi/4, \pi/4]$. Thus the problem of computing $sin(x)$ (respectively $cos(x)$) is reduced to computing $sin(y)$ (respectively, $cos(y)$) for an even n and $cos(y)$ (respectively, $sin(y)$) for an odd n. The range reduction scheme is constructed in such a way that it will have the required high accuracy in the range $(-2^{27}, 2^{27})$ (about the present validity range in the *RT-PC*). It is built in such a way that, on one hand, the range reduction will be fast for the most common argument ranges but, on the other hand, it will be very accurate all over the validity range. Thus it should never lose too many bits due to cancelation for difficult arguments which require many bits in the expansion of π. (The detailed analysis is presented in Appendix B.)

In order to find the maximal number of bits needed in the expansion of π, we wrote a program that finds the most difficult argument, from the point of view of range reduction, in the valid range. This argument was found as follows. Among the numbers $n \times \pi/2$, $n = 1, \ldots, 2^{27}$, considered as infinite binary expansions, the number with the longest pattern of consecutive 0's, or consecutive 1's, starting from the 54th bit, is denoted by Xhard. For the machine number which is obtained by rounding Xhard to the nearest, the bit cancelation in the argument reduction is maximal. Our program found that the mantissa of $X\text{hard} = 1.B951F1572EBA5$ (in hexadecimal notation).

This number is very close to $294600672 \times \pi/2$, which contains 30 zeros after bit 53 in its infinite binary expansion. The number of bits canceled in the range reduction of Xhard is $53 + 30 = 83$. Thus, in order to have last bit accuracy in its range reduction, we need at least $83 + 54$ bits of π (and about 9 bits more in order to maintain 99.8-percent last bit accuracy). However, the arguments for which we need this extra accuracy in the range reduction are very rare and the range reduction for most arguments is fast. Thus the extra work carried out for the difficult arguments does not degrade the average performance.

Our range reduction was constructed so that it will maintain last bit accuracy for Xhard and, obviously, for all the other argument values in the valid range.

We distinguish the following argument ranges:

If $|x| \leq \pi/4$, no range reduction is required.

If $\pi/4 < |x| \leq \pi$, then an extra fast range reduction is performed.

For $|x| > \pi$, x is reduced by $k \times \pi/2$, where $k = INT(x \times 2/\pi)$ (using 103 bits in the expansion of π) in order to get an argument in the range $[0, \pi/4]$. (If the result is very small, then extra bits of π are taken into account).

If $|x| > \pi/4$, then the result of the range reduction is an argument u, $0 \le u \lesssim \pi/4$ plus a small correction term du.

2.3.2 *Calculating* $sin(y)$, $0 \le y \lesssim \pi/4$. (1) For $y < 41.5/256$, calculate $u2 = u^2$ and then use a ninth-degree odd minimax polynomial (plus a correction term): result $= (((C_9 \times u2 + C_7) \times u2 + C_5) \times u2 + C_3) \times u2 \times u + du + u$.

(The relative error of this approximation $< 7.8E - 19$).

(2) For $y \ge 41.5/256$, use the following formula:

$$sin(y) = sin(X_i + z) = sin(X_i) \times cos(z) + cos(X_i) \times sin(z).$$

where

$$X_i = i/256 + \epsilon_i, \qquad i = 16, \dots, 201. \tag{8}$$

$F_i = sin(X_i)$; $G_i = cos(X_i)$, where both F_i *and* G_i contain 11 implicit zeros after bit 53.

We now calculate $sin(y)$ as follows:

$$i = INT(256 \times y)$$
$$z = (u - X_i) + du \qquad -1/512 \lesssim z \lesssim 1/512$$
$$z2 = z^2$$
$$GZ = G_i \times z.$$

Then

$$sin(y) = ((D_5 \times z2 + D_3) \times GZ + (D_4 \times z2 - D_2) \times F_i) \times z2 + GZ + F_i.$$

(Using polynomial approximations to $sin(z)$ and $cos(z)$ in $[-1/512, 1/512]$, $sin(z) \cong z + D_3 \times z^3 + D_5 \times z^5$ with absolute error $< 6.4E - 25$ and $cos(z) \cong 1 + D_2 \times z^2 + D_4 \times z^4$ with absolute error $< 8.2E - 21$.)

If $x < 0$ (or $x = -0$), then $sin(x) = -sin(-x)$.

