60483100

CONTROL DATA CORPORATION

FORTRAN VERSION 5 COMMON LIBRARY MATHEMATICAL ROUTINES REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
- **NOS 1**
- **NOS/BE 1**
- **SCOPE 2**

REVISION RECORD	
REVISION	DESCRIPTION
A	Original release.
(07-17-79)	
B	To incorporate miscellaneous technical and non-technical changes.
(08-30-79)	
Publication No. 60483100	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Cover	–
Title Page	–
ii thru vi	B
vii	A
viii	A
1-1	B
1-2	A
1-3	A
2-1 thru 2-78	A
A-1 thru A-5	A
B-1	A
C-1 thru C-16	A
D-1	A
E-1	A
Index-1	A
Index-2	A
Comment Sheet	A
Mailer	–
Back Cover	–

Page	Revision

Page	Revision

PREFACE

This manual describes the mathematical routines of the FORTRAN Version 5 Common Library which is part of FORTRAN 5.

It is assumed that the reader is familiar with FORTRAN 5 and understands basic numerical techniques.

FORTRAN 5 and the math routines operate under the following operating systems:

NOS Version 1 for the CONTROL DATA® CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

NOS/BE Version 1 for the CDC® CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

SCOPE Version 2 for the CDC CYBER 170 Model 176; CYBER 70 Model 76; and 7600 Computer Systems

Other related publications are listed below.

Publication	Publication Number
FORTRAN Version 5 Reference Manual	60481300
COMPASS Version 3 Reference Manual	60492800

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

CONTENTS

FIGURES

TABLES

INTRODUCTION 1

The FORTRAN Common Library Mathematical Routines (math library) compute frequently occuring math functions, such as sine, cosine, and tangent. They are referenced by the function names described in the FORTRAN 5 Reference Manual. The math library routines can also be accessed from COMPASS programs.

In addition to computing commonly occurring math functions, the library includes routines that perform input and output operations. However, calls to these routines are compiler-generated rather than user-generated, and are not described in this manual. Standard mathematical symbols are used throughout this manual, except for multiplication, which is denoted by an asterisk.

NUMBER TYPES

The math routines perform computations on four number types: integer, single-precision floating-point, double-precision floating point, and complex floating-point. For each number type, there is a set of valid forms, each representing a point on the real number line or in the complex plane. In addition, there is a set of semivalid forms. None of these represent numbers, but give some indication of the erroneous computation that produced them. All other bit configurations in words thought to contain numbers of one of these types is called invalid.

The valid, semivalid, and invalid forms of each number type are described in table 1-1.

Two rules govern the use of these number forms in computation:

1. Unless documented otherwise, if a valid form of a number type is used in a computation, a valid form of the same type will result.

2. Unless documented otherwise, if a semivalid or invalid number is used in a computation, the result is undefined.

An exception to rule 1 is if the answer computed is greater or less than the range of values for valid numbers. Also, if a mathematically invalid operation is attempted, rule 1 does not apply.

If an invalid result is returned from a math routine, the program may continue without issuing a diagnostic message. The program may also terminate with or without a diagnostic, or continue for a short period and then terminate. Results of erroneous computations can vary from run to run.

In some cases, certain types of checking are performed. Also semivalid results can be produced by some routines to indicate an error has occurred.

ROUTINES AND CALLS

The FORTRAN math functions are predefined routines that can be called from a FORTRAN program. There are two types of calling procedures that can be used: calls by name and calls by value.

When a routine is called by name, a parameter list is formed in memory and the first-word-address of this list is stored in register A1 before the routine is invoked.

When a routine is called by value, the arguments are entered directly into registers X1 through X5 before the routine is invoked. The first word of the first argument is entered into register X1, the first word of the second argument is entered into X3, and the first word of the third argument is entered into X5. If an argument is double-precision or complex and requires two words, the second word is entered into the next register (i.e., X2 or X4). The first word of a complex argument is the real part, and the first word of a double-precision argument contains the high-order bits.

For both calls by name and calls by value, the result of the computation is returned in registers X6 and X7. One-word results are returned in X6, and the second word of a two-word result is returned in X7.

ROUTINE ERROR

Error is defined as the computed value of a function minus the true value.

A certain amount of error occurs during the computation of the math library functions, and is composed of two parts: algorithm error and machine round-off error. Algorithm error is caused by inaccuracies inherent in the mathematical process used to compute the result. It includes error in coefficients used in the algorithm.

Machine round-off error is caused by the finite nature of the computer. Because a finite number of bits can be represented in each word of memory, some precision is lost.

A curve representing the algorithm error is usually smooth with discontinuities at breaks in the range reduction technique. The error in the coefficients that are involved in range reduction can also occur. Usually, a good algorithm which uses good coefficients will not have an error greater than one-half in the last bit of the result.

Round-off error is difficult to predict or graph. A graph of round-off error is extremely discontinuous, but maximum and minimum error over small intervals can be shown.

Relative error is the error divided by the true value. The magnitude of relative error can be analyzed in two ways: by using the following formula:

relative error = (routine value - exact value)/exact value

or by figuring out how many bits the routine differs from the exact value. The latter is called bit error.

The first way is used for single-precision algorithms accurate to less than 2E-15, and round-off errors less than

TABLE 1-1. NUMBER FORMS

Number Type	Number Form	Description
Integer	Valid	The one-word right-justified one's-complement binary representation of all integers from $-2^{48} + 1$ to $2^{48} - 1$. Zero can be represented as positive zero (all zero bits), or negative zero (all one bits).
	Semivalid	None.
	Invalid	Any bit configuration in which the top 12 bits are not the same.
Single-precision floating-point	Valid	The normalized one-word forms of the internal floating-point representations. Zero can be represented as positive or negative zero.
	Semivalid	The four forms known as positive infinite, negative infinite, positive indefinite, and negative indefinite.
	Invalid	Any nonzero and nonsemivalid bit configuration where bits 47 and 59 are the same.
Double-precision floating-point	Valid	The forms of the internal double-precision floating-point representations where the first word is normalized and the second word has an exponent that is 48 smaller than the first word or zero. The signs of both words must be the same except when the lower part is zero. Zero can be represented as positive or negative zero.
	Semivalid	The forms where the first word is a single-precision semivalid form. The second word can be anything.
	Invalid	Double-precision representations which have sign disagreement between the two words, or the first word is an invalid single-precision form, and the second word contains an exponent that is 48 smaller than the first word or zero.
Complex floating-point	Valid	All two-word forms where each form is a valid single-precision number.
	Semivalid	All two-word forms where one word is a semivalid single-precision number, and the other is a valid or semivalid single-precision number.
	Invalid	All two-word forms where one word is an invalid single-precision number.

10E-15. Changing the last bit in a single-precision number produces a relative change of between 3.5E-15 for a large mantissa, and 7.1E-15 for a small but normalized mantissa.

The second method of analyzing relative error is by examining the routine's bit error. To determine how many bits off a routine is, the function is evaluated in double-precision and rounded to single-precision. Then, assuming the exponents are the same, the mantissas are subtracted and the integer difference is the bit error.

ERROR PLOTS

In the descriptions of some of the math routines described in this manual, error plots are provided. A typical plot covers a one-argument single-precision function over a range of argument values. These are plotted linearly or logarithmically with the ordinate ranging from -11E-15 to 11E-15 and represent relative error. The saw-tooth curves represent places at which relative error is -3/2, -1/2,1/2, and 3/2 the bit error. Discontinuities occur where the routine produces a result that is a power of 2. The argument values given are found empirically, so only an appropriate number of digits is printed.

Any point that is between the -1/2 and 1/2 saw tooth curves represents a case of the routine being as accurate as possible; anything between 1/2 and 3/2 is 1 bit high.

An algorithm error curve ranges through the middle of the plot. It shows the relative error of the algorithm over the given argument range. Its discontinuities are usually due to the range reduction part of the algorithm. For this curve, the algorithm error is (alg - exact)/exact where alg is a routine rewritten to use double-precision operators and single-precision coefficients. Therefore, a polynomial can't quite equal a transcendental function and pi/2 can't be represented exactly. The coordinates of the highest point are indicated next to it.

The overall error is bounded empirically by two jagged curves with arrowheads on them. The number of different arguments fed to the function is given on the plot; each corresponding point is either at the tip of one of the arrowheads or strictly between the pair of curves. It is likely, that there are points which do not lie between the two curves. However, the curves are close to true least-upper-bound and greatest-lower-bound curves.

The arguments are chosen randomly. After starting with the smallest argument, each argument is the previous argument plus RANF(0)*k, where k is a constant. On a logarithmic scale this algorithm is modified to get an even distribution on the resulting plot.

Ordinary numbers, such as rational numbers and multiples of log 2 or pi, probably are not sampled.

There are usually about 250 points, or arrowheads, on each of the bounding curves. Given arrowheads x and y, the last two on the list, point z (formed by an argument and the relative error of the routine for that value) is added to the arrowhead list if xyz forms a convex curve or the abscissa of x and z are too far apart. Otherwise, arrowhead y is deleted from the list and the test for inclusion is retried. Points going beyond 11E-15 are forced to the boundary. The largest relative error encountered is labeled with its coordinates. Various statistics are printed concerning the distribution of points. Included in these statistics is the percentage of points per segment of width 1E-15, that lie in the interval between -10E-15 and 10E-15. Points that are greater than 10E-15 are included in the segment between 10E-15 and 11E-15. Points that are less than -10E-15 are included in the segment between -10E-15 and -11E-15. Bit errors are similarly handled, with anything above 3 being put with 3. Empty segments are not listed. The "MEAN R.E." is the mean of all ordinates. The "RMS R.F." is the standard deviation of relative error:

SQRT((sum of RE^2)/number of points) - MEAN R.E.)^2

HOW TO READ A PLOT

Here are some cause and effect statements; by taking the inverse of the statement, one has a way to look at a plot and deduce what the algorithm is doing.

1. If f(x) = 2^n * (x+g(x)) where g(x) is small compared to x and rounded addition is used, then the bounding curves roughly parallel the algorithm error and are as far apart as the inner saw-tooth curves. Unrounded addition transposes the curves by 1/2 bit.

2. If f(x) = c+g(x) then the bounds are transposed by the error in c.

3. If f(x) = c*g(x) then the distance between the bounds for f(x) are usually wider than for g(x); in particular f(x) probably has bounds at least 2 bits apart.

4. If f(x) = g(x)+(h(x)+d(x)) where g, h, or d can be constant and one of the additions produces an unnormalized result, then the bound curves can be translated and/or spread farther apart than for a nearby area where the addition happens to be normalized.

5. If f(x) is broken into numerous subintervals (e.g., 16), then the algorithm error curve is dominated by discontinuous lumps in the constants used for table lookup.

ROUTINE DESCRIPTIONS 2

Each of the math routines is described in detail on the following pages. The descriptions include the purpose of each routine, possible entry points, the FORTRAN function names that reference each routine, the formulas used to compute the result, and an error analysis.

Entry points into the routines are places in the routine at which execution can begin. Some routines can evaluate more than one function, and can have separate entry points for each. Also, some routines call others in order to compute a portion of the function.

In the error analysis for some of the routines, the abbreviation ulp is used. This means unit in the last place. Also, four symbols are used throughout to represent four bit configurations. These are summarized in table 2-1.

ACOSIN.

ACOSIN. is an external function which accepts calls from FORTRAN code. It computes the inverse sine and inverse cosine functions (FORTRAN function names ASIN and ACOS). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points ASIN and ACOS, and calls by value are computed at entry points ASIN. and ACOS..

METHOD

The input range is the collection of all valid floating-point quantities in the interval (-1.,1.). Arguments outside this range initiate error processing.

Formulas used in the routine are:

$\arcsin(x) = -\arcsin(-x)$	$x \le -.5$
$\arccos(x) = pi-\arccos(-x)$	$x \le -.5$
$\arcsin(1) = pi/2$	
$\arccos(1) = 0$	
$\arcsin(x) = pi/2-\arccos(x)$	$.5 \le x \le .1$
$\arccos(x) = \arccos(1-g(x,n))/2^n$	$.5 \le x \le 1.$

where:

$g(x,0) = 1-x$	
$g(x,n+1) = 4g(x,n) - 2g(x,n)^2.$	
$\arccos(x) = pi/2 - \arcsin(x)$	$-.5 \le x \le .5$
$\arcsin(x) = x+x^3*s*((w+z-j)*w+a+m/(e-x^2))$	$-.5 \le x \le .5$

where:

$w = (x^2-c)*z+k$
$z = (x^2+r)x^2+i$

The constants used are:

r	= 3.17317007853713
e	= 1.16039462973902
m	= 50.3190559607983
c	= -2.36958885561288
i	= 8.22646797079917
j	= -35.6294815974555
k	= 37.4592309257582
a	= 349.319357025144
s	= $.746926199335419 * 10^{-3}$

The approximation of arcsin (-.5,.5) is an economized approximation obtained by varying r,e,m,...,s. The argument x is supplied to ACOS. or ASIN. in X1, and the result is returned in X6.

```
a.  If ACOS. entry, go to step g.

b.  If |x| ≥ .5, go to step h.

c.  n = 0            (Loop counter).
    q = x
    y = x^2
    u = 0            if ASIN. entry.
    u = pi/2             if ACOS. entry.

d.  z  = (y+r)*y+i
    w  = (y-c)*z+k
    p  = q+s*q*y*((w+z-j)*w+a+m/(e-y))
    p  = u-p
    X6 = p/2^n

e.  If ASIN. entry, go to step k.

f.  If x is in(-.5,1.), return.
    X6 = 2*u-(X6)
    Return.

g.  If |x| < .5, go to step c.

h.  If x = +1,-1 or x is invalid, go to step i.
    n = 0    (Loop counter).
    y = 1-|x|, and normalize y .

i.  h = 4*y-2*y^2
    n = n+1
    If 2*y ≤ 2-sqrt(3) = .267949192431, y = h and go to
    step i.
```

TABLE 2-1. SYMBOL DEFINITIONS

Symbol	Bit Configuration	Meaning
POS.INF.	37770000000000000000_8	Positive Infinite
POS.INDEF.	17770000000000000000_8	Positive Indefinite
NEG.INF.	40000000000000000000_8	Negative Infinite
NEG.INDEF.	60000000000000000000_8	Negative Indefinite

j. q = 1-h, and normalize q.
   y = q^2
   u = pi/2
   Go to step d.

k. X6 = u-(X6), and normalize X6.
   Affix sign of x to X6.
   Return.

l. If x = +1. or -1. , go to step m.
   X6 = pi/2 if x = 1.
   X6 = -pi/2 if x = -1.
   If ASIN. entry, return.
   X6 = 0 if x = 1.
   X6 = pi if x = -1.
   Return.

m. Plug ACOS. entry point with ASIN. entry point, if ASIN. entry.
   Initiate error processing.
   Return through ACOS. entry point.

ERROR ANALYSIS

The maximum absolute value of relative error of the approximation above of arcsin over (-5,.5) is $1.996*10^{-15}$. A graph of the relative error of this approximation is given in figure 2-1. Upper bounds on the absolute value of relative error due to machine error have been established in the following cases:

arcsin on (-.5,.5)	-	$9.232 * 10^{-15}$
arcos on (- 5,.5)	-	$1.673 * 10^{-14}$
arcsin on (-1.,1.)	-	$4.050 * 10^{-14}$
arcos on (-1.,1.)	-	$1.618 * 10^{-13}$

The corresponding upper bounds on the absolute value of relative error in the routine are:

arcsin on (-.5,.5)	-	$1.123 * 10^{-14}$
arcos on (-.5,.5)	-	$1.873 * 10^{-14}$
arcsin on (-1.,1.)	-	$4.250 * 10^{-14}$
arcos on (-1.,1.)	-	$1.638 * 10^{-13}$

For groups of 1000 arguments chosen randomly from given intervals, statistics on relative error were observed. These are summarized in table 2-2.

Algorithm Error

For computation of arcsin (x), where x is in the interval (-.5,.5), the error curve is given in figure 2-2. The curve shows the error between 0 and .5 only, since it is symmetrical about 0.

The curve is not balanced around the axis because the Chebyshev coefficient for x was thrown away, and 1.0 was used instead.

For computation of arcsin (x) outside of the interval (-.5,.5) and arcos (x), a range reduction is performed which produces no algorithm error. At the end of the computation, a multiple of pi is involved, so the curves are offset by an amount dependent on the error in pi. The error for arcos is shown in figure 2-3.

There are breaks in the algorithm error curve at plus and minus .5, and when one-half the square root of three is .866025, .9665926, .991445, .997859, and so on.

Total Error

For the arcsin (x), where x is in the interval (-.5,.5), the routine is: x + x^3 * (...). The total error is dominated by the final addition, so the error curve closely follows the algorithm error curve plus or minus one-half bit.

for x in the interval (.5,.866) the algorithm is:

y = 1 - x
z = (1 - 4 * y) + 2 * y^2
arcsin(x) = pi/2 + (pi/2 - (z + z^3* (...)))/2

where y is in the interval (.5,.134), and z is in (-.5,.5).

No precision is lost computing y, and little is lost computing z. Some accuracy is lost computing the final part. The big jump in the error graph is when x is in the interval (.5,.540302). This occurs when pi/2 - (z + ...) is greater than 2. This peak shows up at other places, such as in the arcsin computation when x is in the interval (.866,.878). It also occurs in the arcos computation just below each peak in the bit error curve.

The arcos computation, except near 1.0, is predominated by pi/2. In particular, for x in (-1.,.5), pi/2 is added twice, first rounded then unrounded, in order to give a near-perfect distribution. Near x=1.0, so much folding occurs that a large error is built up before evaluating the polynomial. The graph gives an indication of the infrequency of error but does not show a worst case (15E-15 relative error has been experienced).

The mean relative error for ASIN and ACOS is graphed in figures 2-2 and 2-3.

TABLE 2-2. RELATIVE ERROR OF ACOSIN.

Entry Point	Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minimum	Maximum
ACOS.	-.5	.5	-9.435E-16	1.547E-15	-5.781E-15	3.856E-15
	-1.	-.5	-4.331E-16	1.746E-15	-4.520E-15	4.546E-15
	.5	1.	-5.098E-16	1.843E-15	-7.150E-15	9.559E-15
ASIN.	-.5	.5	8.401E-16	1.666E-15	-5.328E-15	4.916E-15
	-1.	-.5	6.209E-16	3.268E-15	-7.061E-15	1.489E-14
	.5	1.	7.311E-16	3.307E-15	-7.160E-15	1.554E-14

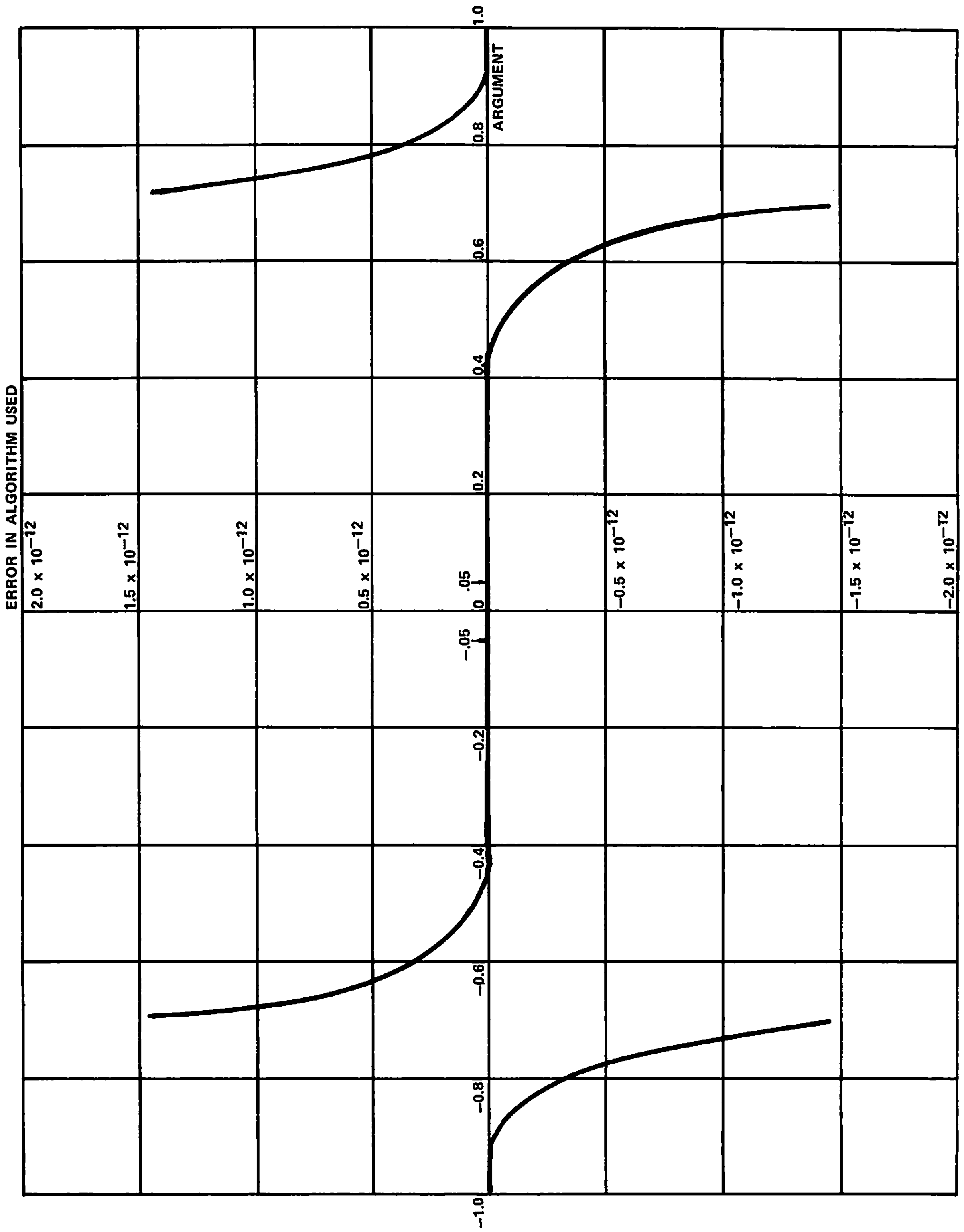

Figure 2-1. Algorithm Error of ACOSIN.

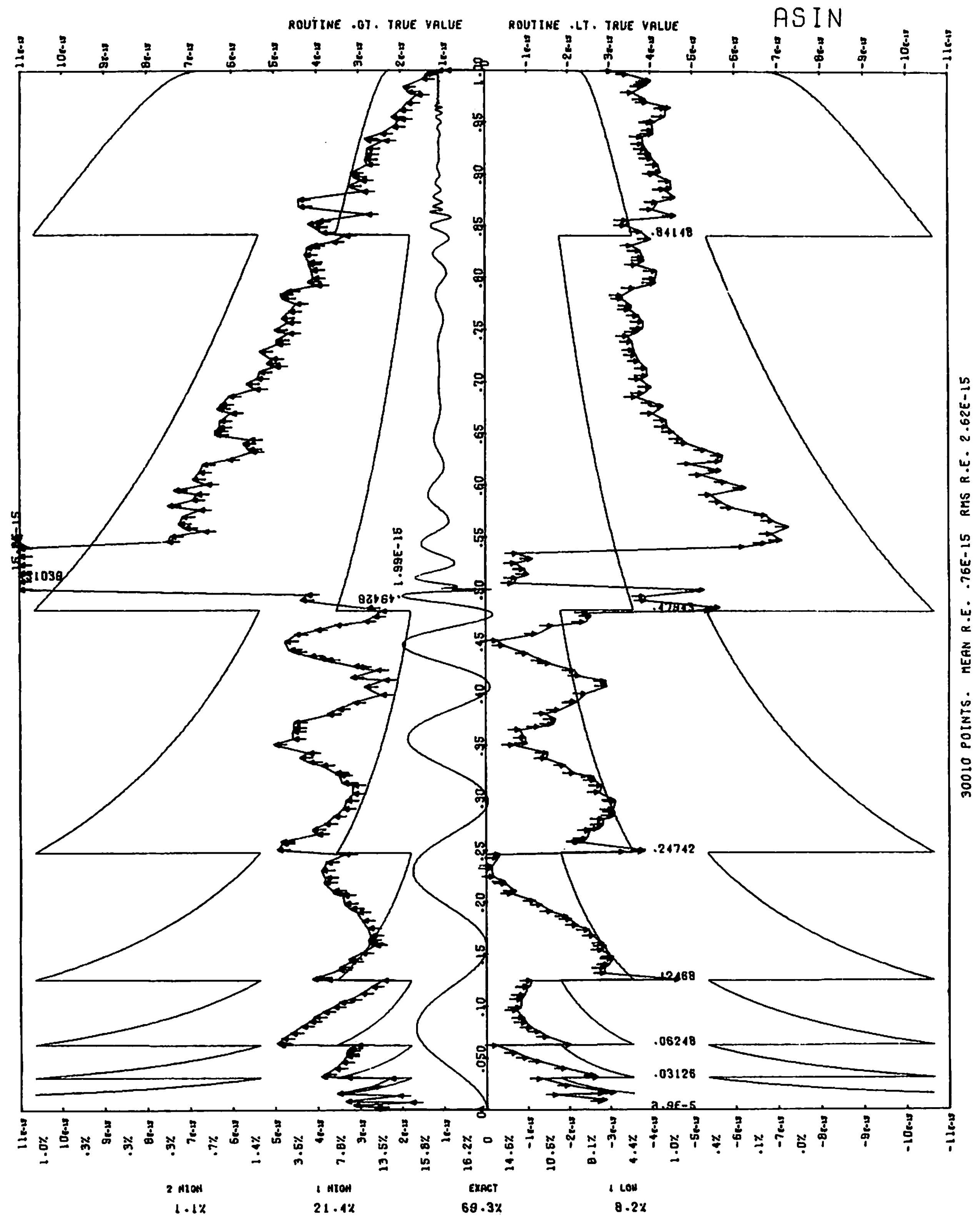

Figure 2-2. Mean Relative Error of ACOSIN. for Arcsin

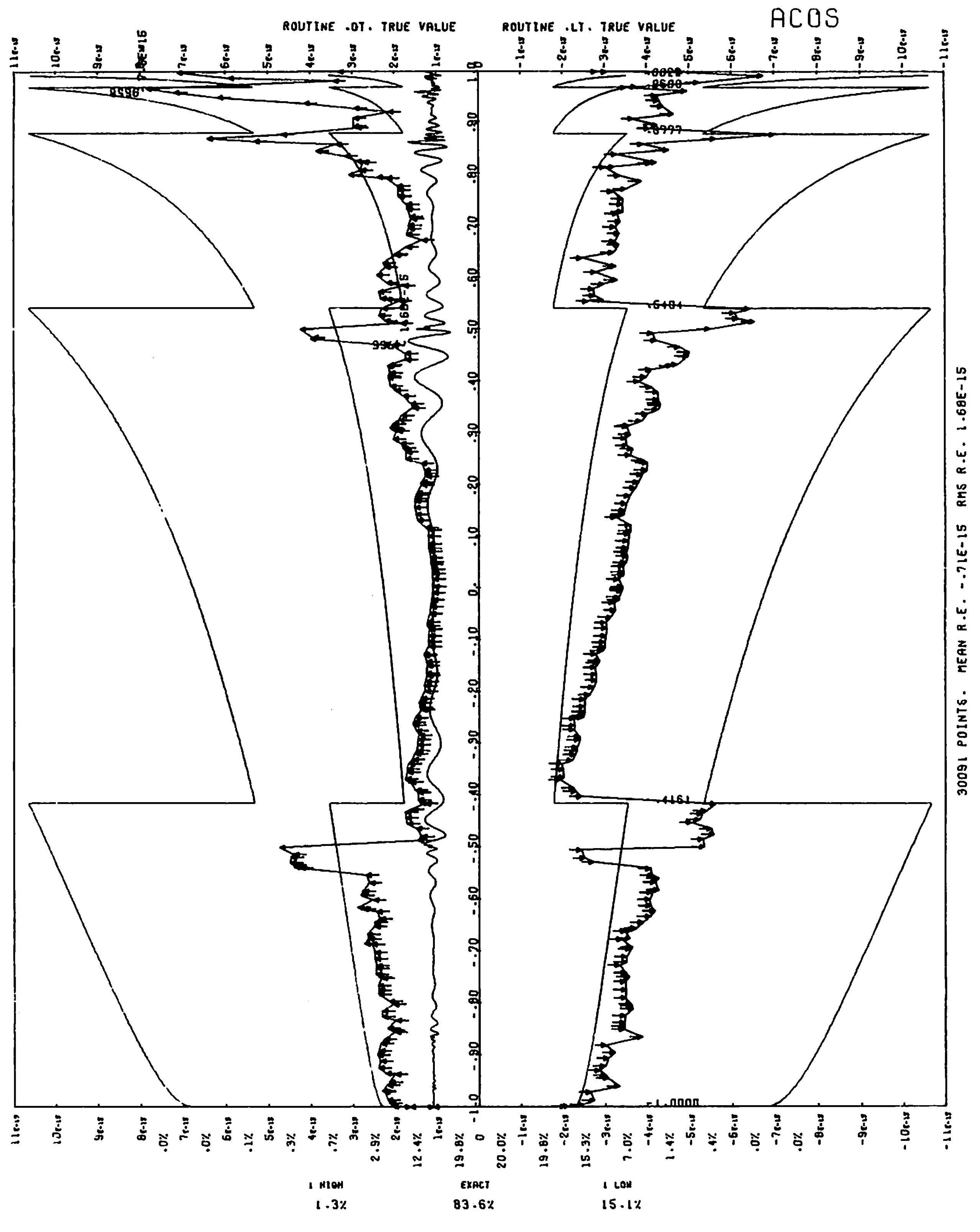

Figure 2-3. Mean Relative Error of ACOSIN. for Arccos

EFFECT OF ARGUMENT ERROR

If a small error, e', occurs in the argument x, the error in the result is given approximately by $e'/(1-x^2)^{.5}$ for ASIN and by $-e'/(1-x^2)^{.5}$ for ACOS.

ALOG

ALOG is an external function which accepts calls from FORTRAN code. It computes the natural and common logarithm functions (FORTRAN function names ALOG and ALOG10). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points ALOG and ALOG10, and calls by value are computed at entry-points ALOG. and ALOG10..

METHOD

The input range to this routine is the collection of all definite in-range non-negative nonzero floating-point quantities. Upon entry, the argument x is put in the form $x = y * 2^n$, where n is an integer, and $1. \leq y \leq 2.$ Then log x is evaluated by:

$$\log x = \log y + 3/4*n + (\log 2 - 3/4)*n$$

To compute log y, the interval (1.,2.) is divided up into the subintervals:

(1., 1.107238769531),
(1.107238769531, 1.357238769531),
(1.607238769531, 1.857238769531),
(1.357238769531, 1.607238769531), and
(1.857238769531, 2.).

Center points 1., 1.225803196513098, 1.475803239208091, 1.735100002271352, 2. are chosen within these intervals. If y is in subinterval (a, b) with center point c, log y is computed by:

$$\log y = \log c + \log((1+t)/(1-t))$$

where $t = (y - c)/(y + c)$

log ((1 + t)/(1 - t)) is then computed by:

$$\log((1+t)/(1-t)) = 2.*t+c(3)*t^3+c(5)*t^5+c(7)*t^7+c(9)*t^9.$$

The coefficients c(3), c(5), c(7) and c(9) are chosen by truncating the Taylor series for log ((1+t)/(1-t)) after the 11th term, and taking a Chebyshev economization to a 9th degree polynomial over the largest interval symmetric about the origin which is applicable. The constants are:

c(3) = .666666666666105
c(5) = .4000000018947
c(7) = .2857120487
c(9) = .22330022

If the argument x is invalid, an error message is issued through SYSAID= , and POS.INDEF. is returned.

ERROR ANALYSIS

The error analysis for ALOG is given. Bounds on machine error are the same for ALOG and ALOG10, while the graph of the algorithm error for ALOG10 can be obtained from the graph for ALOG by multiplying by log (e) 10. The maximum absolute value of the relative error in the algorithm over the interval (1., 2.) is $1.698 * 10^{-16}$, for entry points ALOG and ALOG.. The maximum absolute value error in the algorithm over the interval (1., 2.) is $1.667 * 10^{17}$. A graph of the error in the algorithm over (1., 2.) is given in figure 2-4. An upper bound has been established for the absolute value of the error in the routine due to machine error at $5.045 * 10^{-14} * u$, where u is the greatest integral power of 2. not exceeding the result. Hence an upper bound on the absolute value of the relative error in the routine is $5.062 * 10^{-14}$.

For groups of 10 000 arguments chosen randomly from given intervals at the entry points listed, statistics on relative error were observed. These are summarized in table 2-3.

Algorithm Error

Range reduction first folds arguments into (.9286194,1.857239); the unfolding involves an approximate constant involving log 2; hence, the error graph shows discrete lumps at 2^n *1.857239 in the algorithm error plot. Further range reduction into the subintervals described above involves the use of log c. The values of c were chosen so that the 48-bit representation of log c would be correct to at least 59 bits. Hence, no noticeable error is caused by reducing into the subintervals. Within each subinterval a polynomial is used; the polynomial is accurate enough to show essentially no error except near 1.107239.

Total Error

The final computation is log x = ((((a+t) +t) +p) +b) +b where:

a = log 2 - 3/4) * n ,
$p = c(3) * t^3 \ldots$, and
b = (3/4 * n + log c)/2

TABLE 2-3. RELATIVE ERROR OF ALOG

Entry Point	Interval		Mean	Standard Deviation	Minimum	Maximum
	From	To				
ALOG.	1.	2.	1.743E-16	2.286E-15	-9.040E-15	6.194E-15
	.5	2.	2.325E-16	2.279E-15	-1.058E-14	8.665E-15
	.5	1.	4.101E-17	2.488E-15	-9.450E-15	8.637E-15
	.0001	1000.	4.522E-16	2.223E-15	-5.562E-15	5.234E-15
	10^{-290}	10^{322}	1.228E-15	1.439E-15	-1.616E-15	4.001E-15
ALOG10.	1.	2.	-2.726E-15	2.723E-15	-1.447E-14	4.640E-15
	.5	2.	-2.689E-15	2.770E-15	-1.346E-14	6.506E-15
	.5	1.	-2.826E-15	2.897E-15	-1.546E-14	9.353E-15
	.0001	1000.	-1.795E-15	2.526E-15	-9.208E-15	5.058E-15
	10^{-290}	10^{322}	-2.015E-15	2.178E-15	-7.389E-15	3.453E-15

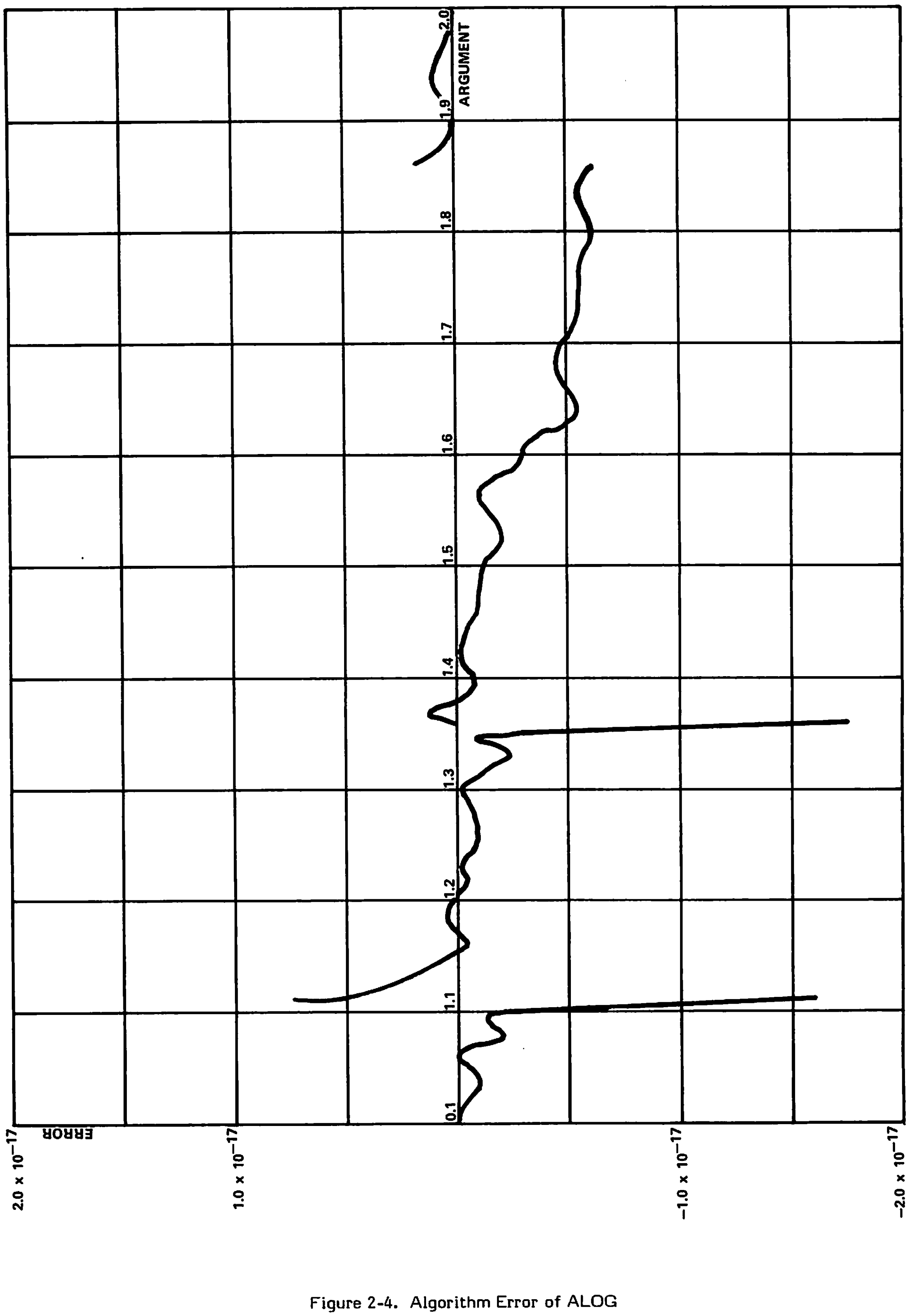

Figure 2-4. Algorithm Error of ALOG

In general $p < t < a < b$ except that a and/or b could be zero. The order was chosen in order to minimize error accumulation; b is added in twice in order to cut down on error and eliminate normalization. Because of this adding, the error graph jumps around at odd times and by fairly small amounts. (A jump probably corresponds to a, t, or one subexpression moving across a power of two.) Note the value of b is exact. When x is outside (.9286194,1.857239), a and b are nonzero and b dominates log x; hence, the error bounds are 1 bit apart. When x is in (.9286194,1.107239), log x collapses to 2t+p. But t = (y-c)/(y+c) where y-c is exact, y+c may lose half a bit, and the quotient involves further error. So those combine with the addition in 2t+p to make the total error. When x is in (1.107239,1.857239), log x = ((2t+p)+p with b = (log c)/2; t and b may be of opposite sign. Figure 2-5 and 2-6 show the mean relative error of ALOG.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the error in the result is given approximately by e'/x.

ATAN

ATAN is an external function which accepts calls from FORTRAN code. It computes the inverse tangent function (FORTRAN function name ATAN). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point ATAN, and calls by value are computed at entry point ATAN.

METHOD†

The input range to this routine is the collection of all definite in-range normalized floating-point quantities. The output range of this routine is included in the set of those floating-point quantities lying between -pi/2 and pi/2.

The argument x is then transformed into an argument y in the interval (0, 1/16) by the range reduction formulas:

arctan (u) = -arctan (-u), u negative
arctan (u) = pi/4 + (pi/4 - arctan (1/u)), $u \geq 1$
arctan (u) = arctan (k/16) + arctan ((u - k/16)/(1 + u*k/16))

where $0 \leq u \leq 1$, and k is the greatest integer not exceeding 16*u.

Finally arctan (y) (for y in (0, 1/16)) is computed by the polynomial approximation:

arctan (y) = y + a (1) *y^3 + a (2) *y^5 + a (3) *y^7 + a (4) *y^9

where:

a (1) = -.3333333333312845
a (2) = .1999999958014464
a (3) = -.1428541305087450
a (4) = .1102281616126149

The coefficients of this polynomial are those of the minimax polynomial approximation of degree 3 to the function f over (0, 1/4) where t (u^2 = (arctan (u) - u)/u^3).

ERROR ANALYSIS

A graph of the relative error of approximation of the algorithm over (0, 1/16) is shown in figure 2-7. The maximum absolute value of this relative error is 3.201 * 10^{-16}. An upper bound on the absolute value of relative error due to machine error has been established at 4.761 * 10^{-13}. Hence, an upper bound on the relative error in the routine is 4.764 * 10^{-13}.

For 1000 arguments chosen randomly from the given intervals, statistics on relative error were observed. These are given in table 2-4.

EFFECT OF ARGUMENT ERROR

If a small error e occurs in the argument, the error in the result y is given approximately by e/(1 * y^2).

ATANH.

ATANH. is an external function which accepts calls from FORTRAN code. It computes the inverse hyperbolic tangent function (FORTRAN function name ATANH). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point ATANH, and calls by value are computed at entry point ATANH..

METHOD

The input range is the collection of all definite, in-range floating-point quantities in the interval (-1.0,+1.0).

The range is reduced to (0,1) using the identity atanh (-x)=-atanh (x). From the definition atanh (x)= (e^x-e^{-x})/(e^x+e^{-x}) one gets atanh (x)=0.5*ln ((1+x)/(1-x)).

Using the property ln (a*b) = ln (a*b) = ln (a) + ln (b), the argument range of the log can be reduced to (.75,1.5) by extracting the appropriate multiple of ln (2):

TABLE 2-4. RELATIVE ERROR OF ATAN

Interval		Mean	Standard Deviation	Minimum	Maximum
From	To				
-1.	1.	-1.589E-17	2.216E-15	-6.823E-15	5.539E-15
-10.	10.	-2.348E-17	1.940E-15	-6.637E-15	7.505E-15

† Algorithm and constants Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455. Coding is by Larry Liddiard, University of Minnesota.

