CONTROL DATA CORPORATION

60498200

FORTRAN COMMON LIBRARY MATHEMATICAL ROUTINES REFERENCE MANUAL

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

FORTRAN Common Library Mathematical Routines

Manual Title Reference Manual Pub. No. 60498200 Rev. C

As part of Control Data's continuing quality improvement program, we invite you to complete this questionnaire so that you may have a more direct influence on the manuals you use.

Please rate this manual for each general and individual category on a scale of 1 through 5 as follows:

1 - Excellent 2 - Good 3 - Fair 4 - Poor 5 - Unacceptable

I. Writing Quality _____

- A. Technical accuracy _____
- B. Completeness _____
- C. Audience defined properly _____
- D. Readability _____
- E. Understandability _____
- F. Organization _____

II. Examples _____

- A. Quantity _____
- B. Placement _____
- C. Applicability _____
- D. Quality _____
- E. Instructiveness _____

III. Format _____

- A. Type size _____
- B. Page density _____
- C. Art work _____
- D. Legibility _____
- E. Printing/Reproduction _____

IV. Miscellaneous _____

- A. Index _____
- B. Glossary _____

V. Please provide a yes or no answer regarding manuals in general:

- A. I prefer that a manual on a software product be as comprehensive as possible; physical size is of little importance. _____
- B. I prefer that information on a software product be covered in several small manuals, each covering a certain aspect of the product. Smaller manuals with limited subject matter are easier to work with. _____
- C. I am interested primarily in reference manuals designed for ease of locating specific information. _____
- D. I am interested primarily in user guides designed to teach the user about a product or certain capabilities of a product. _____

VI. We recognize that we have a wide variety of users. Please identify your primary area of interest or activity:

- A. Student _____
- B. Applications programmer _____
- C. Systems programmer _____
- D. How many years programming experience do you have? _____
- E. What languages
  1. Algol _____
  2. Basic _____
  3. Cobol _____
  4. Compass _____
  5. Fortran _____
  6. PL/I _____
  7. Other _____
- F. Have you ever worked on non-CDC equipment? _____
  1. If yes, approximately what percent of your experience is on non-CDC equipment? _____
  2. How do you rate CDC manuals against other similar manuals using the 1-5 ratings. (Example: XYZ Corp. 2 means XYZ manuals are good as compared to CDC manuals.)
     - Burroughs _____
     - DEC _____
     - Hewlett-Packard _____
     - Honeywell _____
     - IBM _____
     - NCR _____
     - Univac _____
     - Other _____ _____

General Comments _____

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE

CONTROL DATA CORPORATION

60498200

FORTRAN COMMON LIBRARY MATHEMATICAL ROUTINES REFERENCE MANUAL

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

REVISION RECORD	
REVISION	**DESCRIPTION**
A	Original release.
(11-01-75)	
B	This revision documents Phase III of feature 79 for FORTRAN Extended Version 4.6: DSINH,
(03-01-76)	DCOSH, and DTANH routines have been added plus minor revisions to the existing routines.
C	This revision documents feature 191 for FORTRAN Extended Version 4.7 and PL/I Version 1.0
(03-31-78)	at PSR level 472. The following routines have been added: DTAN, DASIN, DACOS, ERF, ERFC,
	ATANH, SIND, COSD, and TAND. TANH, SQRT, DSQRT, and TAN have been modified to
	improve accuracy and error checking.
Publication No. 60498200	

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	C
iii/iv	C
v/vi	C
vii/viii	C
1 thru 208	C
Comment Sheet	C
Mailer	–
Back Cover	–

Page	Revision

Page	Revision

PREFACE

This manual describes the mathematical routines of the FORTRAN Common Library which is part of FORTRAN Extended Version 4. It is assumed that the reader is familiar with FORTRAN Extended. FORTRAN Extended operates under control of the following operating systems:

NOS 1 for the CONTROL DATA® CYBER 170 Models 171, 172, 173, 174, 175; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Sytems.

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

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

CDC manuals can be ordered from Control Data Literature and Distribution Services, 8100 East Bloomington Freeway, Minneapolis, Minnesota 55420.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

INTRODUCTION

The Math Library concerns itself with computations upon four different number types: integer, single [precision floating-point], double [precision floating-point], and complex [floating-point]. For each number type there is a well defined set of *valid* forms [representations], each one representing a particular point on the real line or in the complex plane. In addition, for each of the floating-point forms, there is a well-defined set of *semi-valid* forms, none of which represent numbers, but which instead give some indication of the nature of the [erroneous] computational process that produced them. All other bit configurations in words thought to contain numbers of some one of these types are termed *invalid*.

For these four number types, the valid, semi-valid, and invalid forms are

1. Integer.

   Valid: The ordinary, one word, right-justified, one's-complement binary representations of all integers from $-2^{48}+1$ to $2^{48}-1$. Zero may be represented as either positive zero [all zero bits], or negative zero [all one bits].

   Semi-valid: None

   Invalid: Any bit configuration wherein the top 12 bits are not all the same.

2. Single.

   Valid: The normalized, one word, forms of the internal floating-point representations. (See corresponding Computer Systems' Manual.) Zero may be represented as either positive zero, or negative zero.

Semi-valid: The four forms known as positive infinite, negative infinite, positive indefinite, and negative indefinite.

Invalid: Any non-zero and non-semi-valid bit configuration wherein bit 47 and bit 59 are the same, etc.

3. Double.

Valid: The forms of the internal, double-precision floating-point representations wherein the first word is normalized and the second word either has an exponent that is 48 smaller than the first word, or, if that underflows is zero. The signs of both words must be the same except when the lower part underflows to zero. Zero may be represented as either positive zero or negative zero.

Semi-valid: The forms wherein the first word is a single semi-valid form. The second word may be anything.

Invalid: Sign disagreement between the two words, first word an invalid single, second word with an exponent not as defined above, etc.

4. Complex.

Valid: All the two-word forms wherein each word is a valid single number.

Semi-valid: All the two-word forms wherein one word is a semi-valid single number, and the other is either a valid or a semi-valid single number.

Invalid: All the two-word forms wherein either word is an invalid single number.

The following two general rules apply to the use of these number forms in computational operations, either within the Math Library or within FORTRAN compiled code

1. Unless specially documented otherwise, if a valid form of the appropriate number type is employed in a computational operation, a valid number of the appropriate type will result. The documented exceptions to this cover such things as computing an answer which exceeds the limits of the valid forms, or performing a mathematically invalid operation.

2. Unless specifically documented otherwise, if either:
   a. a semi-valid or invalid number is employed in a computational operation, or

b. the documented limits in rule 1 above are exceeded,
then the result is undefined [i.e. the program may continue without warning, it may terminate abnormally with or without diagnostic, it may continue for a short period and then terminate, etc.]. The documented exceptions to this cover some cases wherein certain forms of checking are done, and also some cases wherein certain semi-valid forms are produced, etc.

These two rules define the limits of CDC support in the area, and also the completeness of CDC supporting documentation. When a result is undefined, there is no guarantee that the actual behavior will be the same from run to run, or that it will remain constant under normal product maintenance.

III. CLASSIFICATION OF ROUTINES AND CALLS

The FORTRAN Common Library mathematical routines (abbreviated: math library routines) compute those mathematical functions explicitly mentioned in FORTRAN. These functions may be divided into two classes - the intrinsic functions and the external functions. Intrinsic functions are simpler functions whose use speeds execution of programs and saves coding effort by occasioning replacement of frequently used sequences of FORTRAN statements with efficient in-line code during an intermediate assembly, or with calls by name to routines (when in traceback mode). The list of intrinsic functions appears in the Appendix. External functions are mathematically more sophisticated functions whose routines require more memory space and execution time. Calls to math library routines may be of two forms - 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 entered into register A1 before a return jump is made to the routine. When the routine is called by value, the arguments are entered directly into operand registers X1,...,X5 according to certain rules, before a return jump is made to the routine. The first word of the first argument is entered into X1, the first word of the second argument is entered into X3 and the first word of the third argument in X5. If an argument should be double-precision or complex and hence take two words, the second word is entered into the next register, viz. X2 or X4. Lastly, the first word of a complex argument is always the real part, and the first word of a double-precision argument is the upper half. For calls by name and calls by value, the result of the computation is returned in registers X6 and X7, a one-word result being returned in X6, and the second-word of a two word result being returned in X71 . (A juxtaposition symbol (^) is sometimes used in the documentation : it denotes that a two-word result occupies the two registers in the order indicated.)

IV. TERMINOLOGY

Some conventions have been introduced in this documentation. Symbolic names are always delimited by blanks, and any latin letters appearing therein are in upper case. ^ denotes juxtaposition, and is

used in referring to complex or double-precision quantities. All values given are in decimal, unless otherwise noted. Error shall mean: (computed value - true value). Relative error shall mean: (error/true value). An argument set is an ordered n-tuple of arguments (x1,...,xn), where x1,...,xn are the arguments in order. For convenience, we identify arguments with corresponding 1-member argument sets. The input range of a routine is the collection of all argument sets for which that routine has been designed to return a result meaningful to the user. For example, the input range to SIN is the collection of all floating-point quantities, whereas the input range to SINCOS= at entry point SIN. is the collection of definite in-range floating-point quantities not exceeding $pi.2^{46}$ in absolute value. POS.INF. abbreviates 3777,0000,0000,0000,0000B , NEG.INF. 4000,0000,0000,0000,0000B , POS.INDEF. abbreviates 1777,0000,0000,0000,0000B , and NEG.INDEF. abbreviates 6000,0000,0000,0000,0000B . In this document, "routine" shall mean the source code or the object code obtained from programs in the UPDATE library mentioned at the begining of this Introduction.

V. ERROR GRAPHS

Dissection of Error

The errors of a routine are composed of two parts: the algorithm error, including errors in the coefficients used in the algorithm; and machine round-off errors. A curve representing the error due to the algorithm and its coefficients is usually a smooth, wavy curve with discontinuities at breaks in the range reduction technique. The error of the coefficients involved in range reduction may also show up. Usually, a good algorithm with good coefficients will not have an error bigger than one-half in the last bit of the result. Round-off (and/or truncation) error is difficult to predict or graph. Suppose f(x) were approximated by $x+c*x^2$ and $x>>c*x^2$. Then by analyzing how an add instruction would work on x and $c*x^2$, one finds that a few bits are dropped off after the last bit in the result. If rounded add is used then the resulting error is between -1/2 and 1/2 in the last bit; the error in computing $c*x^2$ makes it even worse. A graph of round-off error is so discontinuous that little can be done other than showing the maximum and minimum error over small intervals.

The magnitude of a relative error can be analyzed in two ways: relative error = (routine - exact)/exact; or figuring out how many bits the routine differs from the exact value ("bit error"). In the first case, we are talking about single precision algorithms accurate to less than 2E-15 (usually) and round-off errors less than 10E-15 (usually). Note: 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 still normalized, mantissa). In determining how many bits off a routine is, the function is evaluated in double precision and this is rounded to single; then (assuming the exponents are the same) the mantissas are subtracted and the integer difference is the bit error.

Description of Plots

A typical plot covers one single-argument, single-precision function over a range of argument values (plotted linearly or logarithmically) with the ordinate ranging from -11E-15 to 11E-15 representing relative error. The saw-tooth curves represent places at which relative error is -3/2 , -1/2 , 1/2 , and 3/2 bit error. Discontinuities occur where the routine produces a result that is a power of 2 ; the argument values are given (they 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; etc.

An algorithm error curve wiggles around 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 routine rewritten to use double precision operators instead of single but keeping single precision coefficients single. Therefore it incorporates such things as: 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 quite possible, even likely, that there are points which do not lie between the two curves. However, one could, with reservations, assume the curves are "close" to true least upper bound and greatest lower bound curves.

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

Note that "ordinary" numbers (rational numbers, multiples of log 2 or pi, etc.) probably will not be sampled.

There are usually about 250 points (arrowheads) on each of the bounding curves. The algorithm for finding arrowheads goes as follows. 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. The percent within each bin of width 1E-15 with the percent above 10E-15 (below -10E-15) being

stated between 10E-15 and 11E-15 (-11E-15 and -10E-15). Bit errors are similarly handled (with anything above 3 being put with 3). Empty bins are not listed The "MEAN R.E." is the mean of all ordinates. The "RMS P.E." is

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

i.e., the standard deviation of relative error.

