Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic

PING TAK PETER TANG
Argonne National Laboratory

Algorithms and implementation details for the exponential function in both single- and double-precision of IEEE 754 arithmetic are presented here. With a table of moderate size, the implementations need only working-precision arithmetic and are provably accurate to within 0.54 ulp as long as the final result does not underflow. When the final result suffers gradual underflow, the error is still no worse than 0.77 ulp.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*computer arithmetic, error analysis, numerical algorithms*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*

General Terms: Algorithms

Additional Key Words and Phrases: Exponential function, IEEE arithmetic, implementation

1. INTRODUCTION

Since the adoption of the IEEE Standard for Binary Floating-Point Arithmetic [7], there has been a revival of interest in implementing elementary transcendental functions that are comparable in quality to the six basic instructions +, −, *, /, $\sqrt{\ }$, and remainder. Our goal is to provide software implementors with sufficiently detailed guidelines so that much duplication of effort can be saved. As precedents for such guidelines we cite the work of Hart et al. [4] and more recently that of Cody and Waite [2]. Our guidelines, however, are the first that are tailored specifically for IEEE-754 arithmetic.

Although recent activities at the University of California at Berkeley under the direction of W. Kahan have resulted, among other things, in widely distributed elementary-function software suitable for IEEE double precision, implementors must either modify the code to exploit special features of their machines,

This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38, and in part by the Strategic Defense Initiative Organization, Office of the Secretary of Defense, under WPD B411.
Author's address: Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60430-4801.

or must produce a separate library for single precision. Consequently, implementation guidelines (as opposed to implemented code) for both single and double precision would be more desirable as far as code optimization and completeness are concerned. Moreover, we also provide the implementors with a detailed analysis of accuracy. Upon understanding the analysis, implementors can freely change our implementation to another form whose accuracy they can establish themselves.

We intend to publish implementation guidelines for the following list of functions: exp, expm1 ($e^x - 1$), log, log1p ($\log(1 + x)$), pow(x^y), sin, cos, tan, arg (the argument of the complex number $x + \iota y$), and atan. This list contains the most commonly used functions and is also complete in the sense that the remaining standard transcendental functions can be built upon them by simple formulas involving $+$, $-$, $*$, $/$, and $\sqrt{\ }$. This paper deals with the exponential function exp only.

In our experience of implementing elementary functions on various machines with IEEE arithmetic, two demands show up frequently: accuracy and accuracy without the help of extra-precise arithmetic. The first demand is easily understood; the second, however, arises not only when we have to implement functions in the highest precision available, but also when conversion between the various formats or execution in extra-precise arithmetic is expensive. Without extra-precise arithmetic, it is extremely difficult to implement functions efficiently and still achieve nearly perfect accuracy by using the more traditional methods as in [2], [4], and [5]. Thus, we adopt table-driven methods similar to those in [1], [3], and [6] but with smaller tables. As will be illustrated here, this method allows us to achieve efficiently an accuracy very close to 0.5 ulp (units of last place) without using extra-precise arithmetic.

2. GENERAL DISCUSSION

Software implementations of the exponential function usually involve three steps. First, for an integer $L \geq 1$, chosen beforehand, the input argument x is reduced to r in the interval $[-\log 2/2^{L+1}, \log 2/2^{L+1}]$ thus

$$x = (m2^L + j)\log 2/2^L + r,$$

where $j = 0, 1, 2, \ldots, 2^L - 1$. Second, $\exp(r) - 1$ is approximated by a polynomial or rational function, say, $p(r)$. Finally, $\exp(x)$ is reconstructed by the formula

$$\exp(x) = 2^m(2^{j/2^L} + 2^{j/2^L}p(r)).$$

We can classify the errors in such an algorithm into three categories:

(1) *Error in reduction.* The computed reduced argument r does not satisfy the equation

$$x = (m2^L + j)\log 2/2^L + r$$

exactly.

(2) *Error in approximation.* The approximating polynomial or rational function $p(r)$ differs from $\exp(r) - 1$.

(3) *Rounding errors.* Errors will be committed as we compute $p(r)$ and reconstruct the final result, such is the nature of finite-precision arithmetic on computers.

An accurate implementation of the exponential function depends, then, on how well the three kinds of error can be controlled. The first two kinds are very easily controlled. For the error in reduction, all we need is decent arithmetic on the machine and some knowledge of the maximum magnitude for a legal input argument x (details are provided later); both requirements are met in any environment conforming to ANSI/IEEE Std 754-1985. The error in approximation can be made arbitrarily small provided polynomials and rational functions of ever higher orders are employed.

How well, then, can we control the rounding errors? We show that, for a small price in storage, the rounding errors can be minimized so that the implementation will have an overall error below 0.54 ulp over practically the entire input domain.