Accuracy. Testing 300,000 argument values uniformly distributed in $(-\pi, \pi)$ demonstrated last bit accuracy of about 99.9 percent. A similar testing carried out by using a logarithmic distribution for large argument values demonstrated last bit accuracy exceeding 99.9 percent.

It should also be noted that according to the trigonometric routines specification for the *RT–PC*, the argument in the trigonometric subroutine (*sin*, *cos*, *tan*, and *cotan*) has to be smaller, in absolute value, than 2^{27}; otherwise, there will be a loss of accuracy in the range reduction.

2.3.3 *Calculating* $cos(y)$, $0 \le u \lesssim \pi/4$. (1) For $y < 31.5/256$, we use an eighth-degree even polynomial approximation. Let $u2 = u^2$, then

$$cos(y) = (((C_8 \times u2 + C_6) \times u2 + C_4) \times u2 - .5) \times u2 - u \times du + 1$$

with relative error $< 1.8E - 18$.

(2) For $y \ge 31.5/256$, we use the following identity:

$$cos(y) = cos(X_i + z) = cos(X_i) \times cos(z) - sin(X_i) \times sin(z)$$

where X_i, F_i, *and* G_i were defined by (8).

We now calculate $cos(y)$ as follows:

$$\begin{aligned} i &= INT(256 \times y) \\ z &= (u - X_i) + du \qquad -1/512 \lesssim z \lesssim 1/512 \\ z2 &= z^2 \\ FZ &= F_i \times z. \end{aligned}$$

Then

$$cos(y) = (-(D_5 \times z2 + D_3) \times FZ + (D_4 \times z2 - D_2) \times G_i) \times z2 - FZ + G_i$$
$$\textit{for } x < 0 \; cos(x) = cos(-x).$$

Accuracy. Testing 300,000 argument values uniformly distributed in $(-\pi, \pi)$ demonstrated last bit accuracy of about 99.9 percent. A similar testing carried out by using a logarithmic distribution for large argument values demonstrated last bit accuracy exceeding 99.9 percent.

2.4 Tangent and Cotangent

The first step is a range reduction similar to the *sin/cos* routine. Using this range reduction we obtain an argument $y = x - n \times \pi/2$ in the range $0 \le y \lesssim \pi/4$. Thus the problem of computing $tan(x)$ (respectively, $cotan(x)$) is reduced to computing $tan(y)$ (respectively, $cotan(y)$) for an even n and $cotan(y)$ (respectively, $tan(y)$) for an odd n. If $x > \pi/4$, then the reduced argument y is given as $u + du$, where du is a small correction term.

2.4.1 *Computing* $tan(y)$, $0 \le y \lesssim \pi/4$. We distinguish the two following cases.

Case 1. If $0 \le y \le 15.5/256$, then we use an eleventh-degree odd polynomial (with leading coefficient 1) which approximates $tan(y)$, in the minimax sense, in (0, 15.5/256) (with relative error $< 6E - 20$).

$$d_{11} \times y^{13} + d_{11} \times y^9 + \ldots + d_3 \times y^3 + y. \tag{9}$$

Case 2. If $15.5/256 < y \lesssim \pi/4$, then we use the following scheme:

Let $tan(X_i) = F_i$ *and* $cotan(X_i) = G_i (G_i \cong 1/F_i)$, where $X_i = i/256 + \epsilon_i$, $i = 16, \ldots, 201$, such that F_i contains 11 zeros after bit 53 and G_i contains 11 zeros after bit 53.

Let $z = y - X_i$; then

$$\begin{aligned} tan(y) &= tan(X_i + z) = (tan(X_i) + tan(z))/(1 - tan(X_i) \times tan(z)) \\ &= (F_i + tan(z))/(1 - F_i \times tan(z)) \end{aligned}$$

where $-1/512 \lesssim z \lesssim 1/512$.

It follows that

$$\begin{aligned} tan(y) &= F_i + tan(z) \times (1 + F_i^2)/(1 - F_i \times tan(z)) \\ &= F_i + tan(z) \times (G_i + F_i)/(G_i - tan(z)). \end{aligned}$$

The function $tan(z)$ is approximated, in $(-1/512, 1/512)$, by an odd fifth-degree odd polynomial with leading coefficient 1, p, in z (with absolute error $< 1.5E - 22$).

Thus, $tan(y)$ is calculated by the expression

$$F_i + p \times (G_i + F_i)/(G_i - p).$$

2.4.2 *Computing* $cotan(y)$, $0 \le y \le \pi 4$. We distinguish the two following cases.