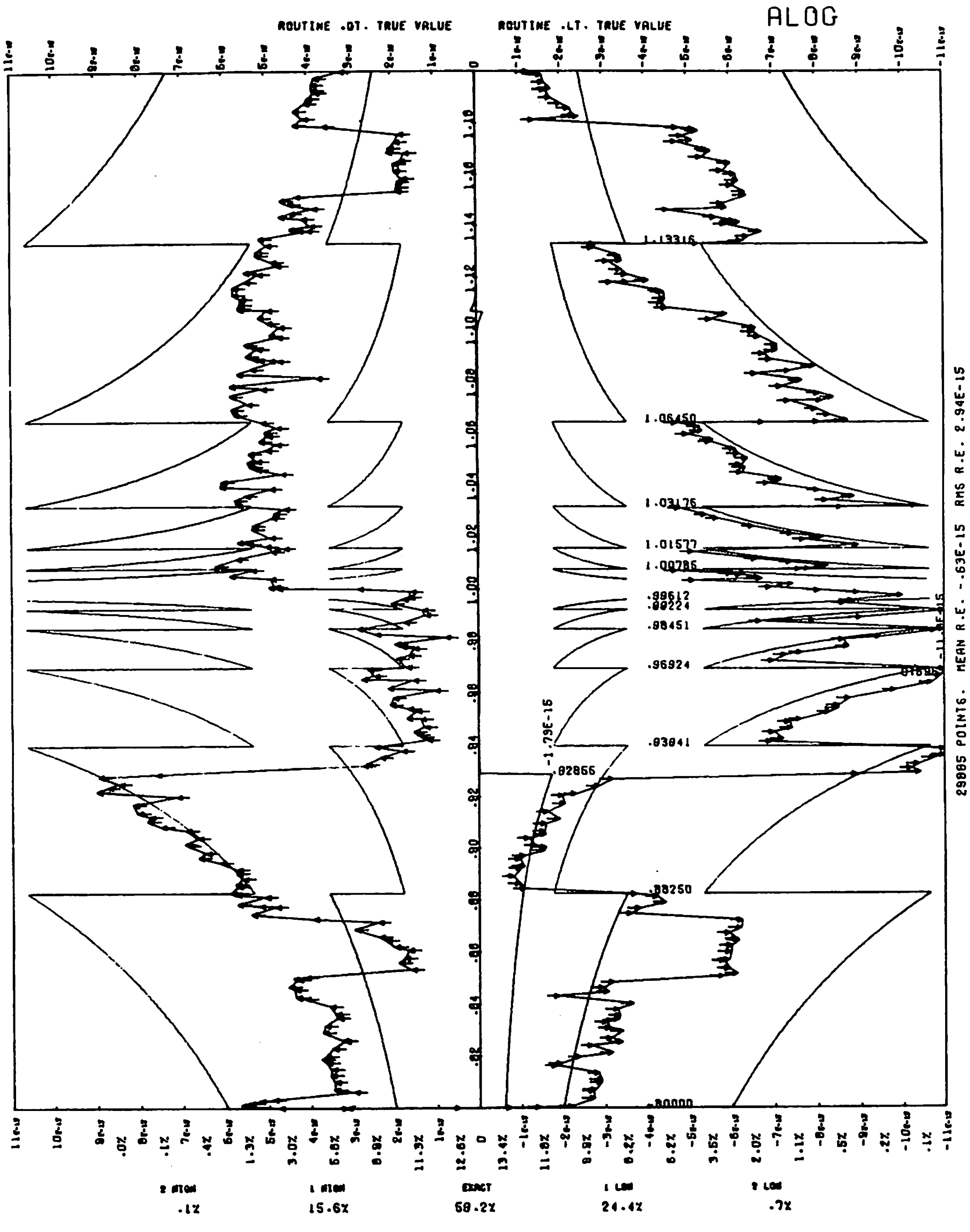

Figure 2-5. Mean Relative Error of ALOG (arithmetic scale)

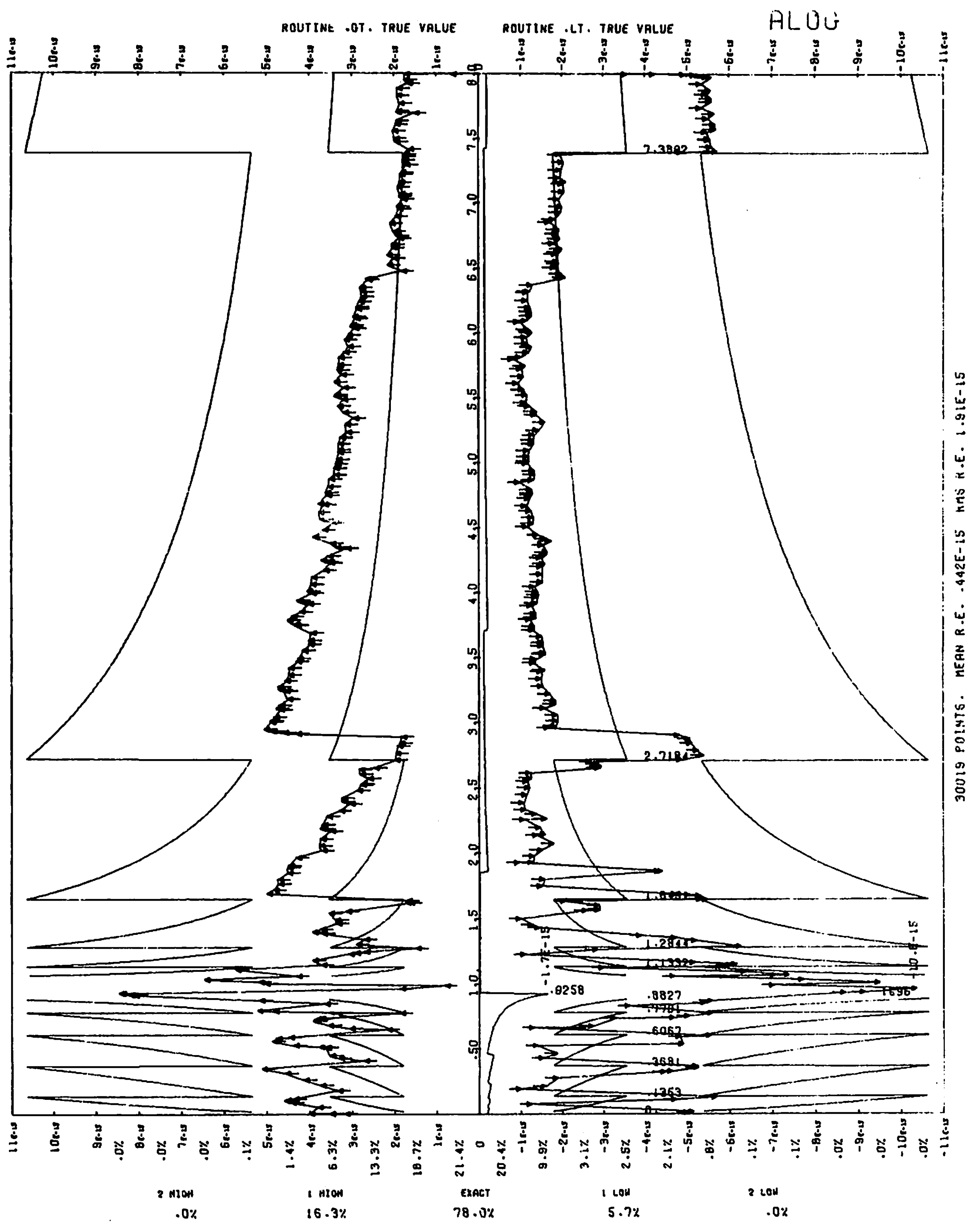

Figure 2-6. Mean Relative Error of ALOG (logarithmic scale)

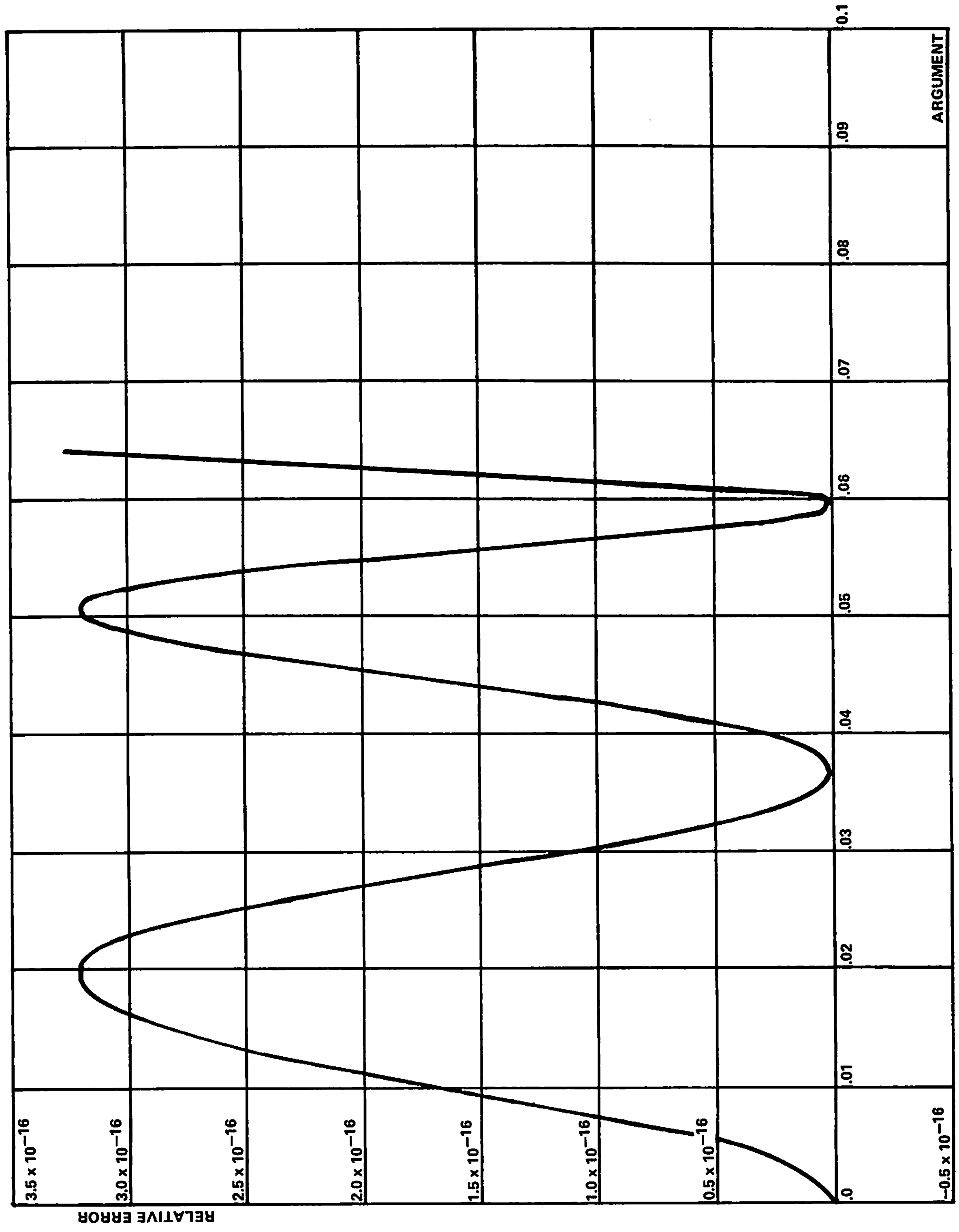

Figure 2-7. Relative Error of ATAN

$\text{atanh}(x) = 0.5 * n * \ln(2) + 0.5 * \ln(2^{-n}) * (1+x)/(1-x))$

Writing the argument of log in the form (1+y)/(1-y), and substituting atanh(y) yields:

$$\text{atanh}(x) = 0.5 * n * \ln(2) + \text{atanh}\left[\frac{2^{-n}*(1+x)-(1-x)}{2^{-n}*(1+x)+(1-x)}\right]$$

This reduces the range to (-0.2,+0.2).

The value of n such that $2^{-n}*(1+x)/(1-x)$ is in (.75,1.5) is the same as that such that $2^{-n}*(1+x)/(0.75*(1-x))$ is in (1,2). If we write 0.75*(1-x) as a*2+m, a in (1,2), then $2^{(-n-m)}*(1+x)/a$ must be in (1,2). If $(1+x) \geq a$ then -n-m=0 and n=-m. If $(1+x) < a$ then -n-m=1 and n=1-m.

The function atanh(z) on (-0.2+0.2) is approximated by $z+z^3*p/q$ where p and q are 4th order even polynomials. The coefficients of p and q were derived from the (7th order odd)/(4th order even) minimax (relative error) rational form on (-0.2,+0.2) for atanh(z).

ERROR ANALYSIS

For abs(x) < 0.2, n equals zero, the form z+...+ is used, and the error stays within the expected bound of 4.8E-15.

For abs(x) ≥ 0.5, the term n*(ln(2)/2) dominates. This term is computed as n*(ln(2)/2-.125)-n*.125-n*.125 because the rounding error in representing ln(2)/2 is large; the above form makes the rounding error relatively small. Since n*.125 is exact and the dominating form, the two additions in (other)+n*.125+n*.125 dominate the error and the expected relative error of 8.3E-15 is the maximum observed error in this region.

For 0.2 ≤ abs(x) < 0.5, n equals one and the term z=(0.5*(1+x)-(1-x))/(0.5*(1+x)+(1-x)) may be relatively large. For abs(x) < 0.25, the subtraction 1-x=0.5-x+0.5 loses two bits of the original argument. Also, z is negative in this range and some cancellation occurs in the final combination of terms, costing about one ulp. The actual upper bound in the region 0.2 < abs(x) < 0.25 is 19.4E-15, which is the overall upper bound.

The errors are summarized in table 2-5.

Figure 2-8 shows the mean relative error for ATANH..

EFFECT OF ARGUMENT ERROR

For small errors in the argument x, the amplification of absolute error is $1/(1-x^2)$ and that of relative error is $x/((1-x^2)*\text{atanh}(x))$. This increases from 1 at 0 and becomes arbitrarily large near 1.0. If x is known to more than single-precision, the following FORTRAN code may be used to get a better result near 1.0:

```
DOUBLE X

   (compute X)
SNGLX=X
SHSNGLX=X-SNGLX
Y=ATANH(SNGLX)+SHSNGLX/((1+SNGLX)*SNGL(1-X)))
```

This method is accurate to single-precision for abs(x) < 1-(1E-8) and less accurate above this point, although still better than ATANH(SNGL(X)).

TABLE 2-5. ERROR OF ATANH.

Source of Error	Error*10^{15}
Rational form	2.2
Coefficient rounding	0.1
Round-off	17.1
Upper bound	19.4
Maximum observed	12.3

ATAN2

ATAN2 is an external function which accepts calls from FORTRAN code. It computes the inverse tangent function of the ratio of two arguments (FORTRAN function name ATAN2). It accepts two floating-point arguments and returns a floating-point result.

Calls by name are computed at entry point ATAN2, and calls by value are computed at entry point ATAN2..

METHOD†

The input range to this routine is the collection of all pairs (x,y) of definite in-range normalized floating-point quantities such that (x,y) ≠ (0,0).

The function ATAN2(x,y) is defined to be the angle (lying in (-pi,pi)) subtended at the origin by the point (y,x) and the first coordinate axis.

The argument (x,y) is reduced to the first quadrant by the range reductions:

ATAN2(x,y) = -ATAN2(-x,y), x < n
ATAN2(x,y) = pi - ATAN2(x,-y), x > 0, y < 0

The argument (x,y) is then reduced to the sector:

$(u,v): u \geq 0 \ \& \ v < u \ \& \ v \geq 0$

by the range reduction:

ATAN2(x,y) = pi/2 - ATAN2(y,x), x ≥ 0 or y ≥ 0

Then ATAN2(x,y) is evaluated as arctan(y/x), using the algorithm described in the method section of the ATAN description.

ERROR ANALYSIS

See the error analysis of ATAN for properties of the algorithm used in computing arctan(y/x). 2 000 000 pairs of arguments (x,y) were randomly generated belonging to sets $((u,v): |u|, |v| \leq 10^k)$, where k = -100, -99, ..., 100. The maximum absolute value of the relative error in the routine for these arguments was observed to be $9.339 * 10^{-15}$ for these random arguments.

For 1000 arguments chosen randomly from given intervals, statistics on relative error were observed. These are summarized in table 2-6.

† Algorithm and constants Copyright 1970 by Krzyztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455. Coding is by Larry Liddiard, University of Minnesota.

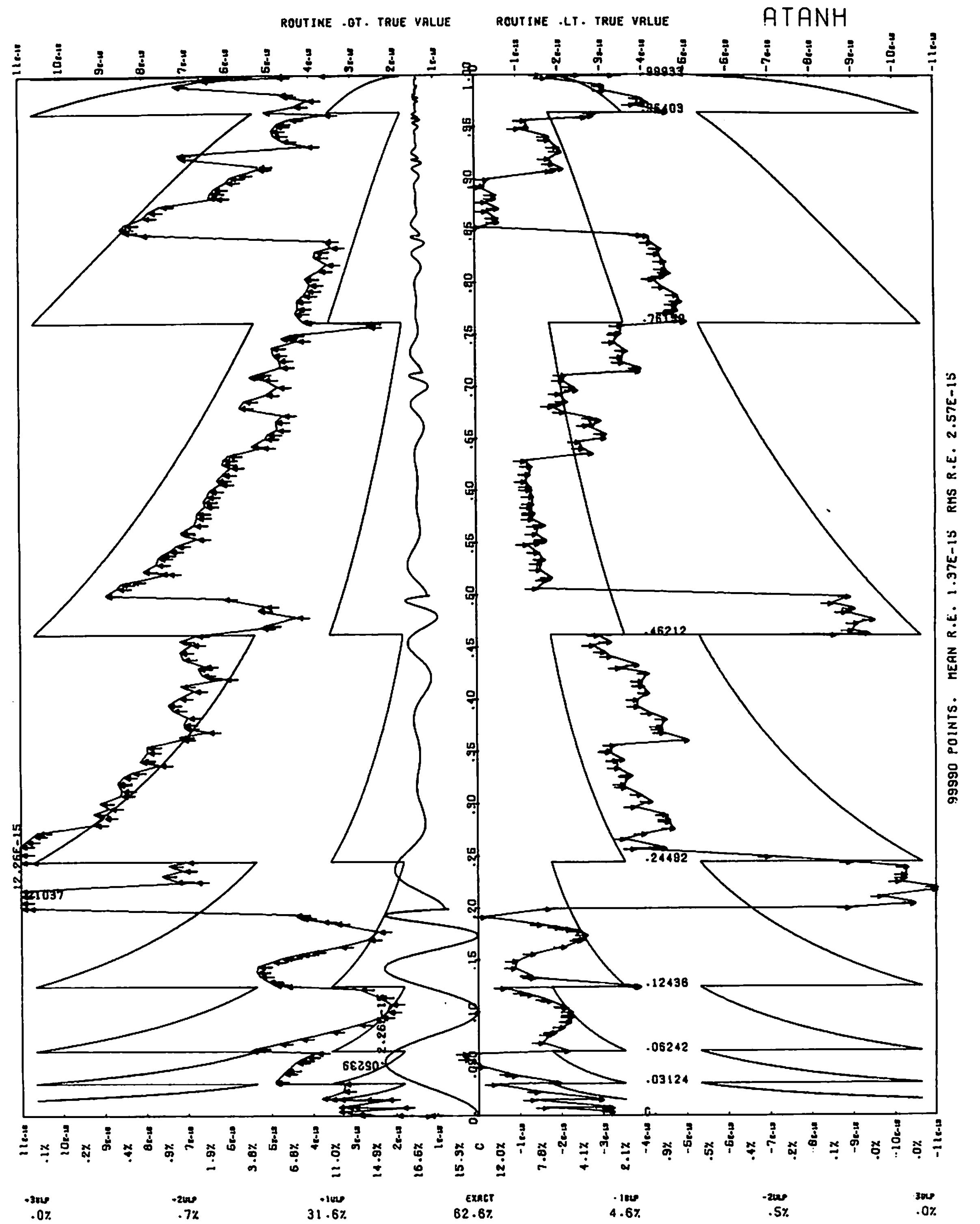

Figure 2-8. Mean Relative Error of ATANH.

TABLE 2-6. RELATIVE ERROR OF ATAN2

Interval of x		Interval of y		Mean	Standard Deviation	Minimum	Maximum
From	To	From	To				
-1.	1.	-1.	1.	-3.182E-16	2.501E-15	-1.001E-14	8.161E-15
-100.	100.	-100.	100.	-2.429E-16	2.512E-15	-1.012E-14	8.374E-15

EFFECT OF ARGUMENT ERROR

If small errors e(x) and e(y) occur in x and y, respectively, the error in the result is given approximately by $(y*e(x) - x*e(y))/(x^2 + y^2)$.

CABS.

CABS. is an external function which accepts calls from FORTRAN code. It computes the complex absolute value function (FORTRAN function name CABS). It accepts a complex argument and returns a floating-point result.

Calls by name are computed at entry point CABS, and calls by value are computed at entry point CABS..

METHOD

The input range is the collection of all valid complex quantities whose absolute value does not exceed $1.265*10^{322}$.

Let x + i*y be the argument. The algorithm used is:

a. u = max(|x|,|y|).
   v = min(|x|,|y|).

b. If u or v fails a test for infinite or indefinite, go to step f.

   If u is zero, return zero to the calling program.

c. r = u/v
   $w = 1 + r^2$
   t = (33/32 + 3/8)(w - 33/32)
   $= 3/8(r^2 + 87/32)$

   (where t is the initial linear approximation to $(1+r^2)^{.5}$)

d. Heron's rule is applied in three stages.
   t(1) = 1/2(t + w/t)
   t(2) = 1/2(t(1) + w/t(1))
   t(3) = 1/2(t(2) + w/t(2))

e. Return with u*t(3) to the calling program if it is not infinite.

f. Call routine SYS=1ST to initiate error processing.

g. Return to the calling program, unless a nonstandard or fatal error recovery has been chosen for this routine.

Note that a number of valid arguments are netted in step b, but these are returned to normal execution after further testing.

Formulas used are:

$|x + i * y| = \mathrm{SQRT}(x + i * y)$
$= \max(|x|,|y|) * (1 + r^2)^{.5}$,

where r = min(|x|,|y|)/max(|x|,|y|).

See the timing information in appendix C for further details.

ERROR ANALYSIS

The maximum absolute value of the error in approximating $t(3) = \mathrm{SQRT}(1+r^2)$ using:

$t = 33/32 + 3/8(1+r^2 - 33/32)$
$t(1) = 1/2(t + (1 + r^2)/t)$
$t(2) = 1/2(t(1) + (1 + r^2)/t(1))$
$t(3) = 1/2(t(2) + (1 + r^2)/t(2))$

is $1.5306*10^{-16}$, assumed when r=0. Hence an upper bound on the absolute value of error in the algorithm is:

$1.5306 * 10^{-16} * \max(|x|,|y|)$

where x+iy is the argument. An upper bound on the absolute value of error in the routine due to machine round-off has been established at $8.512*10^{-14} * \max(|x|,|y|)$. Therefore, an upper bound on the absolute value of error in the routine is $8.527*10^{-14} * \max(|x|,|y|)$, and an upper bound on the absolute value of relative error is $8.527*10^{-14}$.

For 10 000 arguments chosen randomly from the interval (-1.,1.)*(-1.,1.), statistics on relative error were observed. These are summarized in table 2-7.

TABLE 2-7. RELATIVE ERROR OF CABS.

Mean	Standard Deviation	Minimum	Maximum
-2.295E-15	2.658E-15	-1.093E-14	5.967E-15

EFFECT OF ARGUMENT ERROR

If a small error e(z) = e(x)+i*e(y) occurs in the argument z = x+i*y, the error in the result u is given by e(u) = (xe(x)+ye(y))/u.

CCOS

CCOS is an external function which accepts calls from FORTRAN code. It computes the complex cosine function (FORTRAN function name CCOS). It accepts a complex argument and returns a complex result.

Calls by name are computed at entry point CCOS.

METHOD

If u and v are real numbers, then:

cos(u + i * v) = cos(u) * cosh(v) - sin(u) * sinh(v) * i

The argument is checked upon entry. The argument is invalid if the real part or the imaginary part is infinite or indefinite, if the real part or the imaginary part is so large that precision will be lost during the computation, or if floating-point overflow occurs during the computation. If the argument is invalid, POS.INDEF. + i*POS.INDEF. is returned, and a diagnostic message is issued. If the argument is valid, COS.SIN is called at entry point COS.SIN for computation of the cosine and sine of the real part of the argument. HYPERB. is called at entry point HYPERB. for computation of the hyperbolic cosine and sine of the imaginary part of the argument. The result is calculated according to the formula above and is returned to the calling program.

ERROR ANALYSIS

The algorithm used in CCOS is the same as that used at entry point CCOS. in routine CSNCS.. See the description of CSNCS. for the error analysis.

EFFECT OF ARGUMENT ERROR

If a small argument error appears, then the error in the result is given approximately by multiplying the argument error by the negative of the complex sine of the argument. Hence, if a small error occurs in the complex argument and the error has absolute value e', then the absolute value of the error in the result is given approximately by e' * $(\sin(u)^2 + \sinh(v)^2)^{1/2}$, where u+i*v is the complex argument.

CEXP

CEXP is an external function which accepts calls from FORTRAN code. It computes the complex exponential function (FORTRAN function name CEXP). It accepts a complex argument and returns a complex result.

Calls by name are computed at entry point CEXP.

METHOD

If u and v are real, then:

exp (u + i * v) = exp (u) * cos (v) + i * exp (u) * sin (v)

The argument is checked upon entry. It is invalid if: the real part u or the imaginary part v is infinite or indefinite, u is greater than 741.67 in absolute value, v is so large as to lose accuracy during the calculation (i.e., v exceeds pi*2^{46} in absolute value), or floating-point overflow occurs during the calculation. If the argument is invalid, POS.INDEF. + i*POS.INDEF. is returned, and a diagnostic message is issued. If the argument is valid, the result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in CEXP is the same as that used in CEXP.. See the description of CEXP. for the error analysis.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument u + i*v, the error in the result is given approximately by e' * exp(u + i*v). Hence, the absolute value of the error in the result will be approximately e' *exp(u). If the error in the argument is significant, the error in the result should be determined by substitution of possible argument values in the function.

CEXP.

CEXP. is an external function that accepts calls from FORTRAN code. It computes the complex exponential function (FORTRAN function name CEXP). It accepts a complex argument and returns a complex result.

Calls by value are computed at entry point CEXP..

METHOD

The input range is the collection of all definite in-range complex quantities z = x + i*y where |y| does not exceed pi*2^{46} and |x| does not exceed 741.67.

The formula used for computation is:

exp (z) = exp (x+i*y) = exp (x) * cos (y) + i * exp (x) * sin (y)

where x and y are not floating-point quantities.

COS.SIN is called for computation of cos(y) and sin(y), and EXP. is called at entry point EXP. for computation of exp(x). The result is computed according to the formula and is returned to the calling program.

ERROR ANALYSIS

See the descriptions of COS.SIN and HYPERB. for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed $1.378 * 10^{-13} + 1.378 * 10^{-13}$ * exp(|x|). If the real part of the argument is large, the error in the routine will be significant.

For 10 000 arguments chosen randomly from a given interval, statistics on relative error of the components of the results were observed. These are summarized in table 2-8.

TABLE 2-8. RELATIVE ERROR OF CEXP.

Interval x		Interval y		Register	Mean	Standard Deviation	Minimum	Maximum
From	To	From	To					
-1.	1.	-1.	1.	X6	-3.440E-15	3.784E-15	-1.428E-14	1.227E-14
				X7	-5.831E-15	8.853E-15	-4.165E-14	1.242E-14
-670.	670.	-2.210E14	2.2106E14	X6	-8.962E-15	4.669E-14	-3.176E-12	2.235E-14
				X7	-1.071E-14	7.948E-14	-4.977E-12	3.723E-14

EFFECT OF ARGUMENT ERROR

If a small error e(z) occurs in the argument z, the error in the result w is given approximately by w*e(z).

CLOG

CLOG is an external function which accepts calls from FORTRAN code. It computes the complex logarithm function (FORTRAN function name CLOG). It accepts a complex argument and returns a complex result.

Calls by name are computed at entry point CLOG..

METHOD

The argument is checked upon entry. The argument is invalid if the real or complex part is infinite or indefinite, or if both the real part and the complex part are zero. If the argument is invalid, a diagnostic message is written and POS.INDEF. + i*POS.INDEF. is returned (where $i^2 = -1$). Otherwise, CLOG= is called at entry point CLOG. for computation of the complex logarithm. The result is returned to the calling program.

ERROR ANALYSIS

See the description of CLOG=.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument z, the error in the result is given approximately by e'/z. The modulus of this will give approximately the modulus of the error.

CLOG=

CLOG= is an external function which accepts calls from FORTRAN code and from CLOG. It computes the complex logarithm function (FORTRAN function name CLOG). It accepts a complex argument and returns a complex result.

Calls by value are computed at entry point CLOG.

METHOD

The input range to this routine is the collection of all definite in-range complex quantities which are nonzero, and whose absolute values do not exceed the largest floating-point number that can be represented in the machine.

The formula used to compute the complex logarithm is:

log z = log(|z|) + i * arg(z),

where |z| is the modulus of z. The absolute value of z is evaluated by routine CABS., and the logarithm is evaluated by ALOG.. The function arg(z) is evaluated by routine ATAN2.; arg(z) always lies in the interval (-pi, pi) when |z| is nonzero, definite and in-range. The result is returned to the calling program in the register pair X6-X7.

ERROR ANALYSIS

Tests on a sample of 100 000 random numbers distributed over the complex plane, with distribution being the product of two Cauchy distributions of zero mean, returned a maximum absolute value for the relative error of $8.579 * 10^{-13}$.

For 10 000 arguments chosen randomly from the interval (-1.,1.)*((-1.,1.), the components of the results gave statistics on relative error. These are summarized in table 2-9.

EFFECT OF ARGUMENT ERROR

If a small error e(z) occurs in the argument z, the error in the result is given approximately by e(z)/z.

COS.SIN

COS.SIN is an auxiliary routine which accepts calls from other math routines. It simultaneously computes the sine and cosine of an argument. It accepts a floating-point argument and returns two floating-point results. The entry point is COS.SIN.

METHOD

The argument is reduced to the interval (-pi/4, pi/4). Polynomials p(x) and q(x) of degrees 11 and 12 are used to compute sin(x) and cos(x) over that interval. First, the argument x is multiplied by 2/pi. Then, the nearest integer n to 2/pi * x is computed by adding 2/pi * x to 20000000000000000000_8 in double-precision. The upper and lower halves of the result are added using a rounded floating-point addition, and n is normalized. If the shift count in this normalization is zero (i.e., if x exceeds pi * 2^{46} in absolute value), then POS.INDEF. is returned. Otherwise, y = x * pi/2 is computed in double-precision as the reduced argument for input to p(y) and q(y). Then sin(x) and cos(x) are computed from these as indicated by the value mod(n,4). The value of y is in the interval (-pi/4, pi/4).

The polynomials p(x) and q(x) are:

$$p(x) = s(0)x + s(1)x^3 + s(2)x^5 + s(3)x^7 + s(4)x^9 + s(5)x^{11}$$

$$q(x) = c(0) + c(1)x^2 + c(2)x^4 + c(3)x^6 + c(4)x^8 + c(5)x^{10} + c(6)x^{12}$$

TABLE 2-9. RELATIVE ERROR OF CLOG=

Register	Mean	Standard Deviation	Minimum	Maximum
X6	-7.120E-14	4.603E-12	-4.435E-10	4.213E-11
X7	-2.200E-16	2.489E-15	-1.114E-14	8.085E-15

where the coefficients are:

s(0) = .999999999999972
s(1) = -.166666666665404
s(2) = $.833333331696029 * 10^{-2}$
s(3) = $-.19842607353790 * 10^{-3}$
s(4) = $.275548564509884 * 10^{-5}$
s(5) = $-.247320720952463 * 10^{-7}$
c(0) = .999999999999996
c(1) = -.499999999999991
c(2) = .0416666666664705
c(3) = $-.138888888888159 * 10^{-2}$
c(4) = $.248015784673257 * 10^{-4}$
c(5) = $-.275552187277097 * 10^{16}$
c(6) = $.206291063476645 * 10^{-8}$

The coefficients were obtained as follows. The polynomials of degrees 15 and 14 were obtained by truncating the MacLaurin series for sin(x) and cos(x) were telescoped to form the polynomials p(x) and q(x) of degrees 11 and 12. The telescoping is done by removing the leading term of the polynomial. This is accomplished by subtracting an appropriate multiple of T(n)(a(X - x((0))) of the same degree n; 2/a is the length of the interval of approximation, and x(0) is its center.

The Chebyshev polynomial of degree n, T(n)(x), is defined by T(n)(x) = cos(n * arccos(x)). The absolute value of x is no greater than one and satisfies the recurrence relation:

T(0)(x) = 1
T(1)(x) = x
T(n + 1)(x) = 2xT(n)(x) - T(n - 1)(x)

where $n \geq 1$.

For $n \geq 1$, T(n)(x) is the unique polynomial $2(n-1) * x^n + \ldots$ of degree n whose maximum absolute value over the interval (-1, 1) is minimal. This maximum absolute value is one.

The formulas used for the range reduction are:

$\sin(x) = (-1)^n \sin(y)$
$\cos(x) = (-1)^n \cos(y)$
if x = y + n pi, n an integer;
sin(x) = cos(x - pi/2)
cos(x) = -sin(x - pi/2)
if $pi/4 \leq x \leq pi/2$.

The input range is the collection of definite in-range floating-point quantities whose absolute values do not exceed $pi * 2^{46}$.

ERROR ANALYSIS

The maximum absolute error in the approximation of sin(x) by p(x) in the interval (-pi/4, pi/4) is $.1893 * 10^{-14}$. The maximum absolute error in the approximation of cos(x) by q(x) is $.3687 * 10^{-14}$.

Upper bounds on the machine round-off and truncation error over the input range (-pi/4, pi/4) have been established for p(x) at $7.523 * 10^{-15}$, and for q(x) at $1.401 * 10^{-14}$. Therefore, the maximum absolute error in computing sine in the interval (-pi/4, pi/4) is $9.416 * 10^{-15}$, and in computing cosine is $1.770 * 10^{-14}$.

EFFECT OF ARGUMENT ERROR

Not applicable, since this routine is not called directly by the user's program.

CSIN

CSIN is an external function which accepts calls from FORTRAN code. It computes the complex sine function (FORTRAN function name CSIN). It accepts a complex argument and returns a complex result.

Calls by name are computed at entry point CSIN.

METHOD

If x and y are real, then:

sin(x + i * y) = sin(x) * cosh(y) + i * cos(x) * sinh(y)

Upon entry, the argument is checked. It is invalid if the real part x or the imaginary part y is infinite or indefinite, if x or y is so large as to cause loss of precision in the calculation, or if floating-point overflow occurs during the calculation. If the argument is invalid, a diagnostic message is issued, and POS.INDEF. + i*POS.INDEF. is returned. If the argument is valid, the result of the computation is returned to the calling program.

ERROR ANALYSIS

The algorithm used in CSIN is the same as that used at entry point CSIN. of routine CSNCS.. See the description of CSNCS. for the error analysis.

EFFECT OF ARGUMENT ERROR

If a small argument error appears, then the error in the result is given approximately by multiplying the argument error by the complex cosine of the argument. Hence, if a small error occurs in the complex argument and the error has absolute value e', then the absolute value of the error in the result is given approximately by:

$e' * (\cos(x)^2) + (\sinh(y)^2)^{1/2}$

where x + i*y is the complex argument. If the argument error is significant, the error in the result should be found by substitution of the possible argument values in the function.

CSNCS.

CSNCS. is an external function which accepts calls from FORTRAN code. It computes the complex sine and complex cosine functions (FORTRAN function names CSIN and CCOS). It accepts a complex argument and returns a complex result.

Calls by value are computed at entry points CSIN. and CCOS..

METHOD

The input range is the collection of all definite in-range complex quantities z = x + i * y where |y| does not exceed 741.67 and |x| does not exceed $pi * 2^{46}$.

The formula used at entry point CSIN. is:

sin(x + i * y) = sin(x) * cosh(y) + i * + cos(x) * sinh(y)

The formula used at entry point CCOS. is:

$\cos(z) = \cos(x + i * y) = \cos(x) * \cosh(y) - i * \sin(x) * \sinh(y)$

where x and y are floating-point numbers. COS.SIN is called to compute the sine and cosine of x, and HYPERB. is called to compute the hyperbolic sine and cosine of y. The result is returned to the calling program with the real part in register X6, and the imaginary part in register X7.

ERROR ANALYSIS

See the description of HYPERB. and COS.SIN for details. If z = x + i * y is the argument, then the modulus of the error in the routine does not exceed:

$1.276 * 10^{-13} + 1.297 * 10^{-13} * \exp(|y|)$

for CSIN.; and:

$1.241 * 10^{-13} + 1.241 * 10^{-13} * \exp(|y|)$

for CCOS. For 10 000 arguments chosen randomly from the interval (-1.,1.) * (-1.,1.), statistics on relative error were observed for the complex sine and complex cosine methods. These are summarized in table 2-10.

EFFECT OF ARGUMENT ERROR

If a small error e(z) = e(x) + i*e(y) occurs in the argument z = x + i*y, the error in the result is given approximately by -sin(z)*e(z) for CSIN., and cos(z)*e(z) for CCOS.

CSQRT

CSQRT is an external function which accepts calls from FORTRAN code. It computes the complex square root function which maps to the right half of the complex plane (FORTRAN function name CSQRT). It accepts a complex argument and returns a complex result.

Calls by name are computed at entry point CSQRT.

METHOD

For the algorithm, see the description of CSQRT=. Upon entry, the complex argument is checked. The argument is invalid if its real or imaginary part is infinite or indefinite, or if floating-point overflow occurs during the calculation. If the argument is invalid, a diagnostic message is issued, and POS.INDEF. + i*POS.INDEF. is returned. If the argument is valid, CSQRT= is called at entry point CSQRT. for the computation. The result is returned to the calling program. For the purposes of this computation, values returned by the routine will lie in the right half of the complex plane.

ERROR ANALYSIS

See the description of CSQRT=.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument z, the error in the result w is given approximately by e'/(2*w). The modulus of this will give an approximate modulus of the error.

CSQRT=

CSQRT= is an external routine which accepts calls from FORTRAN code. It computes the complex square root function (FORTRAN function name CSQRT). It accepts a complex argument and returns a complex result.

Calls by value are computed at entry point CSQRT..

METHOD

The input range to this routine is the collection of all definite in-range nonzero complex quantities. If the argument is zero, zero is returned.

if z = x + i*y is the argument, the result is given by w = u + i*v where u and v are determined as follows:

$a = (x^2 + y^2)^{1/2}$
$b = ((a + x)/2)^{1/2}$
$c = y/(2 * b)$

If $x \geq 0$, then u = b and v = c. If $x < 0$, then u = c * sign(y) and v = b * sign(y). The result from this routine always lies in the first or fourth quadrant of the complex plane, and complex quantities lying on the axis of negative reals are taken by the routine to the axis of the positive imaginaries.

ERROR ANALYSIS

The routine was tested with a sample of 100 000 random numbers distributed over the complex plane with the distribution being the product of two Cauchy distributions. The maximum observed modulus of relative error was $1.595 * 10^{-14}$.

For 10 000 arguments chosen randomly from a given interval, statistics on relative error of the components of the results were observed. These are summarized in table 2-11.

TABLE 2-10. RELATIVE ERROR OF CSNCS.

Entry Point	Register	Mean	Standard Deviation	Minimum	Maximum
CSIN.	X6	-5.592E-15	8.653E-15	-4.030E-14	1.228E-14
	X7	-4.970E-15	5.877E-15	-3.165E-14	1.550E-14
CCOS.	X6	-3.501E-15	3.827E-15	-1.413E-14	1.182E-14
	X7	-7.313E-15	9.884E-15	-5.059E-14	1.771E-14

TABLE 2-11. RELATIVE ERROR OF CSQRT=

Interval x		Interval y		Register	Mean	Standard Deviation	Minimum	Maximum
From	To	From	To					
-100.	100.	-100.	100.	X6	-4.790E-16	2.652E-15	-9.774E-15	1.107E-14
				X7	-4.320E-16	2.655E-15	-9.726E-15	1.032E-14
$-10.^{100}$	$10.^{100}$	$-10.^{100}$	$10.^{100}$	X6	-4.053E-19	2.632E-15	-1.012E-14	1.036E-14
				X7	-4.098E-16	2.637E-15	-9.520E-15	1.096E-14

EFFECT OF ARGUMENT ERROR

If a small error e(z) = e(x) + i*e(y) occurs in the argument, the error in the result w = u + i*v is given approximately by e(z)/(2*w) = (e(x) + i*e(y))/2(u + i*v).

DASNCS.

DASNCS. is an external function which accepts calls from FORTRAN code. It computes the inverse sine and cosine functions (FORTRAN function names DASIN and DACOS). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry points DASIN and DACOS, and calls by value are computed at entry points DASIN. and DACOS..

METHOD

The input range is the collection of all valid double-precision quantities in the interval (-1.0,+1.0). Arguments outside this range initiate error processing.

The following identities are used to move the interval of approximation to (0,SQRT(.5)):

arcsin(-x) = -arcsin(x)
arccos(x) = pi/2-arcsin(x)
arcsin(x) = arccos(sqrt(1 - x^2)) $x \geq 0$
arccos(x) = arcsin(sqrt(1 - x^2)) $x \geq 0$

The reduced value is called y. If $y \leq .09375$, no further reduction is performed. If not, the closest entry to y in a table of values (z,arcsin(z),sqrt(1-z^2),z=.14,.39,.52,.64) is found, and the formula:

arcsin(x) = arcsin(z) + arcsin(w)

where w = x * sqrt(1-z^2) - z * sqrt(1 - x^2) is used. The value of w is in (-.0792,+.0848)

The arcsin of the reduced argument is then found using a 15th order odd polynomial with quotient:

x + x^3(c(3) + x^2(c(5) + x^2(c(7) + x^2(c(11) + x^2(c(13) + x^2(c(15) + a/(b-x^2))))))

where all constants and arithmetic before c(11) are double-precision and the rest is single-precision. The addition of c(11) has the form single+single=double. The polynomial is derived from a minimax rational form (denominator is (b-x^2)) for which the critical points have been perturbed slightly to make c(11) fit in one word.

To this value, arcsin(z) is added from a table if the last reduction above was done and the sum is conditionally negated. Then 0, -pi/2, +pi/2, or pi is added to complete the unfolding.

ERROR ANALYSIS

Table 2-12 summarizes the maximum relative errors of DASNCS..

The regions of worst error are (.09375,.1446) for DASIN and (.9895,.9966) for DACOS. In these regions the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs, the worst case being .1451-.0629. for DASIN, the polynomial range was extended to cover the region (.0821,.09375), where the worst error occurs. For DACOS, the extension is not used, so that the maximum relative error for either routine occurs in the region (.9956,.9966) in DACOS. For 10 000 points randomly distributed in this region, the maximum observed relative error in DACOS was 12.5E-29.

The mean relative errors for DACOS and DASIN are given in figures 2-9 and 2-10.

EFFECT OF ARGUMENT ERROR

If a small error eps occurs in the argument x, the resulting errors in DASIN and DACOS are approximately eps/(1-x^2)$^{.5}$ and -eps/(1-x^2)$^{.5}$. The amplification of the relative error is approximately x/(f(x)*(1-x^2)$^{.5}$ where f(x) is DASIN or DACOS. The error is attenuated for DASIN of abs(x) < 0.75 and for DACOS of x > -.44, but can become serious for DASIN near -1 or +1 and DACOS near -1. If the argument is generated as 1-y or y-1 then the identities:

asin(x) = acos(sqrt(1 - x^2))
acos(x) = asin(sqrt(1 - x^2))
asin(-x) = -asin(x)
acos(-x) = pi + asin(x)

can be used to get the full significance of y. When computing (1-x^2) one should use a form such as (1-x^2)=(1-x^2)*(1-x)=y*(2-y).

DATAN

DATAN is an external function which accepts calls from FORTRAN code. It computes the inverse tangent

TABLE 2-12. MAXIMUM RELATIVE ERROR OF DASNCS.

	DASIN	DACOS
Minimax rational form error	.082E-29	.082E-29
Algorithm error (double precision coefficients)	.76E-29	.48E-29
Maximum error observed	10.5E-29	12.5E-29

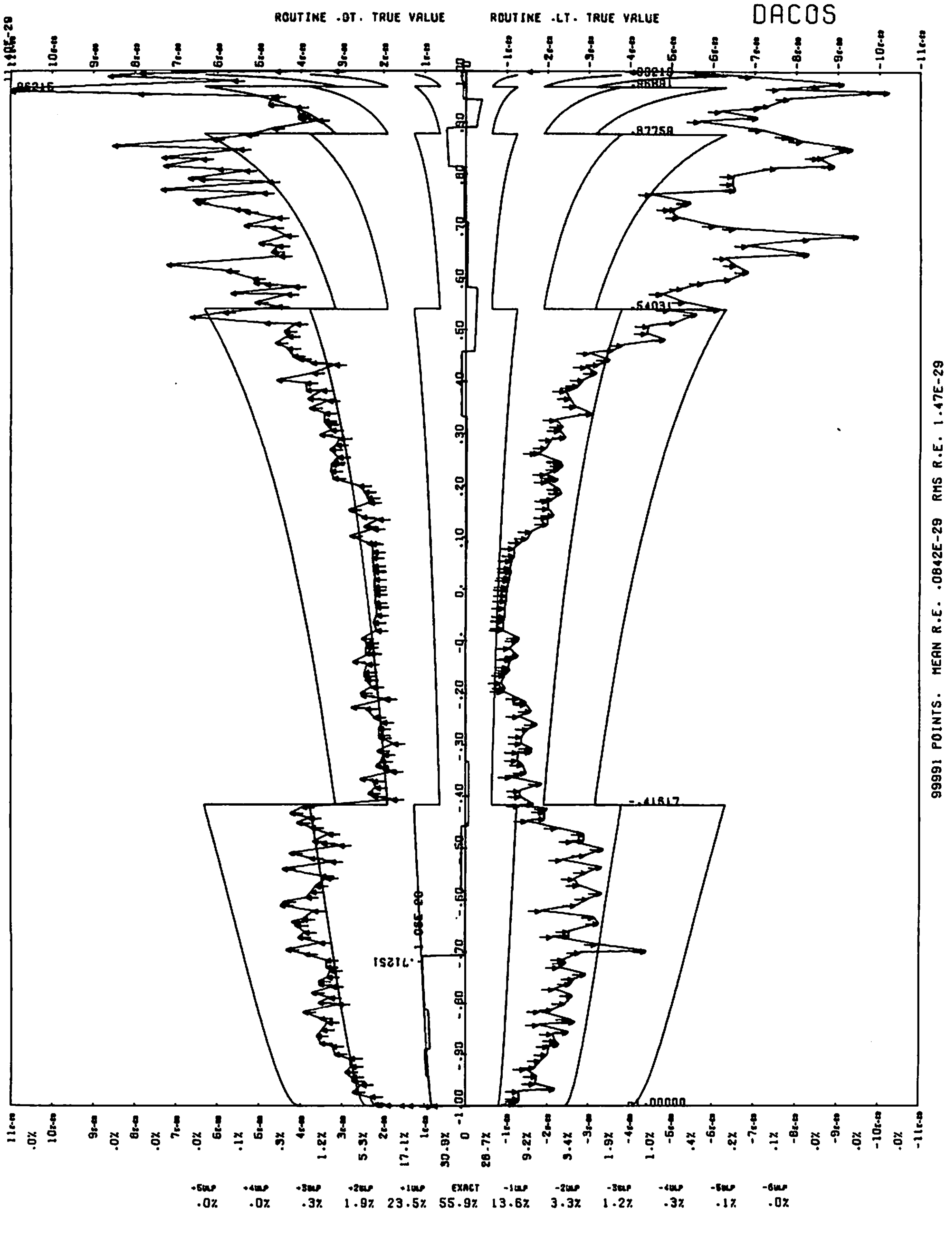

Figure 2-9. Mean Relative Error of DACOS

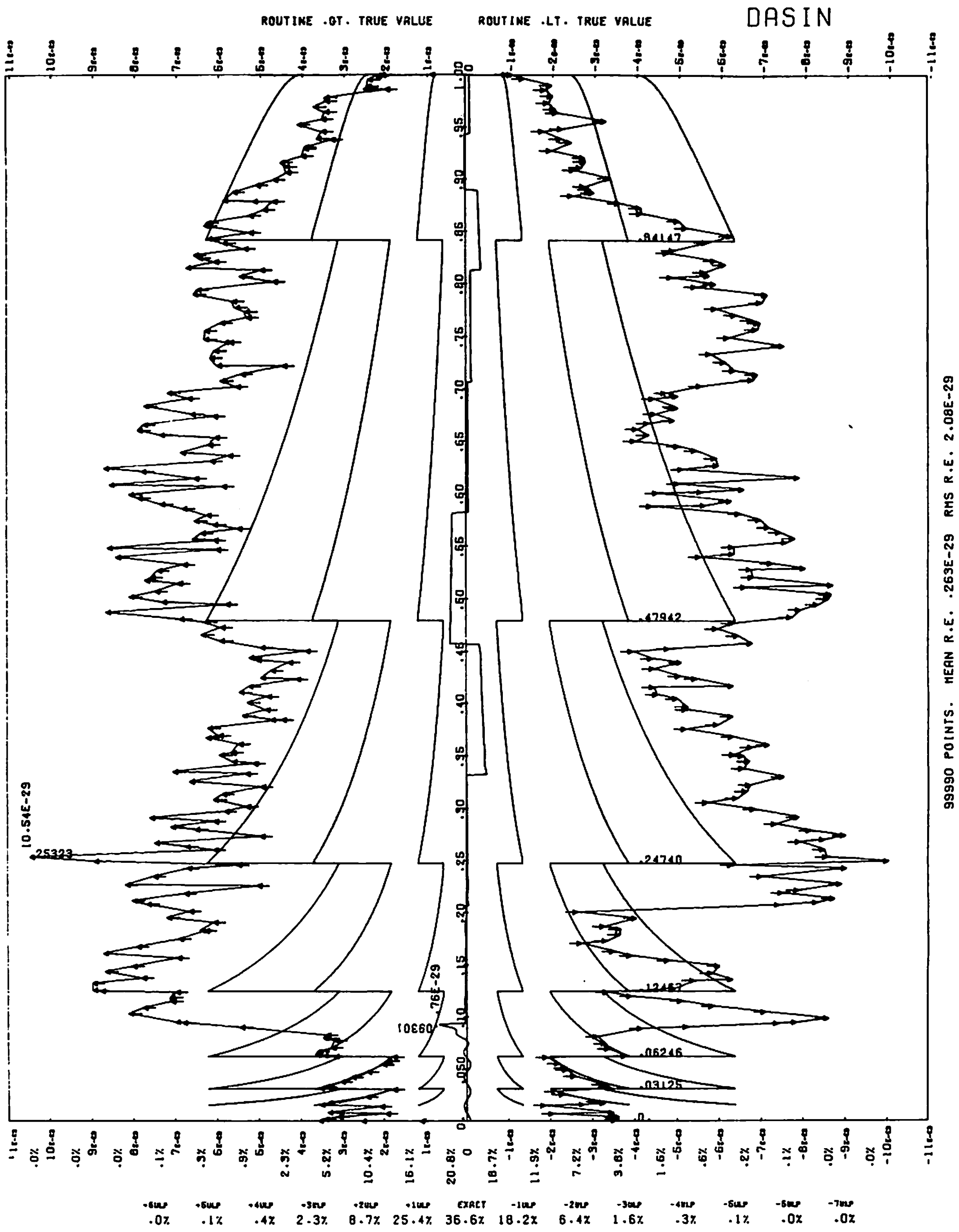

Figure 2-10. Mean Relative Error of DASIN

function (FORTRAN function name DATAN). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DATAN.