3. ALGORITHM

The algorithm is as follows. Implementation details are given in the next section.

Step 1. Filter out the exceptional cases.

Step 2. Reduce the input argument X to $[-\log 2/64, \log 2/64]$. Obtain integers m and j, and working-precision floating-point numbers R_1 and R_2 such that (up to roundoff)

$$X = (32m + j)\log 2/32 + (R_1 + R_2),$$

$|R_1 + R_2| \leq \log 2/64$.

Step 3. Approximate $\exp(R_1 + R_2) - 1$ by a polynomial $p(R_1 + R_2)$, where

$$p(t) = t + a_1 t^2 + a_2 t^3 + \cdots + a_n t^{n+1}.$$

Step 4. Reconstruct $\exp(X)$ via

$$\exp(X) = 2^m(2^{j/32} + 2^{j/32} p(R_1 + R_2)).$$

4. IMPLEMENTATION NOTES

The notes correspond to the algorithm in the previous section. All computations are carried out in working precision in the order prescribed by the parentheses.

Step 1. The exceptional cases are as follows:

—When the input argument X is a NaN (not-a-number), a quite NaN should be returned. In addition, an invalid operation should be signaled whenever X is a signaling NaN.

—When X is $+\infty$, $+\infty$ should be returned without any exception. When X is $-\infty$, +0 should be returned without any exception.

—When the magnitude of X is larger than THRESHOLD_1, a +inf with an overflow signal, or a +0 with underflow and inexact signals, should be returned. When the magnitude of X is smaller than THRESHOLD_2, 1 + X should be returned. The values of THRESHOLD_1 and THRESHOLD_2 for single-precision implementation are 341 log 2 and 2^{-25}, respectively. For double precision, they are 2610 log 2 and 2^{-54}, respectively. The exact values in IEEE format are given in the Appendix. The reasons for the thresholds are as follows. THRESHOLD_1 is chosen so that no result with any significant bit of accuracy can be returned whenever the input argument's magnitude exceeds THRESHOLD_1. Although the IEEE single-precision representation accommodates $\pm[2^{-149}, 2^{128}(1 - 2^{-24})]$, a result that lies in $\pm[2^{-341}, 2^{320}(1 - 2^{-24})]$ can be returned with some (or even full) accuracy through bias adjustment. (For details, see Sections 7.3 and 7.4 of [7].) Thus THRESHOLD_1 for single precision is chosen to be 341 log 2. The value 2610 log 2 for double precision is chosen similarly. THRESHOLD_2 is chosen so that the exponential of any argument with magnitude less than it rounds correctly to 1. Thus the operation 1 + X gives the correct value and generates an inexact-operation exception whenever X is nonzero.

Step 2. To perform the argument reduction accurately, do the following:

—Calculate N as follows:

$$\begin{aligned} \mathrm{N} &:= \mathrm{INTRND}(\mathrm{X} * \mathrm{Inv\_L}) \\ \mathrm{N}_2 &:= \mathrm{N} \bmod 32 \\ \mathrm{N}_1 &:= \mathrm{N} - \mathrm{N}_2 \end{aligned}$$

Inv_L is 32/log 2 rounded to working precision. Note that $\mathrm{N}_2 \geq 0$, regardless of N's sign. The exact values of Inv_L for single and double precision are given in the Appendix. INTRND rounds a floating-point number to the nearest integer in the manner prescribed by the IEEE standard. *It is crucial that the default round-to-nearest mode, not any other rounding mode, is in effect here.*

—The reduced argument is represented in two working-precision numbers, R_1 and R_2. We compute them as follows. First, the value of log 2/32 is represented in two working-precision numbers, L_1 and L_2, such that the leading part, L_1, has a few trailing zeros and $\mathrm{L}_1 + \mathrm{L}_2$ approximates log 2/32 to a precision much higher than the working one. Their exact values are given in the Appendix. If the single-precision exponential is requested and $|\mathrm{N}| \geq 2^9$, then calculate R_1 by

$$\mathrm{R}_1 := (\mathrm{X} - \mathrm{N}_1 * \mathrm{L}_1) - \mathrm{N}_2 * \mathrm{L}_1.$$

Otherwise, calculate R_1 by

$$\mathrm{R}_1 := (\mathrm{X} - \mathrm{N} * \mathrm{L}_1).$$

R_2 is obtained by

$$\mathrm{R}_2 := -\mathrm{N} * \mathrm{L}_2.$$

—To complete this step, we decompose N into M and J thus:

$$\begin{aligned} M &:= N_1/32 \\ J &:= N_2. \end{aligned}$$

Step 3. The polynomial is computed by a standard recurrence:

$$\begin{aligned} \mathrm{R} &:= \mathrm{R}_1 + \mathrm{R}_2 \\ \mathrm{Q} &:= \mathrm{R} * \mathrm{R} * (\mathrm{A}_1 + \mathrm{R} * (\mathrm{A}_2 + \mathrm{R} * (\cdots + \mathrm{R} * \mathrm{A}_n) \cdots)) \\ \mathrm{P} &:= \mathrm{R}_1 + (\mathrm{R}_2 + \mathrm{Q}) \end{aligned}$$

The coefficients are obtained from a Remez algorithm implemented by the author under the direction of W. Kahan of the University of California at Berkeley.