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 add is used, then the bounding curves will roughly parallel the algorithm error and will be as far apart as the inner saw-tooth curves. (Unrounded add would transpose the curves by 1/2 bit.)
2. If f(x) = c+g(x) then the bounds will be transposed by the error in c.
3. If f(x) = c*g(x) then the distance between the bounds for f(x) will usually be wider than for g(x) ; in particular f(x) will probably have bounds at least 2 bits apart.
4. If f(x) = g(x)+(h(x)+d(x)) where g, h, or d may be constant and one of the additions produces an unnormalized result, then the bound curves may 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 sub-intervals (e.g. 16), then the algorithm error curve will be dominated by discontinuous jumps in the constants used for table lookup.

VI. MISCELLANEOUS FACTS

Arguments of trigonometric functions and results of inverse trigonometric functions are always measured in radians. Some statistics concerning the UPDATE library of mathematical routines are given for the CYBER 74. There are 128 routines. The central memory required to UPDATE all routines is 36300 (octal) words. The central memory required to assemble all routines is 50700 (octal) words. The time required on a CYBER 74 to UPDATE all routines is 4.8 CP seconds, and the time required to assemble all routines under COMPASS is 28 CP seconds. The average assembly time for individual routines is .22 CP seconds. These times will be shorter on the CYBER 76 and longer on the CYBER 72 and 73.

VII. REFERENCES

The following references were helpful during the preparation of this document.

1. A. Abramowitz and I. Stegun, Handbook of Mathematical Functions, AMS 55.

2. Control Data Technical Report, number 52.

3. J. Hart, E. Cheney et al, Computer Approximations, John Wiley and Sons, 1968.

4. Hastings, Approximations For Digital Computers, Princeton University Press, 1955.

5. F. B. Hildebrand, Introduction To Numerical Analysis, McGraw-Hill, 1956.

6. C. Lanczos, Applied Analysis, Prentice-Hall

7. H. J. Maehly, "Methods for Fitting Rational Approximations", Part I (J. Assoc. Comp. Mach. 7, pp. 150-162) and Parts II & III (J. Assoc. Comp. Mach. 11, pp. 257-277).

8. H. S. Wall, Analytic Theory Of Continued Fractions, D. Van Nostrand Co. Inc., 1948.

9. J. H. Wilkinson, Rounding Errors In Algebraic Processes, Prentice-Hall, 1963.

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

ROUTINE : ACOSIN.

1. ROUTINE'S FUNCTION.

1.1. Type. FORTRAN external functions. The routine accepts a floating-point argument, and returns a floating-point result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the inverse cosine and inverse sine functions.

2. METHOD.

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

Formulae used in the routine are:

$$\begin{aligned}
\arcsin(x) &= -\arcsin(-x) && x \le -.5\\
\arccos(x) &= \pi - \arccos(-x) && x \le -.5\\
\arcsin(1) &= \pi/2\\
\arccos(1) &= 0\\
\arcsin(x) &= \pi/2 - \arccos(x) && .5 \le x \le .1\\
\arccos(x) &= \arccos(1-g(x,n))/2^{**}n && .5 \le x \le 1.
\end{aligned}$$

where

$$\begin{aligned}
g(x,0) &= 1-x\\
g(x,n+1) &= 4g(x,n) - 2g(x,n)^2.\\
\arccos(x) &= \pi/2 - \arcsin(x) && -.5 \le x \le .5\\
\arcsin(x) &= x+x^3 s\left(\left(w+z-j\right)w+a+m/(e-x^2)\right) && -.5 \le x \le .5
\end{aligned}$$

where

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

and

$$z = (x^2+r)x^2+i$$

The constants employed are:

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

The approximation to arcsin [-.5,.5] is an economized approximation within the class obtained by varying r,e,m,...,s. The algorithm employed is as follows. 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←x²
u←0 if ASIN. entry
←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 l.
n←0 (Loop counter)
y←1-|x|, and normalize y .

i. h←4*y-2*y²
n←n+1
If 2*y≤2-sqrt(3) = .267949192431, y←h and go to step i.

j. q←1-h, and normalize q.
y←q²
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.
←-pi/2 if x = -1.
If ASIN. entry, return.
X6←0 if x = 1.
←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.

3. ERROR ANALYSIS.

The maximum absolute value of relative error of the approximation above to arcsin over [-.5,.5] is 1.996*10**-15 . A graph of the relative error of this approximation is given in figure 6. 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 the following intervals, the following statistics on relative error were observed.

Entry Point	Interval's Lower Bound	Interval's Upper Bound	Mean	Standard Deviation	Minumum	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

6.1. ALGORITHM ERROR.

For ASIN (x) , x in (-.5,.5) the error curve is depicted in the ASIN plot between 0 and .5 . (All of the ASIN plot is symetric about 0.) . The reason for it not being balanced around the axis is because the Chebyshev coefficient for x was thrown away and 1.0 implicitly used instead. For ASIN outside (-.5,.5) and for ACOS , there is range reduction first; this produces no algorithm error. At the end of the computation, some multiple of pi gets involved; hence, the curves are offset by an amount dependant on the error in pi. There are breaks in the algorithm error curve at plus/minus .5 , SQRT(3)/2 = .866025 , .9665926 , .991445 , .997859 , etc.

Each of these is SQRT((1+previous)/2).

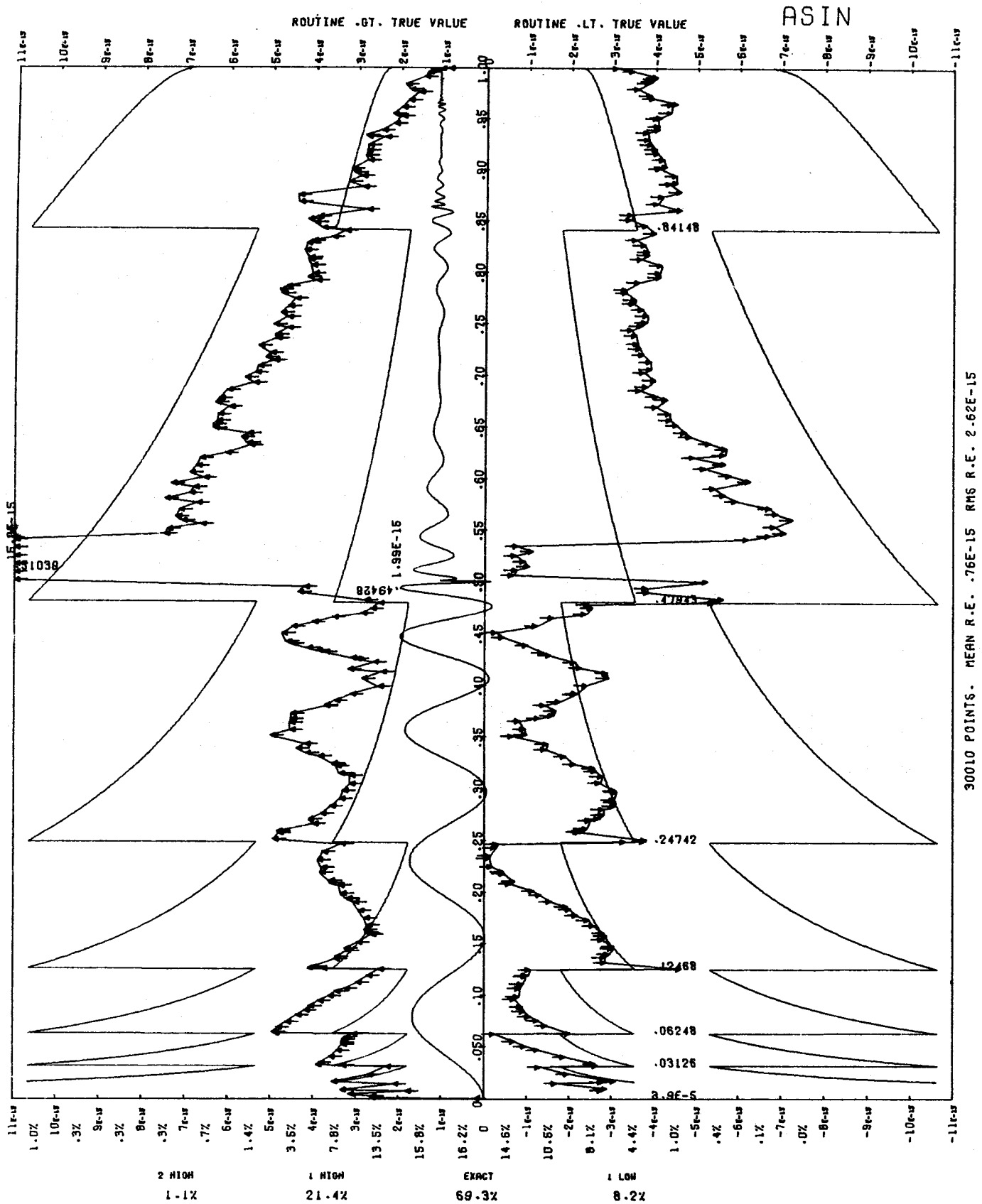
6.2. Total Error.

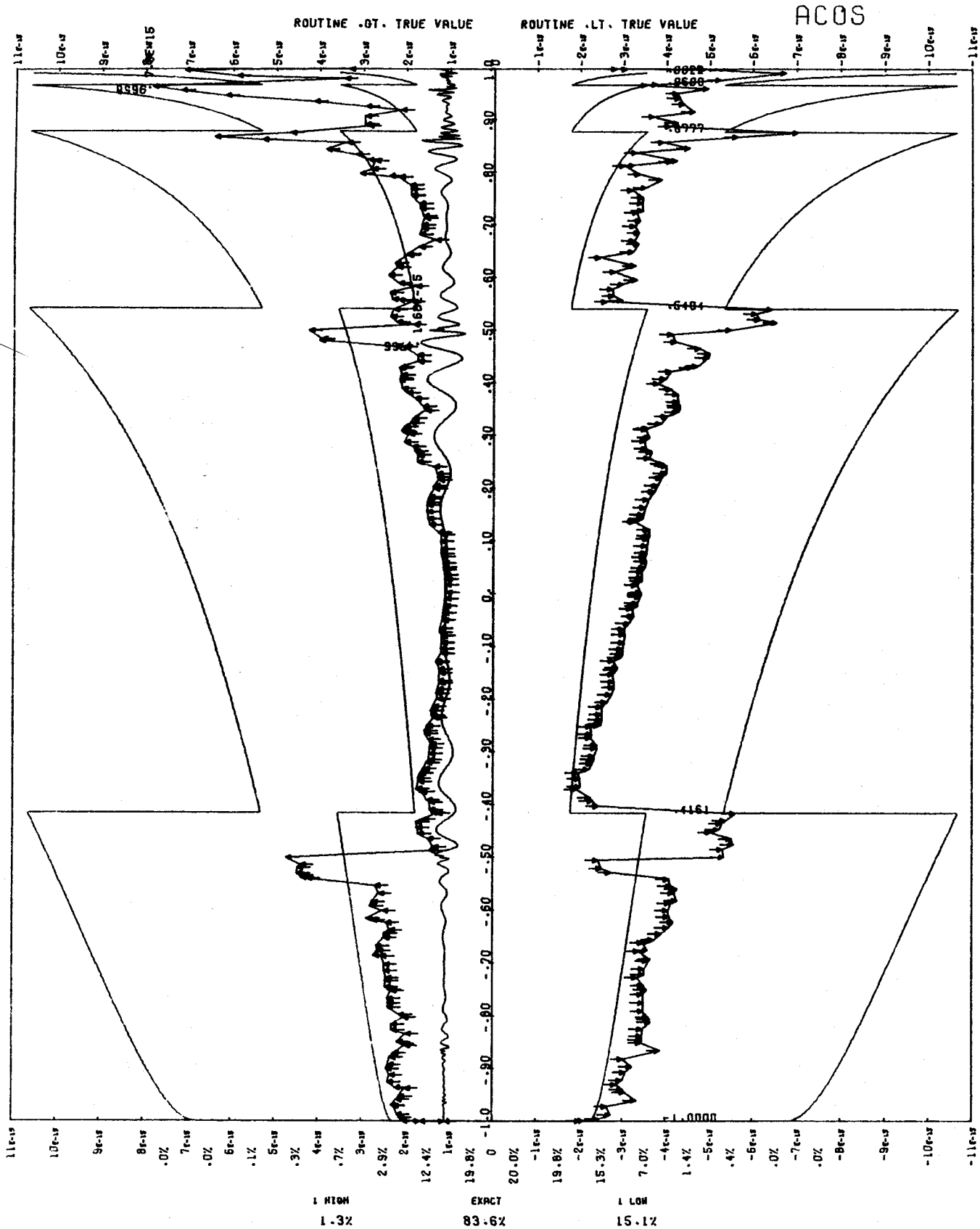
In ASIN for x in (-.5,.5) the routine boils down to $x+x^3*(...)$, hence the total error is dominated by that final addition and the total error curve closely follows the algorithm error curve plus/minus 1/2 bit. For x in (.5,.866) the algorithm is as follows: $y=1-x$, $z=(1-4y)+2y^2$, $ASIN(x)=pi/2+(pi/2-(z+z^3*(...)))/2$. y is in (.5,.134) , z is in (-.5,.5) . Nothing is lost in computing y , little is lost with z , and some is lost in the final part. The big jump when x is in (.5,.540302) is caused by pi/2-(z+...) being greater than 2. ; elsewhere it is less. This peak shows up at other places (in ASIN noticeably in (.866,.878) and ACOS just below each peak in the bit error curve) because of folding into (.5,.54). ASIN gets better near 1.0 because pi/2 predominates the final value.