METHOD

The input range is in the collection of all valid double-precision quantities. Other arguments initiate error processing from DATAN.. Upon entry. the argument is loaded into registers X1 and X2, and routine DATAN. is entered for all remaining computations. See this routine's method description for further details.

ERROR ANALYSIS

See the error analysis of DATAN..

EFFECT OF ARGUMENT ERROR

See the argument error description of routine DATAN..

DATAN.

DATAN. is an external function which accepts calls from FORTRAN code. It computes the inverse tangent function (FORTRAN function name DATAN). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry point DATAN..

METHOD

The input range is the collection of all valid double-precision quantities.

Computation is performed mainly in routine DATCOM., and the constants used are listed there.

a. Transfer return address from entry point word into B6.

b. Test first word of argument for infinite or indefinite. If either, go to step i.

c. B3 = 0. (B3 holds a mask MI.)
B7 = 0. (B7 holds closest multiple of pi/2 to absolute value of result.)

d. B4 = sign mask for argument. (B4 holds MS, a mask for result's sign.)

e. Register pair X7-X3 absolute value of argument.

f. If absolute value of argument $<1.$, jump to routine DATCOM. at entry point DTN. to complete processing.

g. Register pair X5-X3 = absolute value of argument.
Register pair X4-X1 = 1.
B3 = -0.
B7 = 1.

h. Jump to routine DATCOM. at entry point DATCOM. to complete processing.

i. Pick up parameter for error processor. Call error processor, supplying given argument and parameters.

j. If error processor returns control, return pi/2, with the sign that is stored in B4. The value pi/2 is picked up by doubling an entry in a table starting at entry point ATN. in routine DATCOM.

ERROR ANALYSIS

10 000 random arguments were generated in the interval (1/200.,200.). In this sample, the maximum absolute value of relative error is $7.183*10^{-29}$. Groups of 40 double-precision arguments were chosen randomly in given intervals, and statistics on relative error were observed. These are summarized in table 2-13.

The maximum absolute value of relative error in the algorithm is 1.622E-29, and this occurs at 1.069781471095183.

Algorithm Error

Up to the point 1/16, the plot shows the error in the economized polynomial; it is not centered because the first coefficient was forced to be 1. The interval between (2n-1)/16 and (2n+1)/16 is repeated twice (once reflected), but the waviness is damped because of adding $\tan^{-1}$(n/8). Above 1.0, the subranges are delimited by 16/(2n-1).

Total Error

Most of the errors can be traced back to errors in double-precision addition. Note that the lower parts of the constants for pi and some of the atan(n/8)*s are negative. While it allows the constant to be precise to an extra bit or two, the unpredictable sign affects the addition process.

Figure 2-11 shows the mean relative error for DATAN..

EFFECT OF ARGUMENT ERROR

If a small error e occurs in the argument x, the error in the result is given by $e/(1+x^2)$.

DATAN2

DATAN2 is an external function which accepts calls from FORTRAN code. It computes the inverse tangent

TABLE 2-13. RELATIVE ERROR OF DATAN.

Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minimum	Maximum
-8.	8.	-1.995E-30	1.109E-29	-2.063E-29	3.208E-29
.01	10.	-1.505E-30	1.124E-29	-2.907E-29	2.745E-29

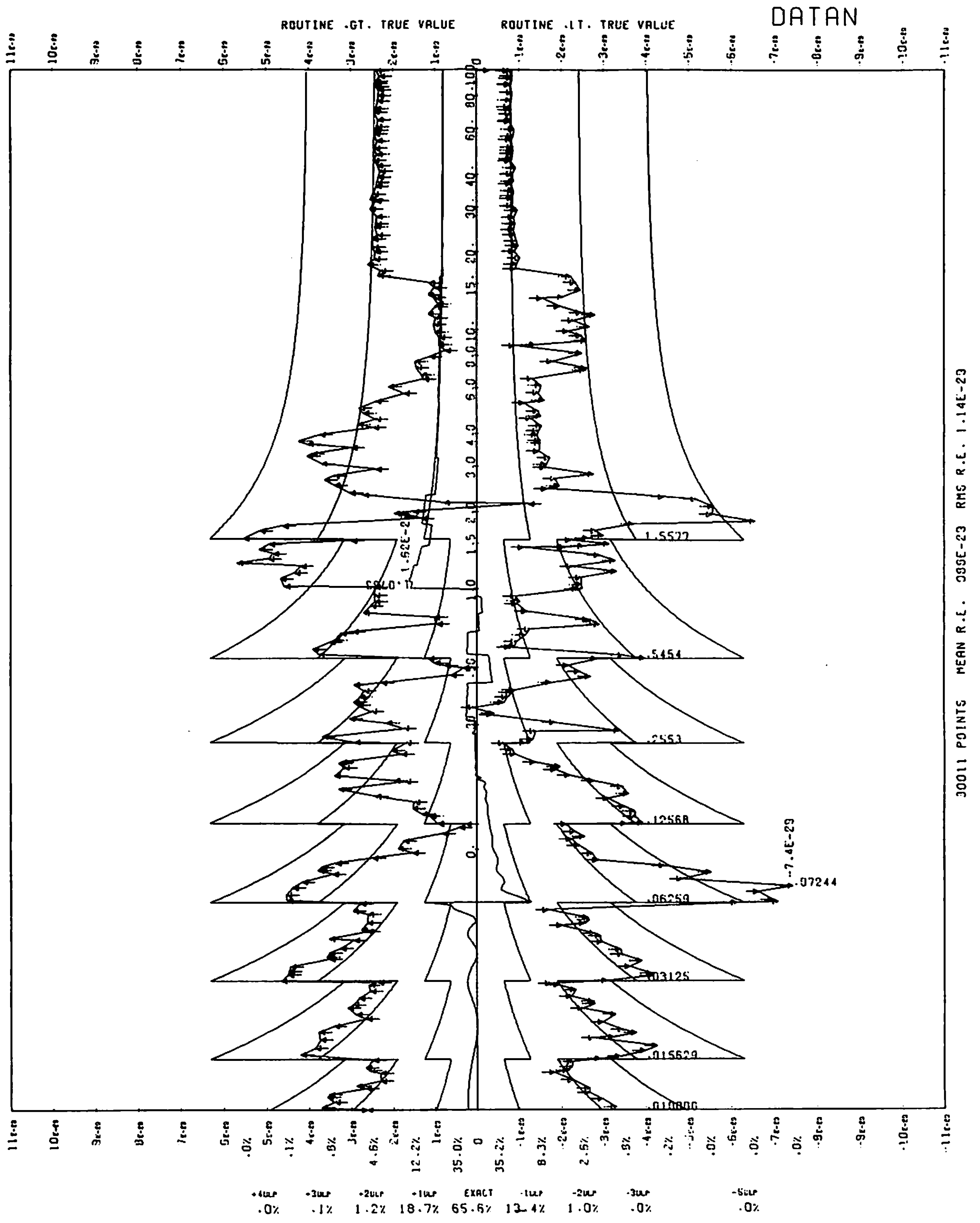

Figure 2-11. Mean Relative Error of DATAN.

function of the ratio of two arguments (FORTRAN function name DATAN2). It accepts two double-precision arguments and returns a double-precision result.

Calls by name are computed at entry point DATAN2.

METHOD

The input range is the collection of all pairs of valid double-precision quantities which are not both zero. Other arguments will initiate error processing from DATAN2.. Upon entry, the arguments are loaded into registers X1, X2, X3, and X4; routine DATAN2. is entered for all remaining computation. See this routine's method description for further details.

ERROR ANALYSIS

See the error analysis of routine DATAN2..

EFFECT OF ARGUMENT ERROR

See the argument error description of routine DATAN2..

DATAN2.

DATAN2. is an external function which accepts calls from FORTRAN code. It computes the inverse tangent function of the ratio between two arguments (FORTRAN function name DATAN2). It accepts two double-precision arguments and returns a double-precision result.

Calls by value are computed at entry point DATAN2..

METHOD

The input domain is the collection of all pairs of valid double-precision quantities which are not both zero.

Computation is performed mainly in routine DATCOM., and the constants used are listed there.

a. Test first words of both arguments to see if either is infinite or indefinite. If so, go to step j.

b. Normalize first words of both arguments.

c. If first words of both arguments are zero, go to step i.

d. B4 = sign mask of first word of first argument.
B3 = complement of sign mask of first word of second argument.
B6 = return address in calling routine.
B7 = 1.

e. Register pair X5-X3 = absolute value of first argument.
Register pair X4-X1 - absolute value of second argument.

f. If X5 > X4, jump to routine DATCOM. at entry point DATCOM. to complete processing.

g. X5 <-> X4
X3 <-> X1
Complement contents of B3.
B7 = 0, if first word of second argument is positive.
B7 = 2, if first word of second argument is negative.

h. Jump to routine DATCOM. at entry point DATCOM. to complete processing.

i. Supply message "ARGUMENT VECTOR 0,0".

j. Pick up parameters for error processor. Call error processor, supplying given arguments and parameters.

k. If control returns from the error processor, return POS.INDEF. to the calling program.

ERROR ANALYSIS

A group of 40 random double-precision arguments was chosen in (.01,10.) x (.01,10.), and statistics on relative error were observed. These are summarized in table 2-14.

The maximum absolute value of relative error in the algorithm is 1.622E-29.

EFFECT OF ARGUMENT ERROR

If small errors e' and e" occur in the arguments x and y, respectively, the error in the result is given approximately by:

$(x * e'' - y * e')/(x^2 + y^2)$

DATCOM.

DATCOM. is an auxiliary routine which accepts calls from DATAN. and DATAN2.. It performs computations that are common between these two routines.

The entry points for the routine are DATCOM., DTN., and ATN..

METHOD

On entry, at both entry points DATCOM. and DTN.:

B3 = mask MI.
B4 = mask MS = sign of final result.
B6 = return address after processing is complete.
B7 = closest multiple of pi/2 to absolute value of result.

In addition, at entry point DATCOM.,

Register pair X4-X1 = DU.
Register pair X5-X3 = DV.

and at entry point DTN.,

Register pair X7-X3 = DU.

Entry point ATN. is the start of an 18-word table containing $\tan^{-1}(n/8)$ $(0 \leq n \leq 8)$ in double-precision. Entry point DATCOM. corresponds to step a., and entry point DTN. corresponds to step b.. Constants used in the algorithm are:

TABLE 2-14. RELATIVE ERROR OF DATAN2.

Mean	Standard Deviation	Minimum	Maximum
-2.649E-30	2.161E-29	-6.188E-29	3.115E-29

d3	-.333 333 333 333 333 333 333 333 285 915
d5	.199 999 999 999 999 999 999 673 046 526
d7	-.142 857 142 857 142 856 280 180 055 289
d9	.111 111 111 111 109 972 932 035 508 119
c11	= -.090 909 090 908 247 503
c13	= .001 351 201 845 778 152
a	= -.085 666 743 757 593 089
b	= -1.133 579 709 202 919 6

where d3, d5, d7, d9 are double-precision constants, c11, c13, a, b are single-precision constants. Arithmetic operations with d subscripts are done in double-precision, those with u subscripts are done in single-precision. Boolean operations have B subscripts.

a. DQ = DU/DV in double-precision. Carry DQ in register pair X7-X3.

b. (DQ = DA-DU at DTN.) (Note that $0 \leq DQ \leq 1.$)

c. n = nearest multiple of 1/8 to DQ * DL = 0.

d. If n = 0, go to step f.

e. DA = (DQ-N/8)/(1 + N/8 * DA), computed in double-precision.

f. If (DA)(u)=0, go to step h.
XX = (DA)(u) *(u) (DA)(u)
X = $XX^{.5}$ (DA)(u) (((DA) (u) *(l) (DA)(u))/((DA)) (u) +(r) (DA)(u)))

g. DC = XX *(d) (d3 +(d) XX *(d) (d5 +(d) XX *(d) (d7 +(d) XX *(d) (d9 +(d) XX *(d) (d11 +(d) XX *(u) (c13 +(u) a/(b -(u) XX)))))))

h. v = (DA)(u) +(d) DC *(d) ((DA)(u) -(d) (DA)(u) *(i) (DA)(u)/ ((DA)(u) +(r) (DA)(u))) w v +(d) ((DA)(l) - X*((DA)(l) + (DA)(u) * (DA)(u)/((DA)(u) +(r) (DA)(u)))

i. $\underline{b}$ = (B7 * pi/2) - (B) B3 (upper and lower)

j. $\underline{c}$ = $\underline{b}$ + (d) $\tan^{-1}(n/8)$. $\tan^{-1}(n/8)$ is obtained as a double-precision quantity from the look-up table.

k. p = ($\underline{c}$ + (d) w) - (B) (B3 - ( ) B4)
Register pair X6-X7 .P, cleaned up.
Return to address B6 by direct jump.

ERROR ANALYSIS

Coefficients d3, d5, d7, d9, c11, c13, a, b were obtained by making the expression using these coefficients a minimax approximation to inverse tangent over (-1/16,1/16), within the class of expressions obtained by varying these coefficients. (See descriptions of routines DATAN. and DATAN2. for error analyses.)

EFFECT OF ARGUMENT ERROR

See descriptions of routines DATAN. and DATAN2. for effect of argument error.

DCOS

DCOS is an external function which accepts calls from FORTRAN code. It computes the cosine function (FORTRAN function name DCOS). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DCOS.

METHOD

See the description of DSNCOS. for the algorithm used in the computation. The argument is checked upon entry. It is invalid if infinite, indefinite, or so large as to lose precision during the calculation. If the argument is invalid, POS.INDEF. is returned, and a diagnostic message is issued. If the argument is valid, DSNCOS. is called at entry point DCOS. for the computation. The result is returned to the calling program.

ERROR ANALYSIS

See the description of DSNCOS..

EFFECT OF ARGUMENT ERROR

See the description of DSNCOS..

DCOSH

DCOSH is an external function which accepts calls from FORTRAN code. It computes the hyperbolic cosine function (FORTRAN function name DCOSH). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DCOSH.

METHOD

The input domain is the collection of all valid double-precision quantities whose absolute value is less than 1071*log(2). Arguments not in the domain initiate error processing in routine DHYP.. Upon entry the argument is loaded into register pair X1-X2 before routine DHYP. is called. (See the description of routine DHYP. for further details.)

ERROR ANALYSIS

See the error analysis of routine DHYP..

EFFECT OF ARGUMENT ERROR

See the argument error description of DHYP..

DEULER.

DEULER. is an auxiliary routine which accepts calls from DEXP., DHYP., and DTANH. It performs computations that are common among these routines.

The entry point for the routine is DEULER..

METHOD

Constants used in the routine are:

1./log(2)
log(2) (in double-precision)
d3 = .166 666 666 666 666 666 666 666 666 709

d5 = .833 333 333 333 333 333 333 331 234 953E-2
d7 = .198 412 698 412 698 412 700 466 386 658E-3
d9 = .275 573 192 239 858 897 408 325 908 796E-5
pc = -.474 970 880 178 988E-10
pa = .566 228 284 957 811E-7
pb = 272.110 632 903 710
c11 = .250 521 083 854 439E-7

The algorithm is:

a. n = nearest integer to x/log 2.
   y = x - n * log (2)
   (Then y is in (-1/2 * log(2), 1/2 * log(2))).

b. a = ((y)(u) * (u) (y)(u))$^{.5}$ = (y)(u) (-(y)(u) * (l)
   q (y)(u) * (u) (y)(u)

c. p = q * (d)(d3 + (d)q * (d)(d5 + (d)q * (d)(d7 + (d)q * (d)
   (d9 + (d)q * (d)(c11 + (d)q * (d)(pa/(pb - q) + pc))))))

d. s = (y)(u) + (d)(y)(u) * (d)p

e. (compute hm = SQRT (1+s^2))
   hi = 3*q+((s)(u))^2 in single-precision.
   hi = hi + hi
   hk = 2 * (1.+hl)
   hl = ((y)(u) * (u)(y)(u) - hj)/hk - hi
   hm = hj + (u)(hk - (u) hl) * (u)(hi/hk)
   (hm now carries cosh-1.0 in single-precision)

f. DS = s + (d)(((y)(l) + (r)(y)(l) * (u) hm) + (r)
   ((s)(l) + (r)((y) 8u) * (l)(p)(u) + (r) (y)(u) * (r)(p)(l))))
   (DS now contains sinh (y) in double-precision)

g. DC = hm + (d) (DS*DS-2*hm-hm*hm)/(2 (1.+hm))
   evaluated in double-precision

h. DX = DS + DC

i. Clean up DS, DC, g with

   Register pair X6-X7 = DS.
   Register pair X0-X1 = DC.
   Register pair X4-X5 = DX.

j. Direct jump to B4.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

DEXP

DEXP is an external function which accepts calls from FORTRAN code. It computes the exponential function (FORTRAN function name DEXP). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DEXP.

METHOD

The input domain is the collection of all valid double-precision quantities lying in the interval:

(-975*log (2),1070*log (2)), (i.e., (-675.84,741.67))

Arguments outside this range initiate error processing from DEXP.. Upon entry, this argument is loaded into register pair X1-X2, and routine DEXP. is entered for the remaining computation. (See the description of routine DEXP. for further details.)

ERROR ANALYSIS

See the description of DEXP..

EFFECT OF ARGUMENT ERROR

See the description of DEXP..

DEXP.

DEXP. is an external function which accepts calls from FORTRAN code. It computes the exponential function (FORTRAN function name DEXP). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry point DEXP..

METHOD

The input domain is the collection of all valid double-precision quantities lying in the interval (-975*log (2),1070*log (2).

The argument reduction performed in routine DEULER. is:

x = < argument >
y =x - n * log (2)

where y = < reduced argument > is in (-1/2 log 2, 1/2 log 2) and n is an integer.

Most of the computation is performed in routine DEULER., and the constants used are listed there.

On input, the argument is in register pair X1-X2, and on output, the result is in register pair X6-X7.

a. x = < argument > . Save x. If |(x) (u)| ≥ 17315640000000000000B, go to step g.

b. Jump to routine DEULER. at entry point DEULER.. Register B4 = address for step c, X7 = upper part of x, X6 = lower part of x, X5 = packed sign mask of x.

   On return from DEULER., B3 = n, X4 = (DX) (u), X5 = (DX) (l), X0 = DC) (u), X1 = (DC) (l), X6 = (DC) (u), X7 = (DS) (l). Here, n = nearest multiple of log 2 to x, y = x-n*log (2), and DS*sinh (y), DC* cosh (y)-1, and DX*exp (y)-1, are all in double-precision.

c. w = 1.0 + (d) (DC + (d) DS). Unpack w, increase exponents by n, and repack into register pair X6-X7.

d. If upper word's exponent overflows, go to step g.

e. If lower word's exponent underflows, go to step i.

f. Return, with result in register pair X6-X7.

g. Set parameters, load original argument, and call error processor.

h. If error processor returns control, return.

i. Set parameters, load original argument, and call error processor.

j. If error processor returns control, return 0. in X6 and X7.

ERROR ANALYSIS

10 000 random arguments were generated in the interval (-1/2 log 2, 3/2 log 2), and the resulting graph of relative error versus argument is shown in figure 2-12. In this interval, the largest absolute value of relative error is 3.858E-29. Groups of 100 double-precision arguments were chosen randomly in given intervals, and statistics on relative error were observed. These are summarized in table 2-15.

The approximation is described in the section on error analysis of routine DEULER.. It is a minimax approximation within the class obtained by varying the coefficients.

Algorithm Error

The curve for the algorithm error is barely distinguishable. It peaks at odd multiples of log 2/2 with a value of about .04E-29. The algorithm error has essentially no effect on the total error.

Total Error

Except for adjusting the exponent, the final computation in DEXP is 1.0+s, where $|s| \leq .3536$. This addition is easy to do exactly when s is small and positive. (See the plot just above 0 and log 2.) For s negative, the sum is less than 1 (i.e., it crosses a band boundary, and it becomes difficult to produce an exact result. The plot is exact or one bit low). When $s \leq .25$ (e.g., $.35 < x < .45$), it becomes even more difficult to prevent bits from dropping off in the low precision word when lower sums overflow.

Figure 2-13 shows the relative error in the algorithm used to approximate exp.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument the error in the result y is given approximately by y*e'.

DHYP.

DHYP. is an external function which accepts calls from FORTRAN code. It computes the hyperbolic sine and cosine functions (FORTRAN function names DSINH and DCOSH). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry points DSINH. and DCOSH..

METHOD

The input domain is the collection of all valid double-precision quantities lying in the interval (-1071*log(2),1071*log(2)).

Most of the computation is performed in routine DEULER., and the constants used are listed there. The argument reduction performed in routine DEULER. is:

x = <argument>
y = <reduced argument>
y = x-n*log(2)

where n is an integer, and y is in the interval (-1/2*log(2), 1/2*log(2)). The recombination formula is:

$$\cosh(y+n*\log 2) = (\cosh(y)+\sinh(y))\,2^{(n-1)} + (\cosh(y)-\sinh(y))\,2^{(-n-1)}$$
$$\sinh(y+n*\log 2) = (\cosh(y)+\sinh(y))\,2^{(n-1)} - (\cosh(y)-\sinh(y))\,2^{(-n-1)}$$

At entry points DSINH. and DCOSH., the argument is in register pair X1-X2, and on exit, register pair X6-X7 holds the result. DSINH. corresponds to entry at step a., and DCOSH. corresponds to entry at step m.

a. a = <argument> = X1-X2.
   b = a Store b in X7-X6.
   B5 = sign of a.

b. B5 = packed zero.
   B4 = address of step g.
   B1 = 1.

c. If (b)(u) < xmax(u), jump to routine DEULER. at entry point DEULER.. If (b)(u) > xmax(u), go to step e. xmax is 1071*log(2).

d. If (b)(l) < xmax(l), jump to routine DEULER. at entry point DEULER..

e. X1-X2 = a
   Set up parameters for error processor call with message "ARGUMENT TOO LARGE". If call was to entry point DCOSH., transfer contents of DCOSH. to DSINH..

f. Call error processor.
   If (a)(u) is indefinite, return through entry point DSINH. with X6-X7 = POS.INDEF. Otherwise, return through DSINH. with X6-X7 = POS.INF. or NEG.INF., the sign determined by B5.

g. Return from DEULER. with parameters:

B3	=	n
X4-X5	=	DX
X0-X1	=	DC
X6-X7	=	DS

where, if y = 1-n log(2),
   DX = exp(y)-1
   DC = cosh(y)-1
   DS = sinh(y)

TABLE 2-15. RELATIVE ERROR OF DEXP.

Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minimum	Maximum
-2.	2.	3.461E-31	8.256E-30	-2.632E-29	2.086E-29
-600.	700.	-8.631E-31	7.310E-30	-1.818E-29	1.446E-29

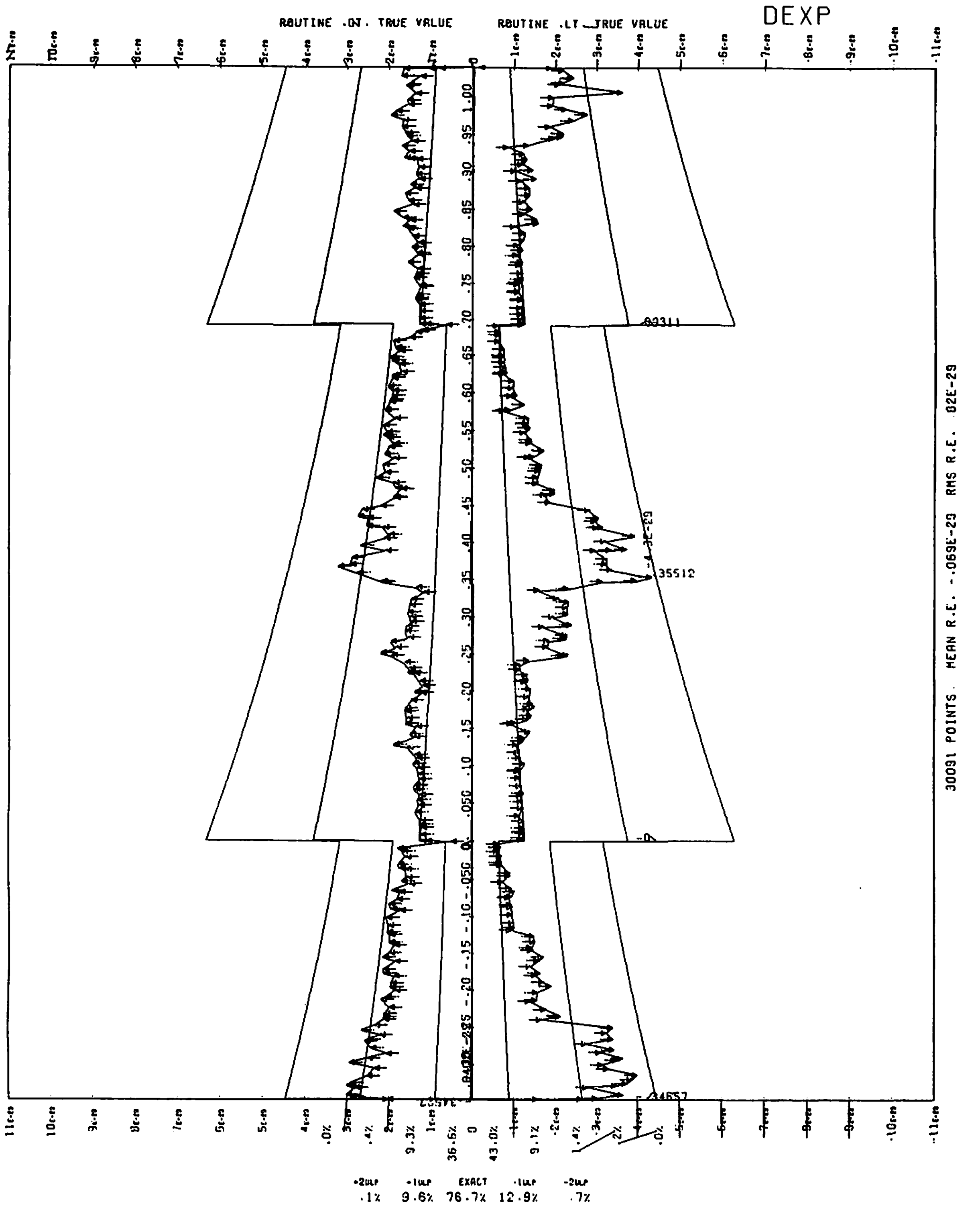

Figure 2-12. Mean Relative Error of DEXP.

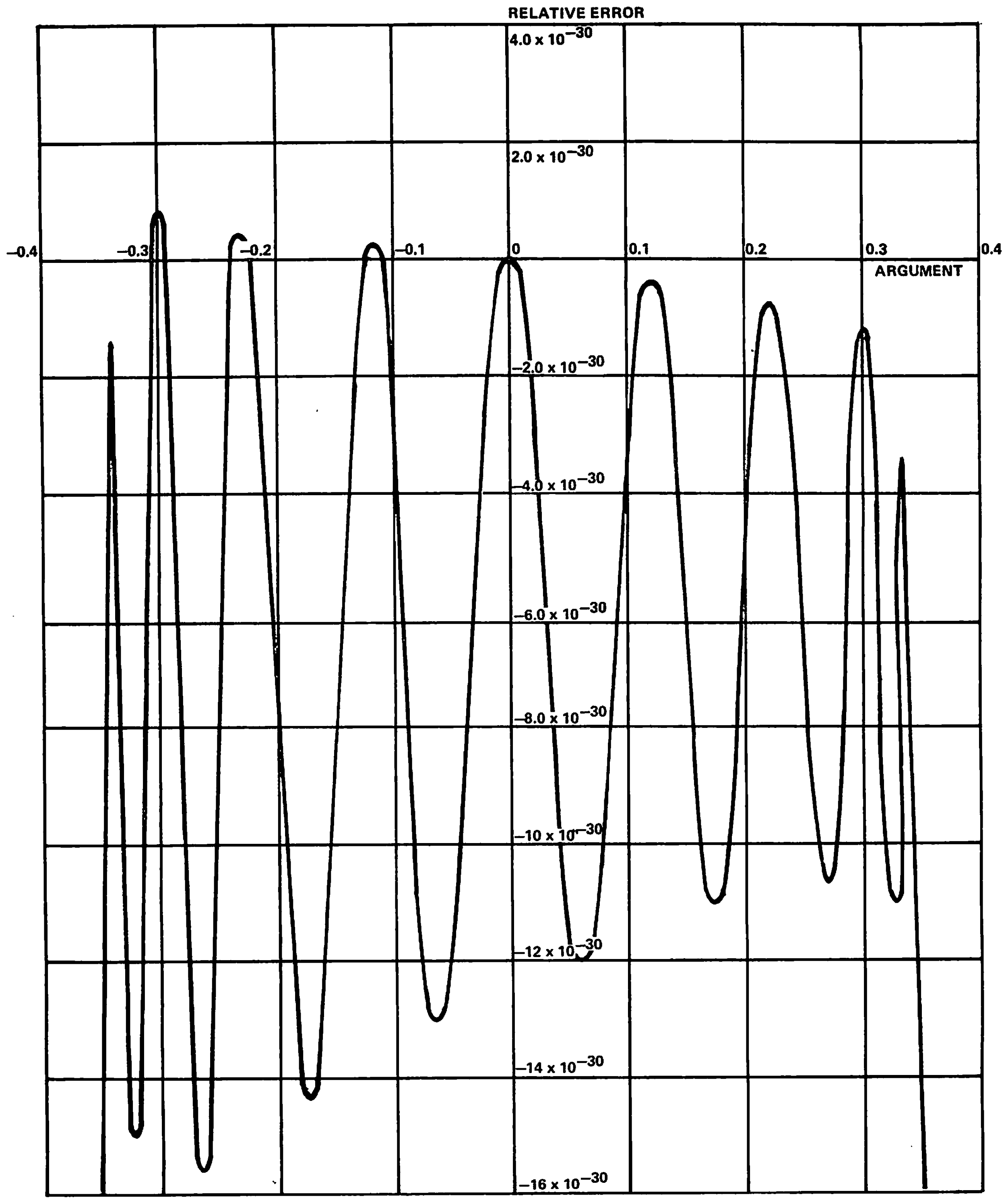

Figure 2-13. Algorithm Error of DEXP.

If n = 0, go to step l.
If $n \geq 48$, go to step k.
u = $2^{(n-1)}$(DC+DS) in double-precision.
v = $2^{(-n-1)}$(DC+DS) in double-precision.
w = $2^{(n-1)}$ + u in double-precision.
if $n \geq 24$, go the step h.
w = w $\pm$ ($2^{(n-1)}$ + v)(l) in double-precision. The sign is determined by B5.

h. w = w $\pm$ ($2^{(n-1)}$ + v)(u) in double-precision. The sign is determined by B5.

i. X6-X7 = w with the sign being the same as that of B5.

j. Return through entry point used to call routine.

k. w = (1.+DC+DS) * $2^{(n-1)}$
Go to step l.

l. If DSINH. entry, return through DSINH.. (Note that X6-X7 = DS)
X6-X7 = 1. + DC in double-precision.
Return through DCOSH.

m. a = X1-S2 = <argument>
b = a Store b in X7-X6.
B5 = 1
Go to step b.

ERROR ANALYSIS

10 000 random arguments were generated in the interval (-1/2 log 2,3^2 log 2) for DSINH and DCOSH, and the resulting graphs of relative error versus argument are shown in figures 2-14 and 2-15. In these samples, the maximum absolute values of relative error were 8.026E-29 for DSINH, and 4.405E-29 for DCOSH. Statistics on relative error were observed in random samples of arguments in given intervals. These are summarized in table 2-16.

Algorithm Error

DCOSH

The curve for the algorithm error is barely distinguishable. It peaks at odd multiples of log 2/2 with a value of about .04E-29. The algorithm error has essentially no effect on the total error.

DSINH

The peaks are at odd multiples of log 2/2 below 33.. At 47.5*log 2, the algorithm error has a sudden peak because at this point the algorithm switches to DSINH(x) =exp(x)/2. This point was chosen because $2^{(n-1)}$ can be done correctly using an IX instruction to add n to the top of 0.5. (48 would produce an indefinite).

Total Error

DCOSH

The total error curves should be symmetric about x=0. The pattern shown should repeat until 47.5*log 2 (about 33.) at which point it will start looking like the DSINH and DCOSH curves. Between 0 and log 2/2 (.3466), DCOSH is computed as 1+c where $0 \leq c \leq .75*SQRT(2)-1=.06066$. This is done accurately, but the addition sometimes drops a bit in the low word. Above log 2/2, the formula ends with a lot of addition and subtraction. For example, DCOSH(1.7443)=(4+1/16)-4*.3+small amount, where the .3 is about what the sinh polynomial produced. Notice that the subtraction crosses a band and the exponent on 4*.3 is only one less than the result; these facts make it difficult to keep from dropping bits.

DSINH

Up to log 2/2, the error is predominated by the final add in the sinh polynomial. Just above log 2/2 the error is especially large because of cancellation. Near log 2/2, DSINH is calculated using (1-1/4)-s+1/4*s where s is greater than 2^{-2} and the result is less than 2^{-1}. The parts of the curve in the two ranges (.35,16.) and (16.,33.), have different shapes because of the shortcut taken in the latter range. The split is at 23.5*log 2. Above 33.0 (47.5*log 2), the error curve is the same as for DEXP.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the error in sinh(x) is approximately cosh(x)*e', and the error in cosh(x) is approximately sinh(x)*e'.

DLOG

DLOG is an external function which accepts calls from FORTRAN code. It computes the natural logarithm function (FORTRAN function name DLOG). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DLOG.

METHOD

The algorithm used is given in the description of DLOG.. Upon entry, the argument is checked. The argument is invalid if it is infinite or indefinite, or is not greater than zero. If the argument is infinite, indefinite, or negative, POS.INDEF. is returned. If the argument is zero, NEG.INF. is returned. In any case, if the argument is invalid, a diagnostic message is issued. If the argument is valid, DLOG. is called at entry point DLOG. for the computation. The result is returned to the calling program.

TABLE 2-16. RELATIVE ERROR OF DHYP.

Entry Point	Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minimum	Maximum
DSINH.	-2.	2.	8.516E-31	1.086E-29	-2.738E-29	3.238E-29
	-600.	700.	-3.274E-31	7.907E-30	-2.645E-29	1.651E-29
DCOSH.	-2.	2.	-2.055E-30	1.217E-29	-3.071E-29	3.706E-29
	-600.	700.	-1.096E-30	9.645E-30	-2.733E-29	1.904E-29

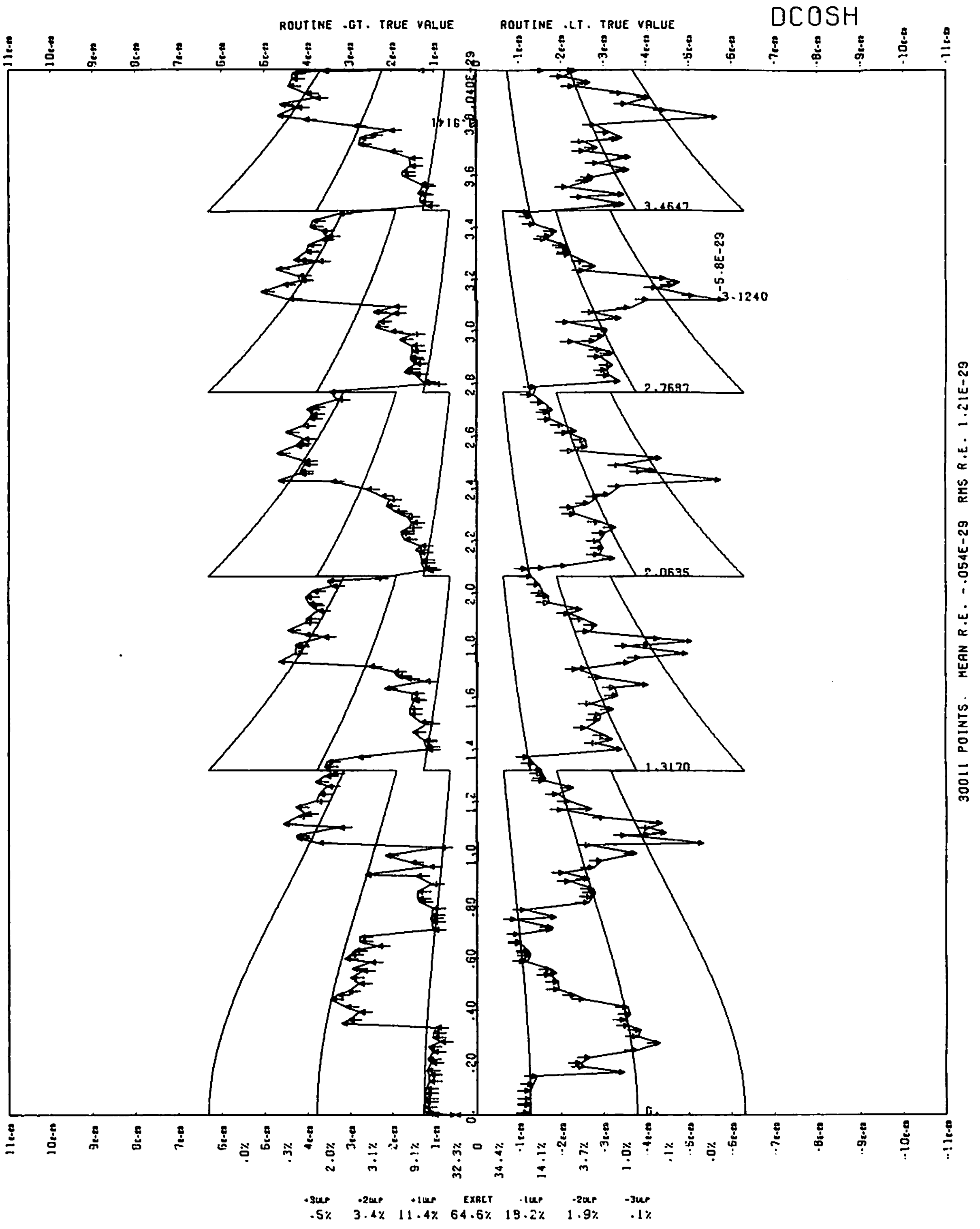

Figure 2-14. Mean Relative Error of DCOSH

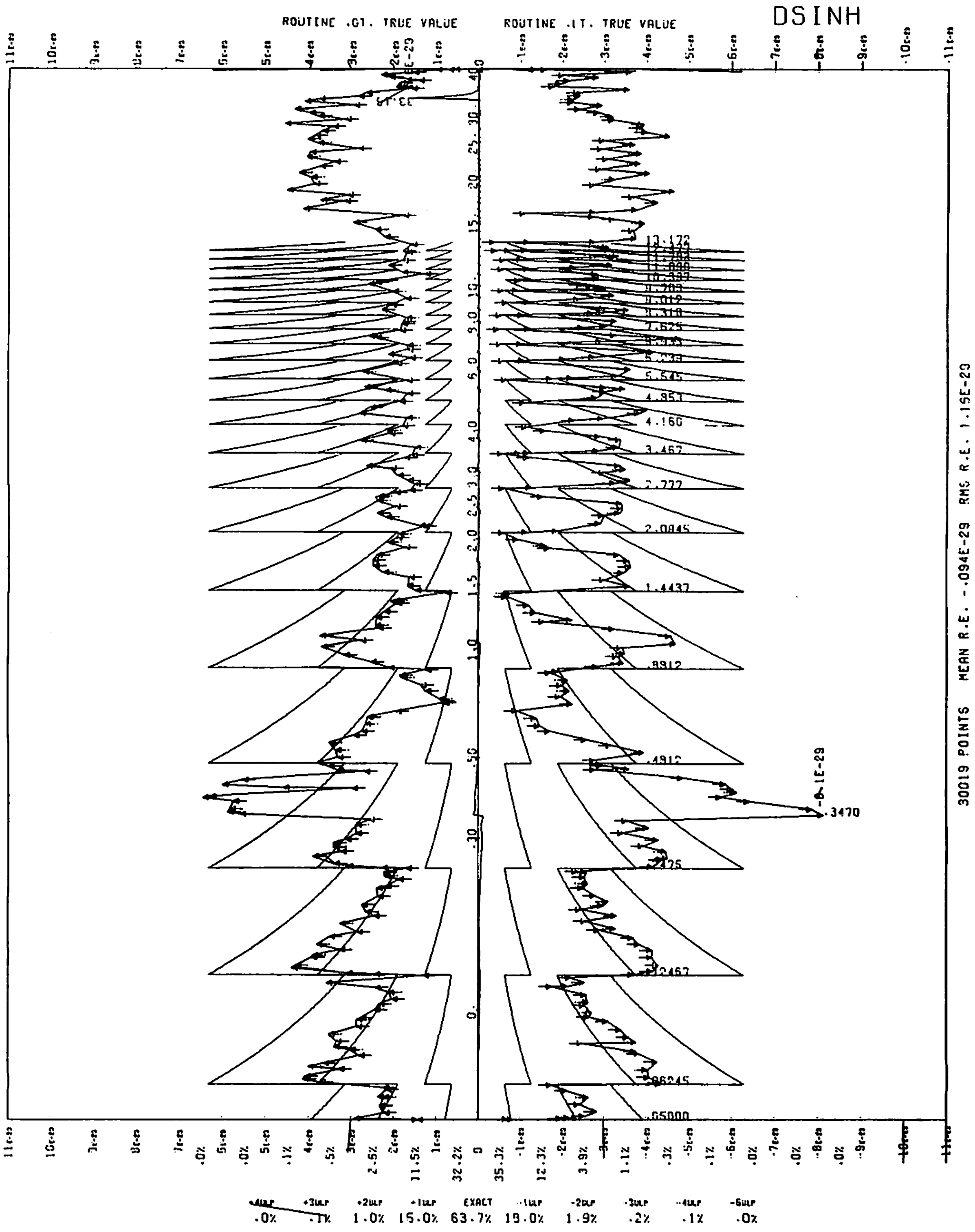

Figure 2-15. Mean Relative Error of DSINH

ERROR ANALYSIS

See the description of DLOG..

EFFECT OF ARGUMENT ERROR

See the description of DLOG..