Step 4. Each of the values $2^{j/32}, j = 0, 1, \ldots, 31$, is calculated beforehand and represented by two working-precision numbers S_lead(J) and S_trail(J). The sum approximates $2^{j/32}$ to roughly double the working precision. Thus, we may consider $2^{j/32}$ = S_lead(J) + S_trail(J) for all practical purposes. Furthermore, these values, as given in the Appendix, are so chosen that the six trailing bits of S_lead are zero. This representation is needed by the function expm1($e^x - 1$).

The reconstruction is as follows:

$$\mathsf{S} := \mathsf{S\_lead(J)} + \mathsf{S\_trail(J)}$$
$$\mathsf{exp} := 2^{\mathsf{M}} * (\mathsf{S\_lead(J)} + (\mathsf{S\_trail(J)} + \mathsf{S} * \mathsf{P}))$$

5. ERROR ANALYSIS

The following analysis assumes the default round-to-nearest rounding mode. Since neither the single-precision nor the double-precision implementation depends on extra-precise arithmetic, the error analyses for them are identical, except for a few obvious modifications. Consequently, we present only the analysis for single precision. The result for double precision is stated at the end of this section.

Our goal is to estimate, in terms of ulps, the final error in the implemented function. We find the following notation useful in error analysis:

—Typefaced letters, X, Y, P, Q, and so on, denote real numbers that are representable exactly in single precision.

—Angle brackets $\langle \cdots \rangle$ denote the rounded value of a real number "$\cdots$" to single precision. Thus, executing the statement

$$\mathsf{A} := \mathsf{B} * \mathsf{C}$$

in single precision gives the value

$$\mathsf{A} = \langle \mathsf{B} \cdot \mathsf{C} \rangle.$$

—Let x be a real number. We define $\xi(x)$ to be the difference between the value of x when rounded to working precision and x itself, thus

$$\xi(x) := \langle x \rangle - x.$$

So, for example, for x such that $2^k \le |x| < 2^{k+1}$,

$$|\xi(x)| \le 2^{k-24}$$

in single precision, and

$$|\xi(x)| \le 2^{k-53}$$

in double precision.

—If one uses $\langle \cdot \rangle$ and $\xi(\cdot)$, the relationship

$$\langle \mathsf{A}\ op\ \mathsf{B} \rangle = \mathsf{A}\ op\ \mathsf{B} + \xi(\mathsf{A}\ op\ \mathsf{B})$$

holds in the absence of overflow and underflow for any single-precision values A and B and for each of the four basic operations $+$, $-$, $\cdot$, and $/$.

We are now ready to carry out the error analysis. The idea is to treat each of the three categories of error independently before combining them.

5.1 Error in Reduction

Let R_1, R_2, N, N_1, N_2, M, and J be the working-precision values as obtained in step 2 of the implementation. We estimate the difference between the value

$R_1 + R_2$ and the correct reduced argument r, where

$$r = X - N \cdot \log 2/32.$$

We observe the following:

—L_1, L_2 are chosen so that

$$|L_1 + L_2 - \log 2/32| < 2^{-49} \qquad \text{and} \qquad |L_2| < 2^{-24}.$$

—Since L_1 has nine trailing zeros, the value $N \cdot L_1$ is representable exactly in single precision as long as $|N| < 2^9$. If $|N| \geq 2^9$, then both $N_1 \cdot L_1$ and $N_2 \cdot L_1$ are representable exactly. This is because N_2 has at most five significant bits and N_1 at most nine. The reasons are that $0 \leq N_2 \leq 31$ and that N_1, having five trailing zeros, has a magnitude strictly less than 2^{14}.