Case 1. $0 \le y \le 15.5/256$. In this case we calculate $tan(y)$ using expression (9) as a sum of two terms $f + g$, where f has 21 trailing zeros in its mantissa and $g < f \times 2^{-31}$. Then $1/tan(y)$ is approximated by $1/(f + g)$, which is approximated using the following scheme:

Get $u0 = 1/f$ using a machine divide.

Express $u0$ as $u1 + u2$, where $u1$ is the 21 leading bits of $u0$ and $u2$ is the low part of $u0$.

Approximate $1/(f + g)$ using a Newton iteration on $u0$, that is,

$$u0 + u0 \times \left(-u0 \times g + \left((1 - u1 \times f) - u2 \times f\right)\right).$$

Case 2. $15.5/256 < y \le \pi/4$. Here $cotan(y) = (1 - F_i \times p)/(F_i + p)$, where F_i *and* p are defined in Case 2. Using the fact that $1/F_i = G_i$, the result is approximated by the following expression:

$$G_i - p \times (F_i + G_i)/(F_i + p).$$

For $x < 0$ (or $x = -0$), $arctan(x) = -arctan(-x)$ and $cotan(-x) = -cotan(x)$.

Accuracy. Testing 300,000 argument values uniformly distributed in $(-\pi/2, \pi/2)$ demonstrated last bit accuracy exceeding 99.7 percent for both *tan* and *cotan*. A similar testing carried out by using a logarithmic distribution for large argument values demonstrated last bit accuracy exceeding 99.9 percent.

2.5 Arctangent

The $arctan(x)$ algorithm is based on the following variant of the accurate tables method:

Let $p_i(x)$ be a fifth-degree minimax polynomial approximating $arctan(x)$ in the interval $[i/256 - 1/512, i/256 + 1/512]$. Then $p_i(x)$ can be represented as

$$p_i(x) = \sum_{j=0}^{j=5} c_{ij} \times (x - X_i)^j. \tag{10}$$

The polynomial p_i can be represented in many forms and its actual representation depends on the number X_i, which can be chosen arbitrarily. Obviously, the accuracy of evaluation of (10) depends on the accuracy of the presentation of the coefficients c_{ij} as machine numbers. Since the values of these coefficients depend on X_i, it is desirable to choose these numbers so that c_{ij} will be accurately presented as machine numbers. In our implementation the numbers $x - X_i$ are small. Thus c_{i0} and c_{i1} are the coefficients that have the maximal influence on the accuracy of the evaluation. Accordingly, for each interval $(i/256 - 1/512,\ i/256 + 1/512)$, we chose the number X_i by adding a small perturbation to the number $i/256$ in such a way that $p_i(X_i)$, which is equal to c_{i0}, considered as an infinite binary fraction, contains 12 zeros after bit 53. (Since we also needed an accurate presentation of $\pi/2 - c_{i0}$, we also required that bits 49–53 of its presentation all be equal to zero.) Also, we required that $p_i'(X_i)$, which is equal to c_{i1}, contain at least 7 zeros after bit 53 of its mantissa.

(In practice, the $p_i(x)$ were first obtained as $\Sigma_{j=0}^{j=5}\ d_{ij} \times x^j$, and then the "special points" X_i, which satisfy the above conditions on $p_i(X_i)$ and $p_i'(X_i)$, were found by a random sampling. All the above calculations were performed by using an extended precision arithmetic).

Our tables contain $\{X_i, c_{i0}, \ldots, c_{i5}\}$, $i = 16, \ldots, 256$. (For $i = 17, \ldots, 256$ they correspond to minimax polynomials in $[i/256 - 1/512,\ i/256 + 1/512]$, but $X_{16}, c_{16,0}, \ldots, c_{16,5}$ correspond to the interval $[1/16,\ 1/16 + 1/512]$.)

2.5.1 *Computing arctan(x).* The function of *arctan*(x) is computed as follows:

If $0 \le x \le 1/16$, then we use the following minimax polynomial (which approximates *arctan*(x) in $(0, 1/16)$ with relative error $< 6.2E - 20$):

$$d_{13} \times x^{13} + d_{11} \times x^{11} + \ldots + d_3 \times x^3 + x. \tag{11}$$

If $x > 1/16$, then the index $i = INT(256 \times x)$ and we extract X_i *and* c_{i0}, $c_{i1}, \ldots, c_{i5}$ from the table plus the number X_i. The coefficients are used to calculate expression (10) using Horner's formula with the argument $z = x - X_i$.