DLOG.

DLOG. is an external function which accepts calls by FORTRAN code and by the DLOG and DLOG10 routines. It computes the natural and common logarithm functions (FORTRAN function names DLOG and DLOG10). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry points DLOG. and DLOG10..

METHOD

The input range is the collection of all definite in-range double-precision quantities which are greater than zero.

Upon entry, the argument x is put into the form $x = 2^k * w$, where k is an integer, and $2^{-1/2} \leq w \leq 2^{1/2}$. Then log x is computed from:

$\log x = k * \log 2 + \log w$

k * log 2 is computed in double-precision, while log w is evaluated as follows. A polynomial approximation u is first evaluated in single-precision using:

$u = c(1) * t + c(3) * t^3 + c(5) * t^5 + c(7) * t^7,$
$t = (w - 1)/(1 + w)$

where the coefficients c(1), c(3), c(5) and c(7) are:

c(1) = 1.9999999937340000
c(3) = 0.666669486638944
c(5) = 0.399657811051126
c(7) = 0.301005922238712

This approximates log with a relative error of absolute value at most $3.133 * 10^{-8}$ over $(2^{-1/2}, 2^{-1/2})$. Newton's rule for finding roots is then applied in two stages to the function exp(x) - w to yield the final approximation to log w. The two stages are algebraically combined to yield the final approximation v:

$$v = u - (1 - x * \exp(-u)) - (1 - x * \exp(-u - (1 - x * \exp(-u)))).$$

Writing $z = 1 - x * \exp(-u)$, z is much less than i, and v is computed using:

$$v = u - z(u) - z(l) - (z(u))^2 * (.5 + z(u)/3)$$

where $z = z(u)+z(l)$. This formula is obtained by neglecting terms which are not significant for double-precision; exp(-u) is evaluated in double-precision by the polynomial of degree 17 which is described in DEXP.. If entry was made at DLOG10., after k * log 2 + log w has been evaluated, the result is multiplied by log(10) * e in double-precision. The result is returned to the calling program.

ERROR ANALYSIS

The maximum absolute value of the error of approximation of the algorithm to log x is $1.555 * 10^{-29}$ over the interval $(2^{-1/2}, 2^{1/2})$. A graph of the error in the algorithm versus argument is given in figure 2-16. An upper bound on the absolute value of the machine round-off and truncation error (for arguments lying in $(2^{-1/2}, 2^{1/2})$) has been established at $5.146 * 10^{-28}$. Hence the absolute value of the error in the routine over the interval $(2^{-1/2}, 2^{1/2})$ is not greater than $5.302 * 10^{-28}$. The maximum absolute value of the relative machine truncation and round-off error has been established at $1.486 * 10^{-27}$. Hence an upper bound on the absolute value of the relative error in the routine over the interval $(2^{-1/2}, 2^{1/2})$ is $1.713 * 10^{-27}$.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the error in the result is given approximately by e'/x.

DLOG10

DLOG10 is an external function which accepts calls from FORTRAN code. It computes the common logarithm function (FORTRAN function name DLOG10). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DLOG10.

METHOD

Upon entry, the argument is checked. It is invalid if it is infinite or indefinite, or if it is not greater than zero. If the argument is infinite, indefinite, or negative, POS.INDEF. is returned. If the argument is invalid, a diagnostic message is issued. If the argument is valid, DLOG. is called at entry point DLOG10. for the computation. The result is returned to the calling program.

ERROR ANALYSIS

See the description of DLOG..

EFFECT OF ARGUMENT ERROR

See the description of DLOG..

DMOD

DMOD is an external function which accepts calls from FORTRAN code. It computes the modulus of an argument relative to a second argument (FORTRAN function name DMOD). It accepts two double-precision arguments and returns a double-precision result.

Calls by name are computed at entry point DMOD.

METHOD

The argument range is all valid double-precision pairs (x,y) such at $(x/y) < 2^{96}$ and y=0. After argument checking,

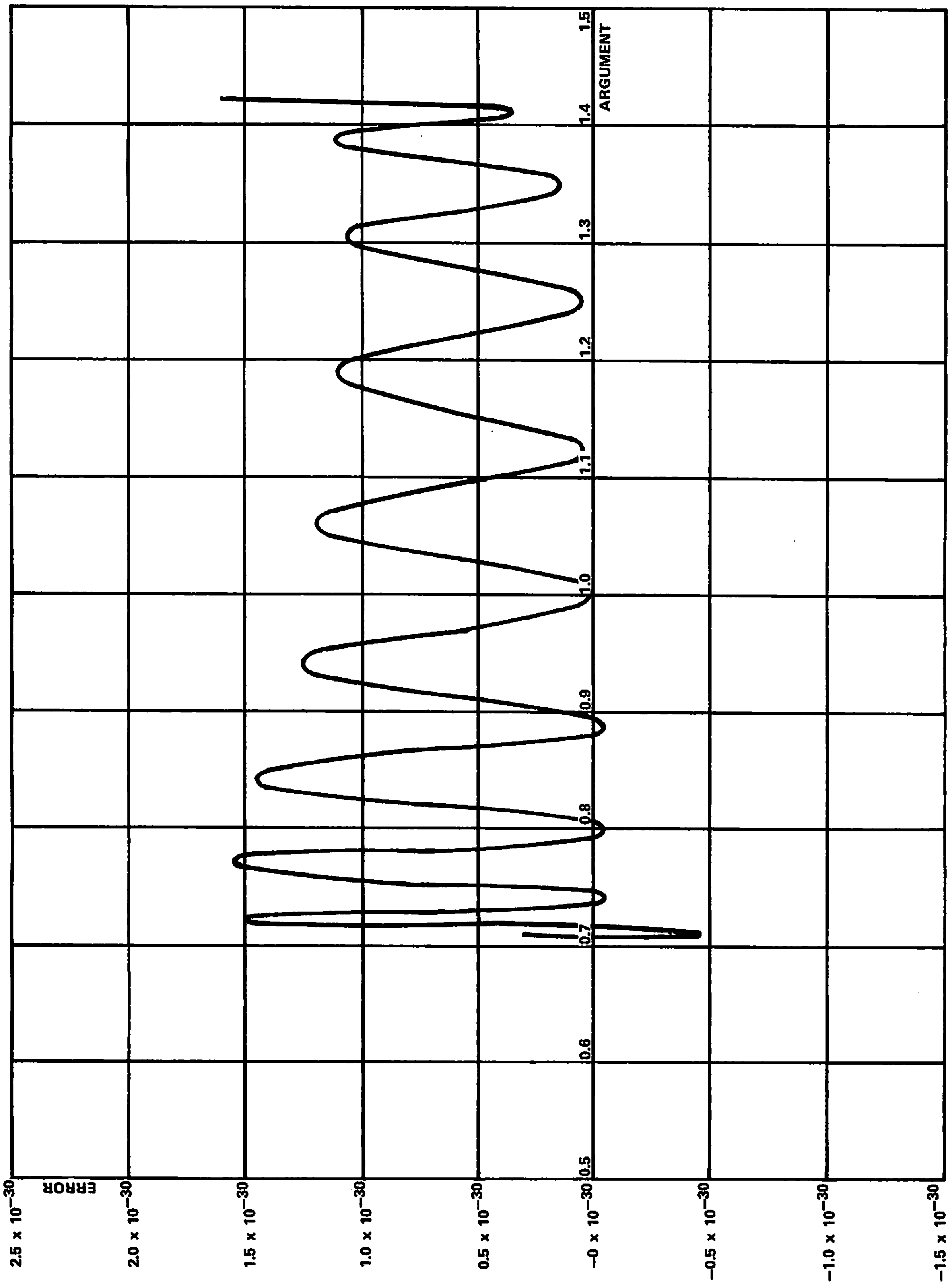

Figure 2-16. Algorithm Error of DLOG.

DMOD. is called to compute the result. The comparison $(x/y):2^{96}$ is done by comparing exponents and, if necessary, coefficients.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

DMOD=

DMOD= is an external function which accepts calls from FORTRAN code. It computes the remainder of an argument relative to a second argument (FORTRAN function name DMOD). It accepts two double-precision arguments and returns a double-precision result.

Calls by value are computed at entry point DMOD..

METHOD

The argument range is all valid double-precision pairs (x/y) such that $(x/y) < 2^{1070}$ and $y \neq 0$. The function computed by DMOD (x,y) is:

x-(x/y)*y

where parentheses denote truncation. The value of x is repeatedly reduced by 45-bit approximations to (x/y) until the reduced value lies in the range (0,sign(y,x)). Since the result does not exceed 96 bits, the intermediate value of x does not exceed 98 bits and the reduction is done in triple precision. The result is always exact.

ERROR ANALYSIS

Not applicable. The only double-precision operations concerned in a determination of error are multiplication and subtraction.

EFFECT OF ARGUMENT ERROR

Not applicable.

DSIN

DSIN is an external function which accepts calls from FORTRAN code. It computes the sine function (FORTRAN function name DSIN). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DSIN.

METHOD

The argument is checked upon entry. It is invalid if it is infinite or indefinite or is so large as to lose accuracy during the computation. If the argument is invalid, POS.INDEF. is returned and a diagnostic message is issued. An argument will lose accuracy if it exceeds pi * 2^{46} in absolute value. If the argument is valid, DSNCOS. is called at entry point DSIN. for the computation. The result is returned to the calling program.

ERROR ANALYSIS

See the description of DSNCOS..

EFFECT OF ARGUMENT ERROR

See the description of DSNCOS..

DSINH

DSINH is an external function which accepts calls from FORTRAN code. It computes the hyperbolic sine function (FORTRAN function name DSINH). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DSINH.

METHOD

The input range is the collection of all valid double-precision quantities whose absolute value is less than 1071*log(2). Arguments outside this range initiate error processing in routine DHYP.. Upon entry, the argument is loaded into register pair X1-X2 and routine DHYP. is called to complete the processing. See the description of routine DHYP. for further details.

ERROR ANALYSIS

See the description of routine DHYP..

EFFECT OF ARGUMENT ERROR

See the description of routine DHYP..

DSNCOS.

DSNCOS. is an external function which accepts calls from FORTRAN code and by the DSIN and DCOS routines. It computes the trigonometric sine and cosine functions (FORTRAN function names DSIN and DCOS). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry points DSIN. and DCOS..

METHOD

The input range is the collection of all definite in-range double-precision quantities which are less than pi*2^{46} in absolute value. Upon entry, the argument x is made positive and is multiplied by 2/pi in double-precision, and the nearest integer n to x * 2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2^{47}. If it does, POS.INDEF. is returned in X6 and a zero in X7. Otherwise, y = x - n * pi/2 is computed in double-precision as the reduced argument, y lies in (-pi/4,pi/4). The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

The sine polynomial approximation is:

$$p(x) = a(1)x + a(3)x^3 + a(5)x^5 + a(7)x^7 + a(9)x^9 + a(11)x^{11} + a(13)x^{13} + a(15)x^{15} + a(17)x^{17} + a(19)x^{19} + a(21)x^{21}$$

and the cosine polynomial approximation is:

$$a(x) = b(0) + b(2)x^2 + b(4)x^4 + b(6)x^6 + b(8)x^8 + b(10)x^{10} + b(12)x^{12} + b(14)x^{14} + b(16)x^{16} + b(18)x^{18} + b(20)x^{20}$$

for x in the interval (-pi/4, pi/4).

The coefficients are:

a(1) = .99999999999999999999999999999
a(3) = -.16666666666666666666666666652
a(5) = .83333333333333333333333270957 * 10^{-2}
a(7) = -.19841269841269841269829134478 * 10^{-3}
a(9) = .27557319223985890639440684401 * 10^{-5}
a(11) = -.25052108385441710113807647355 * 10^{-7}
a(13) = .16059043836817941727119406461 * 10^{-9}
a(15) = -.76471637307988608475534874891 * 10^{-12}
a(17) = .281145706930018 * 10^{-14}
a(19) = -.822042461317923 * 10^{-17}
a(21) = .194362013130224 * 10^{-19}

b(0) = .99999999999999999999999999999
b(2) = -.49999999999999999999999999919
b(4) = .41666666666666666666666613902
b(6) = -.13888888888888888888875543628 * 10^{-2}
b(8) = .24801587301587301569992273730 * 10^{-4}
b(10) = -.27557319223985877555866995711 * 10^{-6}
b(12) = .20876756987861921489874746135 * 10^{-8}
b(14) = -.11470745595858431549595076575 * 10^{-10}
b(16) = .47794769682239311593310626721 * 10^{-13}
b(18) = -.156187668345316 * 10^{-15}
b(20) = .408023947777860 * 10^{-18}

These polynomials are evaluated from right to left in double-precision using an in-stack loop. The sign flag is used to give the result the correct sign before return to the calling program.

ERROR ANALYSIS

Graphs of the errors in approximating sin(x) and cos(x) by p(x) and q(x) over the interval (-pi/4,pi/4) are given in figures 2-17 and 2-18.

The maximum absolute value of the error of approximation of p(x) to sin(x) over (-pi/4,pi/4) is .2570 * 10^{-28}, and of q(x) to cos(x) is .3786 * 10^{-28}. Upper bounds on the machine round-off and truncation error over (-pi/4,pi/4) have been established for p(x) at 1.743 * 10^{-27} and for q(x) at 1.364 * 10^{-27}. Hence an upper bound for the absolute value of error on this routine's computation of sine over (-pi/4,pi/4) is 1.769 * 10^{-27} and of cosine is 1.402 * 10^{-27}.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the resulting error in sin is given approximately by e' * cos(x). The resulting error in cos is given approximately by -e' * sin(x). If the error e' becomes significant, the addition formulas for sin and cos should be used to compute the error in the result.

DSQRT

DSQRT is an external function which accepts calls from FORTRAN code. It computes the square root function (FORTRAN function name DSQRT). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DSQRT.

METHOD

The argument is checked upon entry. It is invalid if it is infinite, indefinite or negative. If the argument is invalid, POS.INDEF. is returned, and a diagnostic message is issued. Otherwise, DSQRT. is called at entry point DSQRT. for the computation. The result is returned to the calling program.

ERROR ANALYSIS

See the description of DSQRT.. Figure 2-19 shows the mean relative error for DSQRT..

EFFECT OF ARGUMENT ERROR

See the description of DSQRT..

<u>H1</u> DSQRT.

DSQRT. is an external routine which accepts calls from FORTRAN code. It computes the square root function (FORTRAN function name DSQRT). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry point DSQRT..

METHOD

The argument range is the set of all valid double-precision numbers which are positive or zero. The identity:

$$\mathrm{sqrt}(y * 2^{(2 * n)}) = \mathrm{sqrt}(y) * 2^n$$

is used to reduce the input range to the interval (.5, 2.0). An initial approximation to sqrt(y) is computed using (31/64) * y + (31/64). This is accurate to five bits.

One Heron's iteration is performed, which produces 11 bits of precision and a positive error. The error is centered, giving 12 bits of precision. Two more Heron's iterations are used to produce a single-precision result. This result is converted to double-precision using one Newton's iteration:

x0 = (31/64) * y + (31/64)
x1 = .5 * (x0 + y/x0)
x1' = x1 - x1 * 2^{-12}
x2 = .5 * (x1' + y/x1')
x3 = .5 * (x2 + y/x2)
upper = pack (exponent, x3)
lower + (y * $2^{(2 * n)}$ - upper^2)/(2 * upper)

The 2^n scaling is performed by computing the final upper exponent and explicitly packing it, ignoring the exponent of x3. The y - x^2 computation is performed in double-precision, giving a one-word result since the upper portions nearly cancel.

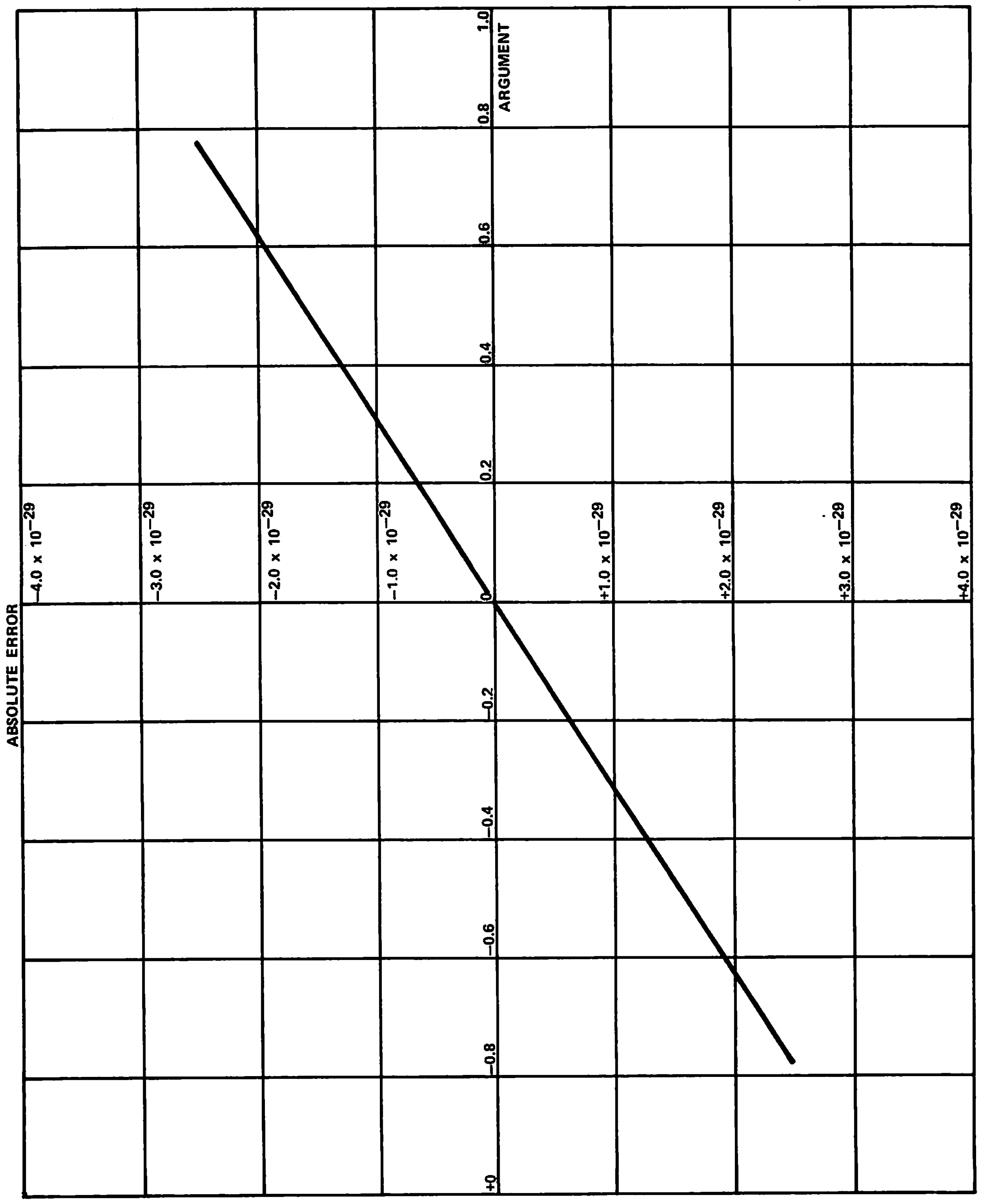

Figure 2-17. Algorithm Error of DSNCOS. for Sine

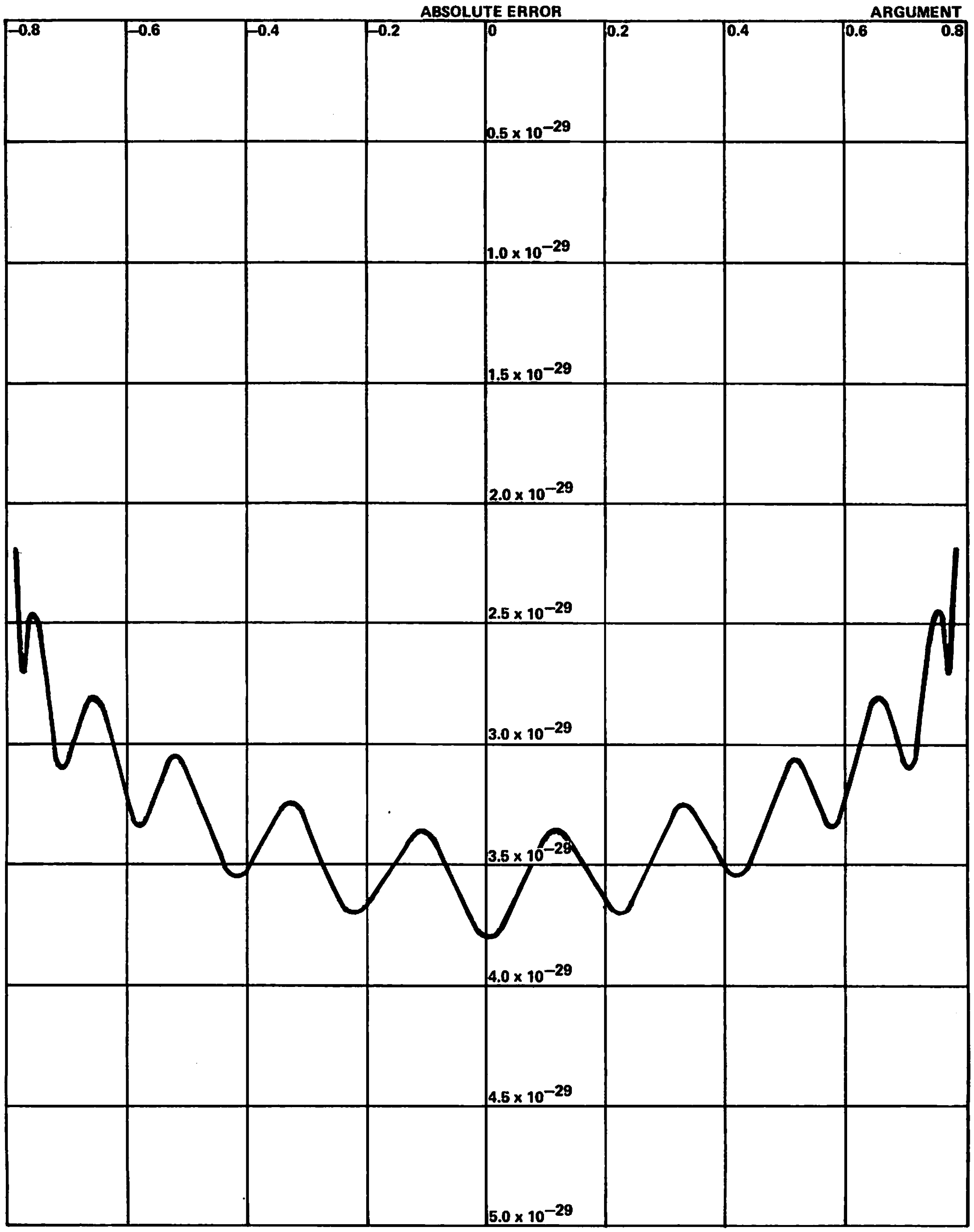

Figure 2-18. Algorithm Error of DSNCOS. for Cosine

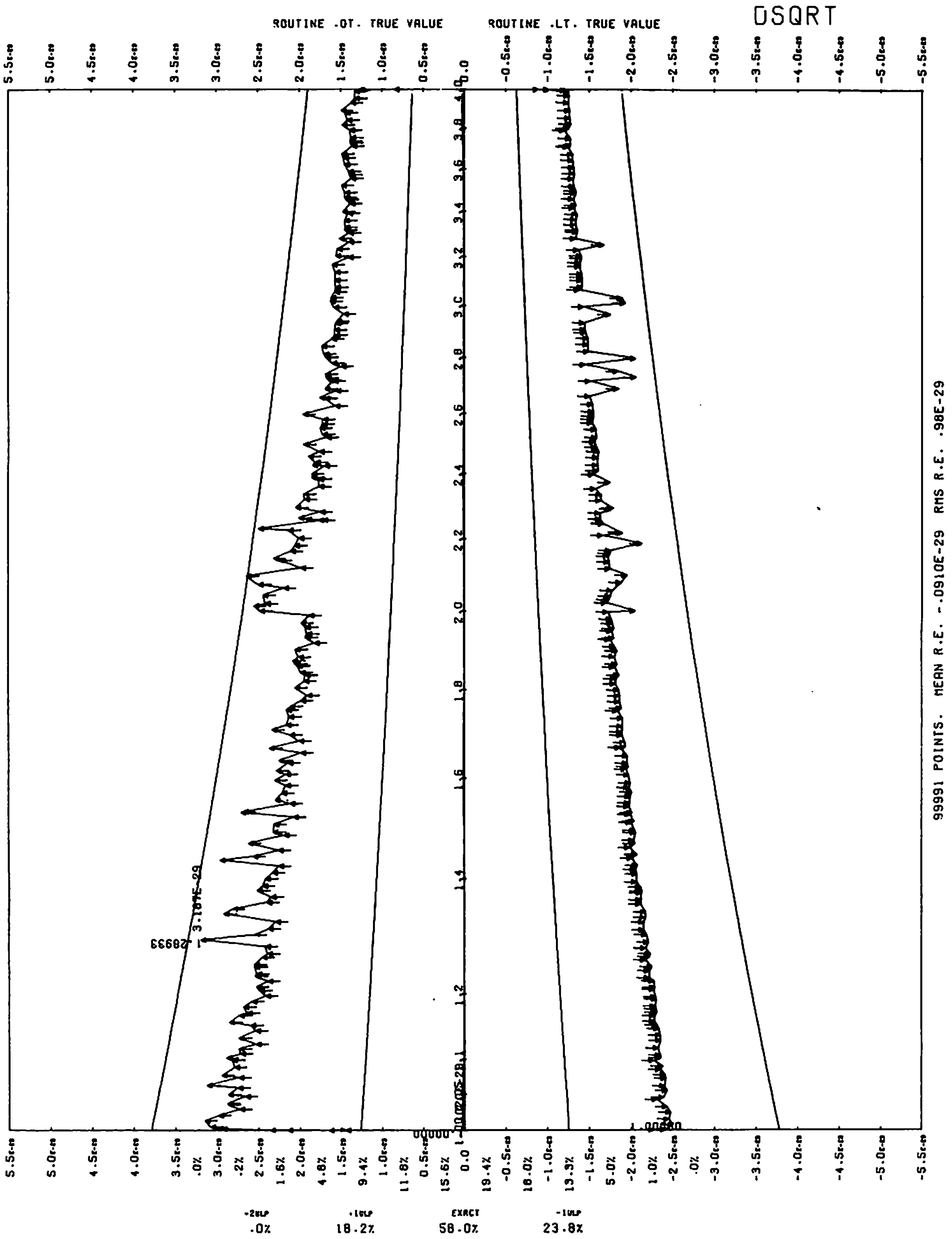

Figure 2-19. Mean Relative Error of DSQRT

ERROR ANALYSIS

The algorithm error is at most 2.05E-31, and is always positive. The round-off error in computing the single-precision approximation of x is exactly 1/2 ulp. The maximum is 7.55E-15.

Including algorithm error, x can have just over 1/2 ulp error. Since x is an approximation of the single-precision part only, the total error in x^2 can exceed 2 ulp when $y > x^2$.

Then $y - x^2$ may contain 50 significant bits. The error range for $y - x^2$ is (-1.78E-15, 3.55E-15), and the error range for $(y - x^2)/(2 * x)$ is (-8.88E-5, 3.55E-15). Relative to x, this error is (-6.71E-29, 2.68E-29).

In order to experience this error, the error in x must be at least 7.11E-15 so that the resulting error after the last Heron iteration is in the interval (-4.18E-29, 5.55E-29). The maximum observed error for 100 000 points randomly chosen in the interval (1, 4) was 3.19E-29. The maximum observed error for 200 000 points randomly chosen from the interval (1, 1.5) was 3.89E-29.

DTAN.

DTAN. is an external routine which accepts calls from FORTRAN code. It computes the tangent function (FORTRAN function name DTAN). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry point DTAN..

METHOD

The input range is the set of all valid double-precision quantities in the range $(-2^{49}, 2^{49})$. Arguments outside this range initiate error processing. The constants used to compute the function are listed in the description of routine DTAN. The argument reduction performed is:

(i) A pi/2 reduction is first performed. If the argument is outside the interval (-pi/4, pi/4), a signed integer multiple n of pi/2 is computed such that, after adding it to the argument, the result z falls in the interval (-pi/4, pi/4).

(ii) A 1/8 reduction is performed. A signed integer m, which is a multiple of 1/8, is subtracted from z such that the result is in the interval (-1/16, 1/16). A small number E(m) is also subtracted from z. The value of E(m) is constant such that the tangent of m/8 + E(m) can be represented to double-precision accuracy in a single-precision word. The lower word is zero. Therefore, the original argument y is reduced to x as follows:

$$x = y - (n * pi/2) - (m/8 + E(m))$$

The following quantities are computed from the reduced argument x, and the range reduction values. The functions U and L represent "upper of" and "lower of" functions.

$T = TAN(m/8 + E(m))$ (table look-up)
$R = L(U(x)^2)/2U(x) + L(x)$
$A = L(U(x)^2) + 2L(x)U(x)$
$B = U(U(x)^2)$

Since:

$$\begin{aligned} TAN(x) &= TAN(SQRT(x^2)) \\ &= TAN(SQRT(U(U(x)^2 + L(U(x)^2) + 2L(x)U(x))) \\ &= TAN(SQRT(B + A)) \\ &= TAN(SQRT(B) + A/2B) \\ &= TAN(SQRT(B) + R) \end{aligned}$$

Then $S = SQRT(B) = X_u - L(U(x)^2)/2U(x)$

The value of the original argument y is:

$TAN(y) = TAN(x + n * pi/2 + m/8 + E(m))$

The effect of the n * pi/2 term on the final result is:

$TAN(y) = TAN(x + m/8 + E(m))$ if n is even

$TAN(y) = 1/TAN(x + m/8 + E(m))$ if n is odd

Applying the tangent addition formula:

$$TAN(x + m/8 + E(m)) = TAN(S + R + (m/8 + E(m)))$$

$$= \frac{TAN(S) + TAN(R) + T - TAN(S) * TAN(R) * T}{1 - TAN(S) * TAN(R) - TAN(R) * T - T * TAN(S)}$$

$$= \frac{TAN(S) + R + T - TAN(S) * R * T}{1 - TAN(S) * R - R * T - T * TAN(S)}$$

The computation of TAN(S) uses the general polynomial form:

$x + x^3/3 + x^5 * 2/315 \ldots$

After applying Chebyshev to the coefficients, the form is:

$TAN(S) = S + S * (((1)S^2 + C(2)S^4 + C(3)S^6 + C(4)S^8 + (a/(B - S^2))S^{10})$

where a = .0218 ... and b = 2.467 ...

The quotient is inverted if n is odd.

ERROR ANALYSIS

For each of the intervals (-pi/4, pi/2) and (.01, 2pi), 100 000 random arguments were generated. The resulting graphs of relative error versus argument are shown in figures 2-20 and 2-21. The first graph has a linear scale, and the second has a logarithmic scale. The worst value for relative error was 1.6145E-28 at .065205. The percentage and mean for the second graph are biased since more points fall within the function's best range.

Algorithm Error

The algorithm error has a negligible effect on the total error. The worst relative error of the algorithm curve is 1.032E-29 at .06853. The discontinuities in the curve are a result of breaks in the range reduction with constants which cannot be represented exactly.

As figure 2-20 shows, there is a large peak in the relative error graph at 1/16. The same peak occurs at any multiple of pi/2 plus or minus 1/16. This is caused by the polynomial term in the quotient. Although the numerator and denominator of the quotient can be single-precision

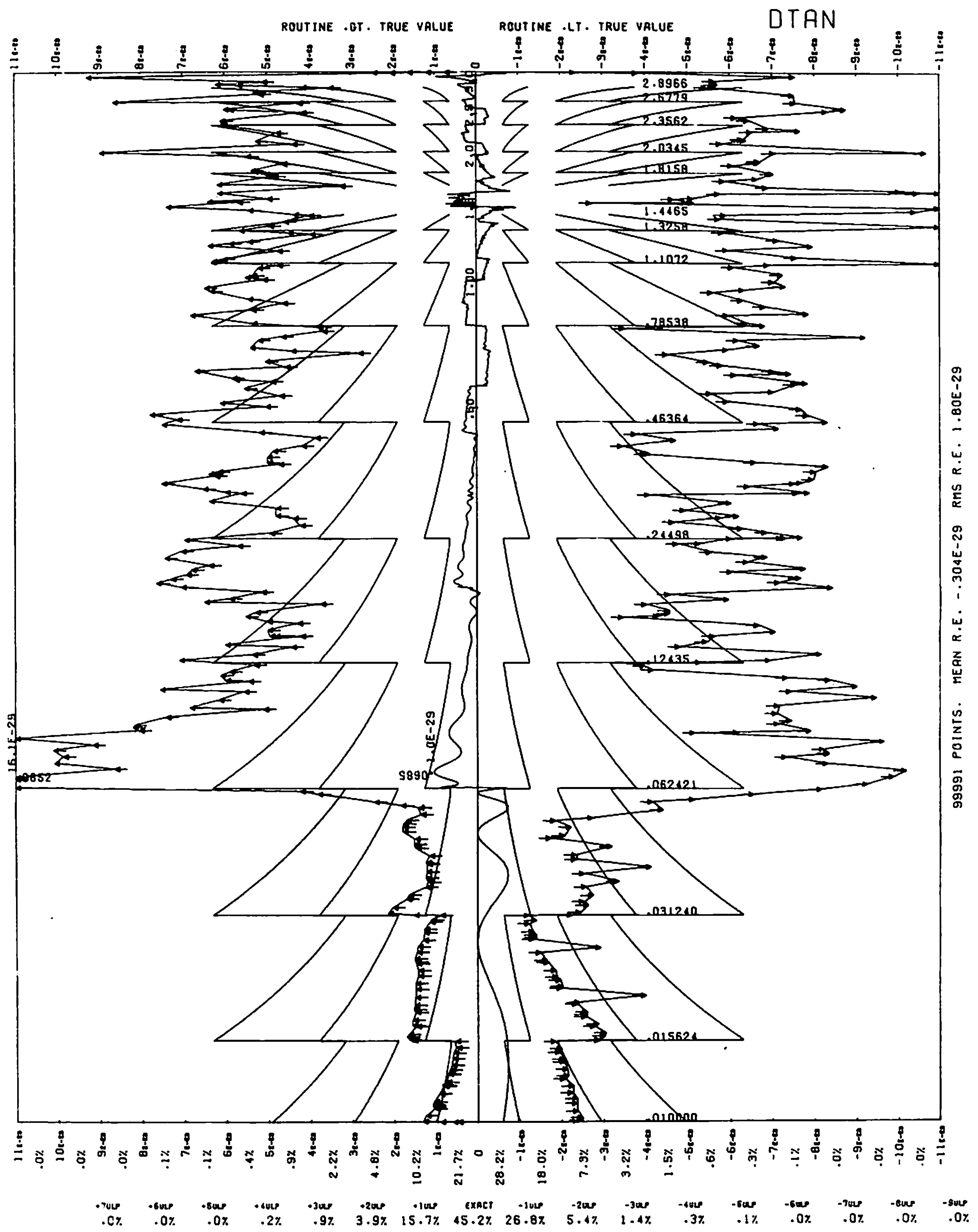

Figure 2-20. Mean Relative Error of DTAN. (arithmetic scale)

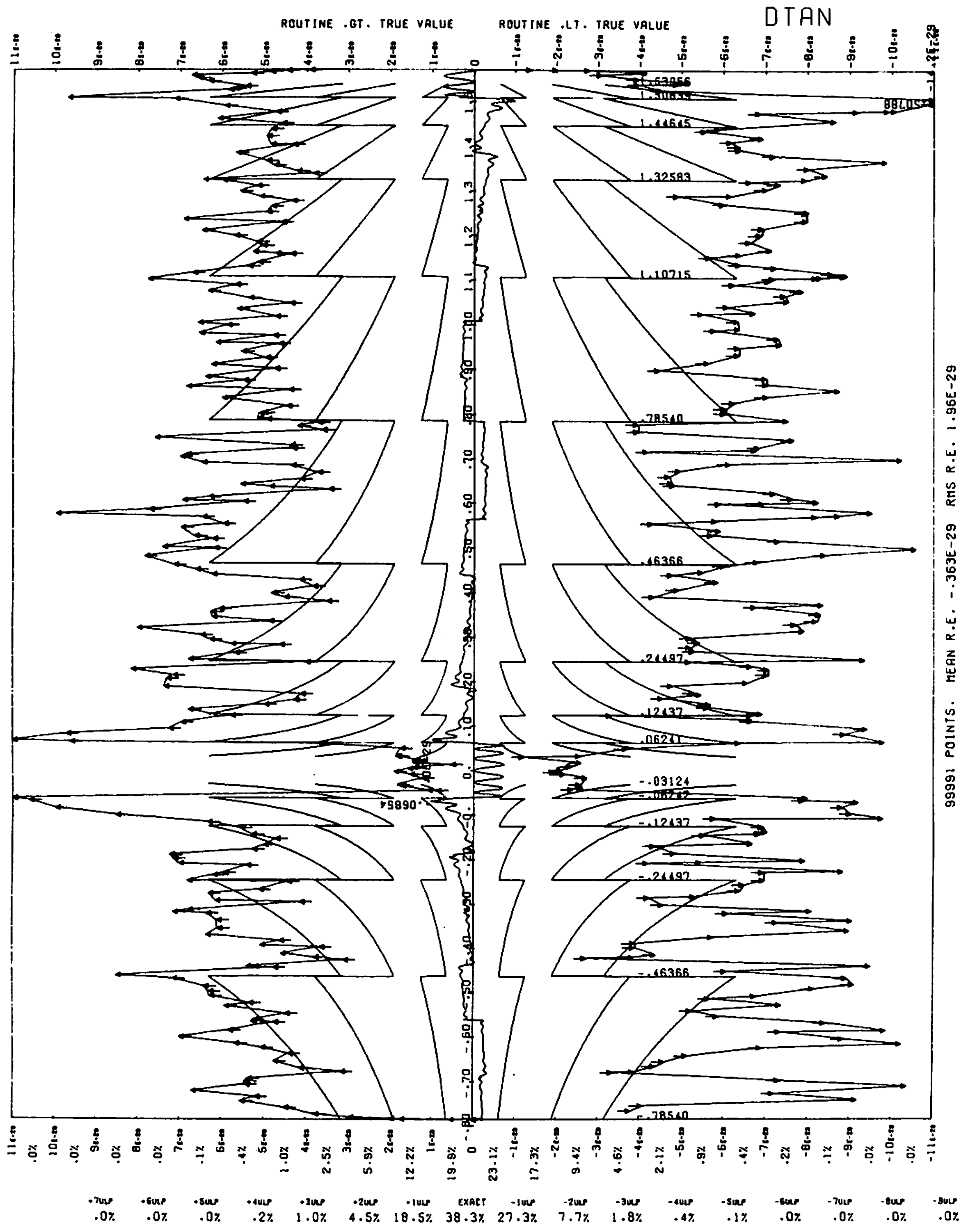

Figure 2-21. Mean Relative Error of DTAN. (logarithmic scale)

quantities, the result should be double-precision. Actually, it is computed in single-precision. If the algorithm error is computed with that division in single-precision, the worst relative error is 3E-29.

There is a negligible error introduced by the pi/2 range reduction except for points close to nonzero multiples of pi/2. Near pi/2 the pi/2 reduction relative error is bounded by $2^{(n - 155)}$ where n is the number of bits of precision to which the argument represents pi/2. At larger multiples of pi/2, similar problems occur.

Total Error

The total error curve is symmetric about all multiples of pi/2, except for variations caused by the range reduction error. For the range (0,pi/2), most of the error is attributed to forming the final quotient, except for the area around 1/16. In the interval (0,1/16) the error is smaller than in the interval (1/16,pi/2). This is because the denominator is 1 - e, where 1/e is less than 2^{52}. In the actual representation, the upper word is an exact 1 and the lower word is small and negative.

The divisions performed in double-precision introduce no error. Because this number is accurate, there is a better area in the range (pi/4 - 1/16, pi/4 + 1/16).

EFFECT OF ARGUMENT ERROR

If a small error e occurs in the argument x, the error in the result is e + e * $\tan^2(x)$.

DTANH

DTANH is an external routine which accepts calls from FORTRAN code. It computes the hyperbolic tangent function (FORTRAN function name DTANH). It accepts a double-precision argument and returns a double-precision result.

Calls by name are computed at entry point DTANH.

METHOD

The input domain is the collection of all valid double-precision quantities. Arguments outside the domain initiate error processing in routine DTANH.. Upon entry, the argument is loaded into register pair X1-X2, and routine DTANH. is entered to complete the computation. See the description of routine DTANH. for further details.

ERROR ANALYSIS

See the description of routine DTANH..

EFFECT OF ARGUMENT ERROR

See the description of routine DTANH..

DTANH.

DTANH. is an external function which accepts calls from FORTRAN code. It computes the hyperbolic tangent function (FORTRAN function name DTANH). It accepts a double-precision argument and returns a double-precision result.

Calls by value are computed at entry point DTANH..

METHOD

The input domain is the collection of all valid double-precision quantities. Arguments outside the domain which are indefinite initiate error processing. Most of the computation is performed in routine DEULER., and the constants used are listed there. The argument reduction performed is:

(i) for argument in (-47*log 2,47*log 2) but not in (-1/*log 2,1/2*log 2):
   x = < argument >
   y = < reduced argument >
   y = 2x - n * log 2
  where n is an integer, and y is in (-1/*log 2,1/2*log 2)
   tanh(x) = u/v where
   u = 1 - 2^{-n} - 2^{-n} * (DC-DS)
   v = 1 - 2^{-n} + 2^{-n} * (DC-DS)

(ii) for argument in (-1/2*log 2,1^2*log 2):
   x = < argument >
   y = < reduced argument >
   y = x
   tanh(x) = DS/(2*+DC)

(iii) for argument outside (-47*log 2,47*log 2):
   x = < argument >
   y = < reduced argument >

   tanh(x) = 1 - 2 ((1+DC-DS) * 2^{-n} - ((1+DC-DS) * $2^{-n})^2$)

In (i), (ii), and (iii), DC=cosh(y)-1 and DS=sinh(y).

On entry to DTANH., register pair X1-X2 holds the argument, and on exit, register pair X6-X7 holds the result.

a. a = X1-X2 = < argument >.
   X7-X6 = b = |a|.
   B5 = sign mask of a.
   X5 = packed zero.
   B1 = 1.
   B4 = address of step e.
   If exponent of first word of a is less than -49, jump to routine DEULER. at entry point DEULER..
   X7 = X7 * 2.
   X6 = X6 * 2.
   B4 = address of step c.
   If exponent of first word of a is less than -42, jump to routine DEULER. at entry point DEULER..

b. X6-X7 = ± 1. with sign obtained from B5.
   If a is definite, return.
   Set parameters for a call to error processor.
   Call error processor.
   If control returns from error processor, return.

c. On return from DEULER.:
   B3 = n = integer offset in argument reduction,
      X7-X6 = n*log 2 + y
   X4-X5 = DX
   X0-X1 = DC
   X6-X7 = DS

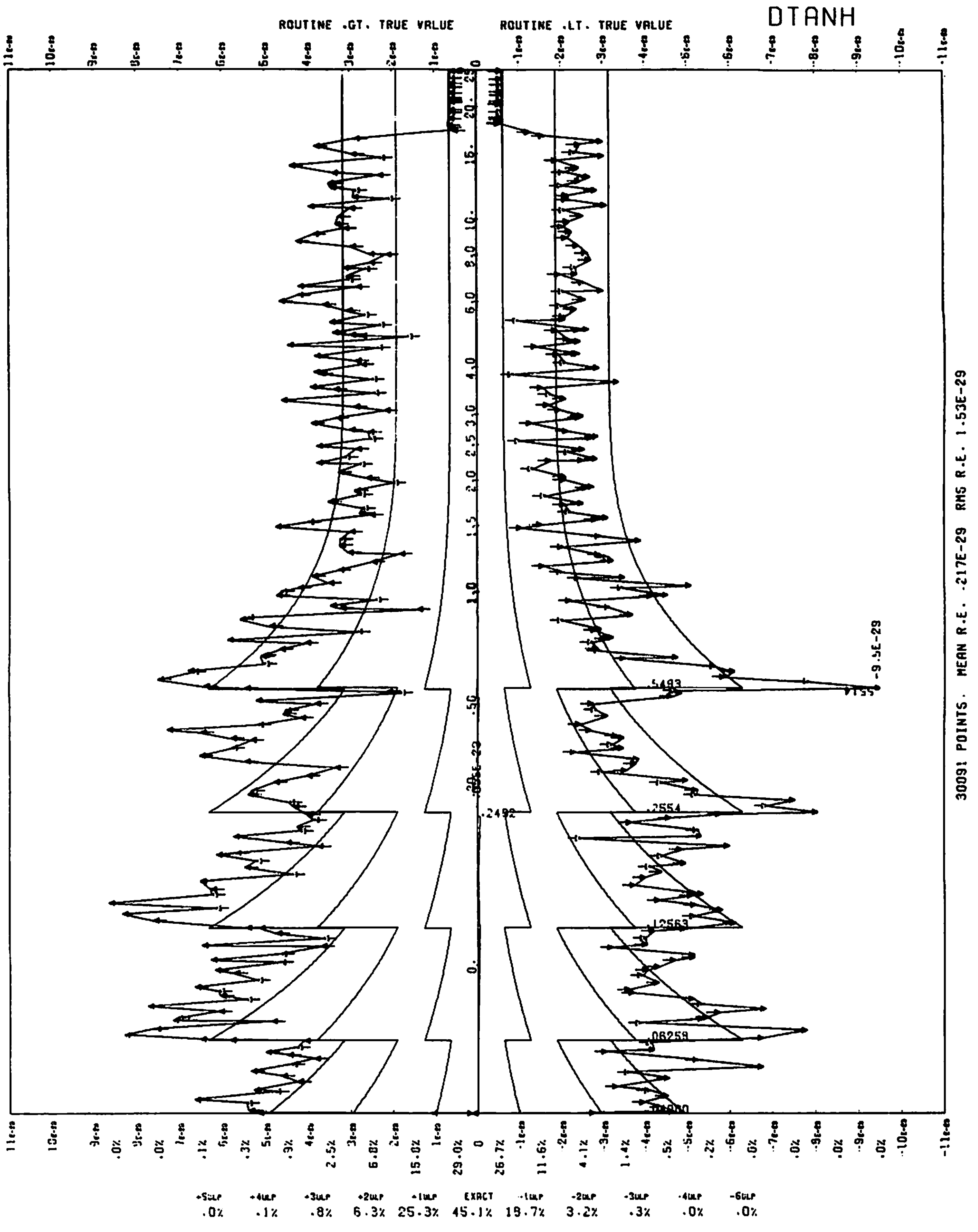

Figure 2-22. Mean Relative Error of DTANH.

where:

DX = exp (y)-1
DC = cosh (y)-1
DS = sinh (y)

If n 47, go to step f.
u = 1.-2^{-n} - 2^{-n} (DC-DS)
v = 1.+2^{-n} + 2^{-n} (DC-DS)

d. w = u/v, in double-precision.
Go to step g.

e. u = DS
v = 1.+DC, in double-precision.
Go to step d.

f. w = 1. - 2 * ((1*+DC-DS) * 2^{-n} - ((1.+DC-DS) * $2^{-n})^2$)
(evaluated in double-precision, although only the second word of 1. is affected.)

g. Clean up w, affix sign in B5 and leave in register pair X6-X7.
Return.