—As a result of cancellation, the computations yielding R_1 are all exact:

$$R_1 = X - N \cdot L_1.$$

Using these observations, we obtain the following:

$$\begin{aligned} R_1 + R_2 &= X - (N \cdot L_1 + \langle N \cdot L_2 \rangle) \\ &= X - N(L_1 + L_2) + \xi(N \cdot L_2). \end{aligned}$$

Now, because $|N| \leq 318 \cdot 32$, $|N \cdot L_2| < 2^{-10.6}$. Consequently,

$$\begin{aligned} |(R_1 + R_2) - (X - N \cdot \log 2/32)| &\leq N \cdot |L_1 + L_2 - \log 2/32| + |\xi(N \cdot L_2)| \\ &\leq 318 \cdot 32 \cdot 2^{-49} + 2^{-11} \cdot 2^{-24} \\ &\leq 2^{-34}. \end{aligned}$$

Moreover, by calculating

$$D_M := M \log 2 - \text{round-to-single}(M \log 2)$$

for $1 \leq M \leq 341$, we found that $|D_M| > 2^{-29}$. Thus, whenever e^X travels across the boundary of a binary interval, $R_1 + R_2$ and r always have the same sign. Consequently, the computed exponential and the true exponential always fall in the same binary interval. The implication is that the last rounding error (see 5.3) never exceeds 1/2 ulp.

5.2 Error in Approximation

We estimate the difference between the transcendental function $e^t - 1$ and the polynomial

$$p(t) = t + A_1 t^2 + \cdots + A_n t^{n+1}$$

for $t \in [-\log 2/64, \log 2/64]$. By locating numerically all the extreme points of $e^t - 1 - p(t)$ in the interval $[-0.010831, 0.010831]$ (slightly wider than $[-\log 2/64, \log 2/64]$), we found that

$$|e^t - 1 - p(t)| < 2^{-33.2}$$

for all $t \in [-0.010831, 0.010831]$.

5.3 Rounding Errors

Here we are concerned with the difference between the value

$$\langle 2^{M} \cdot \langle \text{S\_lead(J)} + \langle \text{S\_trail(J)} + \langle \text{S\_lead(J)} + \text{S\_trail(J)}\rangle \cdot \text{P}\rangle\rangle\rangle$$

obtained by computations with rounding errors and the corresponding value

$$(2^{M} \cdot (\text{S\_lead(J)} + (\text{S\_trail(J)} + (\text{S\_lead(J)} + \text{S\_trail(J)}) \cdot \text{P})))$$

obtained without rounding errors. As long as the final result does not underflow, multiplication by 2^{M} is exact. Thus, it suffices to consider the case when $M = 0$. In this case, the final result lies in the interval $(\frac{1}{2}, 2)$. Note also that the final result lies in $(\frac{1}{2}, 1)$ if and only if $J = 0$ and $r < 0$. Consequently,

$$\begin{aligned} 1 \text{ ulp} &= 2^{-24} && \text{for } \; J = 0 \text{ and } r < 0, \\ 1 \text{ ulp} &= 2^{-23} && \text{otherwise.} \end{aligned}$$

Moreover, the magnitude of Q is approximately $1/2(\log 2/64)^2 < 2^{-13}$. Thus the rounding errors accumulated in its calculation are practically zero. Finally, to shorten the expressions that follow, we use S_1, S_2, and S to denote S_lead(J), S_trail(J) and S_lead(J) + S_trail(J), respectively. We are now ready to begin.

There is no magic for rounding error analysis; we must give a careful account for each deviation of our computed value from the ideal one. We use E_0 to denote the ideal result. E_1 denotes the first corrupted result, E_2 the second, and so on. E_6 is the final computed result, and the rounding error is simply the difference $E_0 - E_6$.

$$\begin{aligned}
E_0 &:= S_1 + S_2 + (S_1 + S_2) \cdot (R_1 + R_2 + Q) \\
E_1 &:= S_1 + S_2 + (S_1 + S_2) \cdot (R_1 + \langle R_2 + Q\rangle) \\
E_2 &:= S_1 + S_2 + (S_1 + S_2) \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle \\
E_3 &:= S_1 + S_2 + S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle \\
E_4 &:= S_1 + S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle \\
E_5 &:= S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle \\
E_6 &:= \langle S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle\rangle
\end{aligned}$$

We also name the following values by $F_1, F_2, \ldots, F_6$ because these values arise often in what follows.

$$\begin{aligned}
F_1 &:= R_2 + Q \\
F_2 &:= R_1 + \langle R_2 + Q\rangle \\
F_3 &:= S_1 + S_2 \\
F_4 &:= S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle \\
F_5 &:= S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle \\
F_6 &:= S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle
\end{aligned}$$

Now the estimates,

$$|\text{ rounding errors }| = |E_0 - E_6| \le \sum_{i=1}^{6} |E_{i-1} - E_i|,$$

and

$$
\begin{aligned}
|E_0 - E_1| &= |\mathsf{S}_1 + \mathsf{S}_2| \cdot |(\mathsf{R}_2 + \mathsf{Q}) - \langle \mathsf{R}_2 + \mathsf{Q} \rangle| \\
&= |\mathsf{F}_3| \cdot |\xi(\mathsf{F}_1)|, \\
|E_1 - E_2| &= |\mathsf{S}_1 + \mathsf{S}_2| \cdot |(\mathsf{R}_1 + \langle \mathsf{R}_2 + \mathsf{Q} \rangle) - \langle \mathsf{R}_1 + \langle \mathsf{R}_2 + \mathsf{Q} \rangle \rangle| \\
&= |\mathsf{F}_3| \cdot |\xi(\mathsf{F}_2)|, \\
|E_2 - E_3| &= |\langle \mathsf{F}_2 \rangle| \cdot |\xi(\mathsf{F}_3)|, \\
|E_3 - E_4| &= |\xi(\mathsf{F}_4)|, \\
|E_4 - E_5| &= |\xi(\mathsf{F}_5)|, \\
|E_5 - E_6| &= |\xi(\mathsf{F}_6)|.
\end{aligned}
$$