If $x \ge 1$, then we use the formula

$$arctan(x) = \pi/2 - arctan(1/x). \tag{12}$$

Thus, we first calculate $y = 1/x$ with extra precision as follows:

Let $y0$ = the result of $1/x$ in double precision. Then using one Newton iteration to obtain a more accurate result y;

$$y = y0 + y1,$$

where $y1$ is obtained using an extra precision multiplication, denoted by $@\times$, as follows:

$$y1 = y0 \times (1 - y0 @\times x).$$

Let $\pi/2$ be given as the sum of two double precision numbers $T + t$ where T represents the high part of $\pi/2$ and t the low part of it.

Then we proceed as before except that

$$z = (y0 - X_i) + y1.$$

$$arctan(x) = t - \sum_{j=1}^{j=5} c_{ij} \times z^j + (T - c_{i0}).$$

If $x > 16$, then we have to subtract expression (11) from $T + t$. In order to maintain high accuracy we use the following expression:

$$arctan(x) = t - \sum_{j=1}^{j=6} d_{2j+1} \times y0^{2j+1} - y1 - COR + D,$$

where $D = T - y0$ and $COR = D - T + y0$.

If $x > (hex)434D02967C31CDB5$ ($\cong 1.63E - 16$), then $arctan(x) = \pi/2$ (rounded according to the required rounding mode).

For $x < 0$ (or $x = -0$) $arctan(x) = -arctan(-x)$.

Accuracy. Testing 300,000 argument values uniformly distributed in (0, 10) demonstrated last bit accuracy exceeding 99.9 percent. A similar testing carried out by using a logarithmic distribution for large argument values demonstrated last bit accuracy exceeding 99.9 percent.

2.5.2 *Computing arctan2(y, x).* The function *arctan* 2(y, x) is defined as

$$arctan\ 2(y, x) = \begin{cases} arctan(y/x), & \text{for } x > 0 \\ \pi + arctan(y/x), & \text{for } x < 0\ and\ y > 0 \\ -\pi + arctan(x/y), & \text{for } x < 0\ and\ y < 0. \end{cases}$$

Thus, it is sufficient to describe the algorithm for $x > 0$ and $y > 0$. Since y/x is usually not an exact machine number, we need to obtain y/x in extra precision in order to maintain last bit accuracy.

We distinguish the following cases:

Case 1. If $y < x$, then we obtain y/x as a sum, $u + du$, where u is obtained by a division operation and $du = (y - x \times u)/x$.

(a) If $0 \le u \le 1/16$, then we use the following minimax polynomial (which approximates $arctan(x)$ in (0, 1/16)):

$$d_{13} \times u^{13} + d_{11} \times u^{11} + \ldots + d_3 \times u^3 + u + du. \quad (13)$$

(b) If $u > 1/16$, then the index $i = INT(256 \times u)$ corresponds to the six coefficients $c_{i0}, c_{i1}, \ldots, c_{i5}$ extracted from the table used for the *arctan* routine and are used to calculate (10) using Horner's formula with the argument $z = u - X_i + du$.

Case 2. If $y \ge x$, then we obtain x/y as a sum, $u + du$, where u is obtained by a division operation and $du = (x - y \times u)/y$.

Let $\pi/2$ be given as the sum of two double precision numbers $T + t$, where T represents the high part of $\pi/2$ and t the low part of it. Then we proceed as in Case 1, but we use the following formula for $u \geq 1/16$:

$$arctan(x/y) = t - \sum_{j=1}^{j=5} c_{ij} \times z^j + (T - c_{i0}).$$

If $u < 1/16$, then we have to subtract expression (13) from $T + t$. In order to maintain high accuracy we use the following expression:

$$arctan\,2(y, x) = t - \sum_{j=1}^{j=6} d_{2j+1} \times y0^{2j+1} - y1 - du - COR + D.$$

where $D = T - u$ and $COR = D - T + u$.

Accuracy. Testing 300,000 argument values for y uniformly distributed in (0, 10), with $x = 1$, demonstrated last bit accuracy exceeding 99.9 percent. A similar testing carried out by using a logarithmic distribution for large y values and $x = 1$ demonstrated last bit accuracy exceeding 99.9 percent.