ERROR ANALYSIS

10 000 random arguments were generated in the interval:

(-1/2*log 2,3/2*log 2),

and the resulting graph of relative error versus argument is shown in figure 2-20. In this sample, the maximum absolute value of the relative error is 8.581E-29. Random samples of 100 arguments were generated in given intervals, and statistics on relative error were observed. These are given in table 2-17.

Algorithm Error

The algorithm error is insignificant. It is predominated by the error in the sinh expression in DEULER., but by various folding actions, the error is reduced even further.

Total Error

The error plot should be symmetric about the origin. In the range (0,.5) the error is dominated by the code to perform s/(1+c); the errors in s and in adding 1+c are secondary. Just above .5, several factors conspire to create errors: an addition of numbers of opposite sign in the numerator, an addition in the denominator, and a division. The errors in evaluating sinh are insignificant in comparison. Up to 16.5, the result is slightly less than 1.0 and the error is almost totally due to imprecise division of slightly imprecise arguments. From 16.5 to 64.0 (2^6), the result is perfect. Above 64.0 (not shown), the error will taper off to zero because the answer will be 1.0 while the true value is closer to 1.0 than 1-2^{-96}.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the error in the result is given approximately by e' * sech^2(x).

DTOD*

DTOD* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements that raise double-precision quantities to double-precision exponents. It accepts two double-precision arguments and returns a double-precision result.

Calls by name are computed at entry point DTOD$.

METHOD

The result is calculated by:

result = exp (exponent * log (base))

Upon entry, the argument set is checked. It is invalid if either argument is infinite or indefinite, if the base is negative, if the base is zero and the exponent is not greater than zero, or if floating-point overflow occurs during the computation. If the argument set is invalid, POS.INDEF. is returned, and a diagnostic message is issued. Otherwise, DTOD* computes the result according to the equation. The result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in routine DTOD* is the same as that used in routine DTOD., the call-by-value counterpart. See the description of DTOD. for the error analysis.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b and a small error e" occurs in the exponent p, the error in the result is given approximately by:

b^p * (p/b*e' + log (b)*e").

The absolute error is approximately the absolute value of this expression. If the errors in the arguments are significant, the error in the result should be found by substitution of the possible argument values in the expression b^p.

DTOD.

DTOD. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements that

TABLE 2-17. RELATIVE ERROR OF DTANH.

Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minimum	Maximum
-2.	2.	3.011E-30	1.7353-29	-6.675E-29	7.436E-29
-30.	30.	1.640E-30	9.7603-30	-3.692E-29	2.544E-29

raise double-precision quantities to double-precision exponents. It accepts two double-precision arguments and returns a double-precision result.

Calls by value are computed at entry point DTOD..

METHOD

The input range is the collection of all argument sets (b,p) for which b and p are definite in-range double-precision quantities, and b is positive. If b is zero, then p is greater than zero, and b^p is in-range.

The formula used is:

$b^p = \exp(p * \log b)$

where $b > 0$. Upon entry, DLOG. computes log b, and DEXP. computes exp (p*log b). The result is returned to the calling program.

ERROR ANALYSIS

10 000 pairs of double-precision random numbers were generated, with distribution being the product of uniform distributions over (.5, 1.5) and (-10, 10). The error in the routine's computation of b^p was determined for each of these pairs. The maximum absolute value of the relative error in this routine for these 10 000 pairs was $2.977 * 10^{-25}$.

EFFECT OF ARGUMENT ERROR

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

$r * (\log b * e(p) + p * e(b)/b)$

DTOI*

DTOI* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements that raise double-precision quantities to fixed-point exponents. It accepts a double-precision argument and a fixed-point argument, and returns a double-precision result.

Calls by name are computed at entry point DTOI$.

METHOD

The argument set is checked upon entry. It is invalid if either argument is infinite or indefinite, or if the base is zero and the exponent is not greater than zero. If the argument set is invalid, a diagnostic message is issued, and POS.INDEF. is returned. Otherwise, DTOI. is called at entry point DTOI. for the computation. The result is returned to the calling program.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b, the error in the result will be given approximately by $n * b^{(n-1)} * e'$, where n is the exponent given to the routine.

DTOI.

DTOI. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements that raise double-precision quantities to fixed-point exponents.

Calls by value are computed at entry point DTOI..

METHOD

A b represents the base and a p represents the exponent. If p is non-negative and has the binary representation 000...0i(n) i(n-1)...i(1) i(0) where each i(j) $(0 \le j \le n)$ is 0 or 1, then:

$p = i(0)*2^0 + i(1)*2^1 + \dots i(n)*2^n$

and n = (log (2)p) = greatest integer not exceeding log (2)p. Then

$b^p = \mathrm{Prod}((b^2)^j : 0 \le j \le n \,\&\, i(j) = 1).$

The numbers $b = b^2$, b^2, $b^4, \dots, b^2$ are generated by successive squarings, and the coefficients i(0),...,i(n) are obtained as the sign bits of successive circular right shifts of p within the computer. A running product is formed during the computation, so that smaller powers of b and earlier coefficients i(j) may be discarded. Thus, the computation becomes an iteration of the algorithm:

$b^p = 1$ if $p = 0$.
$b^p = (b^2)^{p/2}$ if $p > 0$ and p is even.
$b^p = b * (b^2)^{(p-1)}/2$ if $p > 0$ and p is odd.

Upon entry, if the exponent p is negative, p is replaced by -p and b is replaced by 1/b; b is double-precision. For b = x(u)*x(l), 1/b = (1/b)(u)*(1/b)(l) is given in terms of x(u) and x(l) by the following formulas, where n is the normalization operation. The subscript l on one of the operations indicates that the coefficient of the result is taken from the lower 48 bits of the 96 bit result register, and the exponent is 48 less than the single-precision coefficient's exponent. The formulas are:

$$\begin{aligned}(1/b)(u) = {} & n(i/x\,u) + (((n(i-(1-(1/x(u))*x(u)) \\ & +(1-(1)(1/x(u)*x(u)) - (1/x(u)*(l)\,x(u)) \\ & -(1/x(u)*x(l)/x(u))) \\ & +(1/x(u)+(l)(((n(1-(1/x(u))*x*u) \\ & +(1-(l)(1/x(u))*x(u))) - (1/x(u))*(l)\,x(u) \\ & -(1/x(u))*x(l))/x(u)) \\ (1/b)(l) = {} & n(\dots) + (l)(\dots)\end{aligned}$$

In the routine, double-precision quantities x = x(u)*x(l) and y = y(u)*y(l) are multiplied according to:

$x*y = (x*y)(u)*y(l))$

where:

$$\begin{aligned}(x*y)(u) = {} & (((x(u)*y(l)) + (x(l)*y(u))) \\ & +(x(u)*(l)\,y(u))) + (x(u)*y(u))\end{aligned}$$

and

$(x*y)(l) = (((x(u)*y(l)) + (x(l)*y(u)))$
$+ (x(u)*(l)\ y(u))) + (l)(x(u)*y(u))$

The input range is the collection of pairs of arguments (b, p) for which $p \geq 0$ if b is zero, all quantities are definite and in-range, and the result is in-range.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b, then the error in the result is given approximately by $p * b^{(p-1)} * e'$, where p is the exponent. If the error e' is significant, the absolute error in the result is bounded above by:

$p * \max(|b|, |b + e'|)^{(p-1)} * e'$.

DTOX*

DTOX* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise double-precision quantities to floating-point exponents. It accepts a double-precision argument and a floating-point argument, and returns a double-precision result.

Calls by name are computed at entry point DTOX$.

METHOD

The argument set is checked upon entry. It is invalid if either argument is infinite or indefinite, if the base is zero and the exponent is not greater than zero, if the base is negative, or if arithmetic overflow occurs during computation. The result is calculated using:

$\text{base}^{\text{exponent}} = \exp(\text{exponent} * \log(\text{base}))$

If the argument set is invalid, POS.INDEF. is returned and a diagnostic message is issued. If the argument set is valid, the computed result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in DTOX* is the same as that used in DTOX.. See the description of routine DTOX..

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b and a small error e" occurs in the exponent p, the error in the result is given approximately by:

$b^p * (p/b * e' + \log(b) * e'')$

The absolute error is approximately the absolute value of this expression. If the errors in the arguments are significant, the error in the result should be found by substitution of the possible argument values in the expression b^p.

DTOX.

DTOX. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise double-precision bases to floating-point exponents. It accepts a double-precision argument and a floating-point argument, and returns a double-precision result.

Calls by value are computed at entry point DTOX..

METHOD

The input range is the collection of argument sets (b,p) for which b is a definite in-range double-precision quantity, p is a definite in-range floating-point quantity, and b is positive. If b is zero, then p is greater than zero, and b^p is in-range.

The formula used is:

$b^p = \exp(p * \log b)$

where $b > 0$. Upon entry, DLOG. is called to compute log b, and p * log b is computed in double-precision. DEXP. is called to compute exp(p * log b), and the result is returned to the calling program.

ERROR ANALYSIS

10 000 pairs (b,p) of random numbers were generated where b is double-precision and p is single-precision. The distribution was the product of uniform distributions in (.5, 1.5) and (0, 1). The maximum absolute value of the relative error in the routine for these pairs was $6.405 * 10^{-29}$.

EFFECT OF ARGUMENT ERROR

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

$r * (e(p) * \log b + p * e(b)/b)$

DTOZ*

DTOZ* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise double-precision quantities to complex exponents. It accepts a double-precision argument and a complex argument, and returns a complex result.

Calls by name are computed at entry point DTOZ$.

METHOD

If the base is real and the exponent is complex, then:

$\text{base}^{\text{exponent}} = X + i*Y$,

where:

X=exp(re(exponent)*log(base))*cos(im(exponent)*log(base))
Y=exp(re(exponent)*log(base))*sin(im(exponent)*log(base))

Upon entry the double-precision base is rounded to single-precision, and the resulting argument set is checked. The argument set is invalid if: either number is infinite or indefinite, the base is zero and the real part of the exponent is not positive, the base is negative, arithmetic overflow occurs during any stage of the computation, or precision is lost because the argument is too large. If the argument set is invalid, a diagnostic message is issued and POS.INDEF. is returned. Otherwise, the result of the computation is returned to the calling program.

ERROR ANALYSIS

The algorithm used in DTOZ* is the same as that used in DTOZ.. See the description of DTOZ..

EFFECT OF ARGUMENT ERROR

If e' and e" are small errors in the base b and exponent z, respectively, then the corresponding error in b^z is approximately ((z/b) * e' + e" * log (b)) * b^z. The absolute error will be approximately the absolute value of this. If e' or e" becomes significant, the error in the result should be calculated by substitution of the possible values of the arguments in the expression b^z.

DTOZ.

DTOZ. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise double-precision quantities to complex exponents. It accepts a double-precision argument and a complex argument, and returns a complex result.

Calls by value are computed at entry point DTOZ..

METHOD

The input range is the collection of argument sets (b,z) where b is a definite in-range double-precision quantity, z is a definite in-range complex quantity, b is greater than zero, and b^z and $b^{\bar{z}}$ are in-range.

The formula used is:

$$b^{(u+i*v)} = \exp(u*\log b) * \cos(v*\log b) + i * \exp(u*\log b) * \sin(v*\log b)$$

where b > 0. Upon entry, the lower half of the double-precision base b is discarded, and ALOG. is called to compute log b. EXP. is called to compute exp (u*log b), and COS.SIN is called to compute cos (v*log b) and sin (v*log b), where u + i * v is the exponent. The result is computed from the formula, and is returned to the calling program.

ERROR ANALYSIS

10 000 pairs (b,z), where b is double-precision and z is complex, were generated with distribution being the product of uniform distributions over the intervals (.5, 1.5), (-10, 10), and (-2.pi, 2.pi). The maximum modulus of the relative error in the routine was found to be 5.605 * 10^{-14}.

EFFECT OF ARGUMENT ERROR

If a small error e (b) occurs in the base b and a small error e (z) occurs in the exponent z, the error in the result w is given approximately by:

w * (e (z) * log b + z * e (b)/b)

ERF.

ERF. is an external function which accepts calls from FORTRAN code. It computes the error function and the complementary error function (FORTRAN function names ERF and ERFC). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points ERF and ERFC, and calls by value are computed at entry points ERF. and ERFC..

METHOD†

The input range is the collection of all definite floating-point quantities (including out-of-range values INF) except the range (25.92277515027854,+INF) for ERFC, which underflows.

The routine calculates the smaller of erf (abs (x)), erfc (abs (x)), and uses the identities:

erf (-x)=-erf (x)
erf (x)=1-erfc (x)

to compute the final value, which is the sum of a signed function and a constant.

The forms used are given in table 2-18.

where the constants .477 and 25.9 are inverse erf (0.5) and inverse erfc (2^{-975}), which are approximately 0.47693627620447 and 25.92277515027854.

The function p1 is a (5th order odd)/(8th order even) rational form. The functions p2, p3 are exp ($-x^2$) * (rational form), where p2 is (7th order)/(8th order) and p3 is (4th order)/(5th order). Since exp ($-x^2$) is ill-conditioned for large x, exp ($-x^2$) is calculated by exp (u+eps)= exp (u)+eps*exp (u), where u=$-x^2$ upper and eps=$-x^2$ lower.

ERROR ANALYSIS

The large error in p2 and p3 is due to the large size of the rational forms and the additional error in exp ($-x^2$). The polynomials in p2 and p3 are stable, but not as accurate as

TABLE 2-18. FORMS USED IN ERF. (y=ABS(x))

Range	ERF	ERFC
(-INF,-5.625)	-1.0	+2.0
(-5.625,-.477)	-1.0+p2(y)	+2.0-p2(y)
(-.477,0)	-p1(y)	+1.0+p1(y)
(0,+.477)	+p1(y)	+1.0-p1(y)
(.477,5.625)	+1.0-p2(y)	p2(y)
(5.625,8.0)	+1.0	p2(y)
(8.0,25.9)	+1.0	under flow
+INF	+1.0	+0.0

† The coefficients for p2 and p3 are from Hart, Cheney, Lawson, et al., Computer Approximations.

most exponential-type approximations, which, when evaluated using Horner's rule, add the smallest terms first. Inverting x and reversing coefficients does not improve accuracy because of the error involved in division.

The maximum error in the approximations p1, p2, p3, scaled by 10^{15}, is given in table 2-19.

In regions where a constant is added, that constant dominates and the error is less than that shown.

Figures 2-23 and 2-24 show the mean relative errors for ERF. and ERFC..

EFFECT OF ARGUMENT ERROR

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(-x^2) and that of relative error is (2/sqrt(pi))*x*exp(-x^2)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC when $x < 0.53$. For $x > 0.53$ the relative error for ERFC is amplified by approximately 2x.

If the value of x is known to more than single-precision, the following FORTRAN code may be used to compute an accurate value of ERFC when x is large:

```
DOUBLE X
DATA SQRTPI / < 2/sqrt (pi) > /
       .
       .
       .
(compute X)
SNGLX=SNGL (X)
SHSNGLX=SNGL (X-SNGL (X) )
Y=ERFC (SNGLX)+SHSNGLX+SQRTPI*EXP (-SNGLX**2)
       .
       .
       .
(Y is ERFC (X) )
```

EXP

EXP is an external function which accepts calls from FORTRAN code. It computes the exponential function (FORTRAN function name EXP). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point EXP, and calls by value are computed at entry point EXP..

METHOD†

The input range to this routine is the collection of all definite in-range floating-point quantities lying in the interval (-675.84,741.67). Upon entry, the argument x is

TABLE 2-19. MAXIMUM ERROR OF ERF.

Source Of Error	p1	p2	p3
Rational form	1.1	4.9	1.7
Coefficient rounding	0.5	0.8	1.4
Round-off	14.7	110	68
Upper bound	16.3	116	71
Maximum observed	12.8	27.9	28.3

multiplied by 16./log(e)2. in double-precision, and the integral (n) and fractional (u) parts computed. The range reduction formula used here is:

$$\begin{aligned}\exp(x) &= 2^{(x/\log 2)}\\ &= (2^{1/16})^{(16*x/\log 2)}\\ &= (2^{1/16})^{n} * (2^{1/16})^{n}\end{aligned}$$

If $n = 16 * q + r$ where q and r are integers such that $0 \leq r \leq 16$, exp(x) is finally given by:

$$\exp(x) = 2^{q} * (2^{1/16})^{r} * (2^{1/16})^{u}$$

and q will be added to the exponent of the result. Then $(2^{1/16})^{r}$ is obtained from a look-up table, and $(2^{1/16})^{u}$ is obtained from the following approximation:

$$(2^{1/16})^{n} = u + 2 * \frac{u * (p(00) + p(01) * u^2)}{(q(00) + u^2) - u * (p(00) + p(01) * u^2)}$$

where the constants are given by:

q(00) = 20.8137711965230361973 * 256
p(00) = 7.2135034108448192083 * 16
p(01) = .057761135831801928 / 16

ERROR ANALYSIS

The maximum absolute value of the error of approximation of the algorithm is 5.000 * 10^{-17} over the interval (-(log2)/16,(log2)/16). A graph of the error of approximation in the algorithm is given in figure 2-23. An upper bound for the absolute value of the error due to machine round-off is 1.868 * 10^{-14} over the interval ((-log 2)/16,(log 2)/16). Hence an upper bound on the absolute value of the error in the routine over this interval is 1.873 * 10^{-14}. A bound on the routine's error for any given argument x can be obtained by employing the multiplication formula for exp:

$$\exp(x + y) = \exp(x) * \exp(y)$$

The maximum absolute value of the relative error of approximation of the algorithm over (-log2/16, log2/16) is 4.838 * 10^{-17}. An upper bound on the absolute value of the relative error due to machine round-off and truncation is 6.890 * 10^{-15} over ((-log 2)/16,(log 2)/16). So an upper bound on the absolute value of the relative error is 6.938 * 10^{-15} over the interval ((-log 2)/16,(log 2)/16).

For 10 000 arguments chosen randomly from given intervals, statistics on relative error were observed. These are given in table 2-20.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument, the error in the result y is given approximately by y * e'.

HYP.

HYP. is an external function which accepts calls from FORTRAN code. It computes the hyperbolic sine and cosine functions (FORTRAN function names SINH and

† This approximation is described in Hart, Cheney, Lawson, et al., Computer Approximations, (New York) 1968, John Wiley and Sons, pp. 96-104.

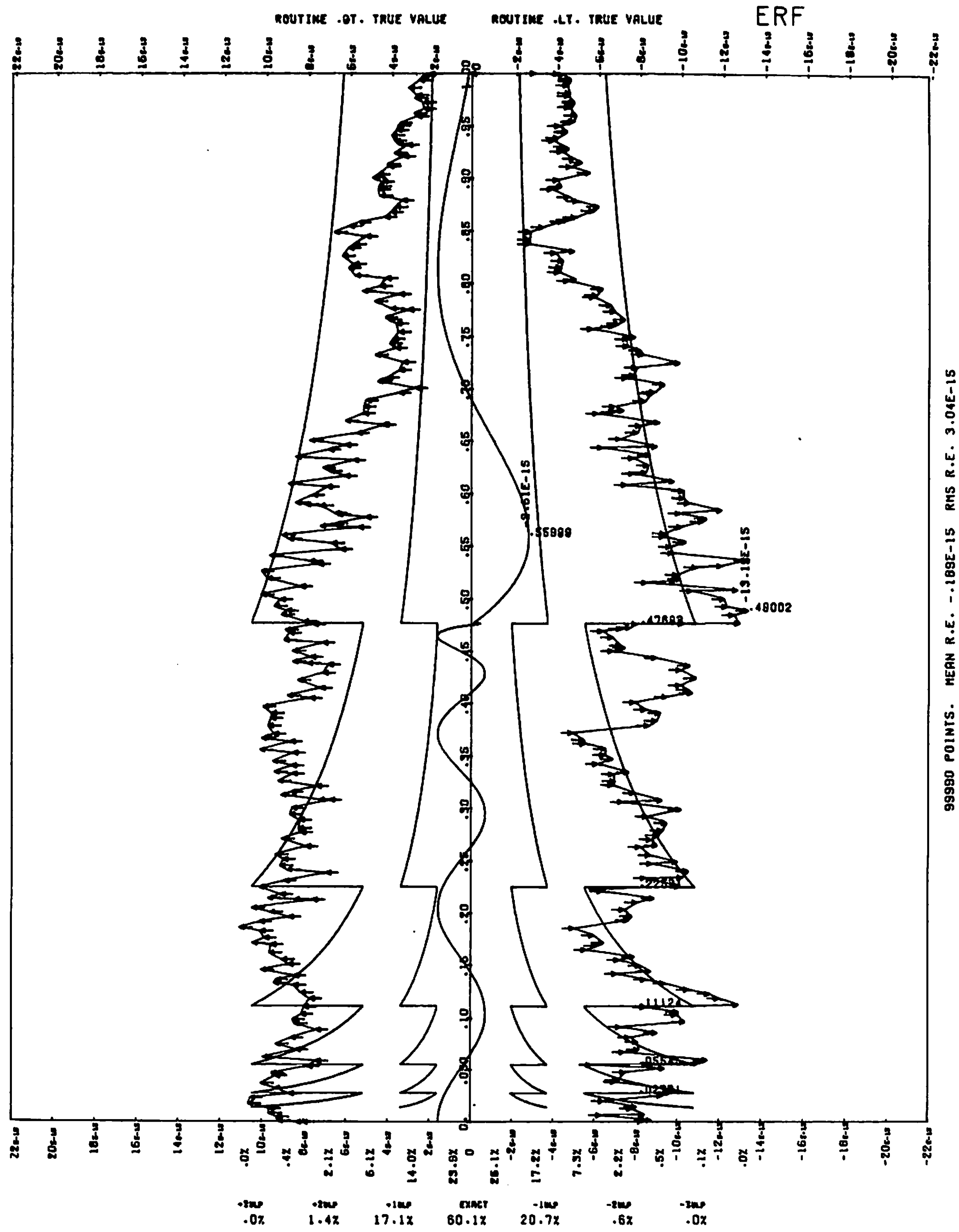

Figure 2-23. Mean Relative Error of ERF.

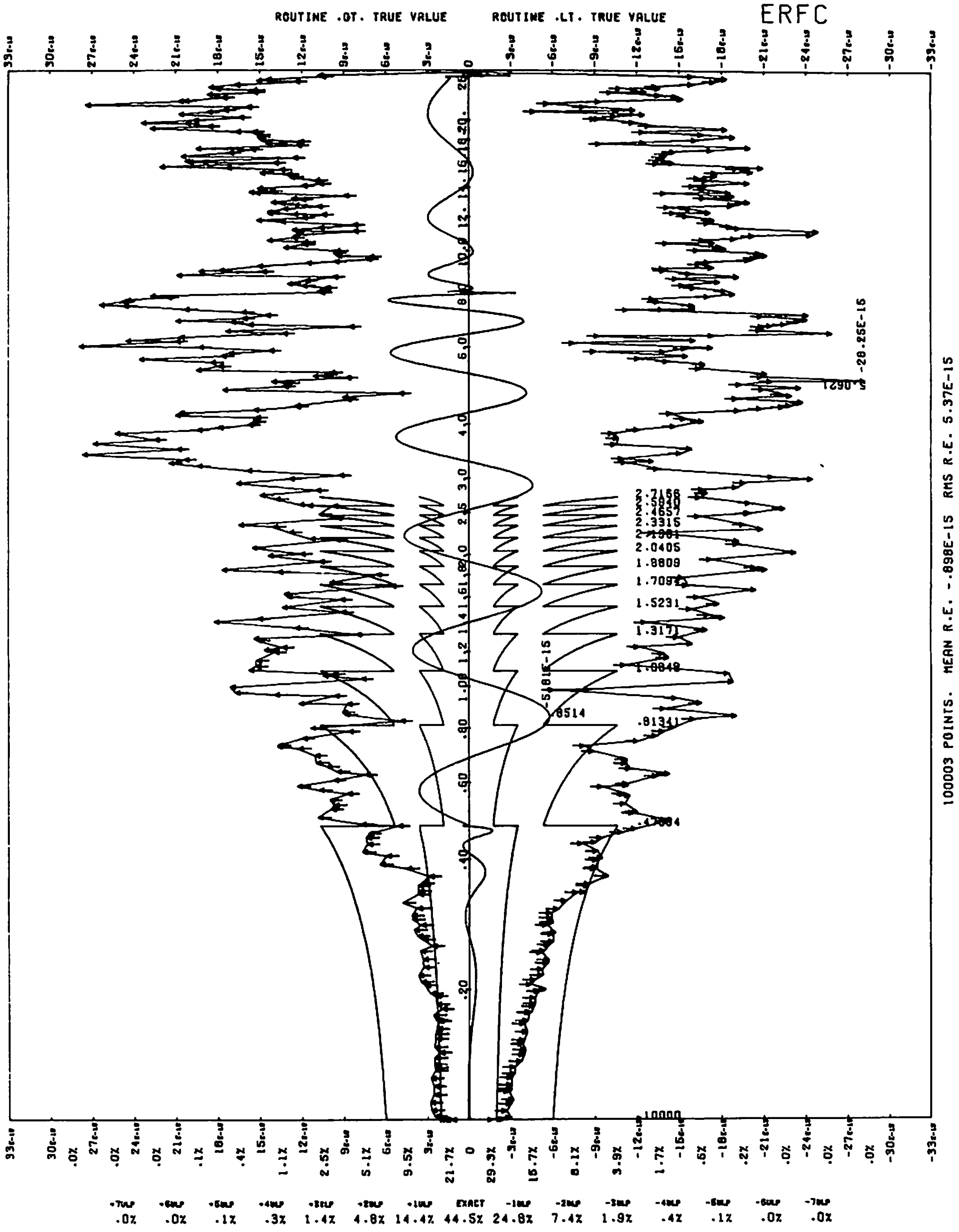

Figure 2-24. Mean Relative Error of ERFC.

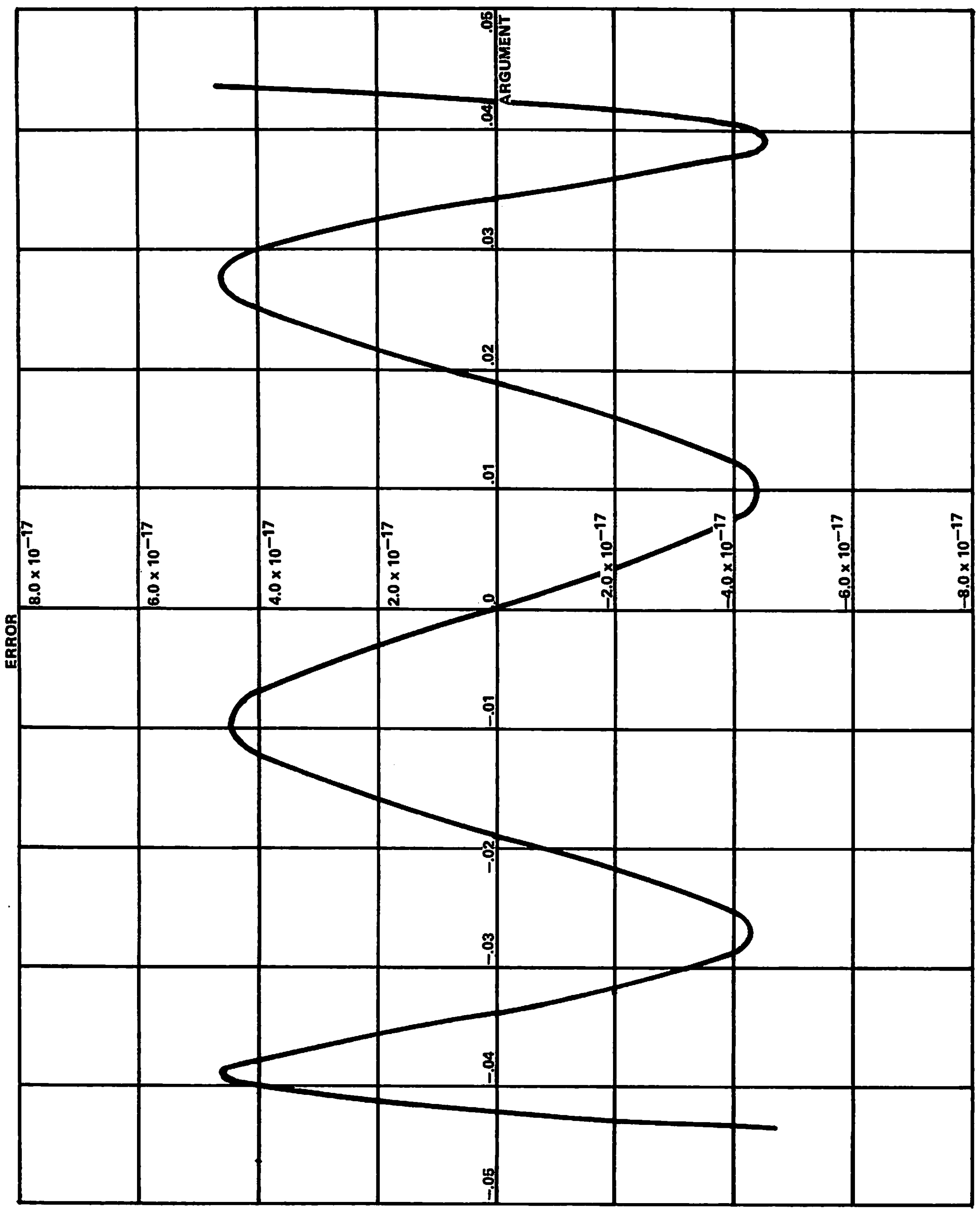

Figure 2-25. Algorithm Error of EXP.

TABLE 2-20. RELATIVE ERROR OF EXP

Interval		Mean	Standard Deviation	Minimum	Maximum
From	To				
-673.	741.	-3.012E-16	2.181E-15	-6.887E-15	5.193E-15
-1.	1.	-3.100E-16	2.223E-15	-6.769E-15	5.028E-15

COSH). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points SINH and COSH, and calls by value are computed at entry points SINH. and COSH..

METHOD

The input range is the collection of all definite in-range, floating-point quantities lying within the interval:

(-1071*log(2),1071*log(2)) = (-742.3606303797,742.3606303 797)

The formulas used to compute sinh(x) and cos(x) are:

$x = n*\log 2 + a, \ |a| \leq 1/2*\log 2,$
$\cosh(x) = 2^{(n-1)} * \cosh(a) + 2^{(n-1)} * \cosh(a) + 2^{(n-1)} * \sinh(a) - 2 * \sinh(a)$
$\sinh(x) = 2^{(n-1)} * \sinh(a) + 2^{(n-1)} * \sinh(a) + 2^{(n-1)} * \cosh(a) - 2^{(n-1)} * \cosh(a)$
$\cosh(a) = 1 + d(a)$
$\sinh(a) = a+a^3*(s(3)+a^2*(s(5)+\underline{b}/(\underline{a}-a^2)))$
$d(a) = a^2*(1/2+a^2*(c(4)+a^2*(c(6)+a^2*(c(8)+a^2)*c(10))))$

where:

$s(3) = .16666666666693558$
$s(5) = -.005972995665652368$
$\underline{b} = 1.031539921161$
$\underline{a} = 72.10374670722$
$c(4) = .04166666666488081$
$c(6) = .0013888888952318045$
$c(8) = 89.75473897315022$
$c(10) = 2.763250805803 * 10^{-7}$

In the following description of the algorithm used, X1 = x = argument on entry; entry is at SINH. or COSH.; and on exit, X6 = result.

a. If $|x| \geq 1071*\log(2)$, go to step j.

b. u = x .
   v = +0 if $x \geq 0$.
   -0 if $x < 0$.

d. n = (u/log 2+.5) = nearest integer to u/log 2.
   w = u - n*log 2, where the right-hand expression is evaluated in double-precision.

e. $s = w+w^3(s(3)+w^2(s(5)+\underline{b}/(\underline{a}-w^2)))$.
   $d = w^2(1/2+w^2(c(4)+w^2(c(6)+w^2(c(8)+w^2)*c(10))))$.
   $a = (1+d-s)*2^{(n-1)}$.
   b = d+s.

f. If COSH. entry, go to step h.

g. $c = (1/4 + (1/4+b))*2^{(n-1)} + (2^{(n-3)} + (2^{(n-3)} - a))$.
   X6 = c with the sign stored in v.
   Go to step i.

h. $c = (1+b)*2^{(n-1)} + a$.
   X6 = c.

i. Return.

j. If infinite or indefinite argument, go to step l.

k. Normalize argument.
   u = |x|.
   v = +0 if $x \geq 0$.
   -0 if $x < 0$.
   If $|x| < 1071*\log 2$, go to step d.

l. Initiate error processing.

m. X6 = POS.INDEF. if x is indefinite.
   POS.INF. if x is infinite or too big, and positive, or COSH.
   NEG.INF. if x is infinite or too big, and negative,
   and SINH.

n. Go to step i.

ERROR ANALYSIS

The maximum absolute value of relative error in the approximation of sinh over (-log 2/2, log 2/2) is $1.282 * 10^{-15}$, and of cosh over (-log 2/2, log 2/2) is $2.421 * 10^{-16}$. Computed upper bounds on the absolute value of relative error due to machine error in the computation of sinh is $2.392 * 10^{-14}$ and of cosh is $1.024 * 10^{-14}$. Hence, upper bounds on the absolute value of relative error in the routine is $2.520 * 10^{-14}$ for sinh, and $1.048 * 10^{-14}$ for cosh. Graphs of the relative errors in the algorithms used to approximate sinh and cosh over (-log 2/2, log 2/2) are given in figures 2-26 and 2-27.

EFFECT OF ARGUMENT ERROR

If a small error u occurs in the argument x, the resulting error in sinh(x) is given approximately by cosh(x)*u, and the resulting error in cosh(x) is given approximately by sinh(x)*u. If the error u is not small, the addition formulas for sinh and cosh should be used to find the resulting error:

sinh(x+u) = sinh(x)cosh(u)+cosh(x)sinh(u)
cosh(x+u) = cosh(x)cosh(u)+sinh(x)sinh(u)

HYPERB.

HYPERB. is an auxiliary routine which accepts calls from CCOS, CSIN, and CSNCS.. It performs incidental computation of the hyperbolic sine and cosine functions. It accepts a floating-point argument and returns two floating-point results.

Calls by value are computed at entry point HYPERB..

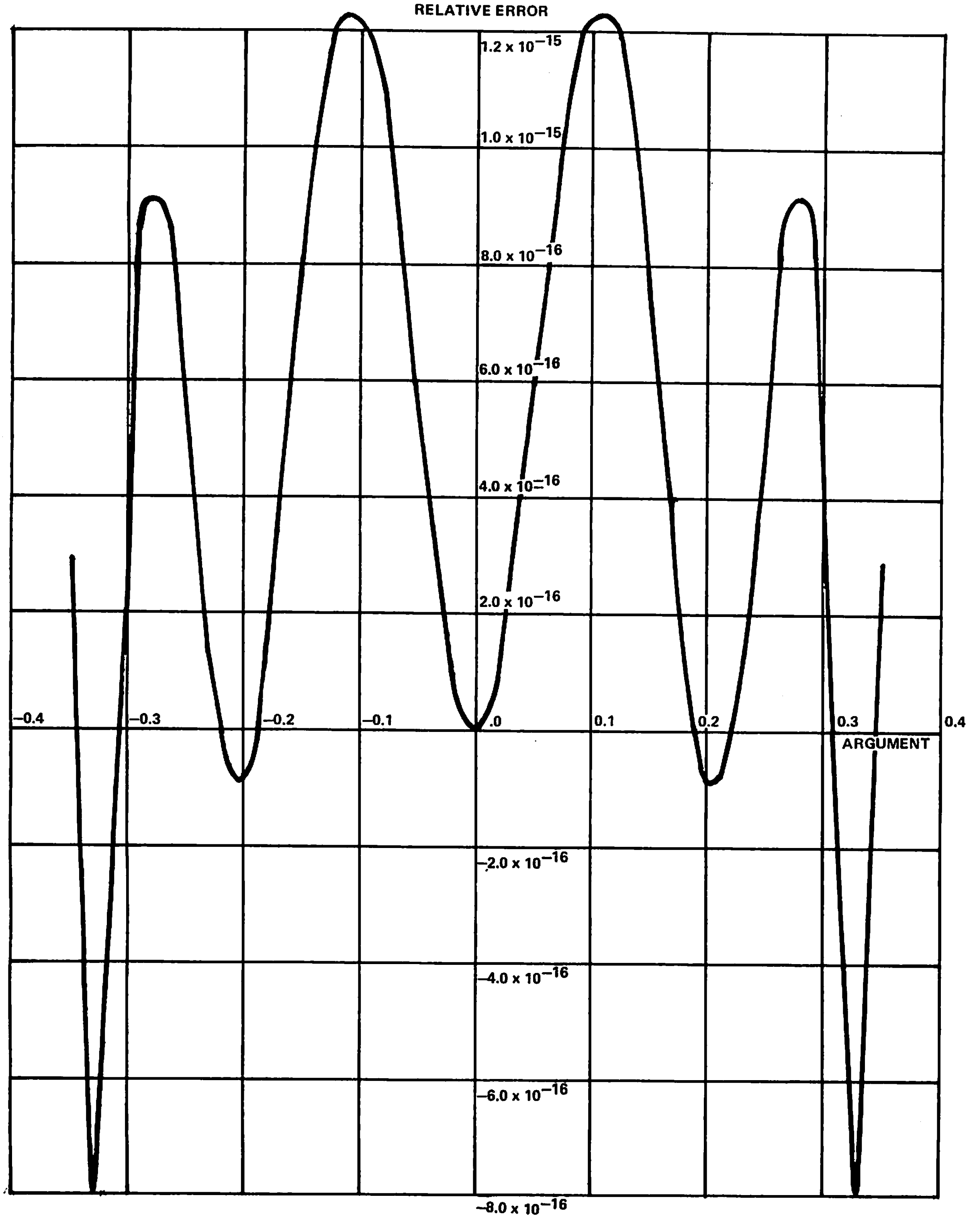

Figure 2-26. Relative Error of HYP. for Sinh

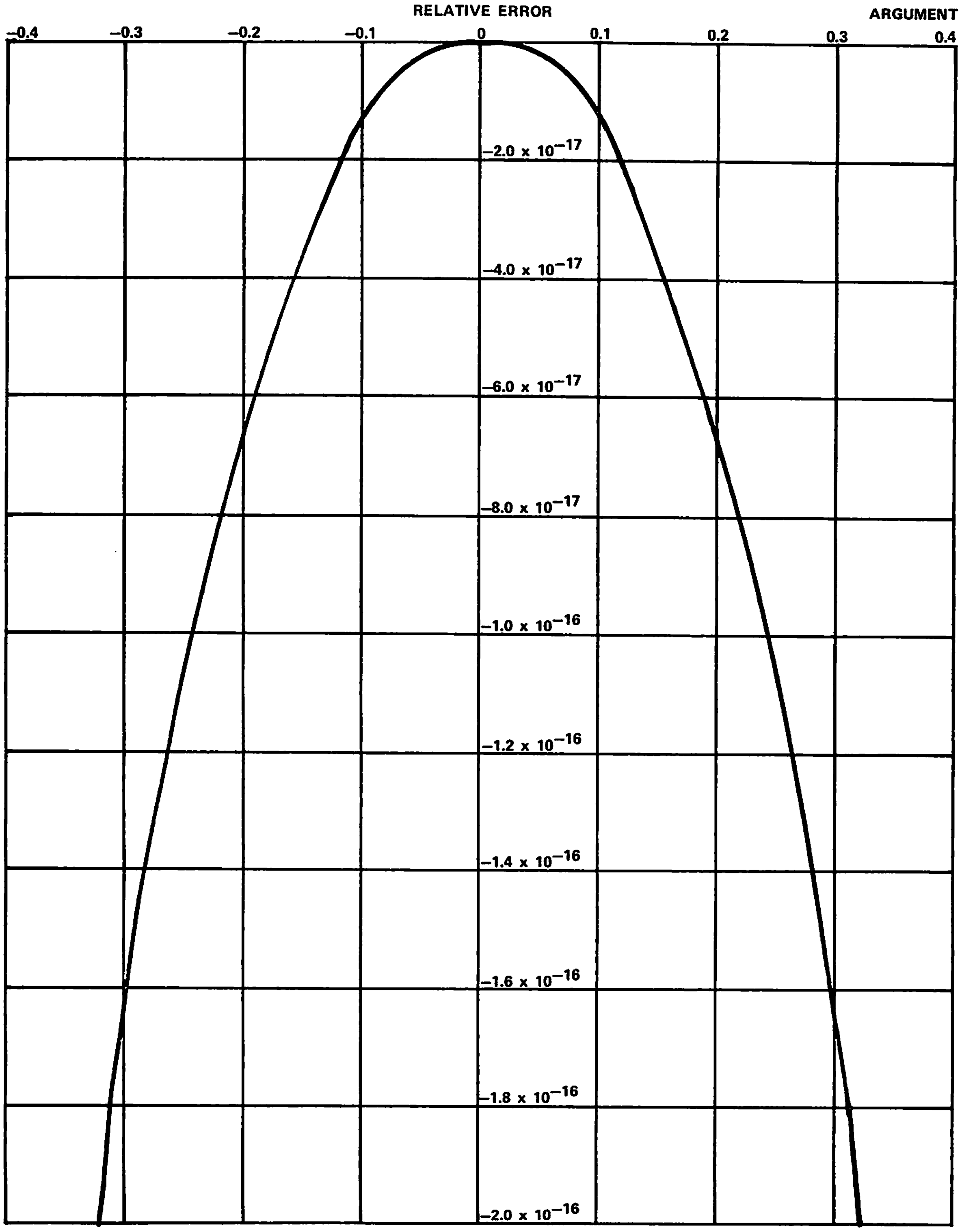

Figure 2-27. Relative Error of HYP. for Cosh

METHOD

The input range is the collection of all definite in-range floating-point quantities which lie in the interval (-741.67, 741.67). The hyperbolic cosine is computed by:

$$\cosh(x) = .5 * (\exp(x) + \exp(-x))$$

If $|x| \geq .22$, the hyperbolic sinh is computed by:

$$\sinh(x) = .5 * (\exp(x) - \exp(-x))$$

For $|x| < .22$, the MacLaurin series for sinh is truncated after the term $x^9/9!$ and the resulting polynomial is taken as approximation:

$$\sinh(x) \quad x + x^3/3! + x^5/5! + x^7/7! + x^9/9!$$

ERROR ANALYSIS

The maximum absolute value of the error of approximation for cosh(x) is $5.000 * 10^{-17}$ and for sinh(x) is $1.464 * 10^{-15}$, over the interval (-log 2, log 2). See the description of EXP. for details concerning the error of approximation to exp. An upper bound for the error due to machine round-off and truncation during computation of the MacLaurin polynomial is $8.198 * 10^{-16}$. A graph of the error of approximation in the polynomial for sinh is given in figure 2-28. An upper bound for the routine's error in the computation of cosh(x) is $7.184 * 10^{-14}$, and in the computation of sinh(x) is $7.148 * 10^{-14}$ over (-log 2, log 2).

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the resulting error in cosh(x) is given approximately by sinh(x)*e', and the resulting error in sinh(x) is given approximately by cosh(x)*e'.

ITOD*

ITOD* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise fixed-point quantities to double-precision exponents. It accepts a fixed-point argument and a double-precision argument, and returns a double-precision result.