To get an estimate of $|\xi(F_i)|$ for $i = 0, 1, \ldots, 6$, it suffices to know the rightmost binary intervals in which the various $|F_i|$s may lie. Note that each of the F_is is the computed result of some value whose range is known. Consequently, unless the largest magnitude achieved by those values lies very close to a power of 2, the rightmost binary intervals in which those values may lie are the binary intervals we seek. We tabulate the results below.

Value	Range	Conclusion drawn	
$\lvert \mathsf{R}_2 \rvert$	$[0, 2^{-10.78}]$	$\lvert \xi(F_1) \rvert \le 2^{-35}$	
$\lvert p(r) \rvert$	$[0, 2^{-6.52}]$	$\lvert \xi(F_2) \rvert \le 2^{-31}$,	$\lvert F_2 \rvert \le 2^{-6.5}$
$\lvert 2^{j/32} \rvert$	$[0, 2^{31/32}]$	$\lvert \xi(F_3) \rvert = 0$	for $j = 0$;
		$\lvert \xi(F_3) \rvert \le 2^{-24}$	otherwise.
$\lvert 2^{j/32} p(r) \rvert$	$2^{j/32}[0, 2^{-6.52}]$	$\lvert \xi(F_4) \rvert \le 2^{-31}$	for $j = 0$;
		$\lvert \xi(F_4) \rvert \le 2^{-30}$	otherwise.
$\lvert S_2 + 2^{j/32} p(r) \rvert$	$2^{j/32}[0, 2^{-6.49}]$	$\lvert \xi(F_5) \rvert \le 2^{-31}$	for $j = 0$;
		$\lvert \xi(F_5) \rvert \le 2^{-30}$	otherwise.
$\lvert 2^{j/32} e^r \rvert$	$2^{j/32}[2^{-1/64}, 2^{1/64}]$	$\lvert \xi(F_6) \rvert \le 2^{-25}$	for $j = 0$ and $r < 0$;
		$\lvert \xi(F_6) \rvert \le 2^{-24}$	otherwise.

Thus, when $j = 0$ and $r < 0$,

$$
\begin{aligned}
|\text{rounding error}| &\le 2^{-24} \cdot (2^{-11} + 2^{-7} + 0 + 2^{-7} + 2^{-7} + 2^{-1}) \\
&\le 0.5240 \cdot 2^{-24}
\end{aligned}
$$

When $j = 0$ and $r \ge 0$,

$$
\begin{aligned}
|\text{rounding error}| &\le 2^{-23} \cdot (2^{-12} + 2^{-8} + 0 + 2^{-8} + 2^{-8} + 2^{-1}) \\
&\le 0.5120 \cdot 2^{-23}
\end{aligned}
$$

When $j \ge 1$,

$$
\begin{aligned}
|\text{rounding error}| &\le 2^{-23} \cdot (2^{-11} + 2^{-7} + 2^{-6.5} + 2^{-7} + 2^{-7} + 2^{-1}) \\
&\le 0.5351 \cdot 2^{-23}
\end{aligned}
$$

5.4 Overall Error

Finally, we estimate the overall error

$$|2^{j/32}e^r - \langle S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle\rangle|$$

in terms of ulps.

$$\begin{aligned}|\text{error}| &= |2^{j/32}e^r - \langle S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle\rangle| \\ &\le 2^{j/32} \cdot |e^r - {}^{R_1+R_2}| + 2^{j/32} \cdot |e^{R_1+R_2} - 1 - p(R_1 + R_2)| \\ &\quad + |2^{j/32}(p(R_1 + R_2) + 1) - \langle S_1 + \langle S_2 + \langle S \cdot \langle R_1 + \langle R_2 + Q\rangle\rangle\rangle\rangle\rangle|\end{aligned}$$

When $j = 0$ and $r < 0$, 1 ulp $= 2^{-24}$ and

$$\begin{aligned}|\text{error}| &\le 1.01\,|r - (R_1 + R_2)| + 2^{-33.2} + 0.5240 \cdot 2^{-24} \\ &\le 2^{-24}(1.01 \cdot 2^{-10} + 2^{-9.2} + 0.524) \\ &\le 0.5267 \text{ ulp}.\end{aligned}$$