2.6 The Hyperbolic Sine and Cosine

2.6.1 *Computing $sinh(x)$.* By its definition, the hyperbolic sine is equal to

$$sinh(x) = 0.5 \times (exp(x) - exp(-x)).$$

In order to calculate $sinh(x)$ we distinguish the following cases:

Case 1. If $|x| \leq 0.16$, we use a (minimax) ninth-degree odd polynomial approximation (relative error $< 8.1E - 19$),

$$x + a_3 \times x^3 + \ldots + a_9 \times x^9.$$

Case 2. For $|x| > 0.16$, we start with the usual algorithm for calculating the exponential function except that the upper bound is larger. (Another small difference is that in the range $0.16 < |x| < 0.65$ we use a fifth- rather than fourth-degree polynomial which approximates $exp(y)$ in $(-1/1024, 1/1024)$ with absolute error $< 6.7E - 23$).

In the following description we assume for convenience that $x > 0$.

It follows from (4) in the exponent algorithm that $exp(x) \cong 2^N \times (p(z) \times F_I + F_I)$, where $p(z)$ is a polynomial in a small argument z and F_I is an accurate table value.

Let $FP = F_I \times p(z)$; then

$$exp(-x) = (2^{-N}) \times (G + DG)$$

where $G = 1/(F_I + FP)$ truncated so that its last 12 bits are all 0 and DG is a Newton iteration increment for the inverse of $F_I + FP$ starting from the value G.

Let @ denote an extra precision operation. Then the increment DG is obtained by

$$DG = G \times (1@ - G@ \times F_I - G \times FP). \qquad (14)$$

The final result is obtained by the expression

$$sinh(x) = 2^{(N-1)} \times (FP - (2^{-2N}) \times DG - (2^{-2N}) \times G + F_I).$$

Accuracy. Practical testing of a 300,000 argument values uniformly distributed in the interval (−90, 90) demonstrated a last bit accuracy of about 99.8 percent.

2.6.2 *Computing cosh(x).* By its definition, the hyperbolic cosine is equal to

$$cosh(x) = 0.5 \times (exp(x) + exp(-x)).$$

Let $y = |x|$. In order to calculate $cosh(x)$ we distinguish the two following cases:

Case 1. If $y < 0.12$, we use a (minimax) eighth-degree even polynomial approximation (with relative error $< 4.5E - 19$),

$$1 + c_2 \times y^2 + \ldots + c_8 \times y^8.$$

Case 2. If $y \geq 0.12$, then we start with the usual procedure for calculating the function $exp(y)$, except that the upper bound is larger. Thus,

$$exp(y) \cong 2^N \times (p(z) \times F_I + F_I)$$

where $p(z)$ is a polynomial in a small argument and F_I is an accurate table value. Also

$$exp(-y) = (2^{-N}) \times (G + DG)$$

where $G = 1/(F_I + FP)$ truncated so that its last 12 bits are all 0 and DG is a Newton iteration increment for the inverse of $F_I + FP$ starting from the value G (obtained, as before, by (14)). The final result is obtained by the following expression:

$$cosh(x) = 2^{(N-1)} \times (p(z) \times F_I + (2^{-2N}) \times DG + (2^{-2N}) \times G + F_I).$$

Accuracy. Practical testing of a 300,000 argument values uniformly distributed in the interval (−90, 90) demonstrated a last bit accuracy of about 99.8 percent.

APPENDIX A. ACCURACY RESULTS

The proportion of correct results for the last bit is given in Table *I*. To demonstrate a last bit accuracy of more than 99.7 percent as indicated by error analysis, using statistical sampling, we required that the standard deviation of the sample average would be less than 0.0001. (Thus, for example, if the observed sample frequency of correctly rounded results is 99.7 percent, then the 95-percent confidence interval for the accuracy of the subroutine is [99.68 percent, 99.72 percent].) In order to achieve this standard deviation, we used a sample size of 300,000 argument values for each