ACOS , except near 1.0 , is predominated by pi/2 . In particular, for x in (-1.,.5), pi/2 is added on twice, first rounded then unrounded in order to give a near-perfect distribution. Near x=1.0 , so much folding goes on that a rather bad error is built up even 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).

4. 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 .


ASIN
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30010 POINTS. MEAN R.E. .76E-15 RMS R.E. 2.62E-15
1.99E-15
.84148
.24742
.12468
.06248
.03126
2 HIGH
1.1%
1 HIGH
21.4%
EXACT
69.3%
1 LOW
8.2%


ACOS
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30091 POINTS. MEAN R.E. -.71E-15 RMS R.E. 1.68E-15
% HIGH
1.3%
EXACT
83.6%
% LOW
15.1%

ROUTINE : ALOG

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by name and by value for ALOG (at entry points ALOG , ALOG10 , ALOG. and ALOG10.) from FORTRAN programs. ALOG computes the natural logarithm function at entry points ALOG and ALOG. , and the common logarithm function at entry points ALOG10 and ALOG10. .

2. METHOD.

The input range to this routine is the collection of all definite in-range non-negative non-zero floating-point quantities. Upon entry, the argument x is put in the form x = y * 2 ** n, where n is an integer, and 1.≤y<2. . Then log x is evaluated from

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

where log y is evaluated as follows. 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.).

"Centre points" 1., 1.225803196513098, 1.475803239208091, 1.735100002271352, 2. are chosen within these intervals. If y is in subinterval (a, b) with centre point c, log y is computed from

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)*t3 + c(5)*t5 + c(7)*t7 + c(9)*t9 .

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.

3. ERROR ANALYSIS.

(We carry out the error analysis for computation of ALOG only. Bounds on machine error are the same for ALOG and ALOG10 here, while

the the graph of algorithm error for ALOG10 may 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 8. 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 10000 arguments chosen randomly from the following intervals at the entry points listed, the following statistics or relative error were observed.

Entry Point	Interval from	to	Mean	Standard Deviation	Minimum	Maximum
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

6.1. 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 jumps 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 .

6.2. TOTAL ERROR.

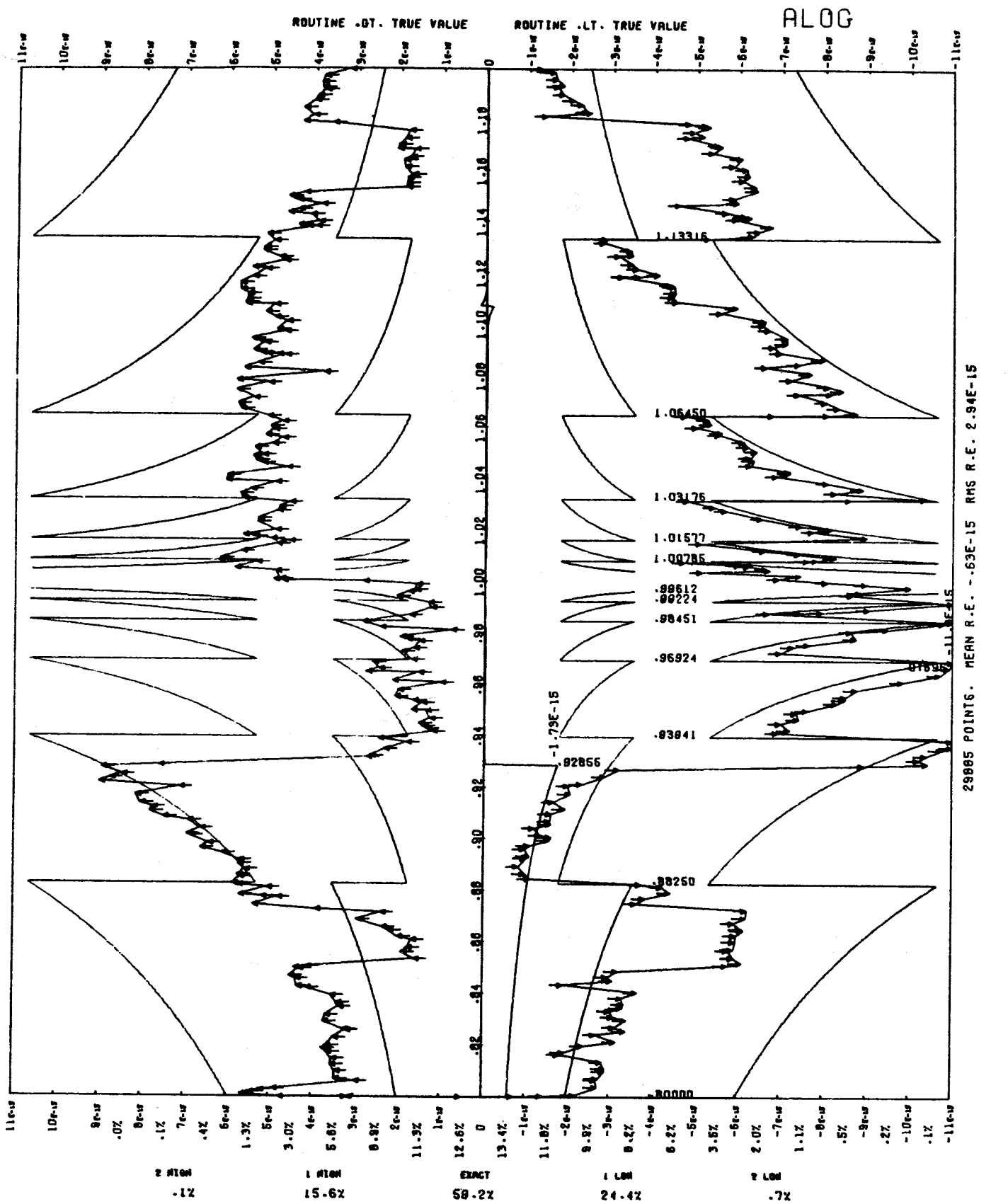
The final computation is log x = ((((a+t)+t)+p)+b)+b where
a = log 2 - 3/4)*n ,
p = c(3)*t^3+... , and
b = (3/4 * n + log c)/2 .

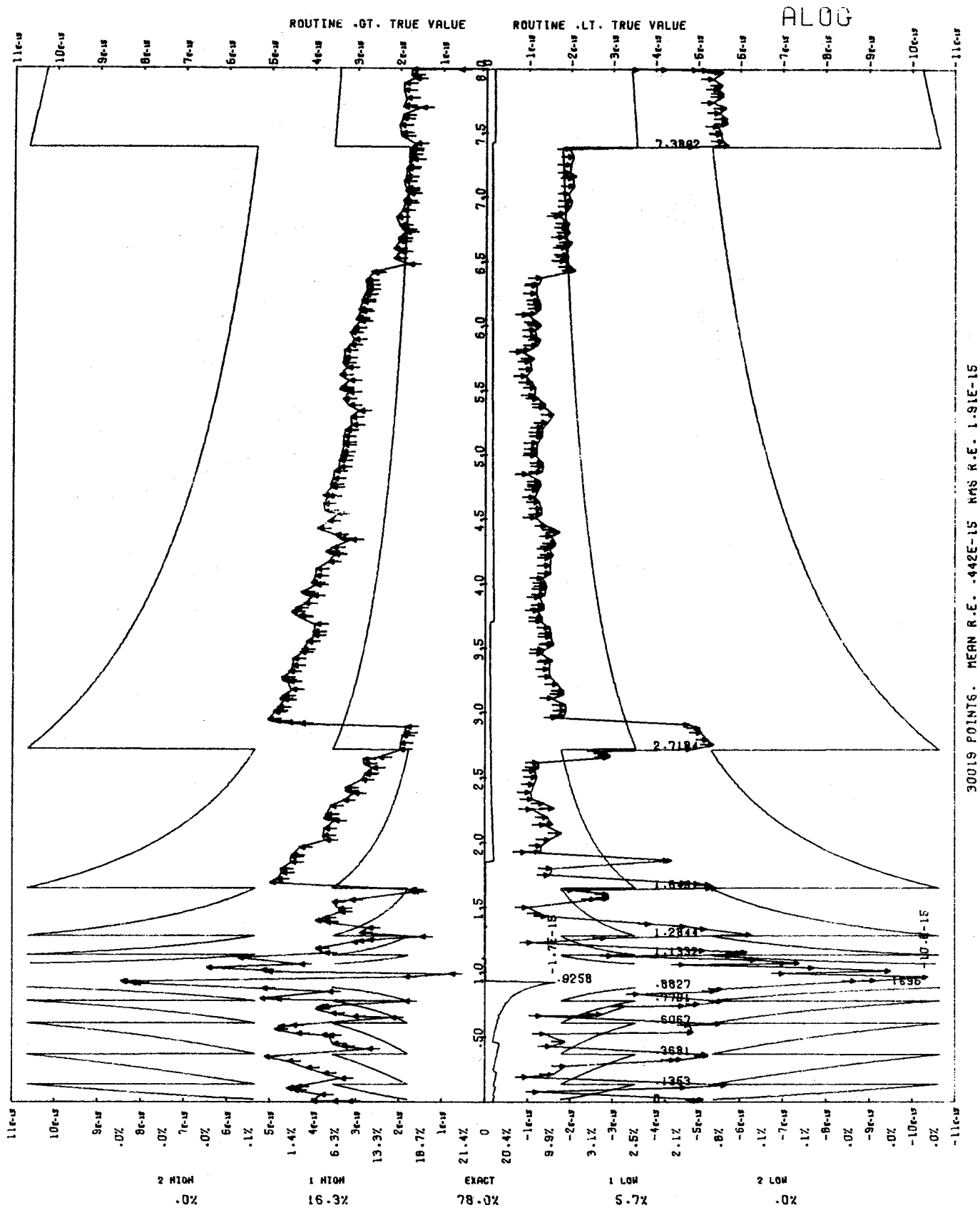
In general p<t<a<b except that a and/or b could be zero. The order was chosen in order to minimize error accumulation. b

is added in twice in order to cut down on error and eliminate a normalize. Because of all this adding going on, the error graph jumps around at odd times and by fairly small amounts. (A jump probably corresponds to a , t , or one subexpression moving accross a power of two.) Note the value of b is effectly exact. For x outside (.9286194,1.857239), a and b are non-zero and b dominates log x ; hence, the error bounds are 1 bit apart. For x 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. For x in (1.107239,1.857239), log x=((2t+p)+b with b=(log c)/2 almost exactly. t and b may be of opposite sign.

4. 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.


ALOG
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
1.13316
1.06450
1.03176
1.01577
1.00785
.98612
.98224
.98451
.96924
.93841
.92856
.88250
.80000
-1.79E-15
29005 POINTS. MEAN R.E. -.63E-15 RMS R.E. 2.94E-15
2 HIGH
.1%
1 HIGH
15.6%
EXACT
58.2%
1 LOW
24.4%
2 LOW
.7%


ALOG
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
7.3882
2.7183
1.2844
1.1332
.9258
.8827
.7781
.6067
.3681
.1353
3001 POINTS. MEAN R.E. .442E-15 RMS R.E. 1.91E-15
2 HIGH
.0%
1 HIGH
16.3%
EXACT
78.0%
1 LOW
5.7%
2 LOW
.0%

ROUTINE : ATAN

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument, and returns a floating-point result.