When $j = 0$ and $r \ge 0$, 1 ulp $= 2^{-23}$ and

$$\begin{aligned}|\text{error}| &\le 1.01\,|r - (R_1 + R_2)| + 2^{-33.2} + 0.5120 \cdot 2^{-23} \\ &\le 2^{-23}(1.01 \cdot 2^{-11} + 2^{-10.2} + 0.512) \\ &\le 0.5134 \text{ ulp}.\end{aligned}$$

When $j \ge 1$, 1 ulp $= 2^{-23}$ and

$$\begin{aligned}|\text{error}| &\le 1.01 \cdot 2^{31/32} \cdot |r - (R_1 + R_2)| + 2^{31/32}2^{-33.2} + 0.5351 \cdot 2^{-23} \\ &\le 2^{-23}(1.01 \cdot 2^{-11}2^{31/32} + 2^{-10.2}2^{31/32} + 0.5351) \\ &\le 0.5378 \text{ ulp}.\end{aligned}$$

Thus, as long as the final result does not underflow, the overall error is below 0.54 ulp.

Suppose now that the final result lies in the first gradual underflow binary interval $[2^{-127}, 2^{-126})$. Then, the final result has only 23 significant bits. Thus the error of 0.54 ulp with respect to 24 significant bits weighs only $\frac{1}{2}(.54)$ ulp = .27 ulp. However, the multiplication by 2^M now introduces an error that can be as large as $\frac{1}{2}$ ulp. Hence the error bound becomes

$$(0.5 + 0.54/2)\text{ulp} = 0.77 \text{ ulp}.$$

Similar reasoning gives an error bound of $(0.5 + 2^{-j}(0.54))$ulp when the result lies in $[2^{-126-j}, 2^{-125-j})$, $j = 1, 2, \ldots, 22$. Hence, even when the final result suffers gradual underflow, the total error still can be no worse than 0.77 ulp.

5.5 Error Bounds for Double Precision

We apply the same analysis for the double-precision implementation to obtain the following:

$$\begin{aligned}|R_1 + R_2 - (X - N \cdot \log 2/32)| &\le 2^{-77} \\ |e^t - 1 - p(t)| &< 2^{-63.2} \\ |\text{rounding error}| &< 0.5235 \cdot 2^{-53} \quad \text{when } J = 0 \text{ and } r < 0; \\ &< 0.5267 \cdot 2^{-52} \quad \text{otherwise.}\end{aligned}$$

Therefore, the overall error stays below 0.54 ulp when the result does not underflow and below 0.77 ulp when it does.

6. TEST RESULTS

The single-precision and double-precision exponential functions have been implemented in C and are running on a SUN 3 and a Sequent Balance, and on a VAX 8700 under the VMS system. (Note that other than the exception-handling mechanism and a slight difference in the exponent range, the error analysis is applicable to single precision and g-format double precision on the VAX.) To test the accuracy of our single-precision function, we compare it with a double-precision exponential function on the particular system in question. To test our double-precision function, we compare it against the h-format (113 significant bits) function supported by FORTRAN under the VMS system. Moreover, to test the accuracy of our function after a bias adjustment, we simply modify step 4 of the implementation from

$$\text{exp} := 2^{M} * (\text{S\_lead(J)} + (\text{S\_trail(J)} + \text{S} * \text{P})),$$

all performed in working precision, to

$$\begin{aligned}\text{exp1} &:= \text{S\_lead(J)} + (\text{S\_trail(J)} + \text{S} * \text{P})\\ \text{exp} &:= 2^{M} * \text{extended\_precision(exp1)},\end{aligned}$$

where extended_precision means double precision when the working precision is single and h-format when double. Note that only the exponent range, not the accuracy, is enhanced by this scheme. The following is the summary of the test results obtained from the SUN, the Sequent, and the VAX.

Interval	Number of random arguments (in millions)	Maximum error observed (in ulps) Single	Maximum error observed (in ulps) Double
I_0	4	[−0.500, +0.500]	[−0.500, +0.501]
I_1	4	[−0.509, +0.501]	[−0.509, +0.500]
I_2	4	[−0.523, +0.508]	[−0.523, +0.510]
I_3	2	[−0.522, +0.512]	[−0.523, +0.510]
I_4	2	[−0.520, +0.507]	[−0.523, +0.511]
I_5	2	[−0.517, +0.505]	[−0.522, +0.510]

The intervals tested were

$$\begin{aligned}
I_0 &= [-2^{-14}, 2^{-14}];\\
I_1 &= [-\log 2/64, \log 2/64];\\
I_2 &= [-2 \log 2, -\log 2/64] \cup [\log 2/64, 2 \log 2];\\
I_3 &= [-20 \log 2, -2 \log 2] \cup [2 \log 2, 20 \log 2];\\
I_4 &= \pm[120 \log 2, 127 \log 2] \quad \text{for single and}\\
& \pm[1010 \log 2, 1023 \log 2] \quad \text{for double;}\\
I_5 &= \pm[128 \log 2, 341 \log 2] \quad \text{for single and}\\
& \pm[1024 \log 2, 2610 \log 2] \quad \text{for double.}
\end{aligned}$$

We comment that the slight biases in the errors for both the single-precision and the double-precision functions are caused by rounding some particular values in the table of $2^{j/32}$ to their respective working precision.