Calls by name are computed at entry point ITOD$.

METHOD

The formula used to perform the exponentiation is:

$$\text{base}^{\text{exponent}} = \exp(\text{exponent} * \log(\text{base})).$$

Upon entry, the fixed-point argument is converted to double-precision and the resulting argument set is checked. The argument set is invalid if: the base is zero and the exponent is not greater than zero, the base is negative, either argument is infinite or indefinite, or floating overflow occurs during the computation. If the base is zero and the exponent is negative, NEG.INF. is returned. If the argument set is otherwise invalid, POS.INDEF. is returned. In all cases, if the argument set is invalid, a diagnostic message is issued. If the argument set is valid, the result is computed and returned to the calling program.

ERROR ANALYSIS

The algorithm used in ITOD* is the same as that used in ITOD.. See the description of routine ITOD..

EFFECT OF ARGUMENT ERROR

If a small error occurs in the double-precision exponent, the resulting error in the result is approximated by multiplying the argument error by the result, and then by the natural logarithm of the base. Thus, if the result is large, the effect of an argument error will be large. If the error in the argument becomes significant, the error in the result should not be calculated by this rule, but should be calculated from the function values.

ITOD.

ITOD. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise fixed-point quantities to double-precision exponents. It accepts a fixed-point argument and a double-precision argument, and returns a double-precision result.

Calls by value are computed at entry point ITOD..

METHOD

The input range is the collection of all argument sets (b,p) where b is a definite in-range fixed-point quantity, p is a definite in-range double-precision quantity, b is greater than zero, and b^p is in-range. Upon entry, b is normalized and converted to double-precision.

The formula used to compute the result is:

$$b^p = \exp(p * \log b)$$

DLOG. is called to compute log b, then p*log b is computed in double-precision. DEXP. is called to compute exp(p*log b), and the result is returned to the calling program.

ERROR ANALYSIS

10 000 random argument sets (b,p) were generated, with distribution being the product of a discrete uniform distribution over the integers 1,2,...,9, and a uniform distribution over (-1, 1). The relative error in the routine was computed for each of the argument sets. The maximum absolute value of the relative error in the routine was $2.466 * 10^{-28}$.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the exponent, the error in the result r is given approximately by r*e'*log b, where b is the base.

ITOJ*

ITOJ* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise fixed-point quantities to fixed-point exponents. It accepts two fixed-point arguments and returns a fixed-point result.

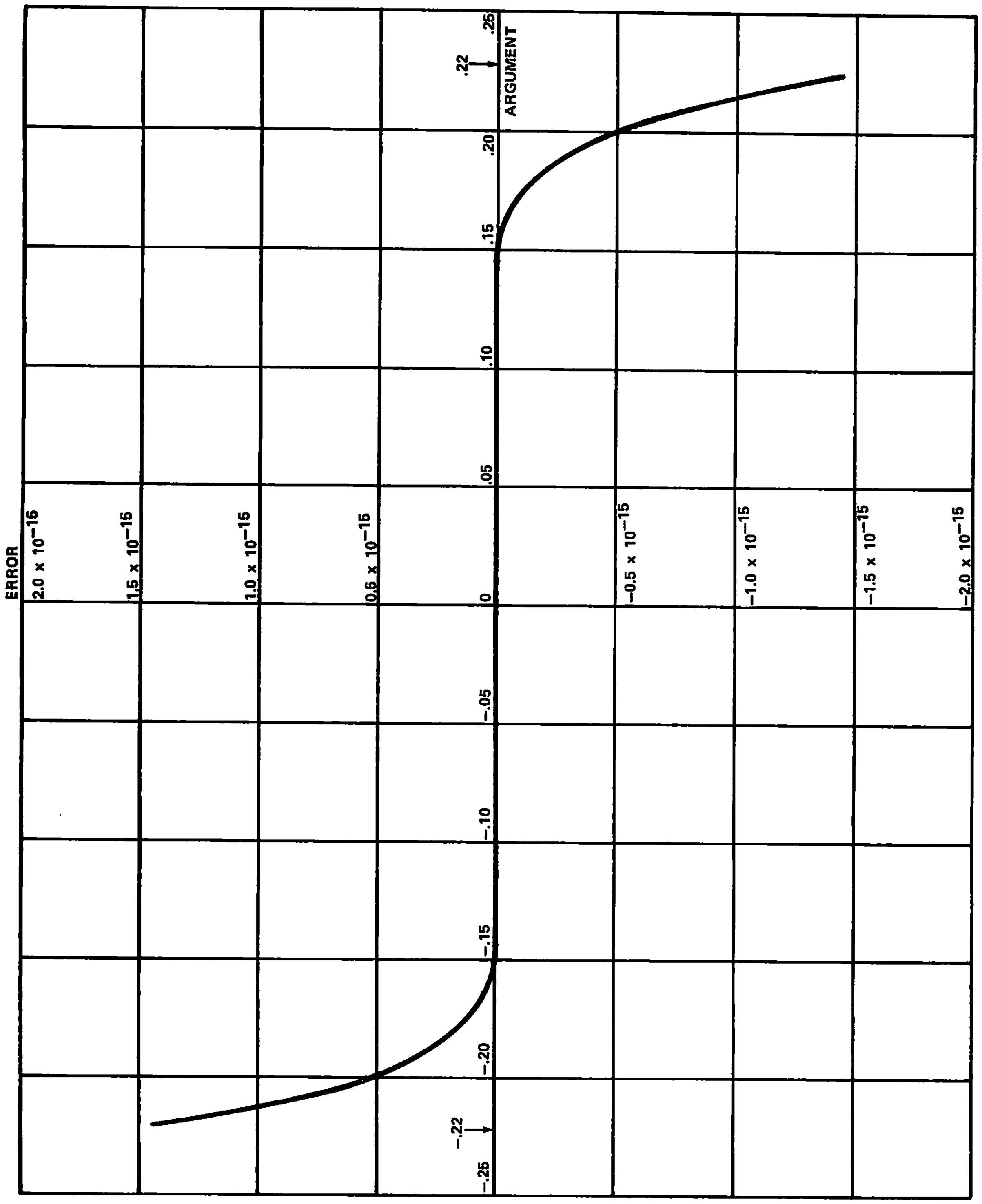

Figure 2-28. Error of HYPERB. for Sinh

Calls by name are computed at entry point ITOJ$.

METHOD

A b represents the base and a p represents the exponent. If p has binary representation 000....000 i(n)i(n-1) ...i(i)i(0) where each $i(j)\ (0 \leq j \leq n)$ is 0 or 1, then:

$p = i(0)*2^0 = i(1)*2^1 +...+ i(n) * 2^n$
$n = (\log(2)p)$ = greatest integer not exceeding $\log(2)p$.

Then:

$b^p = \text{Prod}\,(b^2\ : 0 \leq j \leq n \text{ and } i(j) = 1)$.

The numbers $1 = b^0$, $b = b^1, b^2, b^4, \ldots, b^{(2)^{(\log(2)p)}}$ are generated during the computation by successive squarings, and the coefficients i(0),....,i(n) are generated by sign tests of successive right shifts of p within the computer. A running product is formed during the computation, so that smaller powers of b can be discarded. The computation then becomes an iteration of the algorithm:

$b^p = b$ if $p = 1$
$= (b*b)^{(p/2)}$ if p is even
$= (b*b)^{((p-1)/2)}*b$ if b is odd.

Upon entry, the base is converted to floating-point, and the result of the computation will be later converted to fixed-point for return. The argument set is invalid if the base is zero and the exponent is zero or negative, or if integer overflow occurs during the computation. If the argument set is invalid, zero is returned and a diagnostic message is issued. If the base is nonzero and the exponent is negative, 1, -1, or 0 will be returned as the base. The result of the computation is returned to the calling program.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

ITOJ.

ITOJ. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise fixed-point quantities to fixed-point exponents. It accepts two fixed-point arguments and returns a fixed-point result.

Calls by value are computed at entry point ITOJ..

METHOD

The arguments are checked to determine if the exponentiation conforms to a special case. If it does, the proper value is immediately returned, or if the special case is an error condition, an error message is issued. The special cases are:

0^0 = error
0^J = error if $J < 0$
$-0^1 = +0$
$1^J = 1$
$-1^J = +1$ or -1 (J even or odd)
$I^0 = 1$
$I^J = 0$ if $J < 0$
$I^2 = I*I$
I^J = error if $I \geq 2$ and $J \geq 64$
I^J = error if $I \geq 2^{16}$ and $J \geq 3$

If the exponentiation does not fit any special case, the following algorithm is used:

Variable b represents the base and p represents the exponent. If p is non-negative and has the binary representation 000...00i(n)i(n-1)...i(i)i(0), where each $i(j)\ (0 \leq j \leq n)$ is 0 or 1, then:

$p = i(0)*2^0 = i(1)*2^1 + i(2)*2^2 +....+ i(n)*2^n$

While p is even evaluate:

$b = b^2$, $p = p/2$

$r = b$.

While $p > 1$ evaluate:

$r = r^2$,
if p is odd then $r = r * b$,
$p = p/2$

Now r contains the result. Floating-point is used for r so that the remaining overflows could be detected by examining the final exponent.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

ITOX*

ITOX* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise fixed-point quantities to floating-point exponents. It accepts a fixed-point argument and a floating-point argument, and returns a floating-point result.

Calls by name are computed at entry point ITOX$.

METHOD

Upon entry, the base is converted to floating-point, and the argument set is checked. The argument set is invalid if: either argument is infinite or indefinite, the base is negative, the base is zero and the exponent is not greater than zero, or floating overflow occurs during the calculation. If the base is zero and the exponent is negative, or if floating overflow occurs, POS.INF. is returned. If the argument set is otherwise invalid, POS.INDEF. is returned. In any case, if the argument set is invalid, an appropriate diagnostic message is issued. If the argument set is valid, the result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in ITOX* is the same as that used in ITOX.. See the description of ITOX..

EFFECT OF ARGUMENT ERROR

If a small error occurs in the floating-point exponent, the error in the result is approximated by multiplying the argument error by the result and then by the natural logarithm of the base. Thus, if the result is large, the effect of an error in the exponent is large.

ITOX.

ITOX. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise fixed-point quantities to floating-point exponents. It accepts a fixed-point argument and a floating-point argument, and returns a floating-point result.

Calls by value are computed at entry point ITOX..

METHOD

The input range is the collection of all argument sets (n,x) such that n is a fixed-point quantity, x is a definite in-range floating-point quantity, x is positive and nonzero whenever n is zero, and n^x is in-range.

The formula used is:

$n^x = \exp(x * \log n)$

where $n \geq 1$.

Upon entry, n is packed and normalized. Zero is returned if the base is zero. Otherwise, ALOG. is called to compute log n, and EXP. is called to compute exp(x * log n). the result is returned to the calling program.

ERROR ANALYSIS

500 000 pairs (n,x) of random numbers were generated. The distribution was the product of a discrete form of the right half of a Cauchy distribution, and a Cauchy distribution. n^x was computed for each of these pairs, first using the routine, and then using the double-precision routine. The maximum absolute value of the relative error in the routine was $3.929 * 10^{-12}$ for the 500 000 pairs.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the exponent x, the error in the result r is given approximately by r * e' * log n, where n is the base.

ITOZ*

ITOZ* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements that raise fixed-point quantities to complex exponents. It accepts a fixed-point argument and a complex argument, and returns a complex result.

Calls by name are computed at entry point ITOZ$.

METHOD

If n is a positive integer, and x and y are real, then:

$$n^{(x + i*y)} = \exp(x*\log(n))*\cos(y*\log(n)) + i*\exp(x*\log(n))*\sin(y*\log(n))$$

Upon entry, the argument set is checked. It is invalid if: the first argument is negative or zero, either argument is infinite or indefinite, floating-point overflow occurs during the calculation, or x*log r is greater than 741.67. If the argument set is invalid, a diagnostic message is issued and POS.INDEF. is returned. Otherwise, the computation proceeds and the result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in ITOZ* is the same as that used in ITOZ.. See the description of ITOZ..

EFFECT OF ARGUMENT ERROR

If a small error occurs in the argument, the error in the result is approximated by the product of the argument error, the result, and the natural logarithm of the base. The absolute value of the error in the result is given approximately by the product of the corresponding absolute values. If the argument error is significant, the error in the result should be found from substitution of the possible argument values in the function.

ITOZ.

ITOZ. is an exponentiation routine which accepts calls from compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise fixed-point quantities to complex exponents. It accepts a fixed-point argument and a complex argument, and returns a complex result.

Calls by value are computed at entry point ITOZ..

METHOD

The input range is the collection of all argument sets (n,z) comprising a fixed-point quantity n and a complex quantity z; z is definite and in-range, and if n is zero, z is a positive nonzero real. Also, im(z) * log n must not exceed $pi*2^{46}$, where $n > 0$ and im(z) is the imaginary part of z, and the real number $n^{re(z)}$ must be in-range.

Upon entry, the fixed-point argument is packed and normalized, and routine XTOZ. is called at entry XTOZ. to compute the result. The result is returned to the calling program.

ERROR ANALYSIS

300 000 pairs (n,z) of random numbers were generated with distribution being the product of a discrete form of the right half of a Cauchy distribution, and the product of two Cauchy distributions. For each of these pairs, n^z was computed, first using the routine, and then using double-precision operations. The maximum absolute value of the relative error in the routine was $3.054 * 10^{-10}$ for these pairs.

EFFECT OF ARGUMENT ERROR

If a small error e(z) = e(x) + i*e(y) occurs in the exponent z, the error in the result w is given approximately by w * log n * e(z).

RANF

RANF is an external function which accepts calls from FORTRAN code. It computes random numbers (FORTRAN function name RANF). It accepts a dummy argument and returns a floating-point result.

Calls by name are computed at entry points RANF and RANGET.

METHOD

RANF uses the multiplicative congruential method modulo 2^{48}. The formula is:

$x(n+1) = a * x(n) \pmod{2^{48}}$

The library holds a random seed, RANDOM., and a multiplier, RANMLT.. The random seed can be changed to any value prior to calling RANF by use of the routine RANSET. Upon entry at RANF, the random seed is multiplied by RANMLT. to generate a 96 bit product, and the lower 48 bits become the new random seed. The seed is used to generate subsequent random numbers. RANDOM. has a default initial value of 1717 1274 3214 7741 3155B (241^{463} mod 2^{47}). This new random seed is normalized and returned as the random number.

The multiplier, RANMLT., is constant, and has a value of 2000 1207 2642 7173 0565B. This multiplier passes the Coveyou-MacPherson test and other statistical tests for randomness, including the auto-correlation test with lag $\leq$ 100 and the pair triplet test.†

If RANF is called by name at entry point RANGET, the current seed of the random number generator is returned in the variable whose address is in X1.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

RANSET

RANSET is a subroutine which accepts calls from FORTRAN code. It resets the seed of the random number generator (FORTRAN subroutine name RANSET). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point RANSET.

METHOD

The call supplied the address of a new seed value in X1. If the new seed is 0., the new seed value is made 17171274321477413155_8, which is .17099839404402317200. Otherwise, the coefficient of the new seed is made odd if necessary by adding 1_8, and the exponent of the new seed value is set equal to 1717_8, which is -48.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

SINCOS.

SINCOS. is an external function which accepts calls from FORTRAN code. It computes the trigonometric sine and cosine functions (FORTRAN function names SIN and COS). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points SIN and COS, and calls by value are computed at entry points SIN. and COS..

METHOD††

The input range to this routine is the collection of all definite in-range normalized floating-point quantities whose absolute values do not exceed pi * 2^{46}.

Upon entry, the range reduction:

$y = 2/\text{pi}*x - n$

is performed in double-precision, where x is the argument, n is an integer, and y is in (-1/2, 1/2). Depending upon the sign of x and n(mod 4), the result may be complemented, and a polynomial approximation (p(y) or q(y)) is chosen to give the result. The polynomial approximations p(y) and q(y) are:

$p(y) = \text{pi}/2*y - y^3*(s(0)+s(1)*y^2+s(2)*y^4+s(3)*y^6+s(4)*y^8)^2$

and

$q(y) = 1 - y^2*(c(0)+c(1)*y^2+c(2)*y^4+c(3)*y^6+c(4)*y^8)^2$

The coefficients are:

$s(0) = 8.03718916976708 * 10^{-2}$
$s(1) = -4.95774235001375 * 10^{-2}$
$s(2) = 1.38346449783347 * 10^{-3}$
$s(3) = -1.44725130681196 * 10^{-5}$
$s(4) = 1.54733311005155 * 10^{-7}$
$c(0) = 1.110720734539535$
$c(1) = 1.14191398434002 * 10^{-2}$

† D. E. Knuth, The Art of Computer Programming, Vol. 2.

†† The algorithm and constants are copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455, and are employed under license. Coding is by Larry Liddiard, University of Minnesota.

c(2) = -3.521949713998275 * 10^{-3}
c(3) = 5.172606069276518 * 10^{-5}
c(4) = -4.413282528387191 * 10^{-7}

The polynomial approximations p(y) and q(y) are minimax approximations to their corresponding functions over (-pi/4, pi/4).

ERROR ANALYSIS

A graph of the error of approximation in the algorithm for sin(x) over (-pi/4 pi/4) is given in figure 2-29 and for cos(x) over (-pi/4, pi/4) in figure 2-30. The maximum absolute value of the error of approximation in the algorithm for sin(x) over (-pi/4, pi/4) is 5.670 * 10^{-16}, and for cos(x) is 2.972 * 10^{-15}. Upper bounds for the error due to machine error in the computation of sin(x) and cos(x) were established at 2.898 * 10^{-14}, respectively. Hence upper bounds on the error in the routine are 2.955 * 10^{-14} and 4.741 * 10^{-14} for sin(x) and cos(x), respectively.

The maximum absolute value of the relative error of approximation in the algorithm for sin(x) over (-pi/4, pi/4) is 4.098 * 10^{-14} and for cos(x) is 6.285 * 10^{-14}. Upper bounds for the absolute value of the relative error due to machine error in the computation of sin(x) and cos(x) were established at 8.049 * 10^{-16} and 4.204 * 10^{-15}, respectively. Hence upper bounds on the absolute value of the relative error in the routine were established at 4.178 * 10^{-14} and 6.705 * 10^{-14} for sin(x) and cos(x), respectively.

For 1000 arguments chosen randomly from given intervals for the entry points shown, the associated statistics on absolute or relative error are given in table 2-21.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the argument x, the error in the result is given approximately by e' * cos(x) for sin(x), and -e' * sin(x) for cos(x).

SINCSD.

SINCSD. is an external function which accepts calls from FORTRAN code. It computes the sine and cosine functions in degrees (FORTRAN function names SIND and COSD). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry points SIND and COSD, and calls by value are computed at entry points SIND. and COSD..

METHOD

The input range to this routine is the set of all floating-point values in the interval (-2^{48}, $+2^{48}$).

Routine DEGCOM. is called to subtract the necessary multiple of 90 from the argument to put the result in the interval (-45, +45) and multiply the reduced value by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

sin(X$\pm$360^o)	= sin(X)
sin(X$\pm$180^o)	= -sin(X)
sin(X+90^o)	= cos(X)
sin(X-90^o)	= -cos(X)
cos(X$\pm$360^o)	= cos(X)
cos(X$\pm$180^o)	= -cos(X)
cos(X+90^o)	= -sin(X)
cos(X-90^o)	= sin(X)

ERROR ANALYSIS

The reduction to (-45, +45) is exact; the constant pi/180 has relative error 1.37E-15, and multiplication by this constant has a relative error 5.33E-15, and a total error of 6.7E-15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is at most 5.26E-15 plus the maximum error in SINCOS. over (-pi/4, +pi4), or 7.31E-15, for a total error of 12.57E-15. The maximum observed error for 100 000 points in the interval (0, 360) was 9.96E-15 for SIND and 9.95E-15 for COSD.

Figures 2-31 and 2-32 show the mean relative errors of SIND and COSD.

EFFECT OF ARGUMENT ERROR

Errors in the argument X are amplified by X/tan(X) for SIND and X*tan(X) for COSD. These functions have a maximum value of pi/4 in the interval (-45^0, +45^0) but have poles at even (SIND) or odd (COSD) multiples of 90^o, and are large between multiples of 90^o if X is large. When X is double-precision the following code may be used:

```
  FUNCTION SINDD(X)
  DOUBLE X
  NINT(X)=X+SIGN(0.5,X)
  K=0
  GO TO 1
  ENTRY COSDD
  K=1
1 N=NINT(SINGL(X)/90)
```

TABLE 2-21. ABSOLUTE AND RELATIVE ERROR OF SINCOS.

Entry Point	Error	Interval From	Interval To	Mean	Standard Deviation	Minimum	Maximum
COS.	Relative	-.7854	.7854	-5933E-17	1.596E-15	-7.346E-15	6.962E-15
	Absolute	-3.1416	3.1416	-7.524E-18	1.317E-15	-4.674E-15	4.809E-15
		-10^{12}	10^{12}	8.138E-19	1.248E-15	-5.443E-15	4.843E-15
SIN.	Relative	-.7854	.7854	3.035E-16	1.984E-15	-6.448E-15	6.739E-15
	Absolete	-3.1416	3.1416	-2.504E-18	1.133E-15	-5.648E-15	5.174E-15
		-10^{12}	10^{12}	-6.872E-18	1.254E-15	-4.187E-15	5.353E-15

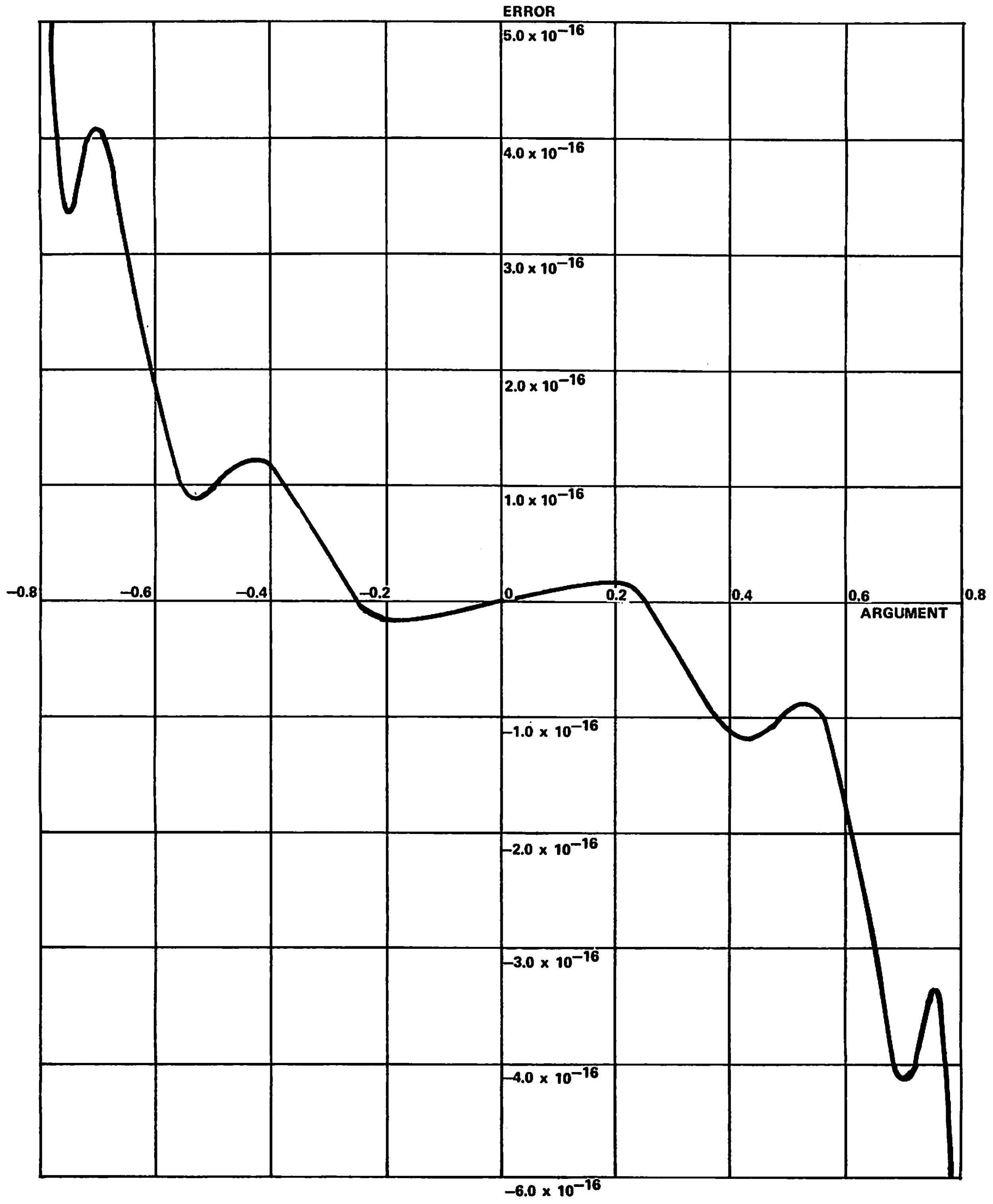

Figure 2-29. Algorithm Error of SINCOS. for Sine

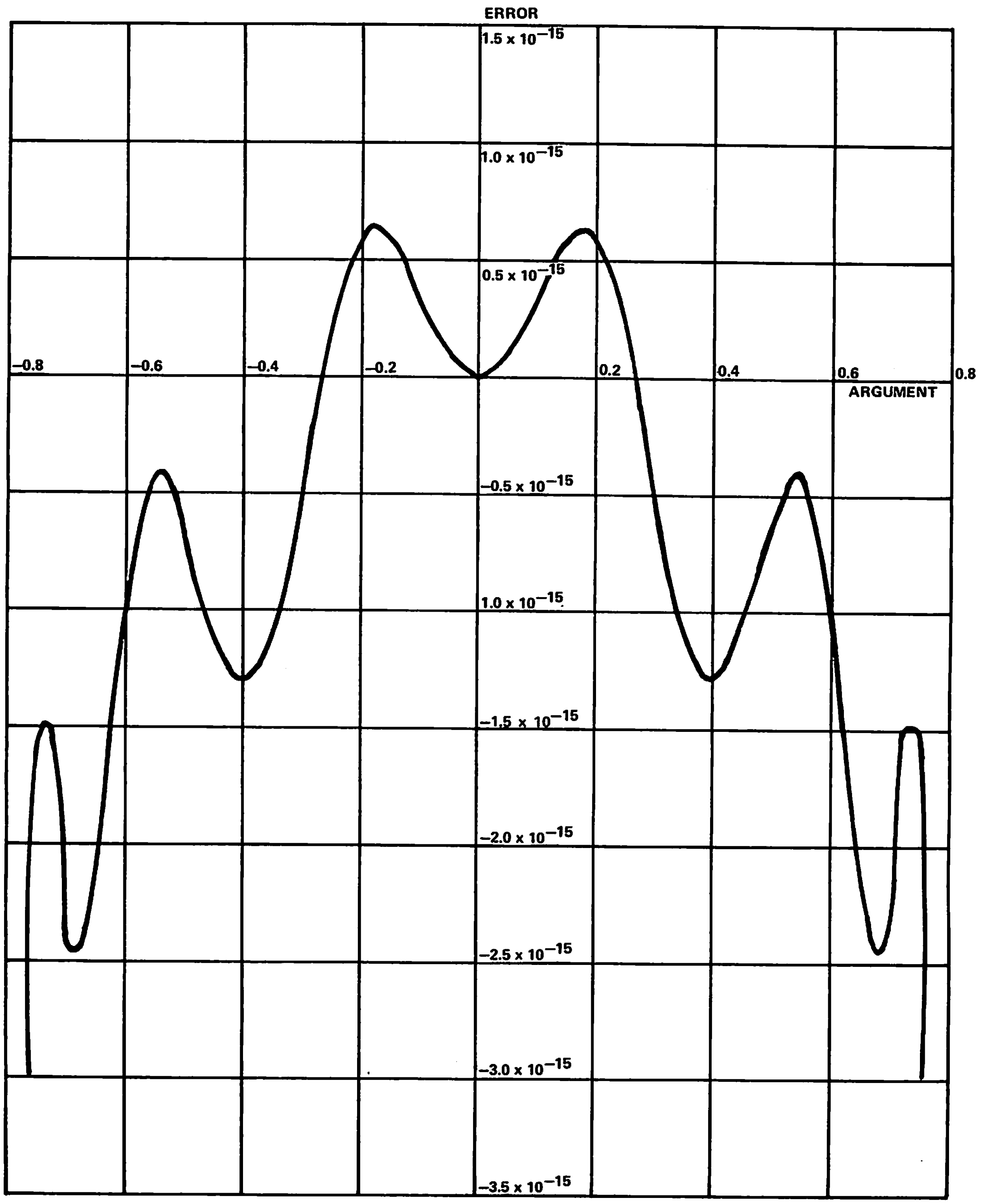

Figure 2-30. Algorithm Error of SINCOS. for Cosine

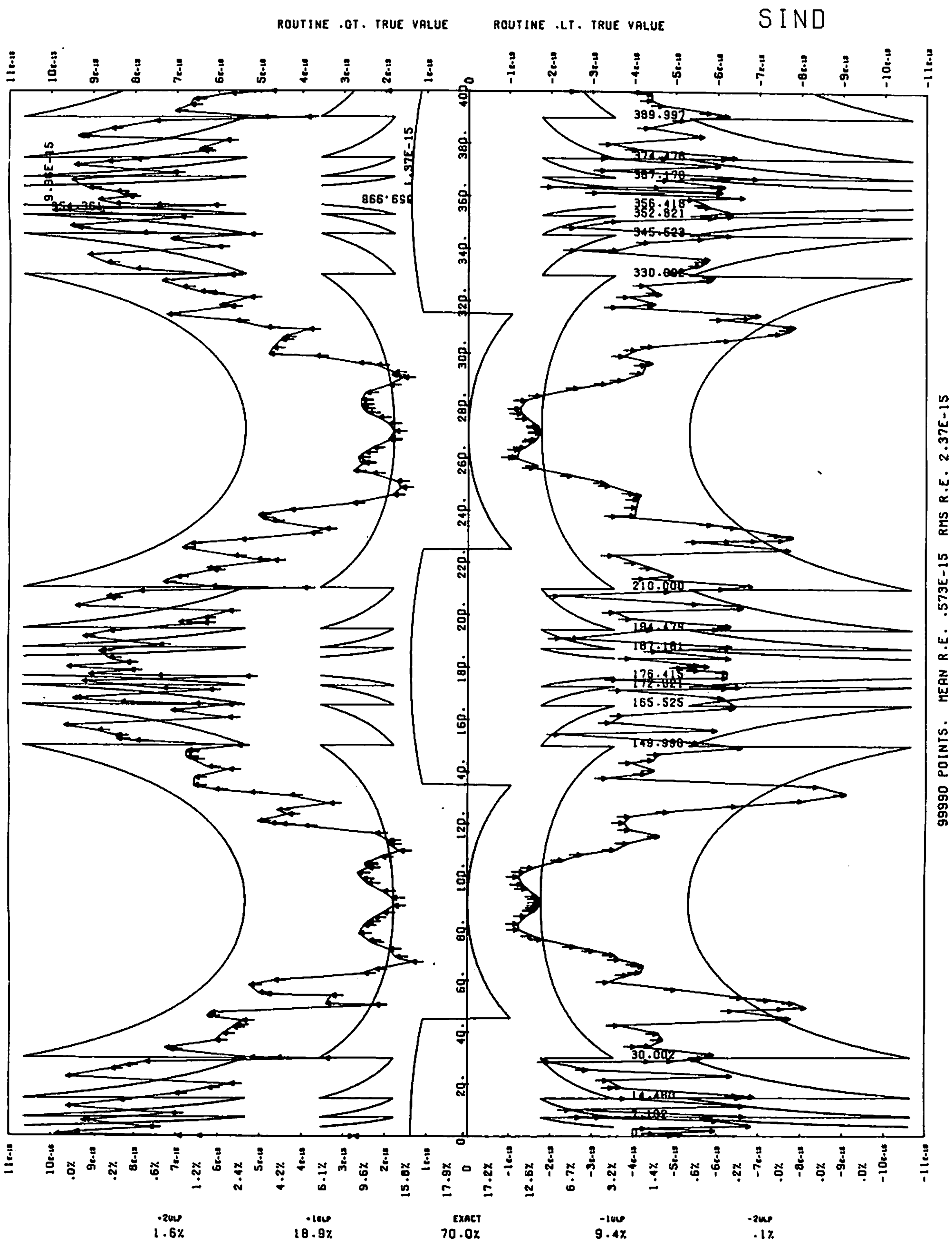

Figure 2-31. Mean Relative Error of SIND

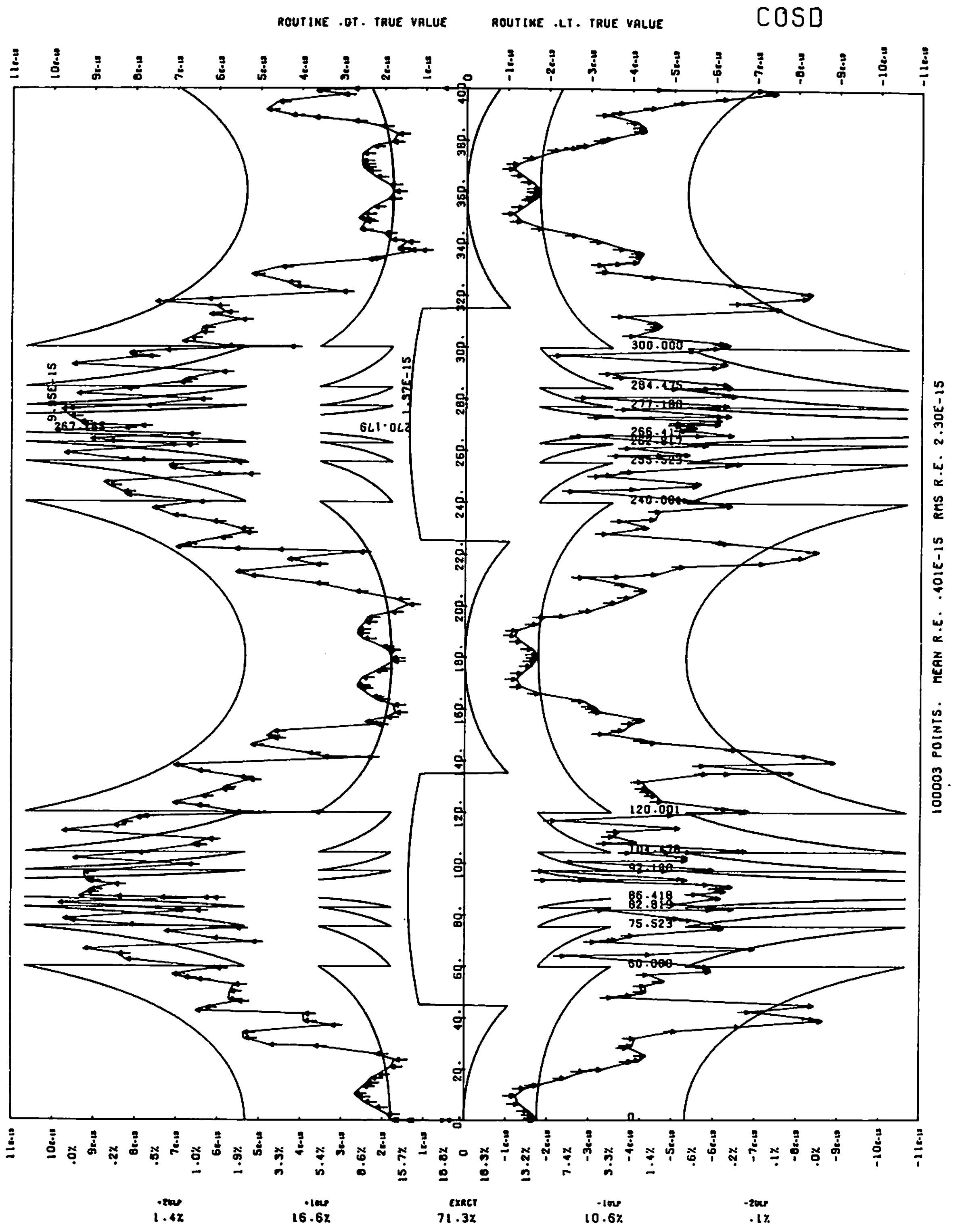

Figure 2-32. Mean Relative Error of COSD

```
  Z=X-N*90
  IF (K.NE.MOD(IABS(B),2)) GO TO 2
  Y=SIND(Z)
  GO TO 3
2 Y=COSD(Z)
3 IF (K*2-1.EQ.MOD(N,2))Y=-Y
  IF (MOD(IABS(N),4).GE.2)Y=-Y
  SINDD=Y
  RETURN
  END
```

SQRT

SQRT is an external function which accepts calls from FORTRAN code. It computes the square root function (FORTRAN function name SQRT). It accepts a real argument and returns a real result.

Calls by name are computed at entry point SQRT.

METHOD IOD

The argument is loaded into X1 and routine SQRT. is called.

ERROR ANALYSIS SIS

See SQRT..

The mean relative error is shown in figure 2-33.

EFFECT OF ARGUMENT ERROR

See SQRT..

SQRT.

SQRT. is an external function which accepts calls from FORTRAN code. It computes the square root function (FORTRAN function name SQRT). It accepts a real argument and returns a real result.

Calls by value are computed at entry point SQRT..

METHOD

The argument range is the set of all positive or zero floating-point numbers. The identity:

sqrt (y*2P)=sqrt (y)*$2^{(p/2)}$

is used to reduce the range to (0.5, 1) with p having an integral value. An initial approximation is made using one of eight linear approximations to the square root on this interval, giving at least 12 bits of accuracy. Two Heron's rule iterations are made to obtain 48 bits of precision.

If p is even, the normal Heron's rule is used:

```
compute x0, an approximation to x=sqrt(y)
x1=0.5*(x0+y/x0)
x2=0.5*(x1+y/x1)
```

If p is odd, scaling is done between steps to prevent affecting the accuracy of the final result:

```
compute x0
x1 =0.5*(x0+y/x0)
x1'=x1*sqrt(2)
x2 =0.5*(x1'+(2*y)/x1')
```

which performs the multiplication by $2^{1/2}$=sqrt (2).

The scaling by $2^{(p/2)}$ ((u) denotes truncation) is done by packing the appropriate exponent with the coefficient of (2*X2). The square root of a number one ulp below an even power of 2 is explicitly forced to one ulp below the square root of that power of 2 to make packing work (e.g., sqrt (4-eps) would be 1.0 but is forced to 2-eps).

Scaling for the square root of 2 is adjusted slightly so that the error is centered after this scaling, picking up one bit of precision at that point.

ERROR ANALYSIS

The maximum error in the initial approximation is .000218. Since the effect of a Heron's iteration is to square and halve the relative error, the algorithm error is 7.08E-17.

Round-off error is insignificant until the last Heron's rule step, which has the form x+y/x, where the quantities being summed are almost equal. Since the error in Heron's rule is always positive, x is too large, so y/x is too small (i.e., $x > y/x$). The error in the division is in the interval (-7.1E-15,0). The error in the rounded addition is in the interval (0, +3.55E-15), so the total round-off error is less than 3.55E-15 in absolute value. Error in division is halved because x is approximately y/x.

The upper bound on relative error is then 3.62E-15. The maximum observed relative error for 100 000 randomly chosen points in the interval (0.5,2) was 3.59E-15.

Figure 2-34 shows the relative error of SQRT in the interval (.5,1.).

EFFECT OF ARGUMENT ERROR

For small error in the argument y the amplification of absolute error is 1/(2*sqrt(y)) and that of relative error is 0.5.

SYS=AID

SYS=AID is an auxiliary routine. It provides a link between the math library and the system error processor. The entry point is SYSAID..

METHOD

Execution proceeds as follows:

a. Enter SYS=AID and save registers X3 and X4.

b. Read entry point SYSAID. and store it at entry point SYS1ST..

c. Long jump to MORGUE..

See the method description of SYS=1ST for further details.

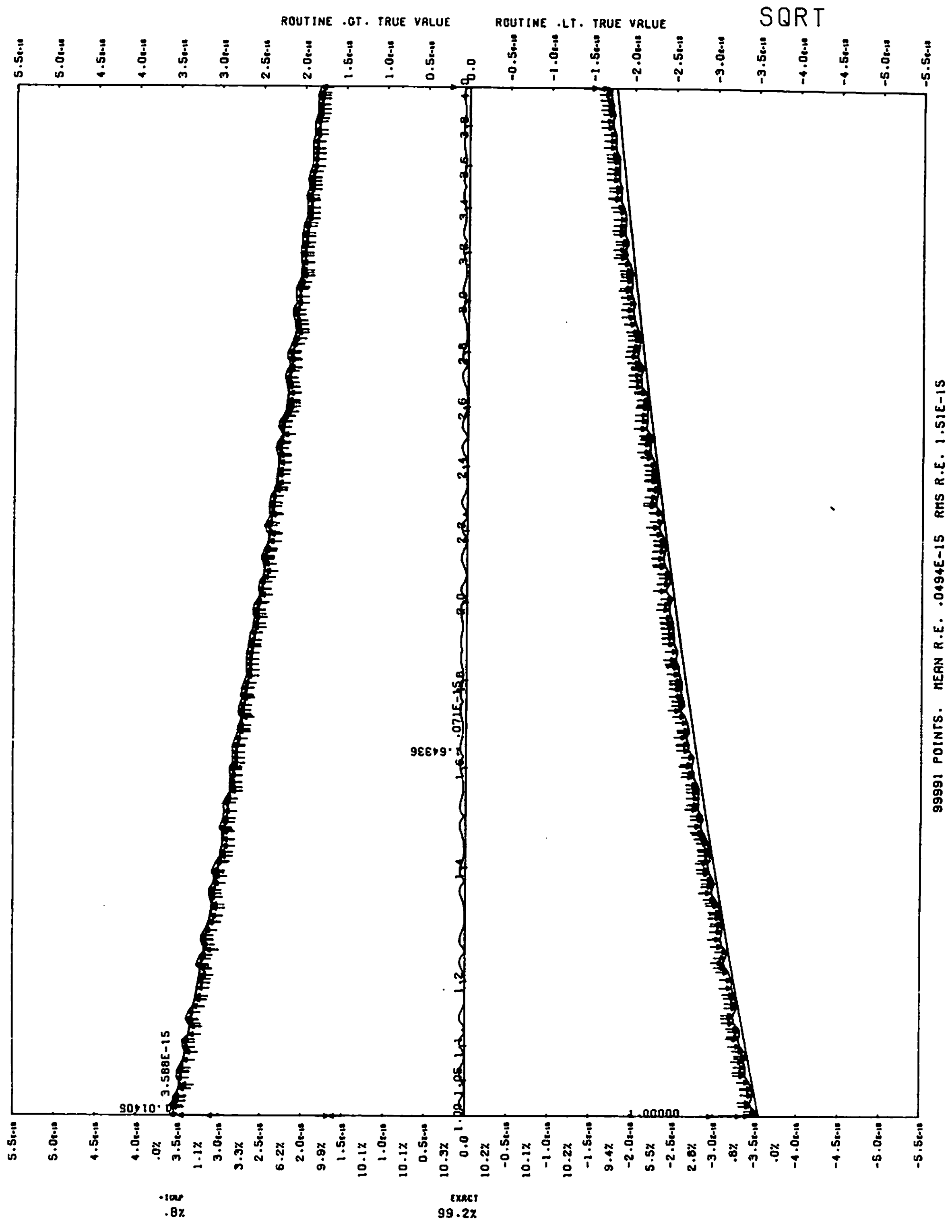

Figure 2-33. Mean Relative Error of SQRT

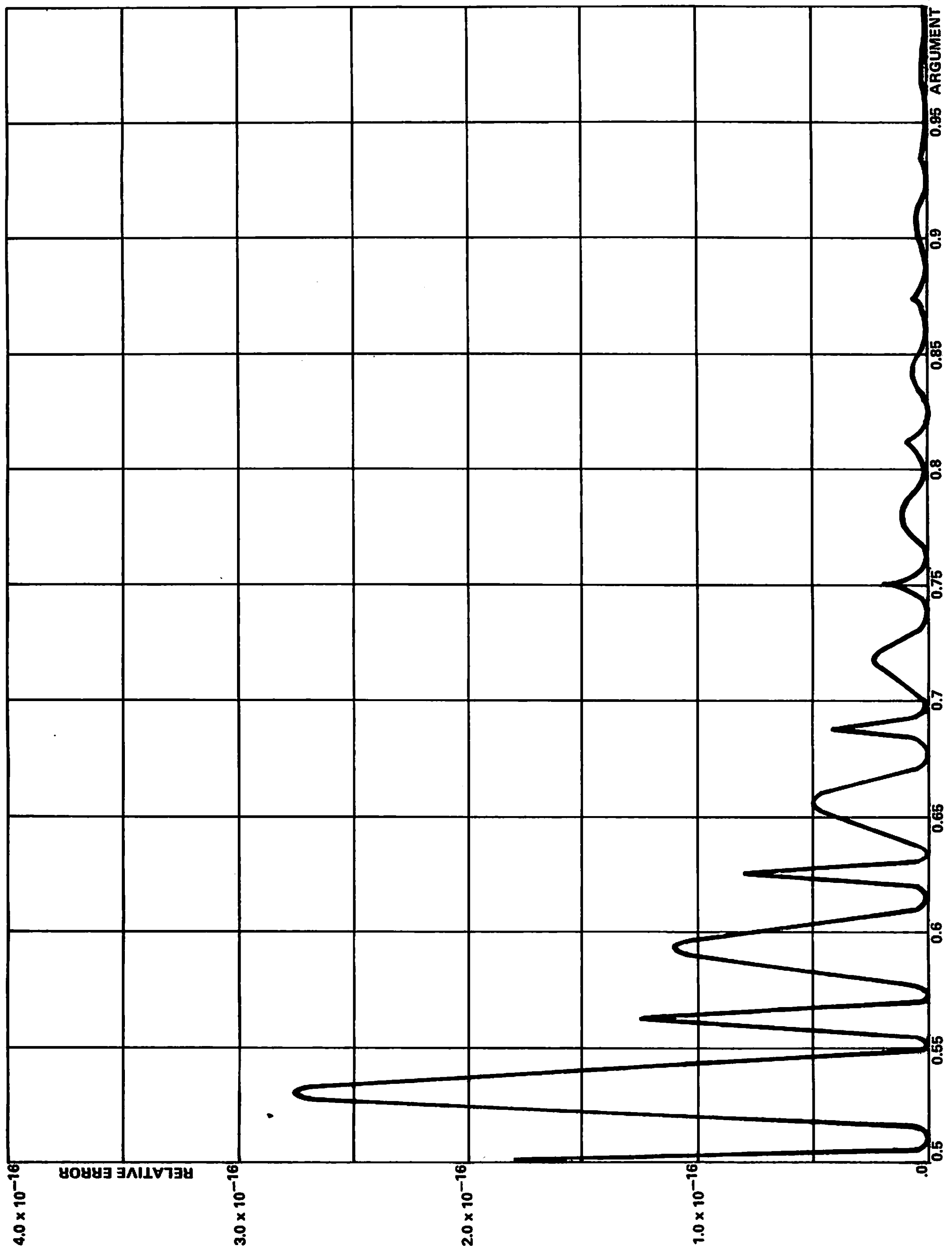

Figure 2-34. Relative Error of SQRT

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

SYS=1ST

SYS=1ST is an auxiliary routine. It provides a link between the math library and the system error processor. The entry points are SYS1ST. and MORGUE..