1.2. Purpose. To accept calls for ATAN by name at entry point ATAN and by value at entry point ATAN. . ATAN computes the inverse tangent function.

2. 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 [0, 1/16] by the range reduction formulae

$\arctan(u) = -\arctan(-u)$, 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) = -.33333333333312845,
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

$$f(u^2) = (\arctan(u) - u)/u^3.$$

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

3. ERROR ANALYSIS.

A graph of the relative error of approximation of the algorithm over [0, 1/16] is shown in figure 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 following intervals, the following statistics on relative error were observed.

Interval from	to	Mean	Standard Deviation	Minimum	Maximum
-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

4. EFFECT OF ARGUMENT ERROR.

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

ROUTINE : ATANH.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for ATANH from FORTRAN programs. ATANH computes the inverse hyperbolic tangent.

2. 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

$$\text{atanh}(-x) = -\text{atanh}(x).$$

From the definition $\tanh(x) = (e^x - e^{(-x)})/(e^x + e^{-x}))$ one gets

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

Using the property $\ln(a*b) = \ln(a) + \ln(b)$, we can reduce the argument range of the above log to [.75,1.5) by extracting the appropriate multiple of ln(2):

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

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

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

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).

3. ERROR ANALYSIS.

For abs(x)<0.2,n=0 and 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 adds 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=1 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. In addition, 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:

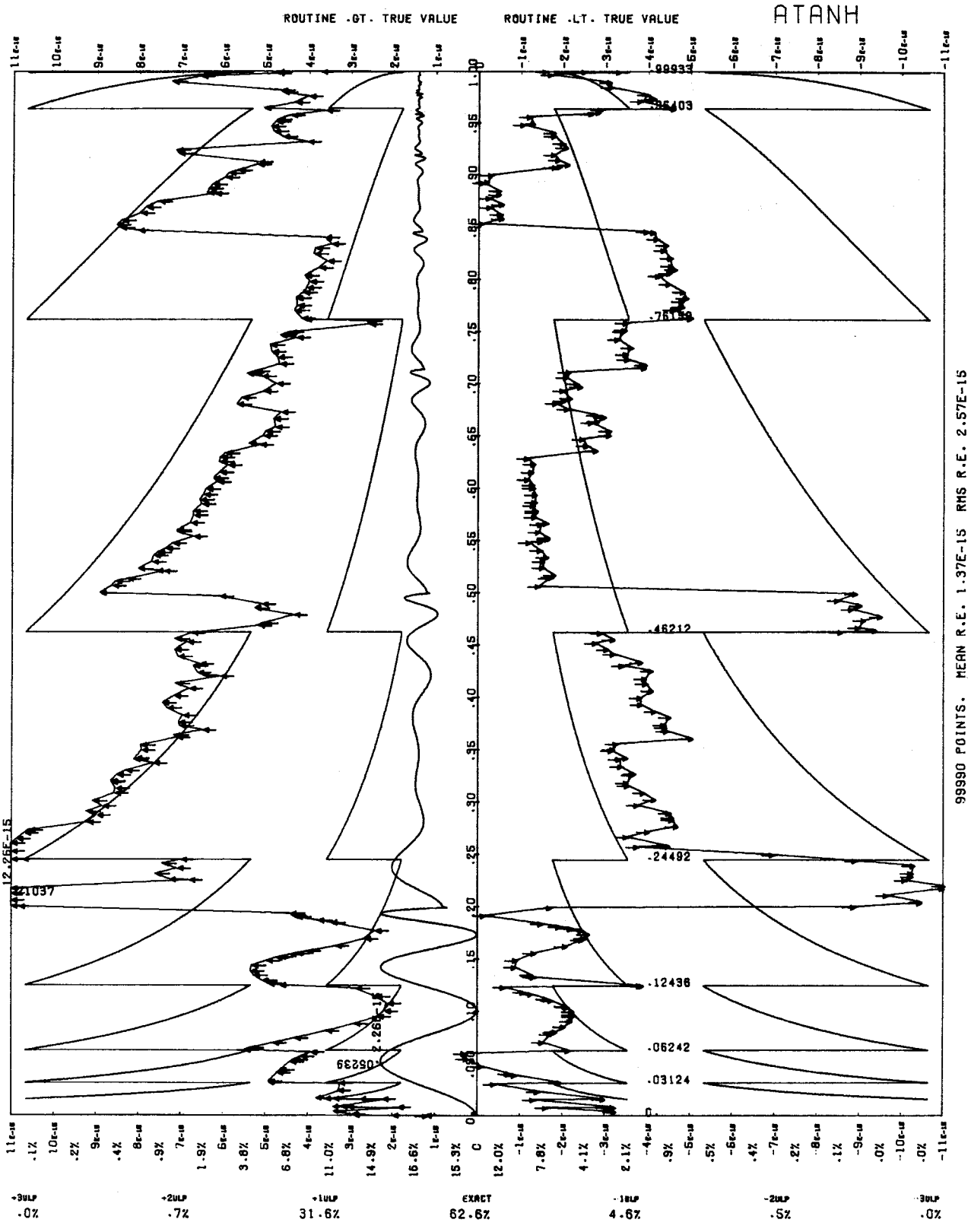
source of error	error *10^15
rational form	2.2
coefficient rounding	<0.1
round-off	17.1
upper bound	19.4
maximum observed	12.3

4. 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 $x/((1-x^2)*\mathrm{atanh}(x))$, which increases from 1 at 0 and becomes arbitrarily large near 1.0, e.g., 18.8 at 0.99 and 132 at 0.999, or approximately -1/(eps*ln(eps)) where x=1-eps. If x is known to more than single precision, the following FORTRAN 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)))
```

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


ATANH
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99990 POINTS. MEAN R.E. 1.37E-15 RMS R.E. 2.57E-15
+3ULP .0%
+2ULP .7%
+1ULP 31.6%
EXACT 62.6%
-1ULP 4.6%
-2ULP .5%
-3ULP .0%

ROUTINE : ATAN2

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts an argument set comprising two floating-point arguments, and returns a floating-point result.

1.2. Purpose. To accept calls for ATAN2 by name at entry point ATAN2 and by value at entry point ATAN2. . ATAN2 computes the inverse tangent function of the ratio of two arguments.

2. 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<0:
ATAN2(x,y) = pi - ATAN2(x,-y), x>0, y<0.

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

{(u,v): u≥0 & v≤u & v≥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 routine ATAN 's description. (The algorithm and constants are copyright 1970 by Krzyztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455. Coding is by Larry Liddiard, University of Minnesota.)

3. ERROR ANALYSIS.

See the error analysis of ATAN for properties of the algorithm used in computing arctan(y/x). 2000000 pairs of arguments (x,y) were randomly generated belonging to sets {(u,v) : |u|, |v| ≤ 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 the following intervals, the following statistics on relative error were observed.

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

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

ROUTINE : CABS.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a floating-point result.

1.2. Purpose. To accept calls by name and by value from FORTRAN programs for computation of the complex absolute value function.

2. 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+r²
   t ← (33/32 + 3/8)(w - 33/32)
     = 3/8(r² + 87/32)
   (t is the initial linear approximation to (1+r²)**0.5)
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 non-standard 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.

Formulae 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 D for further details.

3. ERROR ANALYSIS.

The maximum absolute value of the error in approximating $t(3) = SQRT(1+r^2)$ by

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

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 10000 arguments chosen randomly from the interval [-1.,1.]*[-1.,1.], the following statistics on relative error were observed.

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

4. 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

$$e(u) = (xe(x)+ye(y))/u .$$

ROUTINE : CCOS

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept by name for CCOS from FORTRAN programs. CCOS computes the complex cosine function.

2. 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 either 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 overflows 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, and 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.

3. ERROR ANALYSIS.

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

4. 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.

ROUTINE : CCOS.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by value for computation of the complex cosine.

2. 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 2^{46} . The formula used for computation is

cos(z) = cos(x + i.y) = cos(x).cosh(y) - i.sin(x).sinh(y) ,

where x and y are floating-point quantities. COS=SIN is called for computation of cos(x) and sin(x), and HYPERB= is called for computation of cosh(y) and sinh(y). The result is returned to the calling program - the real part in X6 and the imaginary part in X7 .

3. 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 1.241 . 10**(-13) + 1.241 . 10**(-13) . exp(|y|).

For 10000 arguments chosen randomly from the interval [-1.,1.]*[-1.,1.], the following statistics on relative error were observed.

Register	Mean	Standard Deviation	Minimum	Maximum
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

4. 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) .

ROUTINE : CEXP

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by name for CEXP from FORTRAN programs. CEXP computes the complex exponential function.

2. 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, if u is greater than 741.67 in absolute value, if v is so large as to lose accuracy during the calculation (i.e. v exceeds $\pi.2^{46}$ in absolute value), or if floating 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.

3. ERROR ANALYSIS.

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

4. 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.

ROUTINE : CEXP.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by value for computation of the complex exponential function.

2. 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.

3. 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 10000 arguments chosen randomly from the following interval, the following statistics on relative error of the components of the results were observed.

Interval x from	to	Interval y from	to	Register	Mean	Standard Deviation	Minimum	Maximum
-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

4. 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).

ROUTINE : CLOG

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

   1.2. Purpose. To accept calls by name for CLOG from FORTRAN programs. CLOG computes the complex logarithm function.

2. 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. Otherwise, CLOG. is called at entry point CLOG. for computation of the complex logarithm. The result is returned to the calling program.

3. ERROR ANALYSIS - see the description of CLOG. .

4. 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.

ROUTINE : CLOG.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by value for computation of the complex logarithm, including converted calls from CLOG .

2. METHOD.

The input range to this routine is the collection of all definite in-range complex quantities which are non-zero, and whose absolute values do not exceed the largest floating-point number representable 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. |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) for z nonzero, definite and in-range. The result is returned to the calling program in X6^X7 .

3. ERROR ANALYSIS.

Tests on a sample of 100000 random numbers distributed over the complex plane with distribution the product of two Cauchy distributions of zero mean returned a maximum absolute value for the relative error in the routine of 8.579 * 10**(-13).

For 10000 arguments chosen randomly from the interval [-1.,1.]*[-1.,1.], the components of the results gave the following statistics on relative error.

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

4. 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.

ROUTINE : COS=SIN

1. ROUTINE'S FUNCTION.

1.1. Type. An auxilary routine of the math library. A floating-point argument is accepted and two floating-point results are returned.

1.2. Purpose. To accept calls for COS.SIN from other routines, requiring the simultaneous computation of sine and cosine of the same argument.

2. METHOD.

Polynomials p(x) and q(x) of degrees 11 and 12 are used to compute sin(x) andcos(x) over the interval (-pi/4, pi/4), to which the argument is reduced. Upon entry, the argument x is multiplied by 2/pi, and the nearest integer n to 2/pi . x is computed by double-precision addition of 2/pi . x to 2000000000000000000B , followed by rounded floating-point addition of the upper and lower halves of the result. n is normalized, and the argument x will cause a return of POS. INDEF. if the shift count in this normalization is zero; in other words, if x exceeds pi.2^{46}(i.e., 221069929750888.5...) in absolute value. Otherwise y = x n.pi/2 is computed in double-precision as the reduced argument for input to p(y) and q(y). sin(x) and cos(x) are computed from these as indicated by the value mod(n,4). y lies in the interval (-pi/4,pi/4). The polynomials p(x) and q(x) are respectively

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

and

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

where the coefficients are given by