Next, to confirm the analysis for gradual underflow, we tested our single-precision function for a few intervals in which gradual underflow occurs. We implemented our unmodified algorithm on a SUN 3 and obtained the following results.

Interval	Number of arguments tested	Maximum error observed (in ulps)	Bounds proved (in ulps)
[−127 log 2, −126 log 2)	10,000	[−0.750, +0.751]	[−0.770, +0.770]
[−128 log 2, −127 log 2)	10,000	[−0.628, +0.627]	[−0.635, +0.635]
[−129 log 2, −128 log 2)	10,000	[−0.564, +0.563]	[−0.568, +0.568]
[−130 log 2, −129 log 2)	10,000	[−0.531, +0.532]	[−0.534, +0.534]
[−131 log 2, −130 log 2)	10,000	[−0.515, +0.515]	[−0.517, +0.517]
[−132 log 2, −131 log 2)	10,000	[−0.508, +0.508]	[−0.508, +0.508]

Finally, we tested the accuracy of our functions on an interval slightly larger than [−1, 1] without the help of the extra-precision exponential function. We used a test program written by A. Liu of the University of California at Berkeley. The reported error of this program has been proved to satisfy

$$|\text{ reported error } - \text{ true error }| \leq \tfrac{1}{16} \text{ ulp}.$$

Hence, a reported error below $0.54 + \frac{1}{16}$ will be consistent with our analysis. Indeed, the result is consistent with our claims.

	Maximum error observed	
Number of arguments tested	Single precision	Double precision
256 × 10,000 × 13	[−0.53, +0.51]	[−0.53, +0.52]

APPENDIX. CONSTANTS FOR SINGLE PRECISION

The constants needed in steps 1 through 4 are provided in IEEE single-precision format using hexadecimal representation.

Constants Needed in Step 1

```
THRESHOLD_1 435C 6BBA
THRESHOLD_2 3300 0000
```

Constants Needed in Step 2

```
Inv_L 4238 AA3B
L1    3CB1 7200
L2    333F BE8E
```

Constants Needed in Step 3

The polynomial approximation $p(t)$ is given by

$$p(t) = t + A_1 t^2 + A_2 t^3,$$

where

A_1 3F00 0044
A_2 3E2A AAEC

Constants Needed in Step 4

The thirty-two pairs of constants below are S_lead(J) and S_trail(J) for J from 0 to 31.

J	S_lead(J)	S_trail(J)	J	S_lead(J)	S_trail(J)
0	3F80 0000	0000 0000	16	3FB5 04C0	36CC CFE7
1	3F82 CD80	3553 1585	17	3FB8 FB80	36BD 1D8C
2	3F85 AAC0	34D9 F312	18	3FBD 0880	368E 7D60
3	3F88 9800	35E8 092E	19	3FC1 2C40	35CC A667
4	3F8B 95C0	3471 F546	20	3FC5 6700	36A8 4554
5	3F8E A400	36E6 2D17	21	3FC9 B980	36F6 19B9
6	3F91 C3C0	361B 9D59	22	3FCE 2480	35C1 51F8
7	3F94 F4C0	36BE A3FC	23	3FD2 A800	366C 8F89
8	3F98 37C0	36C1 4637	24	3FD7 44C0	36F3 2B5A
9	3F9B 8D00	36E6 E755	25	3FDB FB80	36DE 5F6C
10	3F9E F500	36C9 8247	26	3FE0 CCC0	3677 6155
11	3FA2 7040	34C0 C312	27	3FE5 B900	355C EF90
12	3FA5 FEC0	3635 4D8B	28	3FEA C0C0	355C FBA5
13	3FA9 A140	3655 A754	29	3FEF E480	36E6 6F73
14	3FAD 5800	36FB A90B	30	3FF5 2540	36F4 5492
15	3FB1 23C0	36D6 074B	31	3FFA 8380	36CB 6DC9

CONSTANTS FOR DOUBLE PRECISION

The constants needed in steps 1 through 4 are provided in IEEE double-precision format using hexadecimal representation.