Table I. Proportion of Last Bit Correct Results

Round 0	1	2	3
exp(x), where x is uniform on $(-170, 170)$; *EMAX* (in *Ulps*), 0.504			
0.99795*	0.99810	0.99804	0.99809
ln(x), where $x = exp(\xi)$, with ξ uniform on $(-170, 170)$; *EMAX* (in *Ulps*), 0.520			
0.99882	0.99871	0.99870	0.99868
$log_{10}(x)$, where $x = exp(\xi)$, with ξ uniform on $(-170, 170)$; *EMAX* (in *Ulps*), 0.514			
0.99936	0.99938	0.99938	0.99938
sin(x) (a) where x is uniform on $(-\pi, \pi)$; *EMAX* (in *Ulps*), 0.515			
0.99867	0.99860	0.99875	0.99865
(b) where $x = exp(\xi)$, with ξ uniform on $(ln(\pi \times 2^{-50}), ln(2^{27})) = (-33.51, 18)$; *EMAX* (in *Ulps*), 0.511			
0.99941	0.99934	0.99933	0.99931
cos(x) (a) where x is uniform on $(-\pi, \pi)$; *EMAX* (in *Ulps*), 0.509			
0.99881	0.99884	0.99866	0.99893
(b) where $x = exp(\xi)$, with ξ uniform on $(ln(\pi \times 2^{-50}), ln(2^{27})) = (-33.51, 18)$; *EMAX* (in *Ulps*), 0.510			
0.99937	0.99944	0.99947	0.99944
tan(x) (a) where x is uniform on $(-\pi/2, \pi/2)$; *EMAX* (in *Ulps*), 0.539			
0.99742	0.99724	0.99752	0.99746
(b) where $x = exp(\xi)$ with ξ uniform on $(ln(\pi \times 2 \times -50), ln(2^{27})) = (-33.51, 18.71)$; *EMAX* (in *Ulps*), 0.542			
0.99878	0.99895	0.99893	0.99896
cotan(x) (a) where x is uniform on $(-\pi/2, pi/2)$; *EMAX* (in *Ulps*), 0.543			
0.99751	0.99794	0.99766	0.99774
(b) where $x = exp(\xi)$ with ξ uniform on $(ln(\pi \times 2^{-50}), ln(2^{27})) = (-33.51, 18.7)$; *EMAX* (in *Ulps*), 0.541			
0.99876	0.99869	0.99869	0.99875
arctan(x) (a) where x is uniform on $(0, 10)$; *EMAX* (in *Ulps*), 0.521			
0.99924	0.99931	0.99910	0.99931
(b) where $x = exp(\xi)$ with ξ uniform on $(ln(1E-10), ln(1E20)) = (-23.02, 46.05)$; *EMAX* (in *Ulps*), 0.523			
0.99984	0.99980	0.99982	0.99980
*arctan*2(y, x) (a) where y is uniform on $(0, 10)$ and $x = 1$; *EMAX* (in *Ulps*), 0.528			
0.99923	0.99931	0.99905	0.99931

Table I—*Continued*

0	1	2	3
(b) where $y = exp(\eta)$ and $x = 1$, with η uniform on $(ln(1E-10), ln(1E20)) = (-23.02, 46.05)$; *EMAX* (in *Ulps*), 0.513			
0.99982	0 99979	0.99984	0.99979
$sinh(x)$, where x is uniform on $(-90, 90)$; *EMAX* (in *Ulps*), 0.507			
0.99803	0.99801	0.99801	0.99793
$cosh(x)$, where x is uniform on $(-90, 90)$; *EMAX* (in *Ulps*), 0.505			
0.99800	0 99801	0 99807	0.99795

* Numbers represent proportion of results in which the last bit was correct.

test run. Our results were compared to the extended precision routines of VS Fortran Rel. 3.0. (For the trigonometric functions the existing extended precision routines are not sufficiently accurate for large arguments. Thus we introduced an accurate extended precision range reduction.) The statistical distribution of the argument values depend on the function to be computed and is indicated below. The accuracy was tested for all the rounding modes:

0: round to the nearest;
1: round to zero;
2: round to plus *INF*;
3: round to minus *INF*.

We also present the maximal error observed in the tests, *EMAX*, for rounding to the nearest. (*EMAX* is the maximal distance between the exact function value and the computed value, measured in units in the last place (*Ulps*).) The theory described in the introduction indicates 0.5 *Ulp* plus a small fraction of a unit (about 1/256). In practice, we observe a larger error which originates from the fact that our implementation usually uses only double precision arithmetic. (For example, if we compute $sin(y)$ near 1/6, then the result is given by a ninth-degree odd polynomial. Calculating this polynomial, using a double precision representation for y^2 increases the maximal error from 0.504 *Ulps* to 0.515 *Ulps*). Actually, the maximal error can always be reduced to 0.504 *Ulps* by using a more accurate arithmetic in a few operations.