```
s(0) = .999999999999972
s(1) = -.16666666666504
s(2) = .833333331696029 . 10 ** -2
s(3) = -.198412607353790 . 10 ** -3
s(4) = .2755548564509884 . 10 ** -5
s(5) = -.247320720952463 . 10 ** -7
c(0) = .999999999999996
c(1) = -.499999999999991
c(2) = .04166666666664705
c(3) = -.138888888698159 . 10 ** -2
c(4) = .2480157846732S7 . 10 ** -4
c(5) = -.275552187277097 . 10 ** -6
c(6) = .206291063476645 . 10 ** -8.
```

These coefficients were obtained as follows. The polynomials of degrees 15 and 14 obtained by truncation of 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 method of telescoping polynomials (c.f., for example, C. Lanczos, Applied Analysis, 1956) consists of the (possibly repeated) removal of the leading term of polynomial by subtraction of an appropriate multiple of T(n)(a(X-x(0))) of the

same degree n, where 2/a is the length of the interval of approximation, and x(0) is its centre. (T(n)(x), the Chebyshev polynomial of degree n, is defined by T(n)(x) = cos(n.arcos(x)) (|x| ≤ 1) and satisfies the recurrence relation:

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

T(n)(x) (for n ≥ 1) is the unique polynomial 2(n-1)*x**n +...of degree n whose maximum absolute value over (-1, 1) is minimal. This maximum absolute value is, of course, 1.)

The formulae used for 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≤x≤pi/2.

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

3. ERROR ANALYSIS.

The maximum absolute error in the approximation of sin(x) by p(x) over (-pi/4,pi/4) is .1893 . 10 ** -14 and 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 . Hence, the maximum absolute error and for q(x) a 1.401 . 10 ** -14. Hence the maximum absolute error in this routine's computation of sine over (-pi/4,pi/4) is 9.416 . 10 * -15 and of cosine is 1.770 * 10 ** -14 .

4. EFFECT OF ARGUMENT ERROR.

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

ROUTINE : CSIN

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by name for CSIN from FORTRAN programs. CSIN computes the sine function for complex arguments.

2. 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 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.

3. ERROR ANALYSIS.

The algorithm used in CSIN is the same as that in CSIN. . See section 3 of the description of CSIN. for the error analysis.

4. EFFECT OF ARGUMENT ERROR.

If a small argument error appears, then the error in the result is given approximately by multipling 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 be found by substitution of the possible argument values in the function.

ROUTINE : CSIN.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by value for computation of the complex sine.

2. 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 for computation is

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

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

3. ERROR ANALYSIS.

(See the descriptions 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|). The error in the routine is significant if the argument has a large (positive or negative) imaginary part.

For 10000 arguments chosen randomly from the interval [-1.,1.]*[-1.,1.], the following statistics on relative error were observed in the components of the results.

Register	Mean	Standard Deviation	Minimum	Maximum
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

4. 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 cos(z).e(z).

ROUTINE : CSQRT

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by name for CSQRT from FORTRAN programs. CSQRT computes the complex square root function which maps to the right half of the complex plane.

2. METHOD.

For the algorithm, see the description of CSQRT. . Upon entry, the complex argument is checked. The argument is invalid if its real part or its imaginary part is infinite or indefinite, or if floating 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.

3. ERROR ANALYSIS - see the description of CSQRT. .

4. 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 a approximately the modulus of the error.

ROUTINE : CSQRT.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a complex argument and returns a complex result.

1.2. Purpose. To accept calls by value for CSQRT. from FORTRAN programs. CSQRT. computes the complex square root function.

2. METHOD.

The input range to this routine is the collection of all definite in-range non-zero complex quantities. If the argument is zero, zero is returned.
If $z = x + i.y$ is the argument, then the result is given by $w = u + i.v$ where u and v are determined as follows.
Let $a = (x^2 + y^2) ** 1/2$,
$b = ((a + |x|)/2) ** 1/2$,
and
$c = y/(2.b)$.
Then if $x \geq 0$, $u = b$ and $v = c$, while if $x < 0$, $u = c . sign(y)$ and $v = b . sign(y)$. The result from this routine will always lie in the first or fourth quadrant of the complex plane, and complex quantities lying on the axis of negative reals will be taken by the routine to the axis of positive imaginaries.

3. ERROR ANALYSIS.

The routine was tested against a sample of 100000 random numbers distributed over the complex plane with distribution the product of two Cauchy distributions. The maximum observed modulus of relative error was 1.595 . 10**(-14) .

For 10000 arguments chosen randomly from the following interval, the following statistics on relative error of the components of the results were observed.

Interval x from	to	Interval y from	to	Register	Mean	Standard Deviation	Minimum	Maximum
-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

4. 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).

ROUTINE : DASNCS.

1. ROUTINE'S FUNCTION

1.1. Type. FORTRAN external functions. It accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls by value for computation of the inverse sine and inverse cosine functions.

2. METHOD.

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

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

$\arcsin(-x)=-\arcsin(x)$

$\arccos(x)=\pi/2-\arcsin(x)$

$\arcsin(x)=\arccos(\sqrt{1-x^2})$ $x\geq 0$

$\arccos(x)=\arcsin(\sqrt{1-x^2})$ $x\geq 0$

Call the reduced value 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 $w=x\cdot\sqrt{1-z^2}-z\cdot\sqrt{1-x^2}$

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 in double, and the rest is in single except the addition of c(11), which 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, and only if the last reduction above was done) and the sum is conditionally negated and 0,-pi/2, +pi/2, or pi is added to complete the unfolding.

3. ERROR ANALYSIS.

The maximum relative errros are:	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

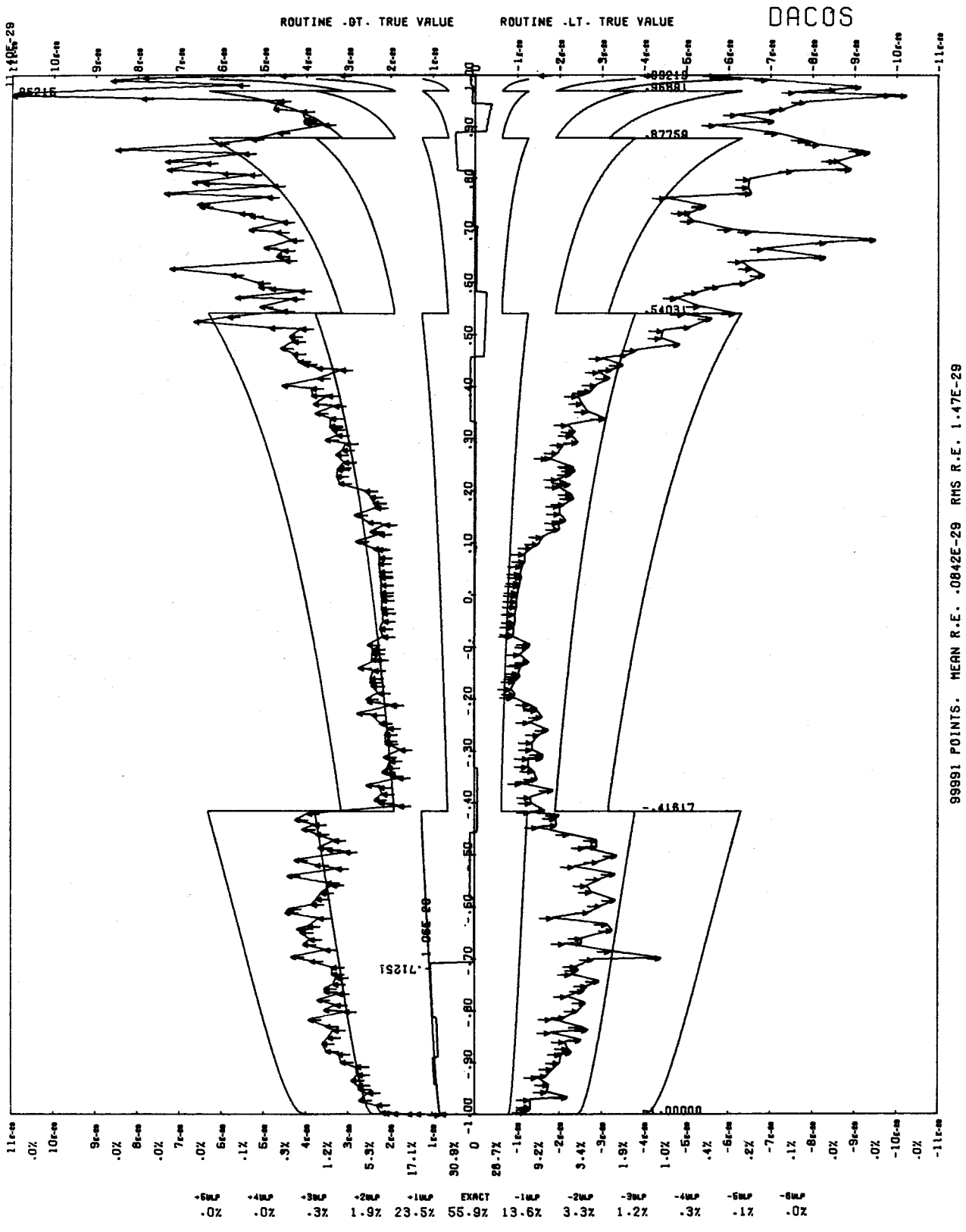
The regions of worst error are (.09375,.1446) for DASIN and (.9895,.9966) for DACOS. In these regions the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs, the worst case being .1451-.0629. For DASIN, the polynomial range was extended 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 10000 points randomly distributed in this region the maximum observed relative error in DACOS was 12.5E-29.

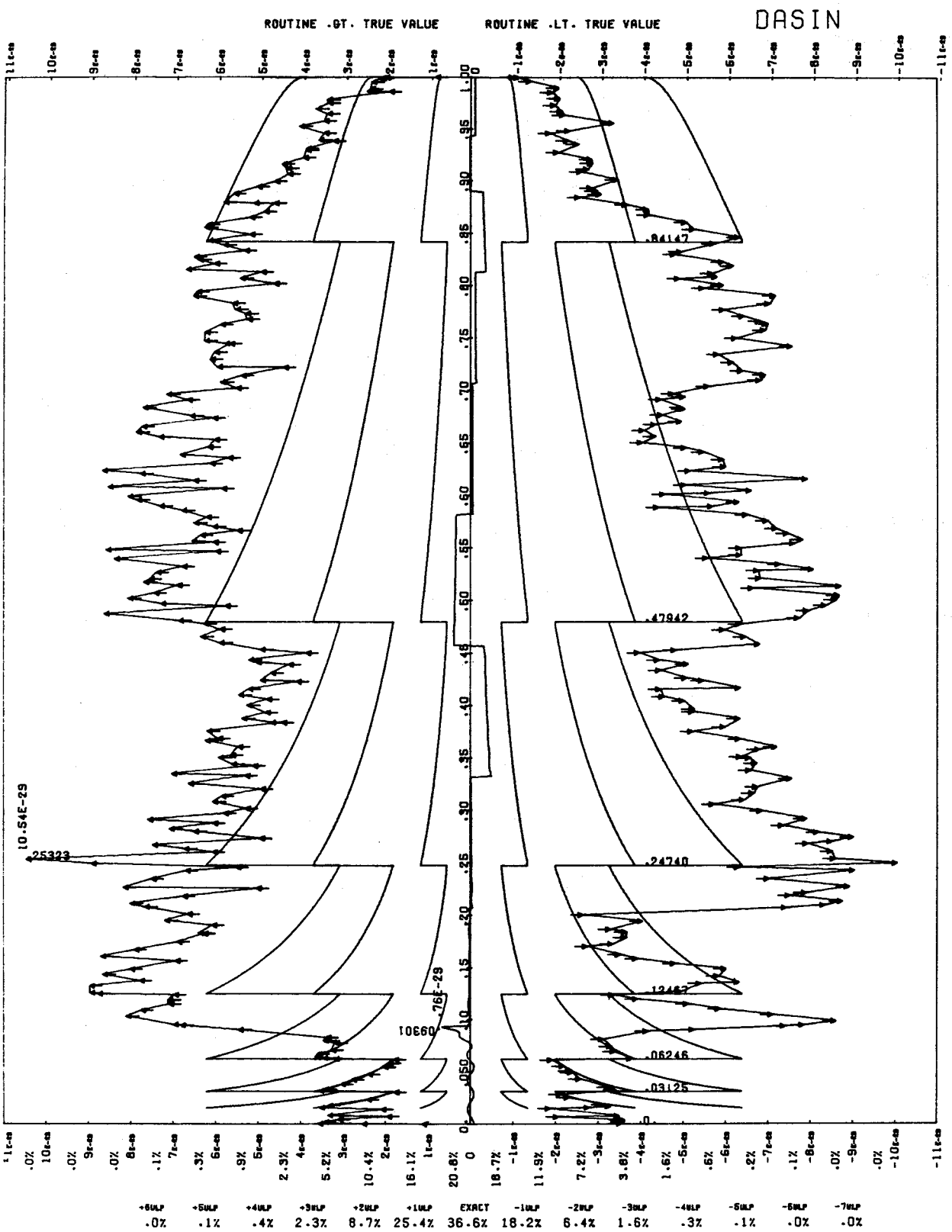
4. EFFECT OF ARGUMENT ERROR.

If a small error eps occurs in the argument x, the resulting errors in DASIN and DACOS are approximately eps/(1-x²)**.5 and -eps/(1-x²)**.5. The amplification of the relative error is approximately x/(f(x)*(1-x²)**.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 may 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-x²))
acos(x)=asin(sqrt(1-x²))
asin(-x)=-asin(x)
acos(-x)=pi+asin(x)
```

may be used to get the full significance of y. When computing (1-x²) one should use a form such as (1-x²)=(1+x)*(1-x)=y*(2-y).