METHOD

Execution proceeds as follows at MORGUE.:

a. Enter SYS=1ST and save registers X1, X2, X6, A0, B5, B6 and B7.

b. Read the return jump word used to enter the routine which called SYS=1ST or SYS=AID. If this word has the format:

```
+ RJ     <entry point>
- VFD  30/1
```

then go to f. below.

c. Read the communication cell SYSAID.. Insert in its lower 18 bits the address of the trace word in routine SYS=1ST. Store the result in cell RJERR which will be executed at step e.

d. Test the argument in the register indicated by the contents of B2. Set X2 to the first word address of an error message as follows:

Condition	Message
Infinite	"ARGUMENT INFINITE"
Indefinite	"ARGUMENT INDEFINITE"
Other	"ARGUMENT" < partial message from address supplied in B2 >

Set X1 to the error number, and A0 to the first word address of the parameter list for non-standard error recovery.

e. Execute word RJERR. This will link the routine to the system error processor.

f. Restore registers X1, X2, A0, B5, B6, B7. Move the contents of X6 into register X5.

g. Set X6 and X7 to +IND..

h. Return to the calling program.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

Not applicable.

TAN

TAN is an external function which accepts calls from FORTRAN code. It computes the trigonometric tangent function (FORTRAN function name TAN). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point TAN.

METHOD

The argument is loaded into X1 and routine TAN. is called.

ERROR ANALYSIS

See TAN..

EFFECT OF ARGUMENT ERROR

See TAN..

TAN.

TAN. is an external function which accepts calls from FORTRAN code. It computes the trigonometric tangent function (FORTRAN function name TAN). It accepts a floating-point argument and returns a floating-point result.

Calls by value are computed at entry point TAN..

METHOD

The input range is the collection of all definite, in-range floating-point quantities in the interval (-2^{47}, +2^{47}).

The identities:

i) tan(x)=tan(x+k*pi/2), k is even
ii) tan(x)=-1.0/tan(tan(x+pi/2)

are used in the form:

iii) tan(x)=tan((pi/2)*(x*2/pi+k)), k is even
iv) tan(x)=-1.0/tan((pi/2)*(x*2/pi+1))

to reduce the evaluation to the interval (-0.5, +0.5). An approximation of tan((pi/2*y) is used. The reduction is done by multiplying x by 2/pi, subtracting the nearest integer, and rounding the result to single-precision.

The function tan((pi/2)*y) is approximated with a rational form (7th order odd)/(6th order even), which has minimax relative error in the interval (-n.5, +0.5). The rational form is normalized to make the last numerator coefficient 1+eps, where eps is chosen to minimize rounding error in the leading coefficients.

Identity (iv) is used if the integer subtracted is odd. The result is negated and inverted by dividing -Q/P instead of P/Q. The mean relative error is graphed in figure 2-35.

ERROR ANALYSIS

The range reduction, the final add in each part of the rational form, the final multiply in P, and the divide

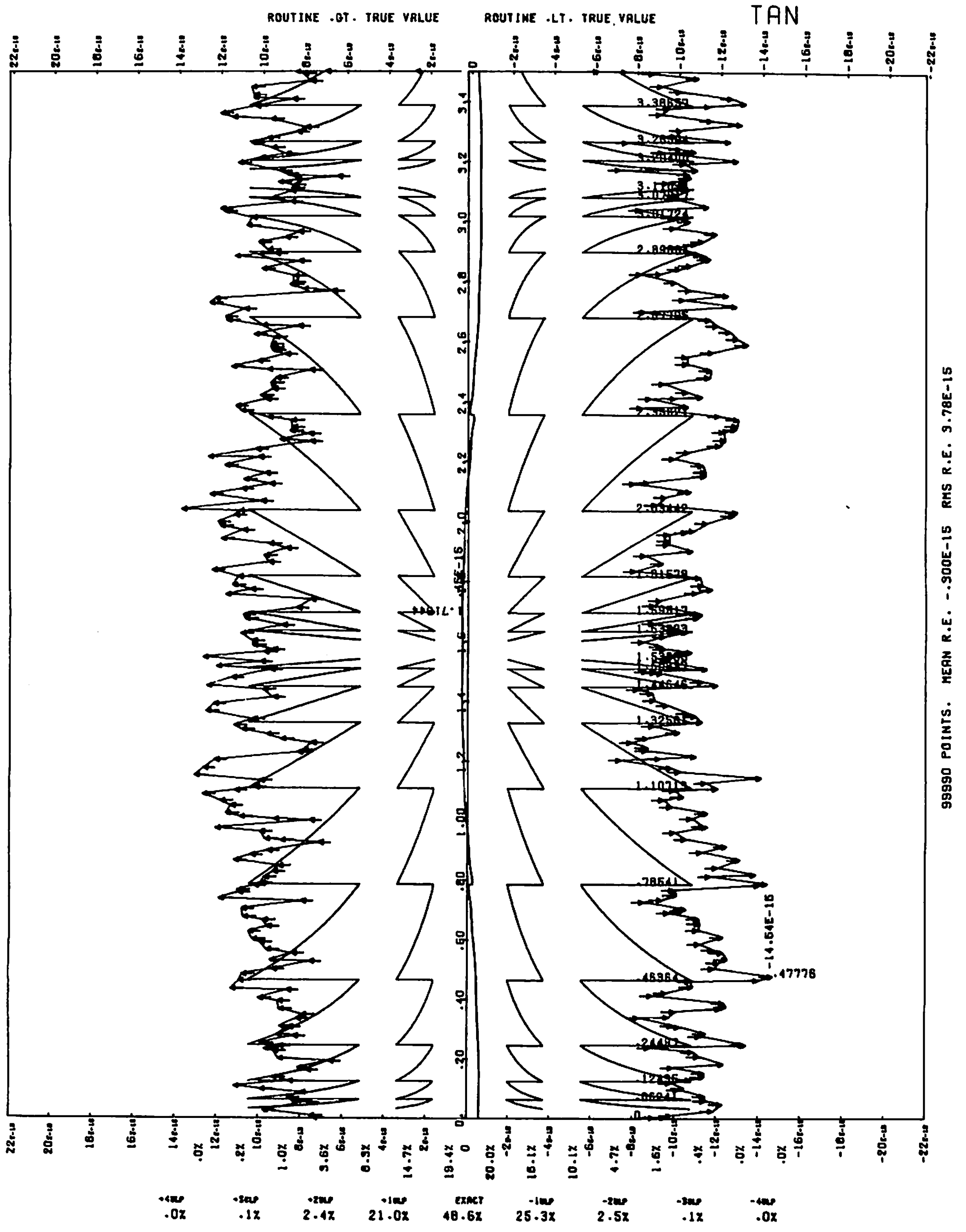

Figure 2-35. Mean Relative Error of TAN

dominate the error. Each of these operations contributes directly to the final error, and each is accurate to about 1/2 ulp. The maximum relative errors are given in table 2-22.

TABLE 2-22. MAXIMUM RELATIVE ERROR OF TAN.

Source of Error	Amount*10^{15}
range reduction	3.6
rational form	.02
coefficient rounding	<.08
round-off	14.2
upper bound	18.0
maximum observed	14.5

EFFECT OF ARGUMENT ERROR

For small errors in the argument x, the amplification of absolute error is $\sec^2(x)$, and that of relative error is x/(sin(x)*cos(x)), which is at least 2x and can be arbitrarily large near a multiple of pi/2. If x is known to more than double-precision, the tangent addition formula can be used if x is less than 3E7:

```
DOUBLE X

(compute X)

T=TAN(SINGL(X))

S=SINGL(X-SNGL(X))

Y=T+S*(1+T**2)/(1-S*T)
```

Thus, S=TAN(S) if X < 3E7. This approximation can give less than single-precision when S*T is near 1.0. It is more accurate than TAN(SNGL(X)) but less accurate than SNGL(DTAN(X)).

TAND.

TAND. is an external function which accepts calls from FORTRAN code. It computes the trigonometric tangent in degrees (FORTRAN function name TAND). It accepts a floating-point argument and returns a floating-point result.

Calls by value name are computed at entry point TAND, and calls by value are computed at entry point TAND..

METHOD

The input range of TAND. is the set of floating-point arguments in the interval $(-2^{48}, +2^{48})$ excluding odd multiples of 90.

Routine DEGCOM. is called to subtract the necessary multiple of 90 from the argument to put the result in (-45, +45) and multiply the reduced value by pi/180. Routine TAN. is called to compute the tangent, and the result is negated and inverted if the multiple was odd, using these identities:

$\tan(X \pm 180^\circ) = \tan(X)$

$\tan(X \pm 90^\circ) = -1/\tan(X)$

ERROR ANALYSIS

The reduction to (-45, +45) is exact; the constant pi/180 has a relative error of 1.37E-15, and multiplication by this constant has a relative error of 5.33E-15, so the total error is 6.7E-15. Since errors in the argument of TAN are amplified at most by pi/2, the error due to reduction and conversion is at most 10.52E-15. The error in the final division is at most 7.11E-15, and the error in TAN. is at most 14.54E-15, so an upper bound on error in TAND is 32.17E-15. The maximum observed error in 100 000 points in the interval (0,360 was 17.72E-15.

Figure 2-36 shows the mean relative error.

EFFECT OF ARGUMENT ERROR

Errors in the argument X are amplified at most X/(sin(X)*cos(X)). This function has a maximum of pi/2 within $(-45^\circ, +45^\circ)$, but has poles at all multiples of 90° except zero and is at least 2*X elsewhere. When X is known to be double-precision and one of the above problems exists, the following code may be used:

```
(compute X in double)

   N=NINT(SNGL(X)/90)
   Y=TAND(SNGL(X-N*90))
   IF (NOD(N,2).EQ.0) GO TO 1
   IF (Y.EQ.0)    error
   Y=-1.0/Y
1  CONTINUE
```

This always returns an accurate value since the range reduction is exact; NINT(X) + IFIX(X+SIGN(0.5,X)), the nearest integer.

TANH

TANH is an external function which accepts calls from FORTRAN code. It computes the hyperbolic tangent function (FORTRAN function name TANH). It accepts a floating-point argument and returns a floating-point result.

Calls by name are computed at entry point TANH.

METHOD

The argument is loaded into X1 and routine TANH. is called.

ERROR ANALYSIS

See TANH..

EFFECT OF ARGUMENT ERROR

See TANH..

TANH.

TANH. is an external function which accepts calls from FORTRAN code. It computes the hyperbolic tangent function (FORTRAN function name TANH). It accepts a floating-point argument and returns a floating-point result.

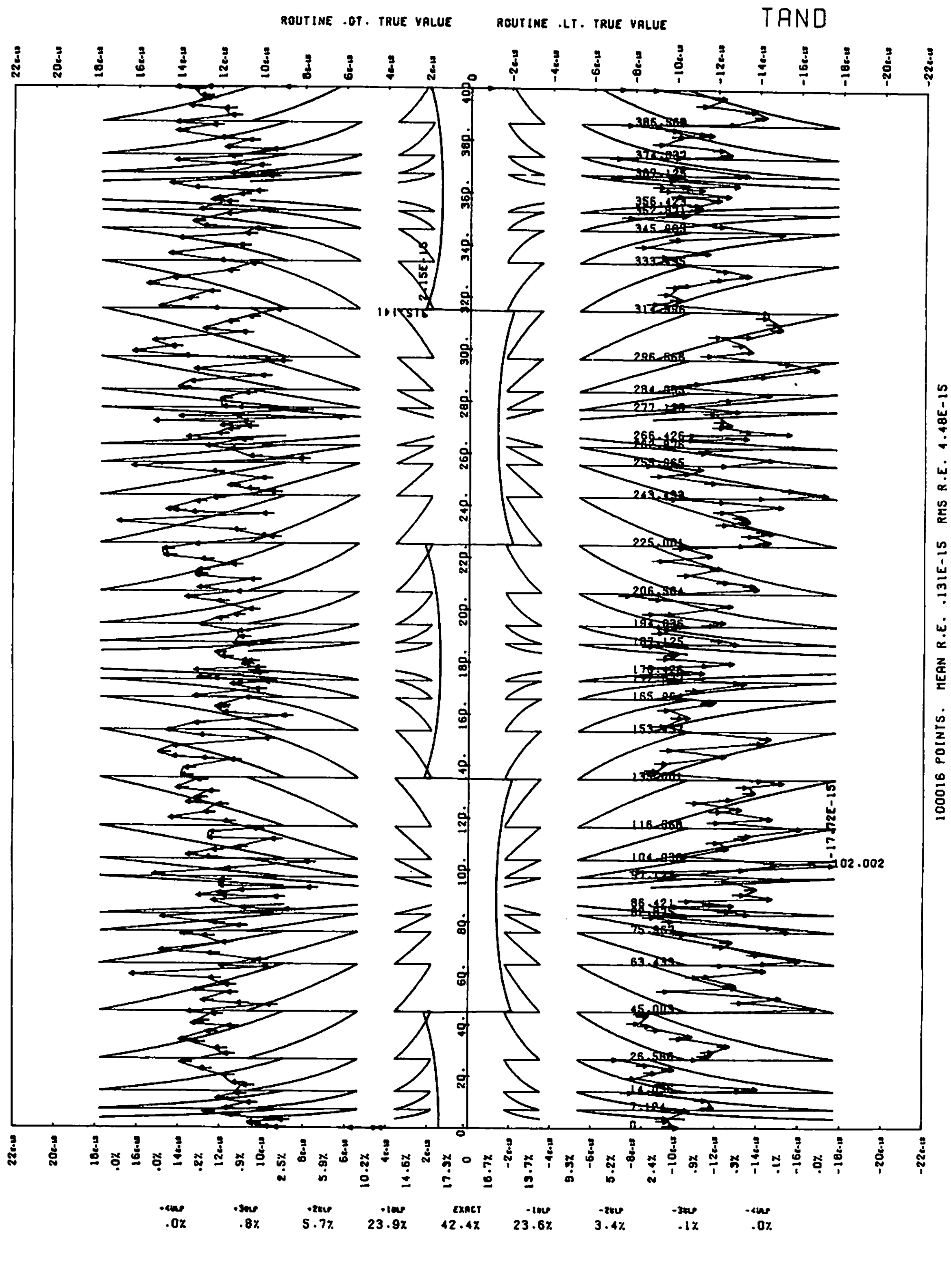

Figure 2-36. Mean Relative Error of TAND.

METHOD

The input range is the collection of all definite floating-point quantities in the range (NEG.INF., POS.INF.).

The identity tanh(-x)=-tanh(x) is used to reduce the range to (0,POS.INF). For abs(x) 17.50, the best machine representation of tanh(x) is sign(1.0,x), so the range is further reduced to (0,17.50).

The identities:

```
  tanh(x)=p(x)/a(x) approximately, on (0,0.55)
  tanh(x)=1-2/(exp(2*x)+1)
exp(2*x)=(1+tanh(x))/(1-tanh(x))
exp(2*x)=2 n*exp(2*(x-n*ln(2)/2))
```

may be combined to get:

tanh(x)=1-2*(q-p)/((q-p)+2 n*(q+p))

where n is chosen to be nint(x*2/ln(2)) and p and q are evaluated on x-n*ln(2)/2. This choice of n minimizes abs(x-n*ln(2)/2).

When $x < 0.55$ the approximation P(x)/q(x) is used. Since tanh($x < 0.55$) < 0.5, the form 1-r would suffer from cancellation in this range.

The approximation p/q is a minimax (relative error) rational form (i.e., (5th order odd)/(6th order even)). The coefficients are scaled so that (x*2/ln(2)-n) may be used instead of (x-n*ln(2)/2), simplifying the range reduction. The coefficients are further scaled by an amount sufficient to reduce truncation error in the leading coefficients without otherwise affecting accuracy.

ERROR ANALYSIS

The algorithm error due to finite approximation and coefficient truncation is 1.7E-15. For abs(x) < 0.55 the form p(x)/q(x) is used. The final operations z=x*2/ln(2) and tanh(z*(p0+small))/(q0+small) dominate the error. The upper bound on the error here is 18.0E-15; the maximum observed was 13.0E-15.

For abs(x) > 1.25 the final subtraction, 1.0-small, dominates and an upper bound on the error is 4.1E-15; the maximum observed was 3.8E-15.

For $0.55 \leq$ abs(x) ≤ 1.25 the final operation is 1-R where R becomes smaller as x approaches 1.25. Thus, the worst relative error is near 0.55, namely (contribution from R) + (error in final sum), where R=2*(q-p)/((q-p) + 4*(q+p)). An upper bound was 16.7E-15; the maximum observed was 10.0E-15. The maximum relative errors are given in table 2-23.

TABLE 2-23. MAXIMUM RELATIVE ERROR OF TANH.

Source of Error	Error*10^{15}
rational form	0.5
coefficient rounding	1.2
round-off	16.5
upper bound	18.2
maximum observed	13.0

Figure 2-37 shows the error in the polynomial approximation of TANH over (-1.2,1.2) and figure 2-38 shows the mean relative error.

EFFECT OF ARGUMENT ERROR

For small errors in the argument x, the amplification of the absolute error is 1/$\cosh^2$(x), and of relative error is x/(sinh(x)*cosh(x)). Both have maximum values of 1.0 at 0 and approach 0 as x gets large.

XTOD*

XTOD* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise floating-point quantities to double-precision exponents. It accepts a floating-point argument and a double-precision argument, and returns a double-precision result.

Calls by name are computed at entry point XTOD$.

METHOD

The formula used is:

$\text{base}^{\text{exponent}}$ = exp(exponent * log(base)).

Upon entry, the argument set is checked. It is invalid if: either argument is infinite or indefinite, the base is negative, the base is zero and the exponent is not greater than zero, or floating-point overflow occurs during the computation. If the argument set is invalid, a diagnostic message is issued and POS.INDEF. is returned. If the argument set is valid, the result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in XTOD* is the same as that used in XTOD.. See the description of routine XTOD..

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b and a small error e" occurs in the exponent p, the error in the result is given approximately by:

b^p * (p/b * e' + log(b) * e")

The absolute error is approximately the absolute value of this expression. If the errors in the argument are significant, the error in the result should be found by substitution of the possible argument values in the expression b^p.

XTOD.

XTOD. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to double-precision exponents. It accepts a floating-point argument and a double-precision argument, and returns a double-precision result.

Calls by value are computed at entry point XTOD..

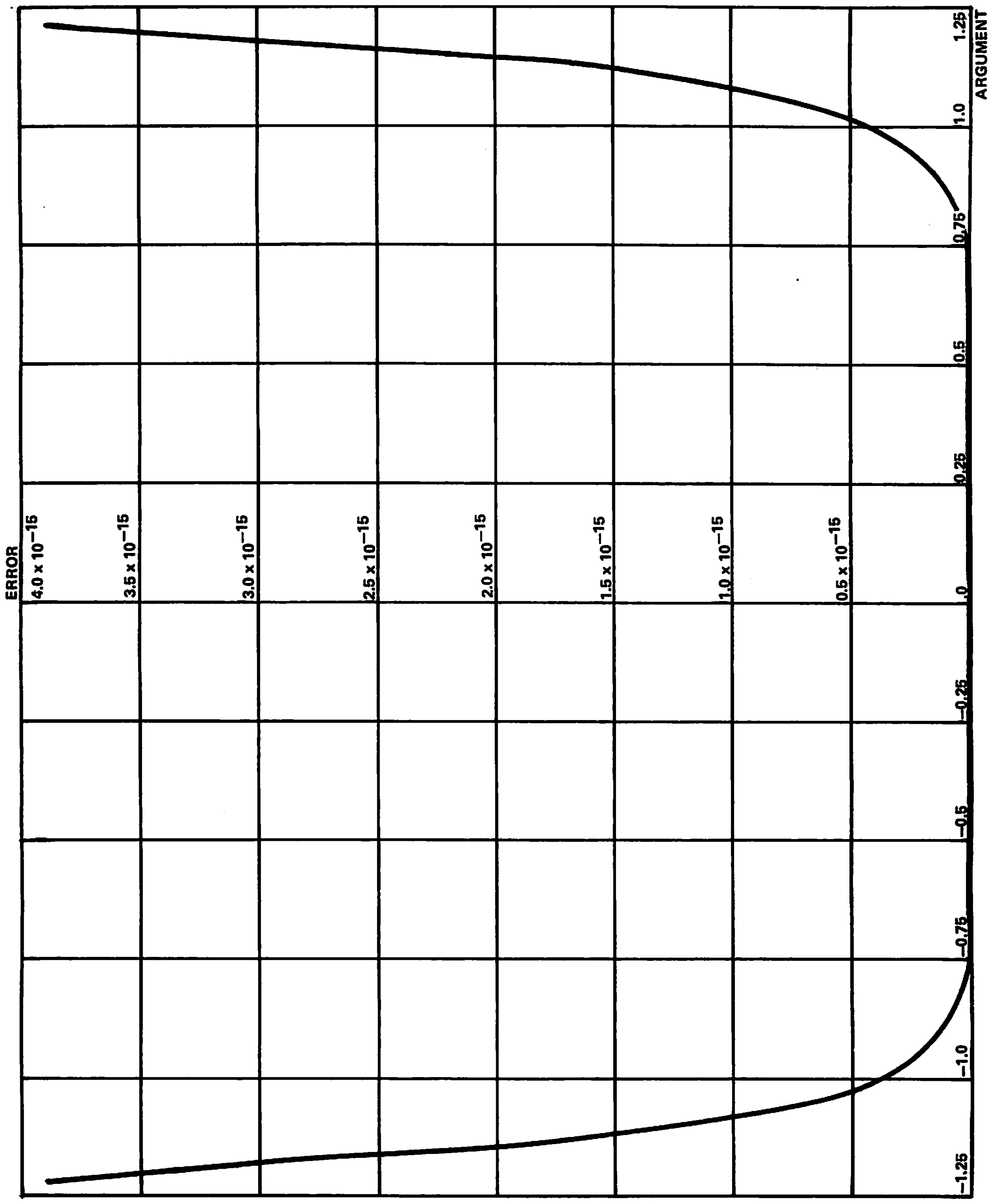

Figure 2-37. Error of TANH. for Tanh

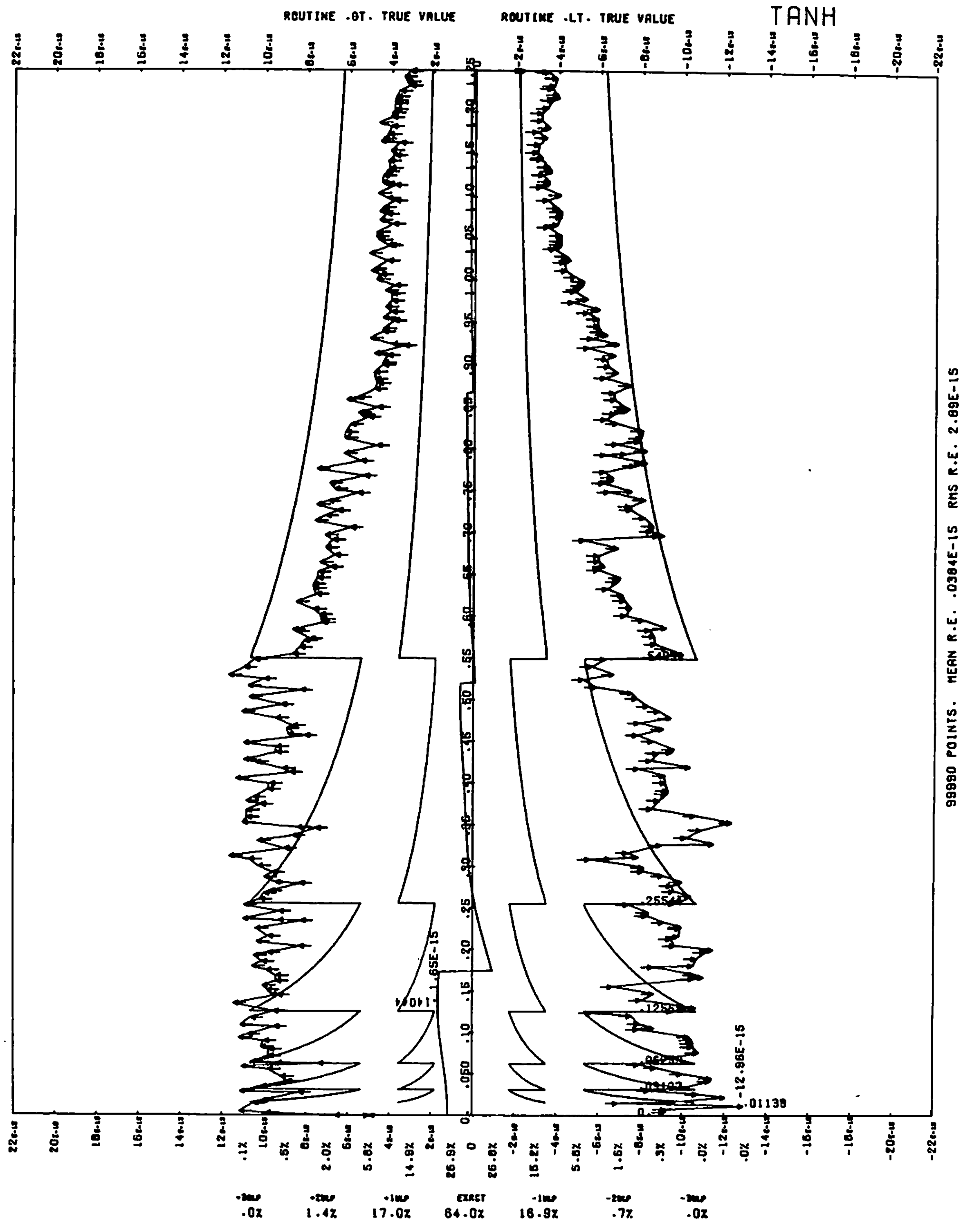

Figure 2-38. Mean Relative Error of TANH.

METHOD

The input range is the collection of argument sets (b,p), where b is a definite in-range floating-point quantity, p is a definite in-range double-precision quantity, b is greater than zero, and b^p = exp (p*log b), where b is converted to double-precision upon entry, and all operations are carried out in double-precision. The result is returned to the calling program.

ERROR ANALYSIS

10 000 argument sets (b,p) were randomly generated, with distribution a product of uniform distribution in (.5,1.5) and (-10,10). The relative error in the routine was computed for each of the argument sets. The maximum absolute value of the relative error was $1.163 * 10^{-25}$.

EFFECT OF ARGUMENT ERROR

If a small error e(b) occurs in the base b and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

r * (e(p) * log b + p * e(b)/b).

XTOI*

XTOI* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to fixed-point exponents. It accepts a floating-point argument and a fixed-point argument, and returns a floating-point result.

Calls by name are computed at entry point XTOI$.

METHOD

The arguments are loaded and XTOI. is called.

ERROR ANALYSIS

Not applicable, since the only errors are round-off errors. See the description of XTOI..

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base p, the error in the result is given approximately by:

$b^{(p-1)}$ * p * e', where p is the exponent.

If the error in the base becomes significant, the error in the result must be found from substitution of the possible values of the base b into the expression b^p.

XTOI.

XTOI. is an exponentiation routine which accepts calls from compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to fixed-point exponents. It accepts a floating-point argument and a fixed-point argument, and returns a floating-point result.

Calls by value are computed at entry point XTOI..

METHOD

The arguments are checked to see if the exponentiation conforms to a special case. If it does, the proper value is immediately returned, or if the special case is an error condition, an error message is issued. The special cases are:

x indefinite	= error
x infinite	= error
0^0	= error
X^0	= 1.0
X^I	= $1.0/X^{-I}$ if $I < 0$

If the exponentiation is not a special case, one of two methods is used to perform the exponentiation. Method 1 is a quick algorithm, and is usually used. Method 2 is used when the number of bits in I plus the number of bits in X is greater than 8.

Method 1

The binary representation of I is scanned starting with the most significant bit. For each bit, the result, which was initialized to X, is squared. If the next bit is one, the result is also multiplied by X.

Method 2

Scaling is performed to make X be between .75 and 1.5, and the exponent is saved. Ten bits of I are scanned as described in Method 1. This procedure is repeated until I is used up. If the exponent is not too large, the result is returned.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b, then the error in the result is given approximately by p * $b^{(p-1)}$ * e', where p is the exponent. If the error e' becomes significant, the absolute error in the result is bounded by:

$|p| * \max(|b, b + e'|)^{(p-1)} * |e'|$

XTOY*

XTOY* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to floating-point exponents. It accepts two floating-point arguments and returns a floating-point result.

Calls by name are computed at entry point XTOY$.

METHOD

The formula is:

$\text{base}^{\text{exponent}}$ = exp (exponent*log (base))

The argument set is checked upon entry. It is invalid if: either base or exponent is infinite or indefinite, the base is negative, the base is zero and the exponent is not greater than zero, or floating-point overflow occurs during the computation. If the argument set is invalid, POS.INDEF. is returned and a diagnostic message is issued. Otherwise, the result of the computation is returned.

ERROR ANALYSIS

The algorithm used in XTOY* is the same as that used in XTOY.. See the description of routine XTOY..

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b and a small error e" occurs in the exponent p, the error in the result is given approximately by:

$b^p * (p/b * e' + \log(b) * e'')$

The absolute error is approximately the absolute value of this expression. If the errors in the arguments are significant, the error in the result should be found by substitution of the possible argument values in the expression b^p.

XTOY.

XTOY. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for FORTRAN statements which raise floating-point quantities to floating-point exponents. It accepts two floating-point arguments, and returns a floating-point result.

Calls by value are computed at entry point XTOY..

METHOD

The input range is the collection of all argument sets (b,e) for which: b and e are definite in-range floating-point quantities, b is positive and nonzero, and b^e is in-range.

$b^p = \exp(p * \log b)$

where $b > 0$.

Upon entry, ALOG. computes log b, and then EXP. computes exp (p * log b). The result is returned.

ERROR ANALYSIS

500 000 pairs (b,p) of random numbers were generated with distribution being the product of the right half of a Cauchy distribution, and a Cauchy distribution. b^p was computed for each of the pairs, first using the routine, and then using the double-precision routine. The maximum absolute value of the error in the routine was $4.583 * 10^{-12}$.

EFFECT OF ARGUMENT ERROR

If a small error e(b) occurs in the base b, and a small error e(p) occurs in the exponent p, the error in the result r is given approximately by:

$r * (\log b * e^p + p * (e(b))/b)$

XTOZ*

XTOZ* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to complex exponents. It accepts a floating-point argument and a complex argument, and returns a complex result.

Calls by name are computed at entry point XTOZ$.

METHOD

If the base b is real and the exponent z is x + i * y, where x and y are real, then:

$b^z = u + i * v$

where:

$u = \exp(x * \log(b)) * \cos(y * \log(b))$
$v = \exp(x * \log(b)) * \sin(y * \log(b))$

ALOG., EXP. and COS.SIN are called to evaluate these expressions. The argument set is checked upon entry. It is invalid if: either base or exponent is infinite or indefinite, the base b is negative, the base is zero and the real part of the exponent is greater than zero, y * log(b) is so large that precision is lost in the computation, or floating-point overflow occurs during the computation. If the base b is zero, y is zero, and x is less than zero, POS.INF. is returned. If the argument set is otherwise invalid, POS.INDEF. is returned. In either case, a diagnostic message is issued. If the argument set is valid, ALOG., EXP. and COS.SIN are called during computation. The result is returned to the calling program.

ERROR ANALYSIS

The algorithm used in XTOZ* is the same as that used in XTOZ.. See the description of routine XTOZ..

EFFECT OF ARGUMENT ERROR

If a small error e(b) occurs in the base b, and small errors e(x) and e(y) occur in the real and imaginary parts, x and y, respectively, of the exponent z, then the error e(r) in the result is given approximately by:

$e(r) = b^z * \log(b)*z*((e(x) + i*e(y))/z + e(b)/(b*\log(b)))$

The absolute error in the result is approximately the absolute value of this expression. If the error in an argument becomes significant, the error in the result should be found from substitution of possible argument values in the expression b^z.

XTOZ.

XTOZ. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise floating-point quantities to complex exponents. It accepts a floating-point argument and a complex argument, and returns a complex result.

Calls by value are computed at entry point XTOZ..

METHOD

The input range is the collection of all argument sets (x,z) (= x, u + i*v)), such that: x is positive, if x is zero then u = 0 and v is positive and nonzero, both x and z are definite and in-range, and floating-point overflow does not occur during the computation of x^u (i.e., $|u*\log(x)| \leq 741.67$, and $|v*\log(x)| \leq pi*2^{46}$).

The formula used is:

$$x^{(u+i*v)} = e^{(u*\log(x)} * \cos(v*\log(x)) + i * e^{(u*\log(x)} * \sin(v*\log(x))$$

Upon entry, the base is checked. If it is zero, zero is immediately returned to the calling program. Otherwise, ALOG. is called for computation of log x, and COS.SIN is called for computation of cos(v*log(x)) and sin(v*log(x)). Then EXP. is called for computation of exp(u*log(x)). The result is calculated according to the formula and is returned to the calling program.

ERROR ANALYSIS

400 000 pairs (x,z) of random numbers were generated with distribution being the product of a right half of a Cauchy distribution, and the product of two Cauchy distributions. Then x^z was computed for each of these pairs, first using the routine, then using double-precision operations. The maximum absolute value of the relative error in the routine was $7.196 * 10^{-10}$.

EFFECT OF ARGUMENT ERROR

If a small error e(x) occurs in the base x, and a small error e(z) (or e'(x)+i*e'(y)) occurs in the exponent z, the error in the result w is given approximately by:

$$w * (\log x * e(z) + z * e(x)/x)$$

ZTOI*

ZTOI* is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements that raise complex quantities to fixed-point exponents. It accepts a complex argument and a fixed-point argument, and returns a complex result.

Calls by name are computed at entry point ZTOI$.

METHOD

See the description of ZTOI. for the algorithm. The argument set is checked upon entry. It is invalid if: either argument is infinite or indefinite, or the base is zero and the exponent is not greater than zero. In these cases, POS.INDEF. is returned and a diagnostic message is issued. Otherwise, the result of the computation is returned to the calling program.

ERROR ANALYSIS

Not applicable, since the only errors are round-off errors.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the base b, the error in the result is given approximately by $n * b^{(n-1)} * e'$, where n is the exponent. The absolute value of this expression is approximately the absolute error. If the error e' is significant, the error in the result should be found by substitution of the possible argument values in the expression b^n.

ZTOI.

ZTOI. is an exponentiation routine which accepts compiler-generated calls from FORTRAN code. It performs exponentiation for statements which raise complex quantities to fixed-point exponents. It accepts a complex argument and a fixed-point argument, and returns a complex result.

Calls by value are computed at entry point ZTOI..

METHOD

A b represents the base and p represents the exponent. If p is non-negative and has the binary representation 000...0i(n-1)...i(1)i(0), where each $i(j) (0 \leq j \leq n)$ is 0 or 1, then:

$$p = i(0) * 2^0 + i(1) * 2^1 + \ldots + i(n) * 2^n$$

and n = (log(2)p) = greatest integer not exceeding log(2)p. Then:

$$b^p = \mathrm{Prod}(b^2 : 0 \leq j \leq n \text{ and } i(j) = 1).$$

The numbers $b = b^2$, b^2, b^4 ..., b^2 are generated by successive squarings, and the coefficients i(0),..., i(n) are obtained as sign bits of successive circular right shifts of p within the computer. A running product is formed during the computation, so that smaller powers of b may be discarded. Thus, the computation becomes an iteration of the algorithm:

$b^p = 1$ if $p=0$
$b^p = (b^2)^{(p/2)}$ if p 0 and p is even
$b^p = b*(b^2)^{((p-1)/2)}$ is p 0 and p is odd

Upon entry, if the exponent p is negative, p is replaced by -p and a sign flag is set. b^p is computed according to this algorithm, and if the sign flag was set, the result is reciprocated before being returned to the calling program.

The input range is the collection of pairs of bases b and exponents p such that b is nonzero if p is negative, both arguments are definite and in-range, and the result is in-range.

ERROR ANALYSIS

Not applicable.

EFFECT OF ARGUMENT ERROR

If a small error e' occurs in the complex base b, the error in the result is given approximately by $p * b^{(p-1)} * e'$. If e' is significant, the absolute value of the error in the result is less than or equal to:

$$|p| * (|b + b + e'|)^{(p-1)} * |e'|$$

SUMMARY OF ROUTINES

The following tables summarize the major math functions in the math library. The routines are listed in alphabetical order according to the math routine name. For each routine, the follwing information is given.

FCL Routine Name	The name of the FORTRAN Common Library math routine.
Entry Points	Possible points at which execution of the routine may begin.
Type of Call	Specifies whether the routine is called by name or by value.
Checking	Indicates if the arguments and/or result is checked for errors.
Argument Type	The type of numbers that are accepted by the routine.
Result Type	The type of numbers that are produced by the routine.
Function Type	The classification of the math routine.
FORTRAN Function Name	The FORTRAN symbolic names which cause the FCL math routine to be executed.

The codes used to indicate the argument type and result type are:

FL	Floating-point
FI	Fixed-point
D	Double-precision
C	Complex
A	Any type

The codes used to indicate the function type are:

Ext.	Externally callable FORTRAN 5 routines
Aux.	Auxiliary routine (not called directly by a FORTRAN function call)
Expon.	Exponentiation routine (compiler-generated call to perform exponentiation)
Sub.	Subroutine

TABLE A-1. MATH ROUTINES

FCL Routine Name	Entry Point	Type of CALL	Checking	Argument Type	Result Type	Function Type	FORTRAN Function Name
ABS	ABS IABS	Name	No	FL	FL	Ext.	ABS IABS
ACOSIN.	ASIN ACOS ASIN. ACOS.	Name Name Value Value	Yes Yes Yes Yes	FL FL FL FL	FL FL FL FL	Ext. Ext. Ext. Ext.	ACOS ASIN
AIMAG	AIMAG	Name	No	C	FL	Ext.	AIMAG
AINT	AINT	Name	No	FL	FL	Ext.	AINT
ALOG	ALOG ALOG10 ALOG. ALOG10.	Name Name Value Value	Yes Yes Yes Yes	FL FL FL FL	FL FL FL FL	Ext. Ext. Ext. Ext.	ALOG ALOG10
AMAXØ	AMAXØ	Name	No	(FI,FI,...)	FL	Ext.	AMAXØ
AMAX1	AMAX1	Name	No	(FL,FL,...)	FL	Ext.	AMAX1
AMINØ	AMINØ	Name	No	(FI,FI,...)	FL	Ext.	AMINØ
AMIN1	AMIN1	Name	No	(FL,FL,...)	FL	Ext.	AMIN1
AMOD	AMOD	Name	No	(FL,FL)	FL	Ext.	AMOD
AND	AND	Name	No	(A,A,...)	A	Ext.	AND
ATAN	ATAN ATAN.	Name Value	Yes Yes	FL FL	FL FL	Ext. Ext.	ATAN

TABLE A-1. MATH ROUTINES (Continued)

FCL Routine Name	Entry Point	Type of CALL	Checking	Argument Type	Result Type	Function Type	FORTRAN Function Name
ATANH.	ATANH	Name	Yes	FL	FL	Ext.	ATANH
	ATANH.	Value	Yes	FL	FL	Ext.	
ATAN2	ATAN2	Name	Yes	(FL,FL)	FL	Ext.	ATAN2
	ATAN2.	Value	Yes	(FL,FL)	FL	Ext.	
CABS.	CABS	Name	Yes	C	FL	Ext.	
	CABS.	Value	Yes	C	FL	Ext.	
CCOS	CCOS	Name	Yes	C	C	Ext.	CCOS
CEXP	CEXP	Name	Yes	C	C	Ext.	CEXP
CEXP.	CEXP.	Value	Yes	C	C	Ext.	CEXP
CLOG	CLOG	Name	Yes	C	C	Ext.	CLOG
CLOG=	CLOG.	Value	No	C	C	Ext.	CLOG
COMPL	COMPL	Name	No	A	A	Ext.	COMPL
CONJG	CONJG	Name	No	C	C	Ext.	CONJG
COS.SIN	COS.SIN	Value	No	FL	(FL,FL)	Aux.	-
COUNT	COUNT	Name	No	A	FI	Ext.	
CMPLX	CMPLX	Name	No	(FL,FL)	C	Ext.	CMPLX
CSIN	CSIN	Name	Yes	C	C	Ext.	CSIN
CSNCS.	CSIN.	Value	Yes	C	C	Ext.	CSIN
	CCOS.	Value	Yes	C	C	Ext.	CCOS
CSQRT	CSQRT	Name	Yes	C	C	Ext.	CSQRT
CSQRT=	CSQRT.	Value	No	C	C	Ext.	CSQRT
DABS	DABS	Name	No	D	D	Ext.	DABS
DASNCS.	DASIN	Name	Yes	D	D	Ext.	DASIN
	DACOS	Name	Yes	D	D	Ext.	DACOS
	DASIN.	Value	Yes	D	D	Ext.	
	DACOS.	Value	Yes	D	D	Ext.	
DATAN	DATAN	Name	Yes	D	D	Ext.	DATAN
DATAN.	DATAN.	Value	Yes	D	D	Ext.	DATAN
DATAN2	DATAN2	Name	Yes	(D,D)	D	Ext.	DATAN2
DATAN2.	DATAN2.	Value	Yes	(D,D)	D	Ext.	DATAN2
DATCOM.	DATCOM.	Value	No	-	-	Aux.	-
	DTN.	Value	No	-	-	Aux.	-
	ATN.	Value	No	-	-	Aux.	-
DBLE	DBLE	Name	No	FL	D	Ext.	DBLE
DIM	DIM	Name	No	(FL,FL)	FL	Ext.	DIM
DMAX1	DMAX1	Name	No	(D,D,...)	D	Ext.	DMAX1
DMIN1	DMIN1	Name	No	(D,D,...)	D	Ext.	DMIN1
DSIGN	DSIGN	Name	No	(D,D)	D	Ext.	DSIGN

TABLE A-1. MATH ROUTINES (Continued)

FCL Routine Name	Entry Point	Type of CALL	Checking	Argument Type	Result Type	Function Type	FORTRAN Function Name
DTAN.	DTAN.	Value	Yes	D	D	Ext.	DTAN
DTANH	DTANH	Name	Yes	D	D	Ext.	DTANH
DTANH.	DTANH.	Value	Yes	D	D	Ext.	DTANH
DTOD*	DTOD$	Name	Yes	(D,D)	D	Expon.	†
DTOD.	DTOD.	Value	No	(D,O)	D	Expon.	†
DTOI*	DTOI$	Name	Yes	(D,FI)	D	Expon.	†
DTOI.	DTOI.	Value	No	(D,FI)	D	Expon.	†
DTOX*	DTOX$	Name	Yes	(D,FL)	D	Expon.	†
DTOX.	DTOX.	Value	No	(D,FL)	D	Expon.	†
DTOZ*	DTOZ$	Name	Yes	(D,C)	C	Expon.	†
DTOZ.	DTOZ.	Value	No	(D,C)	C	Expon.	†
ERF.	ERF	Name	Yes	FL	FL	Ext.	ERF
	ERFC	Name	Yes	FL	FL	Ext.	ERFC
	ERF.	Value	Yes	FL	FL	Ext.	
	ERFC.	Value	Yes	FL	FL	Ext.	
EXP	EXP	Name	Yes	FL	FL	Ext.	EXP
	EXP.	Value	Yes	FL	FL	Ext.	
FLOAT	FLOAT	Name	No	FI	FL	Ext.	FLOAT
HYP.	SINH	Name	Yes	FL	FL	Ext.	SINH
	COSH	Name	Yes	FL	FL	Ext.	COSH
	SINH.	Value	Yes	FL	FL	Ext.	
	COSH.	Value	Yes	FL	FL	Ext.	
HYPERB.	HYPERB.	Value	No	FL	(FL,FL)	Aux.	-
IDIM	IDIM	Name	No	(FI,FI)	FI	Ext.	IDIM
INT	INT	Name	No	FL	FI	Ext.	INT
	IDINT						IDINT
	IFIX						IFIX
ISIGN	ISIGN	Name	No	(FI,FI)	FI	Ext.	ISIGN
	SIGN						SIGN
DCOS	DCOS	Name	Yes	D	D	Ext.	DCOS
DCOSH	DCOSH	Name	Yes	D	D	Ext.	DCOSH
DEULER.	DEULER.	Value	No			Aux.	-
DEXP	DEXP	Name	Yes	D	D	Ext.	DEXP
DEXP.	DEXP.	Value	Yes	D	D	Ext.	DEXP
DHYP.	DSINH.	Value	Yes	D	D	Ext.	DSINH
	DCOSH.	Value	Yes	D	D	Ext.	DCOSH
DLOG	DLOG	Name	Yes	D	D	Ext.	DLOG
DLOG.	DLOG.	Value	No	D	D	Ext.	DLOG
	DLOG10.	Value	No	D	D	Ext.	DLOG10

† The Compiler generates the call for FORTRAN statements which perform exponentiation with certain number types.