We also used an independent test program to confirm our results. Such a program was provided by Liu [8]. This test performed 1000 function evaluations per subregion for each elementary function. (Such a choice corresponds to 32,000 function evaluations of $sin(x)$, 32,000 $cos(x)$, 208,000 $exp(x)$, 832,000 $arctan(x)$ and 224,000 $ln(x)$). No monotonicity failure was detected in this test, but the maximal error was sometimes greater than *EMAX*. The largest error, 0.62 *Ulps*, was detected in $ln(x)$ near $x = 0.984$. This error can be reduced to 0.504 by an improved arithmetic implementation of (6) (*e.g.*, in order to avoid a cancellation error, one can use negative z, $-1/192 \leq z \leq 0$ for $y < 1$ and positive z, $0 \leq z \leq 1/192$ for $y > 1$).

APPENDIX B. ERROR ANALYSIS OF THE TRIGONOMETRIC RANGE REDUCTION

Assume that $\pi/2$ is approximated by a J bit number P,

$$P = P_1 + P_2 + \dots + P_{L-1} + P_L,$$

where $P_1, \dots, P_L$ have (at least) 28 zeros at the end. Denote the approximated reduced argument by AR:

$$AR = (\dots((x - N \times P_1) - N \times P_2) - \dots) - N \times P_L \; 0 < N \leq 2^{27}.$$

The multiplication operations are exact. Thus the error in AR originates only from the fact that only J bits were used in the representation of $\pi/2$. The size of the absolute error is thus smaller than $N \times 2^{-(J+1)}$.

The exact reduced argument is $R = X - N \times \pi/2$. We are interested in the relative error

$$RE = \mathit{absolute\ error}/R = |AR - R|/R \leq (N/R) \times 2^{-(J+1)}. \tag{15}$$

Since J is fixed, we should find the maximal value of N/R in the range $|x| \leq 2^{27}$ denoted as the worst case. We actually found the worst case in the range $|x| \leq 2^{28}$. This can be done as follows: Let $XN = N \times \pi/2$, where $\pi/2$ is represented by, say, two extended precision words and rounded to the nearest machine number. Assume that

$$XN = (2^{exp}) \times m = (2^{exp}) \times (1.U_1, U_2, \dots, U_{52}, U_{53}, \dots) \tag{16}$$

where $1 \leq m < 2$. Let $R(N) = (2^{(exp-52)}) \times min((.U_{53}, \dots),(1 - .U_{53}, \dots))$. We are looking for the maximal $N/R(N)$ for $N = 1, 2, \dots, 2^{28}$.

$$\begin{aligned} N/R(N) &= (XN/(\pi/2))/R(N) \\ &= ((2^{exp}) \times m/(\pi/2))/(2^{(exp-52)}) \\ &\quad \times min((.U_{53}, \dots),(1 - .U_{53}, \dots)). \end{aligned} \tag{17}$$

Let k be the number of consecutive $0's$ or $1's$ starting from U_{53}. Then expression (17) is smaller than

$$4/\pi \times 2^{(52+k)}. \tag{18}$$

Thus, in order to have a relative error smaller than, say, 2^{-64}, it is sufficient by (15) and (18), that the number of bits J in the representation of $\pi/2$ will satisfy

$$4/\pi \times 2^{(52+k)} \times 2^{-(J+1)} \leq 2^{-64}.$$

In order for the above inequality to hold it is sufficient that $J \geq 116 + k$. Since the maximal k over all the relevant range was found to be $k = 30$, it follows that 146 bits are sufficient to obtain the desired accuracy.

APPENDIX C. INFINITY AND *NaN* RESULTS

In all our routines an *NaN* argument always produces an *NaN* result, so we do not indicate this fact separately for each routine. We denote infinity by

INF and use the notation of Appendix A for the rounding modes:

0: round to the nearest;

1: round to zero;

2: round to plus *INF*;

3: round to minus *INF*.

C.1 Exceptional Results

C.1.1 *exp*(x). If $x = INF$, then the result is always *INF*.

If $x = -INF$, then the result is always 0.

If $x > (hex)40862E42FEFA39EF$, then the result is *INF* for rounding mode 0 and 2, and *HUGE* for rounding modes 1 and 3.

If $x < (hex) - 40874910D52D3052$, then the result is 0 for rounding modes 0, 1, and 3, and *TINY* for rounding mode 2.

C.1.2 *Log*(x) *and* $Log_{10}(x)$. If x is negative (or -0), then the result is always *NaN*. (The decision to define *log*(-0) as *NaN* follows from the fact that -0 can be obtained from a negative underflow.)