DACOS
ROUTINE -DT- TRUE VALUE
ROUTINE -LT- TRUE VALUE
99991 POINTS. MEAN R.E. .0842E-29 RMS R.E. 1.47E-29
+6ULP .0%
+4ULP .0%
+3ULP .3%
+2ULP 1.9%
+1ULP 23.5%
EXACT 55.9%
-1ULP 13.6%
-2ULP 3.3%
-3ULP 1.2%
-4ULP .3%
-5ULP .1%
-6ULP .0%


DASIN
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99990 POINTS. MEAN R.E. .263E-29 RMS R.E. 2.08E-29
10.54E-29
.25323
.76E-29
.09301
.94147
.47942
.24740
.12467
.06246
.03125
+6ULP .0%
+5ULP .1%
+4ULP .4%
+3ULP 2.3%
+2ULP 8.7%
+1ULP 25.4%
EXACT 36.6%
-1ULP 18.2%
-2ULP 6.4%
-3ULP 1.6%
-4ULP .3%
-5ULP .1%
-6ULP .0%
-7ULP .0%

ROUTINE 1 DATAN

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the inverse tangent function.

2. METHOD.

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

3. ERROR ANALYSIS.

   See section 6 of the description of routine DATAN, for the error analysis.

4. EFFECT OF ARGUMENT ERROR.

   See section 7 of the description of routine DATAN.

ROUTINE : DATAN.

1. ROUTINE* FUNCTION.

1.1. Type. FORTRAN external function. The routine accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls from FTN compiler code for computation of the inverse tangent function.

2. 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 will hold 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. 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. X5 X3← absolute value of argument.
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 precessor. Call error processor, supplying given argument and parameters.

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

3. ERROR ANALYSIS.

10000 random arguments were generated in the interval [1/200.,200.], and the resulting graph of relative error versus argument is shown in the figure following this routine's description. 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 each of the following intervals, and the following statistics on relative error were observed.

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

The maximum absolute value of relative error in the algorithm is 1.622E-29, and this occurs at 1.06978147109518 3.

3.1. ALGORITHM ERROR.

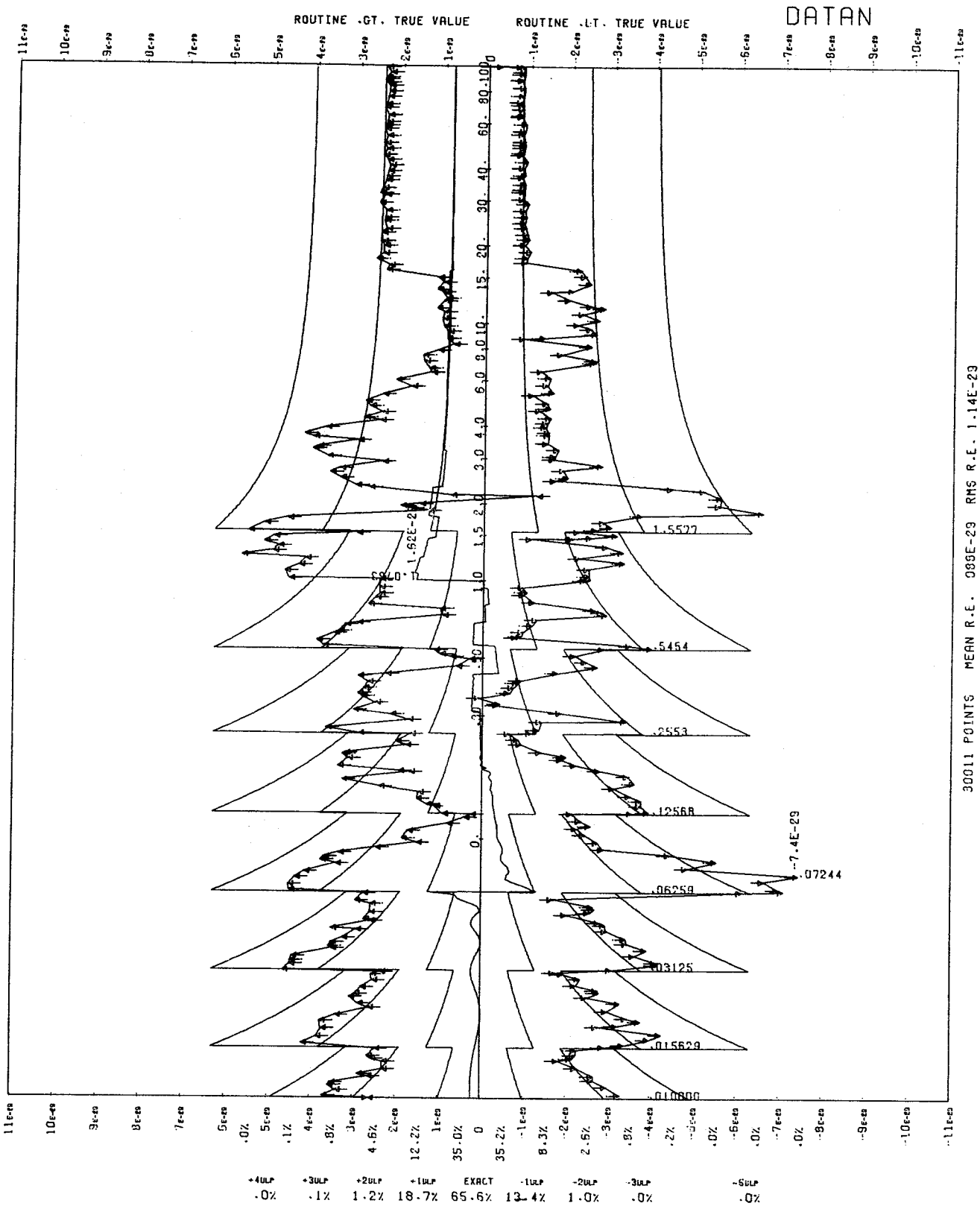
Up to 1/16, the plot shows the error in the economized polynomial; it is not centered because the first coefficient was forced to be 1. The interval between (2n-1)/16 and (2n+1)/16 is repeated twice (once reflected), but the waviness is damped because of adding atan(n/8) . The descrete jumps at (2n-1)/16 are caused by the inaccuracies in atan(n/8) . Above 1.0, the subranges are delimited by 16/(2n-1) .

3.2. TOTAL ERROR

Most of the errors can be traced back, with difficulty, to quirks 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 wreaks havoc on the addition process.

4. EFFECT OF ARGUMENT ERROR.

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


DATAN
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30011 POINTS MEAN R.E. .89E-29 RMS R.E. 1.14E-29
1.5527
.5454
.2553
.12568
.07244
.06259
.03125
.015629
.010800
7.4E-29
+4ULP .0%
+3ULP .1%
+2ULP 1.2%
+1ULP 18.7%
EXACT 65.6%
-1ULP 13.4%
-2ULP 1.0%
-3ULP .0%
-5ULP .0%

ROUTINE : DATAN2

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. The routine accepts a double-precision argument and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the inverse tangent function with two arguments.

2. 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 and routine DATAN2. is entered for all remaining computation. See this routine's METHOD description for further details.

3. ERROR ANALYSIS.

   See section 6 of the description of routine DATAN2. for the error analysis.

4. EFFECT OF ARGUMENT ERROR.

   See section 7 of the description of routine DATAN2. for the effect of argument error.

ROUTINE : DATAN2.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. The routine accepts an argument vector comprising two double-precision arguments, and returns a double-precision result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the inverse tangent function with two arguments.

2. 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. Of 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. X5 X3 ← absolute value of first argument.
X4 X1 ← absolute value of second argument.

f. 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
← 2, if first word of second argument is negative.

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

i. Supply message for "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 +INDEFINITE. to the calling program.

3. ERROR ANALYSIS.

A group of 40 random double-precision arguments was chosen in [.01,10.] x [.01,10.], and the following statistics on relative error were observed.

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

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

4. 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

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

ROUTINE : DATCOM.

1. ROUTINE'S FUNCTION.

1.1. Type. An auxiliary routine containing common code for DATAN, and DATAN2. .

1.2. Purpose. Common code for computation of DATAN and DATAN2 .

2. 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. ,
X4 X1 = DU
X5 X3 = DV ,
and at entry point DTN. ,
X7 X3 = DU .

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

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

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 X7 X3 .

b. (DQ DA←DU at DTN.) (Note that $0 \le DQ \le 1$.)

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**0.5 (DA)(u) (((DA)(u) *(1) (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)(1) - X*((DA)(1) +
(DA)(u) * (DA)(u)/
((DA)(u) +(r) (DA)(u)))

i. b ← (B7 * pi/2) -(B) B3 (upper and lower)

j. c ← b +(d) tan-1(n/8). tan-1(n/8) is obtained as a double-precision quantity from the look-up table.

k. p ← (c +(d) w) -(B) (B3 -() B4)
X6 X7 ← p , cleaned up.
Return to address B6 by direct jump.

3. 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.)

4. EFFECT OF ARGUMENT ERROR.

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

ROUTINE : DCOS

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

   1.2. Purpose. To accept calls by name for DCOS from FORTRAN programs. DCOS computes the cosine function.

2. METHOD.

   See the description of DSNCOS. for the algorithm used in the computation. The argument is checked upon entry. It is invalid if infinite or 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.

3. ERROR ANALYSIS - see the description of DSNCOS. .

4. EFFECT OF ARGUMENT ERROR - see the description of DSNCOS. .

ROUTINE : DCOSH

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. The routine accepts a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic cosine function.

2. 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 will initiate error processing in routine DHYP. . Upon entry the argument is loaded into X1 X2 before routine DHYP. is alled. (see the description of routine DHYP. for further details.)

3. ERROR ANALYSIS.

(See the description of routine DHYP. for error analysis.)

4. EFFECT OF ARGUMENT ERROR.

(See the description of routine DHYP. for effect of argument error.)

ROUTINE : DEULER.