TABLE A-1. MATH ROUTINES (Continued)

FCL Routine Name	Entry Point	Type of CALL	Checking	Argument Type	Result Type	Function Type	FORTRAN Function Name
DLOG10	DLOG10	Name	Yes	D	D	Ext.	DLOG10
DMOD	DMOD	Name	Yes	(D,D)	D	Ext.	DMOD
DMOD=	DMOD.	Value	No	(D,D)	D	Ext.	DMOD
DSIN	DSIN	Name	Yes	D	D	Ext.	DSIN
DSINH	DSINH	Name	Yes	D	D	Ext.	DSINH
DSNCOS.	DSIN. DCOS.	Value Value	No No	D D	D D	Ext. Ext.	DSIN DCOS
DSQRT	DSQRT	Name	Yes	D	D	Ext.	DSQRT
DSQRT.	DSQRT.	Value	Yes	D	D	Ext.	DSQRT
DTAN	DTAN	Name	Yes	D	D	Ext.	DTAN
ITOD*	ITOD$	Name	Yes	(FI,D)	D	Expon.	†
ITOD.	ITOD.	Value	No	(FI,D)	D	Expon.	†
ITOJ*	ITOJ$	Name	Yes	(FI,FI)	FI	Expon.	†
ITOJ.	ITOJ.	Value	Yes	(FI,FI)	FI	Expon.	†
ITOX*	ITOX$	Name	Yes	(FI,FL)	FL	Expon.	†
ITOX.	ITOX.	Value	No	(FI,FL)	FL	Expon.	†
ITOZ*	ITOZ$	Name	Yes	(FI,C)	C	Expon.	†
ITOZ.	ITOZ.	Value	No	(FI,C)	C	Expon.	†
LOCF	LOCF	Name	No	A	FI	Ext.	LOCF
MASK	MASK	Name	Yes	FI	A	Ext	MASK
MAXØ	MAXØ	Name	No	(FI,FI,...)	FI	Ext.	MAXØ
MAX1	MAX1	Name	No	(FL,FL,...)	FI	Ext.	MAX1
MINØ	MINØ	Name	No	(FI,FI,...)	FI	Ext.	MINØ
MIN1	MIN1	Name	No	(FL,FL,...)	FI	Ext.	MIN1
MOD	MOD	Name	No	(FI,FI)	FI	Ext.	MOD
OR	OR	Name	No	(A,A,...)	A	Ext.	OR
RANF	RANF RANGET	Name	No No	A FL	FL -	Ext. Sub.	RANF
RANSET	RANSET	Name	No	FL	-	Sub.	RANSET
REAL	REAL SNGL	Name	No	C	FL	Ext.	REAL SNGL
SHIFT	SHIFT	Name	No	(A,FI)	A	Ext.	SHIFT
SINCOS.	SIN COS SIN. COS.	Name Name Value Value	Yes Yes Yes Yes	FL FL FL FL	FL FL FL FL	Ext. Ext. Ext. Ext.	SIN COS

† The Compiler generates the call for FORTRAN statements which perform exponentiation with certain number types.

TABLE A-1. MATH ROUTINES (Continued)

FCL Routine Name	Entry Point	Type of CALL	Checking	Argument Type	Result Type	Function Type	FORTRAN Function Name
SINCSD.	SIND	Name	Yes	FL	FL	Ext.	SIND
	COSD	Name	Yes	FL	FL	Ext.	COSD
	SIND.	Value	Yes	FL	FL	Ext.	
	COSD.	Value	Yes	FL	FL	Ext.	
SQRT	SQRT	Name	Yes	FL	FL	Ext.	SQRT
SQRT.	SQRT.	Value	Yes	FL	FL	Ext.	SQRT
SYS=AID	SYSAID.					Aux.	-
SYS=1ST	SYS1ST.					Aux.	-
	MORGUE.					Aux.	-
TAN	TAN	Name	Yes	FL	FL	Ext.	TAN
TAN.	TAN.	Value	Yes	FL	FL	Ext.	TAN
TAND.	TAND	Name	Yes	FL	FL	Ext.	TAND
	TAND.	Value	Yes	FL	FL	Ext.	
TANH	TANH	Name	Yes	FL	FL	Ext.	TANH
TANH.	TANH.	Value	Yes	FL	FL	Ext.	TANH
XOR	XOR	Name	No	(A,A,...)	A	Ext.	XOR
XTOD*	XTOD$	Name	Yes	(FL,D)	D	Expon.	†
XTOD.	XTOD.	Value	No	(FL,D)	D	Expon.	†
XTOI*	XTOI$	Name	Yes	(FL,FI)	FL	Expon.	†
XTOI.	XTOI.	Value	Yes	(FL,FI)	FL	Expon.	†
XTOY*	XTOY$	Name	Yes	(FL,FL)	FL	Expon.	†
XTOY.	XTOY.	Value	No	(FL,FL)	FL	Expon.	†
XTOZ*	XTOZ$	Name	Yes	(FL,C)	C	Expon.	†
XTOZ.	XTOZ.	Value	No	(FL,C)	C	Expon.	†
ZTOI*	ZTOI$	Name	Yes	(C,FI)	C	Expon.	†
ZTOI.	ZTOI.	Value	No	(C,FI)	C	Expon.	†

† The Compiler generates the call for FORTRAN statements which perform exponentiation with certain number types.

ERROR RECOVERY

All FORTRAN common library routines that check arguments and issue error messages allow for standard and non-standard error recovery, as described in the FORTRAN 5 Reference Manual. The structure of these routines satisfies:

Word 1: VFD 42/,< routine's name >, 18/ < relative position of entry point >

When executing under traceback mode, register A0 holds the field length when in the main program. Otherwise, it has the first-word-address of the parameter list in the previous call. In normal execution, each routine must save the contents of A0 before using this register, and before calling any other routine. A0's contents must be restored upon return to the calling routine.

The symbols SYSARG. and SYSERR. are two entry points in the FORTRAN Common Library utility package FORSYS.. A call at SYSARG. with a bad argument (i.e., negative, zero, infinite or indefinite) in X1 returns with X2 holding the address of the text of an appropriate error message. A call to SYSERR. with an error number in X1 and the address of a diagnostic message in X2 prints the diagnostic message and a traceback listing, provided that the first two words of each routine are as above, the return jump to SYSERR. is in the upper half of a word, and the lower 18 bits contain a pointer from word 1 to the return jump.

The sequence of events on executing math library routines which issue diagnostic messages is:

(a.) Enter routine.

(b.) Check arguments. If valid, compute result and return through entry point. (Some routines also check the result before return.) If invalid, go to (c.).

(c.) Enter contents of register A0 in TEMPA0. and enter the first-word-address of the parameter list (now in A1) into A0.

(d.) Call SYSARG. to obtain the address of an error message in X2 if the argument is infinite, indefinite, zero, or negative; in this case, go to (f.).

(e.) Otherwise, enter the address of an appropriate error message directly into register X2.

(f.) Enter the error number into X1. (See the FORTRAN 5 Reference Manual.) (Step (f.) can precede step (d.).)

(g.) Return to SYSERR. to initiate error actions. (Lower part of RJ word = trace pointer.) If non-standard error recovery is specified through a previous call to SYSTEMC, transfer will return to the supplied recovery routine. If standard error was inhibited, the job aborts. Otherwise, control will return to the calling routine, at step (h.).

(h.) The appropriate indefinite or infinite quantity is entered into X6, and the contents of A0 are restored from TEMPA0..

(i.) Return through the entry point.

A list of error numbers and diagnostic messages is given in the FORTRAN 5 Reference Manual.

Some routines listed in appendix A now detect errors and issue messages for all bad arguments passed to them. These routines call routines SYS=AID or SYS=1ST at entry points SYSAID. or SYS1ST., respectively, for error processing. The sequence of events on executing these routines is:

(a.) Enter routine.

(b.) Check arguments. If valid, compute result and return through entry point. (Some routines also check the result before return.) If invalid, go to (c.).

(c.) Set B2 with pointers indicating error number, partial message, and register residence of bad argument. The format is given in the method description of routine SYS=1ST. The partial message will be ignored if the argument is infinite or indefinite.

(d.) Set up the arguments in registers X1, X2, X3 and X4 (or just X1, X2 if one argument) according to the rules in the Introduction.

(e.) Call SYS1ST. or SYSAID. to initiate error processing. SYSAID. must be chosen if there is more than one argument. The return jump must be in the upper 30 bits of a word. The next 12 bits are zero, and the next 18 bits must include a pointer to a trace word, as described above.

(f.) Testing commences. A parameter list is built up from values in X1, X2, X3, X4 to allow non-standard error recovery. If the routine calling the routine calling SYS=AID made this call in the format:

```
+ RJ   =X<routine>
- VFD  30/1
```

go to step g below. Otherwise, set A0 to point to the reconstructed parameter list. Set X1 to the error number, X2 to the first-word-address of the constructed message, and execute the communication cell SYSAID., after traceback linkage information has been inserted in its lower 18 bits.

(g.) Return POS.INDEF. in registers X6 and X7, and restore registers A0, X1, X2 (and X3 and X4, if entry was to SYS=AID).

TIMING OF ROUTINES

The external and intrinsic math routines were timed. As many arguments as possible were chosen to cover all the possibilities for times for each routine.

CYBER 76 timing was done using the machine instruction 01610, which accesses a hardware clock. CYBER 72, 73, and 74 timing was done by observing variations in speed of two equivalent loops in central memory, one of which called the routine being timed. These variations in speed were obtained by using a system-maintained real-time clock which was synchronized with a hardware clock on one of the data channels. These times do not include time for setting up arguments and parameter lists; they represent the time which elapses between the jump to the routine and execution of the next instruction in sequence.

The timing information is summarized in table C-1. The times given are in minor cycles, or clock periods. On the CYBER 72, 73, and 74, one minor cycle equals 100 nanoseconds. On the CYBER 76, one clock period equals 27.5 nanoseconds. On the CYBER 171, 172, and 173 one minor cycle equals 50 nanoseconds.

On the CYBER 76, a return jump can be delayed if the instruction stack control has requested one or more instruction words that have not arrived at the instruction stack. Therefore, CYBER 76 routine times depend upon how the routine is called.

On the CYBER 72 and 73, a floating-point instruction executes at least 48 minor cycles faster if one of the operands is zero, infinite, or indefinite. If, while an algorithm is being executed, a routine happens to produce an intermediate zero result, it will execute faster by at least 48 minor cycles if this result is combined arithmetically with anything else.

Some routines will call others; however, the times listed in table C-1 represent only the time spent at the specified entry point in the routine. To find the total execution time for a particular computation, first determine the routine and entry point which is initially called. Then find the time for that routine and entry point in table C-1. If other routines are called, an ampersand will appear in the table followed by an entry point name. Locate that entry point in table C-1, and add that time to the total, and so on.

The timings are for valid argument sets only. Use the time for the first alternative which covers the argument concerned. Times do not include the time spent in jumps to other routines, but represent only the time actually spent in a specific routine.

TABLE C-1. TIMING OF ROUTINES

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
ABS					
ABS (any valid)		100	79	58	66
ACOSIN.					
ACOS (x)			56	35	47
x valid					
& ACOS. (x)					
ASIN (x)					
x valid			59	35	41
& ASIN. (x)					
ACOS. (x)					
x valid and:					
x = 0.	812		741	159	119
x = 1.	306		234	127	85
x = -1.	307		237	127	87
x in (-.5,.5)	950		897	159	116
x not in (-.5,.5), time = a*b*n where n is the loop count, as defined in the ACOSIN. description.					
a =			1138	207	147
b =			114	18	12
x in (-1.,-.5), add to I x I time:			10	5	4
ASIN. (x)					
x valid and:					
x = 0.	823		763	153	123
x = 1.	292		220	120	87
x = -1.	293		219	120	90

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
x in (-.5,.5)	958		904	152	126
|x| in (.5,1.), time = a*b*n, where n is defined in the ACOSIN. description.					
a =			1170	226	168
b =			115	15	12
AIMAG					
AIMAG (any valid)		101	81	54	62
AINT					
AINT (any valid)		121	98	66	60
ALOG					
ALOG10 (x) & ALOG10. (x)			67	42	46
ALOG (x) & ALOG. (x)			67	40	49
ALOG10. (x)					
x infinite or indefinite & SYSAID.			253	192	110
0. & SYSAID.			266	199	120
x valid,x<0. & SYSAID.			285	213	129
x valid, x = $y*2^n$ n integral, 1≤y<2, and					
1≤y<1.1072	860		892	179	129
1.1072≤y<1.3572	860		892	176	129
1.3572≤y<1.6072	861		891	177	128
1.6072≤y<1.8572	860		892	179	129
1.8572≤y<2	990		1012	212	141
ALOG. (x)					
x infinite or indefinite & SYSAID.			298	176	97
0. & SYSAID.			311	183	112
x valid,x>0 & SYSAID.			330	192	117
x valid,x= $y*2^n$ n integral					
1.≤y<1.8572	941		814	197	119
1.8572<2	1072		933	218	143
AMAX0					
AMAX0 (x(1),..., x(n))					
n=2		240	178	121	112
n=3		338	250	159	140
each add.		99	73	42	32
AMAX1					
AMAX1 (x(1),..., x(n))					
n=2		232	178	104	106
each add.		110	83	45	34
AMIN0					
AMIN0 ((x(1)) ..., x(n)					
n=2		237	179	112	105
n=3		338	252	148	133
each add.		100	72	43	32

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
AMIN1					
AMIN1 (x(1),..., x(n))					
n=2		227	179	108	105
n=3		333	252	163	142
n=4		436	328	189	172
each add.		105	78	44	34
AMOD					
AMOD (x,y)					
y≠0		248	207	111	133
AND					
AND (x(1),..., x(n))					
n=2		217	163	103	103
n=3		282	212	112	118
n=4		347	262	133	141
each add.		65	49	22	19
ATAN					
ATAN (x)			66	32	53
& ATAN. (x)					
ATAN. (x)					
x valid | x |<1.	1059		756	187	141
x valid | x |≥1.	1092		784	203	201
ATAN2					
ATAN2 (y,x)			78	53	78
& ATAN2. (x)					
ATAN2. (y,x)					
(y,x) valid and ...					
x=0,y≠0.	898		850	246	190
x≠0,y=0	981		835	276	187
| x |>| y |>0	1167		1085	249	161
| y |≥| x |≥0	1165		1077	241	172
ATANH.					
ATANH (x)					
& ATANH. (x)					
ATANH.(x)					
x valid and:					
x=0	682			203	
.75≤x<1.5	908			202	
x≥1.5					
CABS.					
CABS (z)					
z valid			105	38	43
& CABS. (z)					
CABS. (x+i*y)					
x+i*y valid					
and x=y=0.	276		225	138	85
x≠0. or y≠0. ,					
special case. (See					
routine's description)	715		786	283	197
and otherwise valid	715		684	283	181
CCOS					
CCOS (z)					
z valid		546	436	180	119
& HYPERB. (im(z))					
& COS.SIN (re(z))					
|im(z)| > 741.67		468	348	363	131
& SYSERR.					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
CEXP					
CEXP (z)					
|re(z)| > 741.67		356	273	158	120
& SYSERR.					
z valid		487	403	155	115
& EXP. (re(z)) & COS.SIN (im(z))					
CEXP.					
CEXP. (z)					
z valid		262	225	74	60
& EXP. (re(z)) & COS.SIN (im(z))					
CLOG					
CLOG (z)					
z=0.		291	213	118	76
& SYSARG=SYSERR.					
z valid		163	131	102	69
& CLOG. (z)					
CLOG=					
CLOG. (z)					
z valid		253	199	95	50
& ATAN2. ((im(z), re(z)))					
& CABS. (z) & ALOG. (|z|)					
CMPLX					
CMPLX (x,y)					
x,y valid		126	103	64	84
COMPL					
COMPL (x)		83	69	55	54
CONJG					
CONJG (z)					
z valid		128	101	58	68
-					
COS. -- see SINCOS.					
-					
COSH. -- see HYP.					
COS.SIN					
COS.SIN (x)					
|x| > pi*2^{46}		307	244	108	90
|x|≡y(mod2pi) ,					
0<y<2pi 0≤y≤pi/4	1463	1561	1380	242	215
pi/4≤y≤pi/2	1715	1880	1649	269	234
pi/2<y≤3pi/4	1716	1879	1649	269	234
3pi/4<y≤pi	1734	1885	1655	282	245
pi<y≤5pi/4		1884	1657	323	245
5pi/4<y≤3pi/2		1886	1659	319	244
3pi/2<y≤7pi/4		1887	1658	319	244
7pi/4<y≤2pi	1693	1885	1635	267	232
COUNT					
COUNT (x)		148	133	49	62
CSIN					
CSIN (z)					
|re(z)| > pi*2^{46}		386	295	162	121
& COS.SIN (re(z))					
& SYSERR.					
|im(z)| > 741.67		470	362	221	136
& COS.SIN (re(z))					
& SYSERR.					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
z valid & COS.SIN (re(z)) & HYPERB. (im(z))		551	436	181	123
CSNCS.					
CCOS. (z)					
z valid & HYPERB. (im(z)) & COS.SIN (re(z))		327	279	78	71
CSIN. (z)					
z valid & COS.SIN (re(z)) & HYPERB. (im(z))		315	248	77	79
CSQRT					
CSQRT (z)					
z valid & CSQRT. (z)		153	115	93	67
CSQRT=					
CSQRT. (z)					
z=0. & CABS. (0.) & SQRT. (0.)		287	219	103	58
z valid, z≠0 & CABS. (z) & SQRT. (1/2(|z| * |re(z)|))		477	376	265	90
DABS					
DABS (x)					
x valid		144	111	70	72
DASNCS.					
DASIN & DASIN.					
DACOS & DACOS.					
DACOS.					
$0 \leq x \leq .09375$	3344			529	
$.09375 < x < .7071$	4844			853	
$.701 < x < .9956$	4823			841	
$.9956 < x < 1$	4228			756	
DASIN.					
$0 \leq x \leq .09375$	3260			492	
$.09375 < x < .7071$	4756			814	
$.701 < x < .9956$	4779			820	
$.9956 < x < 1$	4197			736	
DATAN					
DATAN (x)					
x valid & DATAN. (x)			130	42	143
DATAN.					
DATAN.					
x valid, and:					
$\|x\| < 1.$ & DTN. (see routine description)		144	74	40	
$\|x\| \geq 1.$ & DATCOM. (see routine description)		320	134	73	
DATAN2					
DATAN2 (y,x)					
y,x valid, and (y,x)≠(0,0) & DATAN2. ((y,x))		124	46	66	

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
DATAN2.					
DATAN2. (y,x)					
where both are valid, and					
$(y,x) \neq (0,0)$, and:					
$\lvert y \rvert \leq \lvert x \rvert$		276	144	65	
& DATCOM. (see routine description)					
$\lvert y \rvert > \lvert x \rvert$		283	175	71	
& DATCOM. (see routine description)					
DATCOM.					
DATCOM. (y,x) (from DATAN2.)					
argument set validated. If n					
is nearest integer to					
$8*\min(\lvert x \rvert,\lvert y \rvert)/\max(\lvert x \rvert,\lvert y \rvert)$,					
then:					
n=0		3150	521	337	
$n \neq 0$ and $\min(\lvert x \rvert,\lvert y \rvert)-n/8*\max(\lvert x \rvert,\lvert y \rvert) \neq 0$		3735	664	417	
otherwise		3725	663	417	
DTN.					
y (from DATAN.), valid.					
If n is nearest integer to					
8*y, then:					
n=0		2736	451	287	
$n \neq 0$ and $(y - n/8) \neq 0$		3356	587	367	
otherwise		1212	307	200	
DBLE					
DBLE (x)					
x valid		98	78	52	54
DCOS					
DCOS (x)					
x valid		144	121	71	67
& DCOS. (x)					
DCOSH					
DCOSH (x)					
x valid			130	52	45
& DCOSH. (x)					
DEULER.					
DEULER.					
(See description of routine DEULER.)			3719	623	361
DEXP					
DEXP (x)					
x valid			117	45	49
& DEXP.(x)					
DEXP.					
DEXP. (x)					
x valid and:					
$x < -643.2405835596292471391914 09$:			515	163	107
& DEULER.					
x otherwise:			378	147	100
& DEULER.					
DHYP.					
DCOSH. (x)					
x valid and:					
$\text{abs}(x) > 42.3606303797014263858556 02079$:			560	215	127
& DEULER.					
$\text{abs}(x)/\log 2 \geq 48$:			546	182	101
& DEULER.					
$\text{abs}(x)/\log 2 \geq 24$:			658	203	125
& DEULER.					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine / Entry Points / Arguments / & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
x in [-1/2 log 2,1/2 log 2]:			233	136	86
& DEULER.					
x otherwise			719	229	132
& DEULER.					
DSINH. (x)					
x valid and:					
abs(x) > 42.360630379701426385855602079:			575	202	119
& DEULER.					
abs(x)/log 2 ≥ 48:			515	160	93
& DEULER.					
abs(x)/log 2 ≥ 24:			625	206	136
& DEULER.					
x in [-1/2 log 2,1/2 log 2]:			155	94	64
& DEULER.					
x otherwise			720	226	134
& DEULER.					
DIM					
DIM (x,y)					
x,y valid		191	150	84	96
DLOG.					
DLOG10. (x)					
$x=(2^n)*y$					
$1/2 \le y < 1/2^{.5}$	7104	7931	6946	1220	761
$1/2^{.5} < y < 1$	6962	7799	6802	1221	762
DLOG. (x)					
$x=(2^n)*y$					
$1/2 \le y < 1/2^{.5}$	6797	7576	6631	1158	731
$1/2^{.5} \le y < 1$	6636	7444	6487	1144	731
DLOG					
DLOG (x)					
x=0.		284	215	136	83
& SYSARG. SYSERR.					
x<0		332	251	142	105
& SYSARG. SYSERR.					
x valid		150	96	101	68
& DLOG. (x)					
DLOG10					
DLOG10 (x)					
x=0.		284	216	130	89
& SYSARG. SYSERR.					
x, x<0		333	255	216	105
& SYSARG. SYSERR.					
x valid		177	144	97	68
& DLOG10. (x)					
DMAX1					
DMAX1 (x(1),x(2))		909	675	320	135
DMIN1					
DMIN1 (x(1),x(2))		863	644	310	134
DMOD					
DMOD (x,y)					
x valid, y=0		332	243	137	77
& SYSARG. SYSERR.					
(x,y) valid		266	203	34	97
& DMOD. (x,y)					
DMOD=					
DMOD. (x,y)					
x,y valid, y≠0 $\vert x/y\vert \ge 2^{96}$	2007			582	
$\vert x/y\vert \le 2^{48}$	1426			431	
$\vert x/y\vert < 2^{48}$	841			281	

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
DSIGN					
DSIGN (x,y)					
x,y any		205	157	81	101
DSIN					
DSIN (x)					
x valid		162	102	83	76
& DSIN. (x)					
DSINH					
DSINH (x)					
x valid			124	52	43
& DSINH. (x)					
DSNCOS.					
DCOS. (x)					
$\lvert x \rvert > pi*2^{94}$		605	501	181	129
x≡y(mod2pi), 0≤y≤2pi					
0≤y<pi/4,	4671	5129	4475	778	516
pi/4≤y<pi/2,	5140	5703	4971	844	563
pi/2≤y<3pi/4,	5140	5703	4971	846	563
3pi/4≤y<pi,	5059	5679	4904	851	558
pi≤y<5pi/4,		5658	4923	920	558
5pi/4≤y<3pi/2,		5703	4980	908	563
3pi/2≤y<7pi/4		5722	4971	909	563
7pi/4≤y<2pi	5063	5677	4904	850	563
DSIN. (x)					
$\lvert x \rvert > pi*2^{94}$		624	511	181	137
x≡y(mod2pi), 0≤y≤2pi,					
0≤y<pi/4,	4750	5093	4446	786	520
pi/4≤y<pi/2,	5078	5695	4933	867	566
pi/2≤y<3pi/4,	5083	5689	4904	864	571
3pi/4≤y<pi,	5139	5715	4971	856	575
pi≤y<5pi/4,		5715	4980	924	571
5pi/4≤y<3pi/2,		5689	4933	934	566
3pi/2≤y<7pi/4		5687	4904	935	566
7pi/4≤y≤2pi	5141	5718	4980	853	571
DSQRT					
DSQRT (x)					
x<0.		282	234	125	85
& SYSARG. SYSERR.					
x valid		140	107	93	60
& DSQRT. (x)					
DSQRT.					
DSQRT. (x)					
x=0.					
$x=y*2^n$					
n odd	745			228	
n even	746			231	
DTAN					
DTAN					
& DTAN.					
DTAN.					
DTAN.					
x valid and:					
x=0	2371			579	
pi/4<x<pi/4	3247			579	
pi/4<x<3pi/4	3663			639	
3pi/4<x<5pi/4, etc.	3474			633	
5pi/4<x<7pi/4, etc.	3666			638	

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DTANH					
DTANH (x)					
x valid			124	120	42
& DTANH. (x)					
DTANH.					
DTANH. (x)					
x valid and:			765	217	134
$\lvert x \rvert < 1/8$:					
& DEULER.(x)					
$\lvert x \rvert \geq 32$:			214	103	62
If x (or 2x)=y+n*log(2), n>47:			619	163	122
& DEULER. (2x)					
Otherwise:					
& DEULER. (2x)			1055	311	171
DTOD*					
DTOD$ (x,y)					
(0.,0.)		441	341	192	158
& SYSERR.					
(0,y), to y>0		352	387	153	208
(0,y), to y<0		439	340	195	152
& SYSERR.					
x<0		410	318	169	130
& SYSERR.					
(x,y) valid		863	740	236	138
& DLOG. (x)					
& DEXP. (y*log x)					
DTOD.					
DTOD. (x,y)					
(0,y), y>0		114	63	66	62
x>0, x,y valid		517	466	113	79
& DLOG. (x)					
& DEXP. (y*log x)					
DTOI*					
DTOI$ (x,n)					
(0.,0)		404	312	189	136
& SYSERR.					
(0.,n),n<0		418	311	195	142
& SYSERR.					
(0.,n),n>0		230	188	123	192
x>0		264	231	110	69
& DTOI. (x,n)					
DTOI.					
DTOI. (x,n)					
if n<0, add, and replace		467	415	114	73
n with -n					
(x,0)		83	65	51	51
(x,1)		364	301	126	90
(x,2)		672	575	190	128
if n>2,time=t, a(1)+b(1)$\leq$					
log(2)n$\leq$+$\leq$a(2)/+b(2)log(2)n					
a(1)=		380	316	227.9	94.3
a(2)=		69.3	14.5	111.6	105
b(1)=		292.	257.	38.8	24.2
b(2)=		530	489	41.9	33.6
DTOX*					
DTOX$ (x,y)					
(0.,0)		426	337	199	184
& SYSERR.					
(0.,y),y<0		299	239	245	228
(0.,y).y<0		426	335	107	178
& SYSERR.					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
x<0		383	299	176	145
& SYSERR.					
(x,y) valid, x>0		708	606	236	158
& DLOG. (x)					
& DEXP. (y*log 2)					
DTOX.					
DTOX. (x,y)					
(0.,y)		95	74	59	54
(x,y) valid		460	415	85	63
& DLOG. (x)					
& DEXP. (y*log 2)					
DTOZ*					
DTOZ$ (x,z)					
(0.,0.+i.0.)		403	311	189	148
& SYSERR.					
x<0		342	263	168	117
& SYSERR.					
(0.,z), re(z)≥0		277	211	102	136
& SYSERR,					
(0.,z), re(z)<0 im(z)≠0		432	312	221	136
&SYSERR.					
(0.,z), re(z)<0 im(z)=0		432	312	223	136
& SYSERR.					
(x,z) valid		762	636	231	90
& ALOG. (x)					
& EXP. (re(z)*log x)					
& COS.SIN (im(z)*log x)					
DTOZ.					
DTOZ. (x,z)					
x=0.		103	81	63	59
x,z valid, x≠0		480	426	149	85
& ALOG. (x)					
& EXP. (re(z)*log x)					
& COS.SIN (im(z)*log x)					
ERF.					
ERF					
& ERF					
ERFC					
& ERFC					
ERF. (x)					
x<-5.625 or -inf	526			189	
-5.625<x<-.477	3094			489	
-.477≤x<0	1172			234	
x=0	904			230	
0<x≤.477	1172			235	
.477<x≤5.625	3090			495	
x>5.625 or +inf	527			185	
ERFC. (x)					
x<-5.625 or -inf	588			213	
-5.625<x<-.477	3155			518	
-.477≤x<0	1234			255	
x=0	965			252	
0<x≤.477	1234			253	
.477<x≤8	3154			513	
x>8					
x infinite					
EXP					
EXP (x)			34	57	38
& EXP. (x)					
EXP. (x)					
x infinite			268	140	89
& SYSAID.					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine / Entry Points / Arguments / & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
x indefinite			201	103	58
& SYSAID.					
x valid,x>741.67			304	155	97
& SYSAID.					
x valid x≥512.	932		864	184	130
x valid,x<-675.84			298	157	119
& SYSAID.					
x valid,x<-512	931		865	182	140
x valid	843		804	145	112
FLOAT					
FLOAT (x)					
x valid		102	82	65	56
HYP.					
COSH. (x)					
x valid					
|x|<1/2 log 2	1296		1313	233	164
x otherwise valid	1385		1426	233	167
SINH. (x)					
x valid					
|x|<1/2 log 2	1325		1351	250	178
x otherwise valid	1457		1498	257	177
COSH (x)					
x valid			1495	283	200
SINH (x)					
x valid			1559	306	214
HYPERB.					
HYPERB. (x)					
x valid, |x|<.22	1649	1772	1540	347	265
x valid, |x|≥.22		398	311	136	95
& EXP. (x)					
IDIM					
IDIM (x,y)					
(x,y) valid		163	127	85	103
INT					
IFIX (x)					
x valid		101	81	59	56
INT					
IDINT					
ISIGN					
ISIGN (x,y)		161	125	75	96
ITOD*					
ITOD$ (n,x)					
(0.,0.)		365	337	164	132
& SYSERR.					
(0,x),x<0		582	483	166	123
& SYSERR.					
(0,x),x>0		496	451	109	173
n<0		322	153	128	92
& SYSERR.					
n>0, x*log n overflows		695	584	238	914
& SYSERR.					
& DLOG. (n)					
(n,x) valid, n>0		598	613	264	125
& DLOG. (n)					
& DEXP. (x*log n)					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine / Entry Points / Arguments / & Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
ITOD.					
ITOD. (n,x)					
(0.,x)		146	110	78	64
(n,x) valid, n>0		457	397	98	80
& DLOG. (n)					
& DEXP. (x*log n)					
ITOJ*					
ITOJ$ (m,n)					
& ITOJ. (m,n)					
ITOJ.					
ITOJ. (m,n)					
$m^n<2^{48}$					
(m,0),m valid	181			95	
(m,1),m valid	218			131	
(m,2),m valid	283			139	
if n>2,m>1, look at n in binary:					
for each 1 bit, add					
for each 0 bit, add					
ITOX*					
ITOX$ (n,x)					
(0,0.)		389	267	175	158
& SYSERR.					
(0,x),x>0		352	313	114	208
(0,x),x<0		346	268	178	149
& SYSERR.					
n<0		289	223	136	102
& SYSERR.					
n>0, | x*logn |≥741.67		459	354	246	122
& ALOG. (n)					
& SYSERR.					
(n,x) valid		315	237	245	95
& ALOG. (n)					
& EXP. (x*log n)					
ITOX.					
ITOX. (n,x)					
(0,x)		113	85	66	62
(n,x) valid n>0		215	185		64
& ALOG. (n)					
(n,z) valid					
& ALOG. (n)		113	85	175	
& EXP. (x*log n)					
ITOZ*					
ITOZ$ (n,z)					
(0,0.+10.)		376	291	165	129
& SYSERR.					
(0,z), re(z) < 0,im(z)=0		395	287	199	120
& SYSERR.					
(0,z),re(z)>0		238	187	210	91
im(z)=0 (0,z),im(z)≠0,		376	291	165	120
& SYSERR.					
re(z)<0 (n,z),n<0		316	241	144	104
& SYSERR.					
(n,z) valid		632	515	211	139
& ALOG. (n)					
& COS.SIN (im(z)*log n)					
& EXP. (re(z)*log n)					
ITOZ.					
ITOX. (n,z)			84	42	24
& XTOZ. (n,z)					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine / Entry Points / Arguments / & Times at Entry Points (argument)	Times for CYBER				
	173	72	73	74	76
LOCF					
LOCF (x)		72	60	46	49
MAX0					
MAX0 (x(1),..., x(n))					
n=2		222	168	105	113
n=3		324	240	148	134
n=4		422	314	191	166
each additional argument		100	73	43	31
MAX1					
MAX1 (x(1)..., x(n))					
n=2		249	187	111	111
n=3		357	270	157	141
n=4		467	355	202	175
each additional argument		110	83	45	34
MASK					
MASK (n)					
n>60		263	207	111	91
& SYSERR.					
n<0		274	210	127	83
& SYSERR.					
n valid		181	133	103	87
MIN0					
MIN0 (x(1),..., x(n))					
n=2		228	169	105	102
n=3		328	241	148	130
n=4		429	312	191	162
each additional argument		100	72	43	28
MIN1					
MIN1 (x(1),..., x(n))					
n=2		242	182	110	107
n=3		347	259	155	137
n=4		454	337	199	171
each additional argument		105	77	44	38
MOD					
MOD (x,y)					
(x,y) valid		316	268	114	133
OR					
OR (x(1),..., x(n))					
n=2		209	161	103	88
n=3		274	210	124	106
n=4		335	258	145	130
each additional argument		63	48	21	20
RANF					
RANF (anything)		189	165	63	80
RANGET (x)		96	79	67	66
x will be modified					
RANSET					
RANSET (x)		176	134	89	82
REAL					
REAL (u)					
SNGL (u)		83	69	55	54
u valid					
SHIFT					
SHIFT (u,n)					
n valid		128	104	60	86

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)			Times for CYBER		
	173	72	73	74	76
SINCOS.					
SIN (x)			64	34	42
& SIN. (x)					
COS (x)			64	34	42
& COS. (x)					
SIN. (x)					
x infinite or indefinite			169	115	75
& SYSAID.					
x=0.	888		821	193	141
x valid, $\lvert x\rvert>pi*2^{46}$			166	109	79
& SYSAID.					
x valid, $\lvert x\rvert\leq pi*2^{46}$	1283		1256	194	141
COS. (x)					
x infinite or indefinite			169	115	75
& SYSAID.					
0.	831		757	188	165
x valid $\lvert x\rvert\leq pi*2^{46}$	1190		1230	188	178
x valid $\lvert x\rvert>pi*2^{46}$				220	165
& SYSAID.					
SQRT					
SQRT (x)			78	40	37
& SQRT. (x)					
SQRT. (x)					
x infinite, indefinite or negative			222	180	279
& SYSAID. (Append. B)					
x valid,$x\neq0$	527		523	119	101
0.	244		393	196	97
SYS=AID					
SYSAID.					
(1 in lower half of RJ word)			359		133
& SYSERR.					
(other than 1 in lower half of RJ word)			986	423	267
& SYSERR.					
SYS=1ST					
SYS1ST.					
(1 in lower half					
of RJ word			299		106
& SYSERR.					
(other than 1 in					
lower half of RJ word)			892	377	239
& SYSERR.					
TAN					
TAN (x)					
x valid, not an odd multiple of pi/2		216	175	142	109
& TAN. (x)					
TAN.					
TAN. (x)					
x=0	617			155	
$\lvert x\rvert<2^{47}$, x=n(pi/2)*y, pi/4<x<pi/4					
n=0	1065			156	
n odd	1061			147	
n even	1061			157	
TANH					
TANH (x)					
x valid		98	75	65	58
& TANH. (x)					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
TANH.					
TANH. (x)					
x valid and:					
$\mid x \mid \leq .55$	812			153	
$.55 < \mid x \mid \leq 17.1$	970			210	
$\mid x \mid > 17.1$	388			126	
XOR					
XOR (x(1),..., x(n))					
n=2		213	164	96	100
n=3		276	213	117	118
n=4		340	262	139	144
n=5		404	309	160	156
each additional argument		64	49	21	19
XTOD*					
XTOD$ (x,y)					
(0.,0.)		445	341	197	147
& SYSERR.					
(0.,x),x valid x>0		476	389	158	204
x<0		454	343	199	147
&SYSERR.					
(x,y)x<0, x valid		403	304	167	132
& SYSERR.					
x,y valid,x<0, y*logx>741.67		753	606	280	188
& DLOG. (x)					
& SYSERR.					
y*logx<741.67		684	558	239	149
& DLOG. (x)					
& EXP. (y*log x)					
XTOD.					
XTOD. (x,y)					
x=0.		129	99	66	62
(x,y) valid, x=0		406	352	120	79
& DLOG. (x)					
& DEXP. (y*log x)					
XTOI*					
XTOI$ (x,n)					
& XTOI. ((x,n))					
XTOI.					
XTOI. (x,n)					
x valid n valid, n>0 when x=0					
n=0					
n<0,replace n by -n and					
x by 1/x, add:					
n=1					
n=2					
n=3					
n=4					
XTOY*					
XTOY$ (x,y)					
(0.,0.)		283	179	186	155
& SYSERR.					
(0.,x),x valid,x>0		396	330	92	198
(0.,x), x valid,x<0		368	284	189	155
& SYSERR.					
x,y valid, x<0		309	243	157	114
& SYSERR.					
x,y valid, x>0 valid, x≠0		399	315	201	141
& ALOG. (x)					
& EXP. (y*log x)					

TABLE C-1. TIMING OF ROUTINES (Continued)

Routine Entry Points Arguments & Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
XTOY.					
XTOY. (x,z)					
(0,x) valid, x>0		80	62	58	53
(x,y) valid,x≠0		174	150	45	53
& ALOG. (x)					
& EXP. (y*log x)					
XTOZ*					
XTOZ$ (x,z)					
(0.,z)					
z valid,re(z)>0		401	341	135	178
z valid, re(z)<0		398	296	212	121
& SYSERR.					
z valid,re(z)=0		355	294	180	121
& SYSERR.					
x,z valid, x<0		312	240	156	104
& SYSERR.					
x,z valid, x>0 re(z)*log x>741.67		632	469	251	130
& ALOG. (x)					
& SYSERR.					
(x,z) valid, x≠0		705	573	221	85
& ALOG. (x)					
& COS.SIN (im(z)*log x)					
& EXP. (re(z)*log x)					
XTOZ.					
XTOZ. (x,z)					
(0.,z)					
z valid, re(z)>0		82	341	58	55
(x,z) valid, x>0		476	422	94	91
& ALOG. (x)					
& EXP. (re(z)*log x)					
& COS.SIN (im(z)*log x)					
ZTOI*					
ZTOI$ (z,n)					
(0.,0.)		379	303	189	140
& SYSERR.					
(0,x), x>0		232	178	137	111
(0.,x),x<0		369	287	182	61
& SYSERR.					
z≠0, z,n valid		204	181	113	99
& ZTOI. (z,n)					
ZTOI.					
ZTOI. (z,n)					
(z,n) valid n = 0		85	66	51	54
n=1		233	230	115	85
n=2		710	602	179	125
n=3		725	614	178	122
n=-1		656	571	151	118
n=-2		1036	899	215	156
n=-3		1101	955	214	160
If n<0, replace n by -n, and add					
n odd		374	337	46	32
n even		327	291	36	32
If n>3, t=time a(1)+b(1)*log(2)n≤+≤ a(2)+b(2)*log(2))n, where					
a(1)=		477.	295.	142.5	86.9
b(1)=		233.	222.	36.8	55.0
a(2)=		162.	127.	87.9	74.5
b(2)=		390.	337.	62.3	28.1

GLOSSARY

Algorithm Error	Error caused by the mathematical formulas used in an FCL routine.
Argument	A variable or constant that is passed to a routine and used by that routine to compute a function. The actual value of the variable is passed when a routine is called by value; the address of the variable is passed when the routine is called by name.
Argument Set	An ordered list of one or more arguments.
Auxiliary Routine	An FCL routine which is not directly called from FORTRAN code, but assists in the computation of a math library function.
Bit Error	A way of analyzing the magnitude of the relative error of a routine. The number of last place positions that the coefficient parts of the true value and computed value differ from each other.
Call by Name	A method of referencing a subprogram in which the addresses of the arguments are passed.
Call by Value	A method of referencing a subprogram in which the values of the arguments are passed. Modifications of the arguments within the subprogram are not reflected in the calling program.
Dummy Argument	A variable or constant that is passed to a routine, but is not used by the routine to compute a function.
Entry Point	A statement within an FCL routine at which execution can begin. There may be more than one entry point into an FCL routine.
Error	The computed value of a function minus the true value.
Exponentiation Routine	An FCL routine which accepts compiler-generated calls from a source program to perform exponentiation. These calls are generated when a FORTRAN statement involves exponentiation of certain number types. Exponentiation routines are not called directly using FORTRAN function names.
External Routine	A predefined subprogram that accepts calls from FORTRAN code to compute certain mathematical functions.
FORTRAN Function Name	A symbolic name which appears in a FORTRAN program and causes a math routine to be executed. See the FORTRAN 5 Reference Manual for a full description of the available FORTRAN function names.
Input Range	A collection of argument sets for which a given FCL routine will return a meaningful result.
Invalid Form	All bit configurations in words thought to contain numbers which do not represent valid or semivalid forms for a particular number type.
NEG. INDEF.	An abbreviation for the constant 60000000000000000000_8. It represents the negative indefinite semivalid form.
NEG. INF.	An abbreviation for the constant 40000000000000000000_8. It represents the negative infinite semivalid form.
Number Types	A classification of the numbers processed by the math routines. The math routines perform computations on four number types: integer, single-precision floating-point, double-precision floating-point, and complex floating-point.
POS. INDEF.	An abbreviation for the constant 17770000000000000000_8. It represents the positive indefinite semivalid form.
POS. INF.	An abbreviation for the constant 37770000000000000000_8. It represents the positive infinite semivalid form.
Relative Error	The error of a function divided by the true value.
Round-off Error	Error caused by the finite nature of the computer hardware.
Routine, FCL	A computer subprogram, written in assembly language, which computes commonly occurring math functions, and perform other tasks such as input and output.
Semivalid Form	Bit configurations that do not represent numbers, but indicate what erroneous computation produced it.
Valid Form	A bit configuration which represents a number on the real number line or in the complex plane.

REFERENCES

A. Abramowitz and I. Stegun, Handbook of Mathematical Functions, AMS 55.

Control Data Technical Report, Number 52.

J. Hart, E. Cheny et al, Computer Approximations, John Wiley and Sons, 1968.

Hastings, Approximations for Digital Computers, Princeton University Press, 1955.

F. B. Hildebrand, Introduction to Numerical Analysis, McGraw-Hill, 1956.

C. Lanczos, Applied Analysis, Prentice Hall.

H. J. Maehly, Methods for Fitting Rational Approximations, Part I, (J. Assoc. Comp. Mach. 7, pp. 150-162) and Parts II and III (J. Assoc. Comp. Mach. 11, pp. 257-277).

H. S. Wall, Analytic Theory of Continued Fractions, D. Van Nostrand Co., Inc., 1948.

J. H. Wilkinson, Rounding Errors to Algebraic Processes, Prentice-Hall, 1963.

D. E. Knuth, The Art of Computer Programming, Vol. 2.

INDEX

CONTROL DATA CORPORATION

COMMENT SHEET

MANUAL TITLE: FORTRAN 5 Common Library Mathematical Routines Reference Manual

PUBLICATION NO.: 60483100 REVISION: B

NAME: ____________________

COMPANY: ____________________

STREET ADDRESS: ____________________

CITY: ____________ STATE: ____________ ZIP CODE: ____________

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

CUT ALONG LINE

AA3419 REV. 4/79 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

FOLD FOLD

TAPE TAPE