Constants Needed in Step 1

THRESHOLD_1 409C4474 E1726455
THRESHOLD_2 3C900000 00000000

Constants Needed in Step 2

Inv_L 40471547 652B82FE
L1 3F962E42 FEF00000
L2 3D8473DE 6AF278ED

Constants Needed in Step 3

The approximation polynomial $p(t)$ is given by

$$p(t) = t + A_1t^2 + A_2t^3 + A_3t^4 + A_4t^5 + A_5t^6,$$

where

A_1	3FE00000	00000000
A_2	3FC55555	55548F7C
A_3	3FA55555	55545D4E
A_4	3F811115	B7AA905E
A_5	3F56C172	8D739765

Constants Needed in Step 4

The thirty-two pairs of constants below are S_lead(J) and S_trail(J) for J from 0 to 31.

J	S_lead(J)	S_trail(J)
0	3FF00000 00000000	00000000 00000000
1	3FF059B0 D3158540	3D0A1D73 E2A475B4
2	3FF0B558 6CF98900	3CEEC531 7256E308
3	3FF11301 D0125B40	3CF0A4EB BF1AED93
4	3FF172B8 3C7D5140	3D0D6E6F BE462876
5	3FF1D487 3168B980	3D053C02 DC0144C8
6	3FF2387A 6E756200	3D0C3360 FD6D8E0B
7	3FF29E9D F51FDEC0	3D009612 E8AFAD12
8	3FF306FE 0A31B700	3CF52DE8 D5A46306
9	3FF371A7 373AA9C0	3CE54E28 AA05E8A9
10	3FF3DEA6 4C123400	3D011ADA 0911F09F
11	3FF44E08 60618900	3D068189 B7A04EF8
12	3FF4BFDA D5362A00	3D038EA1 CBD7F621
13	3FF5342B 569D4F80	3CBDF0A8 3C49D86A
14	3FF5AB07 DD485400	3D04AC64 980A8C8F
15	3FF6247E B03A5580	3CD2C7C3 E81BF4B7
16	3FF6A09E 667F3BC0	3CE92116 5F626CDD
17	3FF71F75 E8EC5F40	3D09EE91 B8797785
18	3FF7A114 73EB0180	3CDB5F54 408FDB37
19	3FF82589 994CCE00	3CF28ACF 88AFAB35
20	3FF8ACE5 422AA0C0	3CFB5BA7 C55A192D
21	3FF93737 B0CDC5C0	3D027A28 0E1F92A0
22	3FF9C491 82A3F080	3CF01C7C 46B071F3
23	3FFA5503 B23E2540	3CFC8B42 4491CAF8
24	3FFAE89F 995AD380	3D06AF43 9A68BB99
25	3FFB7F76 F2FB5E40	3CDBAA9E C206AD4F
26	3FFC199B DD855280	3CFC2220 CB12A092
27	3FFCB720 DCEF9040	3D048A81 E5E8F4A5
28	3FFD5818 DCFBA480	3CDC9768 16BAD9B8
29	3FFDFC97 337B9B40	3CFEB968 CAC39ED3
30	3FFEA4AF A2A490C0	3CF9858F 73A18F5E
31	3FFF5076 5B6E4540	3C99D3E1 2DD8A18B

ACKNOWLEDGMENTS

An early implementation of the single-precision function was done while the author was with the iWarp project at Intel Corp.; thanks for encouragement are due to George Cox and Brent Baxter. Cleve Moler helped test some of the early implementations. The author is much indebted to W. J. Cody for his constructive comments.

Special thanks are due to W. Kahan, the author's mentor, for his guidance, patience, and inspiration.

The author's work on elementary functions had been financially supported in the past by the Air Force grant AFOSR-84-0158, a grant from ELXSI Inc., and an IBM predoctoral fellowship.

REFERENCES

1. AGARWAL, R. C., ET AL. New scalar and vector elementary functions for the IBM System/370. *IBM J. Res. Dev. 30*, 2 (Mar. 1986), 126–144.
2. CODY, W., AND WAITE, W. *Software Manual for the Elementary Functions.* Prentice-Hall, Englewood Cliffs, N.J., 1980.
3. GAL, S. Computing elementary functions: A new approach for achieving high accuracy and good performance. In *Accurate Scientific Computations, Lecture Notes in Computer Science, 235,* Springer, New York, 1985.
4. HART, J., ET AL. *Computer Approximations.* Wiley, New York, 1968.
5. HOUGH, D. Elementary functions based upon IEEE arithmetic. *Mini/Micro West Conference Record,* Electronic Conventions Inc., Los Angeles, 1983.
6. IBM Elementary Math Library. *Programming RPQ P81005, Program 5799-BTB, Program Reference and Operations Manual.* SH20-2230-1, Aug. 1984.
7. *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985.* Institute of Electrical and Electronic Engineers, New York, 1985.

Received December 1987; accepted October 1988