1. ROUTINE'S FUNCTION.

1.1. Type. An auxiliary routine.

1.2. Purpose. Common code for routines DEXP. , DHYP. , and DTANH .

2. 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))**0.5 = (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
    hi ← hi + hi
    hk ← 2 * (1.+hi)
    hl ← ((y)(u) *(u) (y)(u) - hi)/hk-hi
    hm ← hi +(u) (hk -(u) hl) *(u) (hl/hk)
        (hm now carries cosh-1. 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
```

h. DX ← DS+DC

i. Clean up DS, DC, g with
X6 X7 ← DS
X0 X1 ← DC
X4 X5 ← DX
3 ← n

j. Direct jump to B4

3. ERROR ANALYSIS.

Not applicable.

4. EFFECT OF ARGUMENT ERROR.

Not applicable.

ROUTINE : DEXP

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the exponential function.

2. 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 will initiate error processing from DEXP. . Upon entry, the argument is loaded into registers X1 X2, and routine DEXP. is entered for the remaining computation. (See the description of routine DEXP. for further details.)

3. ERROR ANALYSIS - see the description of DEXP. .

4. EFFECT OF ARGUMENT ERROR - see the description of DEXP. .

ROUTINE : DEXP.

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the exponential function.

2. 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]
    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 X1 X2 , and on output, the result is in X6 X7 .

a. Let x = <argument>. Save x in core. If
   |(x)(u)| ≥ 1731564000000000000B , go to step g.

b. Jump to routine DEULER. at entry point DEULER. with 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 Ssinh(y), DC cosh(y)-1, DX exp(y)-1, all in double-precision.

c. w ← 1. +(d) (DC +(d) DS). Unpack w , increase exponents by n , and repack into 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 X6 X7 .

g. Set parameters. Load up original argument. Call error processor.

h. If error processor returns control, return.

i. Set parameters. Load up original argument. Call error processor.

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

3. ERROR ANALYSIS.

10000 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 the figure following this routine's description. In this sample, the largest absolute value of relative error is 3.858E-29 . Groups of 100 double-precision arguments were chosen randomly in each of the following intervals, and the following statistics on relative error were observed.

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

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

3.1. ALGORITHM ERROR.

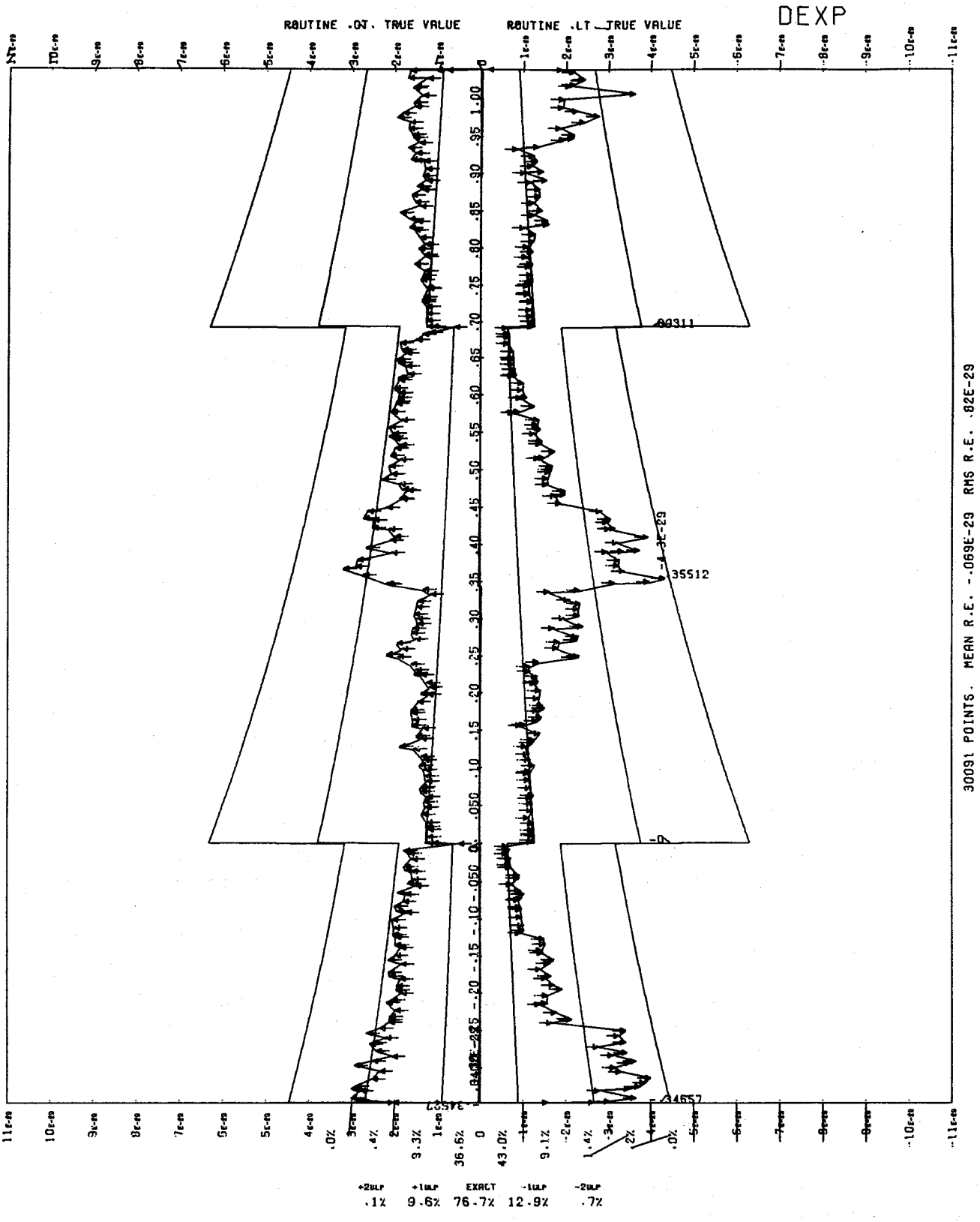
The curve for the algorithm error is barely distinguishable. It peaks at odd multiples of log 2/2 with a value of about .04E-29 . The algorithm error has essentially no effect on the total error.

3.2. TOTAL ERROR.

Except for fiddling with the exponent, DEXP ends with 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 shows 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 when lower sums overflow.

4. 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* .


DEXP
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30091 POINTS. MEAN R.E. -.069E-29 RMS R.E. .82E-29
+2ULP +1ULP EXACT -1ULP -2ULP
.1% 9.6% 76.7% 12.9% .7%

ROUTINE : DHYP.

1. ROUTINE'S FUNCTION.

1.1. Type. FORTRAN external functions. The routine accepts a double-precision argument at either entry point, and returns a double-precision result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic sine and cosine functions.

2. 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-r*log(2)
```

where n is an integer, and y is in [-1/2*log(2),1/2*log(2)] . The re-combination 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 X1 X2 , and on exit, X6 X7 holds the result. DSINH. corresponds to entry at step a., and DCOSH. corresponds to entry at step m.

a. Let 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)(1) < xmax(1), 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
    S6 X7 = +INDEFINITE .  Otherwise, return through DSINH. with  S6
    X7 = +-INFINITE , the sign determined by B5 .

g.  (Return from DEULER. with parameters
      B3 = n
      X4 = (DX)(u)
      X5 = (DX)(1)
      X0 = (DC)(u)
      X1 = (DC)(1)
      X6 = (DS)(u)
      X7 = (DS)(1)

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

    If n=0 , go to step l.
    If n≥48 , go to step k.
    u ← 2**(n-1) (DC+DS) in double
    v ← 2**(-n-1) (DC+DS) in double
    w ← 2**(n-1) + u in double
    If n≥24 , go to step h.
    w  ←  w  $+$  (2**(-n-1)+v)(1) in double, sign $+$ determined by
    B5 .

h.  w ← w $+$ (2**(-n-1)+v)(u) in double,  sign  $+$  determined  by
    B5 .

i.  X6 X7 ← w with sign same as sign of B5 .

j.  Return through entry point used to call routine.

k.  w ← (1.+DC+DS) * 2**(n-1)
    Go to step 1.