If $x = 0$, then the result is always $-INF$.

If $x = INF$, then the result is always *INF*.

C.1.3 *Trigonometric Routines* (*sin, cos, tan, cotan*). If $|x| \geq 2^{27}$, then the result is *NaN*.

The only possibility of getting an *INF* or a *HUGE* result is in the *cotan* routine in the case that the argument is either zero or in the gradual underflow range. In the case of a gradual underflow: if x is positive, then we get either *INF* (in rounding modes 0 and 2) or *HUGE* (in rounding modes 1 and 3); and if x is negative, then the corresponding results are $-INF$ (for rounding modes 0 and 3) and $-HUGE$ (for rounding modes 1 and 2).

cotan(0) is always *INF*.

cotan(-0) is always $-INF$.

C.1.4 *arctan*(x). *arctan*(*INF*) $= \pi/2$ rounded according to the appropriate rounding mode.

arctan$(-INF) = -\pi/2$ rounded according to the appropriate rounding mode.

C.1.5 *arctan* $2(y, x)$. The exceptional results are obtained according to Table *II*.

Remark. The decision to define *arctan*(0, 0) as *NaN* follows from the fact that this situation is similar to $0/0$, which is defined as *NaN* in the IEEE standard. This choice may create a problem when transforming the origin from rectangular to polar coordinates. In order to avoid this problem it is possible to define *arctan*(0, 0) as any fixed number in $[-\pi, \pi]$.

C.1.6 *sinh*. If $x = INF$, then the result is always *INF*.

If $x = -INF$, then the result is always $-INF$.

Table II. Exceptional Results for *arctan* 2(*y*, *x*)

y	x	Result
0	0	*NaN* (see remark in Appendix C.1.5)
INF or *−INF*	*INF* or *−INF*	*NaN*
0	<0	π
−0	<0	$-\pi$
0	>0	0
−0	>0	−0
>0	*−INF*	π
<0	*−INF*	$-\pi$
>0	*+INF*	0
<0	*+INF*	−0
>0	0 or −0	$\pi/2$
<0	0 or −0	$-\pi/2$
+INF	any	$\pi/2$
−INF	any	$-\pi/2$

If $x > (hex)\ 408633CE8FB9F87D$, then the result is *INF* for rounding mode 0 and 2, and *HUGE* for rounding modes 1 and 3.

If $x < (hex) - 408633CE8FB9F879$, then the result is *−INF* for rounding modes 0 and 3, and *−HUGE* for rounding modes 1 and 2.

C.1.7 cosh. If $x = INF$ or $-INF$, then the result is always *INF*.

If $x > (hex)\ 408633CE8FB9F87D$ or $x < (hex) - 408633CE8FB9F879$, then the result is *INF* for rounding modes 0 and 2, and *HUGE* for rounding modes 1 and 3.

REFERENCES

1. Agarwal, R. C., Cooley, J. W., Gustavson, F. G., Shearer, J. B., Slishman, G., and Tuckerman, B. New scalar and vector elementary functions for the *IBM* system/370. *IBM J. Res. Develop. 30,* 2 (Mar. 1986), 126-144.
2. Clarification. *IBM J. Res. Develop. 31*, 2 (Mar. 1987), p. 274.
3. Black, C. M., Burton, R. P., and Miller, T. H. The need for an industry standard of accuracy for elementary-function programs. *ACM Trans. Math. Soft. 10*, 4 (Dec. 1984), 361-366.
4. Dekker, T. J. Floating point technique for extending the available precision. *Num. Math. 18* (1971), 224-242.
5. Gal, S. Computing elementary functions: A new approach for achieving high accuracy and good performance. In *Accurate Scientific Computations. Lecture Notes in Computer Science*, no. 235. Springer-Verlag, New York, 1986, pp. 1-16.
6. *IBM* Elementary Mathematics Library. Programming *RPQ* (5799-*BTB*). In *Program Reference and Operation Manual, IBM*, 1984.
7. Institute of Electrical and Electronics Engineers, Inc. IEEE standard for binary floating point arithmetic (An American National Standard). IEEE New York, 1985.
8. Liu, Z. A. Test program for the elementary functions. Personal Communication, Department of Computer Science, University of California, Berkeley.
9. Knuth, D. E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd ed. Addison-Wesley, Reading, MA, 1980.

Received July 1987; revised October 1988; accepted December 1989