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

m.  Let a ← X1 S2 = <argument>
    b ← |a| Store b in X7 X6 .
    B5 ← 1
    Go to step b.
```

3. ERROR ANALYSIS.

10000 random arguments were generated in the interval

[-1/2 log 2,3² log 2]

for each of DSINH and DCOSH , and the resulting graphs of relative error versus argument are shown in the figures following this routine's description. In these samples, the maximum absolute values of relative error were 8.026E-29 for DSINH , and 4.405E-29 for DCOSH . The following statistics on relative error were observed in random samples of arguments in the intervals described.

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

3.1. 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.

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; the reason is that it is at this point that 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 Indefinite.) Anyway, exp(x)/2 is accurate enough.

3.2. TOTAL ERROR.

DCOSH

The total error curves should be symmetric about the 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≤c<.75*SQRT(2)-1=.06066 ; this is done fairly 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 stuff, 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.

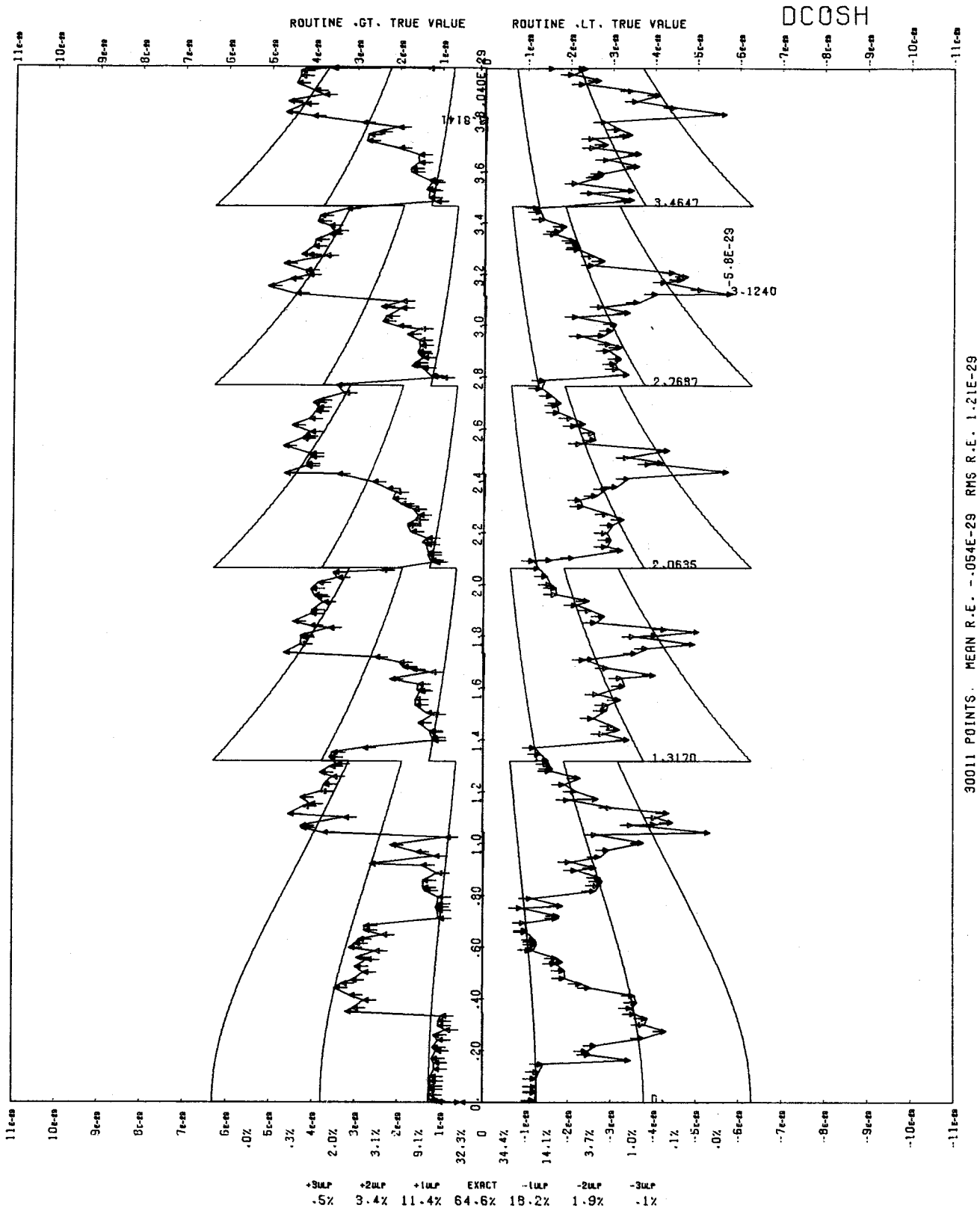
DSINH

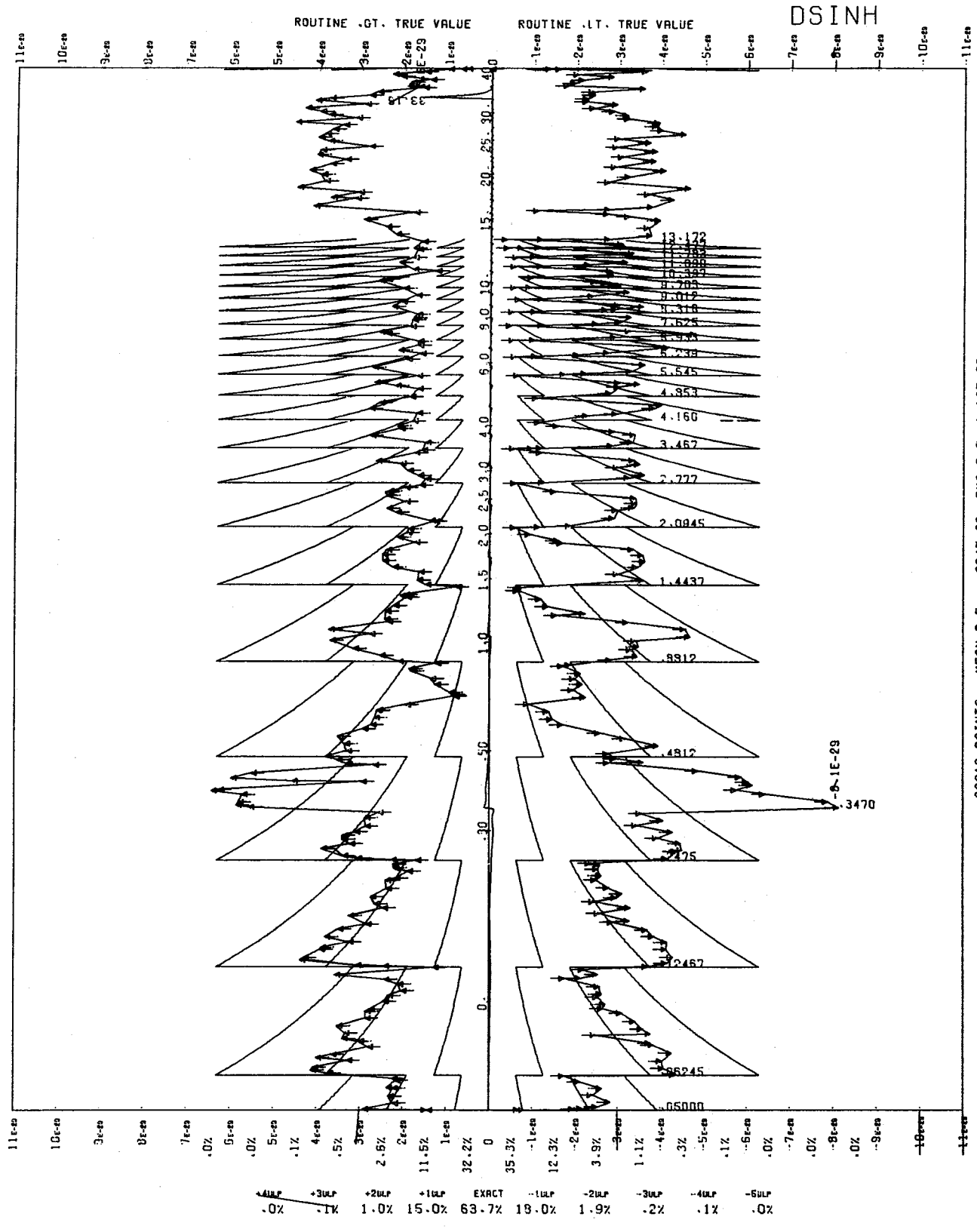
Up to log 2/2 , the error is predominated by the final add in the sinh polynomial. Just above log 2/2 the error is

especially bad because of cancellation. Near log 2/2 , DSINH is calculated via (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. (47.5*log 2), the error curve is the same as for DEXP .

4. 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* .


DCOSH
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
3.4647
3.1240
2.7687
2.0635
1.3170
30011 POINTS. MEAN R.E. -.054E-29 RMS R.E. 1.21E-29
+3ULP +2ULP +1ULP EXACT -1ULP -2ULP -3ULP
.5% 3.4% 11.4% 64.6% 18.2% 1.9% .1%


DSINH
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30019 POINTS MEAN R.E. -.094E-29 RMS R.E. 1.16E-29
EXACT
63.7%
15.0%
19.0%
1.0%
1.9%
.2%
.1%
.0%

ROUTINE : DLOG

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls by name for DLOG from FORTRAN programs. DLOG computes the natural logarithm function.

2. 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.

3. ERROR ANALYSIS - see the description of DLOG. .

4. EFFECT OF ARGUMENT ERROR - see the description of DLOG. .

ROUTINE ! DLOG. ! = DLNLOG. !

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision arg ument and returns a double-precision result.
1.2. Purpose. To accept calls by value for DLNLOG. , calls generated by the use of DLOG or DLOG10 within FORTRAN programs. DLNLOG. computes the natural and common logarithm functions.

2. 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

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

k . log2 is computed in double-precision, while log w is evaluated as follows. A polynomial approximation u is first evaluated in single-precision by

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

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

c(1) = 1.9999999993734000,
c(3) = 0.6666694866638944,
c(5) = 0.3996578110511126,
c(7) = 0.3010059222387l2.

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:

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

Writing z = 1 - x . exp(-u) , z is much less than 1, and v is computed by

$$v = u - z(u) - z(1) - (z(u))^2 \cdot (.5 + z(u)/3)$$

where z = z(u)+z(1) . 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 section 5 of the description of routine 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.

3. 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 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 interv al [2**-1/2, 2**1/2] is not greater than 5.302 . 10**-28. The maximum absolute value of the relative error of approximation of the algorithm to log x over [2**-1/2, 2**1/2] is 2.266 . 10**-28. An upper bound on the 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.

4. 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 .

ROUTINE : DLOG10

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls by name for DLOG10 from FORTRAN programs. DLOG10 computes the common logarithm function.

2. 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 or 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, DLNLOG= is called at entry point DLOG10. for the computation. The result is returned to the calling program.

3. ERROR ANALYSIS - see the description of DLNLOG .

4. EFFECT OF ARGUMENT ERROR - see the description of DLNLOG .

ROUTINE : DMOD

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts two double-precision arguments and returns a double-precision result.

1.2. Purpose. To accept calls by name from DMOD from FORTRAN programs. DMOD computes the modulus of an argument relative to a second argument.

2. METHOD.

The argument range is all valid double precision (x,y) such that [x/y]<2↑96 and y≠0. After argument checking, DMOD. is called to compute the result. The comparison [x/y]:2↑96 is done by comparing exponents and, if necessary, coefficients.

3. ERROR ANALYSIS - not applicable.

4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : DMOD.

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts argument sets comprising two double-precision arguments, and returns double-precision results.

   1.2. Purpose. To accept calls by value for DMOD. , calls generated by the use of DMOD within FORTRAN programs. DMOD. computes the remainder of an argument relative to a second argument.

2. METHOD.

   The argument range is all valid double-precision (x/y) such that [x/y]<2↑1070 and y≠0. The function computed by DMOD (x,y) is

   x-[x/y]*y

   where [u] denotes 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 (see AMOD), the intermediate value of x does not exceed 98 bits and the reduction is done in triple precision, the result is always exact.

3. ERROR ANALYSIS.

   Not applicable. Note that the only double-precision operations concerned in a determination of error are double-precision multiplication and double-precision subtraction. 4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : DSIN

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls by name for DSIN from FORTRAN programs. DSIN computes the sine function.

2. 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.

3. ERROR ANALYSIS - see the description of DSNCOS. .

4. EFFECT OF ARGUMENT ERROR - see the description of DSNCOS. .

ROUTINE : DSINH

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. The routine accepts a double-precision argument, and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic sine function.

2. 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 will initiate error processing in routine DHYP. . Upon entry, the argument is loaded into X1 X2 , and routine DHYP. is called to complete the processing. (See the description of routine DHYP= for further details.)

3. ERROR ANALYSIS.

   See the description of routine DHYP. for the error analysis.

4. EFFECT OF ARGUMENT ERROR.

   See the description of routine DHYP. for the effect of argument error.

ROUTINE : DSNCOS.

1. ROUTINE'S FUNCTION.

1.1. Type. FORTRAN external functions. The routine accepts a double-precision argument and returns a double-precision result.

1.2. Purpose. To accept calls by value for DSNCOS., calls generated by the use of DSIN or DCOS within FORTRAN programs. DSNCOS. computes the trigonometric sine and cosine functions.

2. METHOD.

The input range is the collection of all definite in-range double-precisionquantities 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. will be returned in X6 and a zero word 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, and also a flag 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

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

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) = -.25052108385441710113807647325 . 10 ** -7

a(13) = .16059043836817941727119406461 . 10 ** -9

a(15) = -.76471637307988608475534874891 . 10 ** -12

a(17) = .2811457069300l8 . 10 ** -14

a(19) = -.822042461317923 . 10 ** -17

a(21) = .194362013130224 . 10 ** -19

and

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.

3. 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 13 and 14. 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).

4. 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 formulae for sin and cos should be used to compute the error in the result.

ROUTINE : DSQRT

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

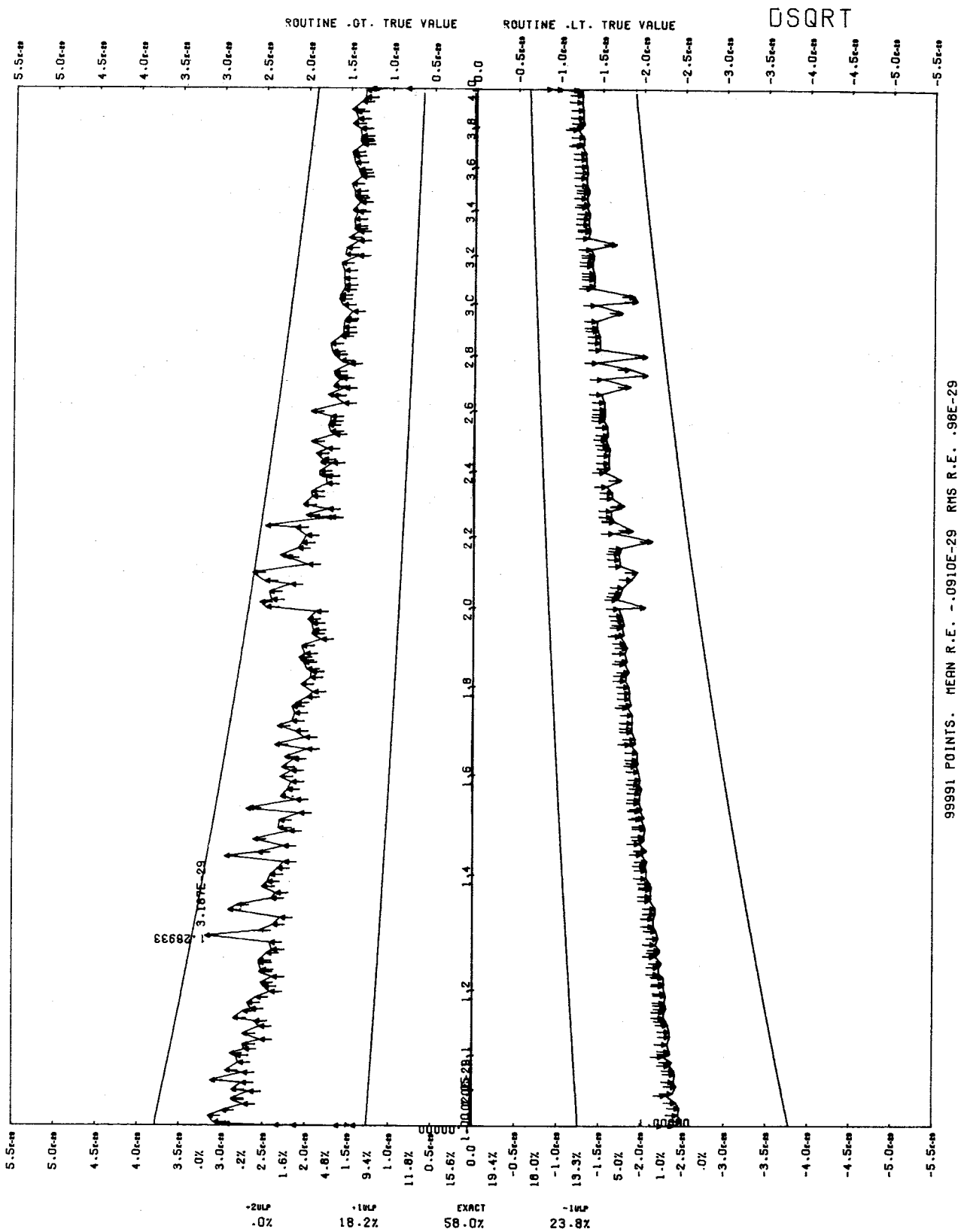
1.2. Purpose. To accept calls by name for DSQRT from FORTRAN programs. DSQRT computes the square-root function.

2. 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.

3. ERROR ANALYSIS - see the description of DSQRT. .

4. EFFECT OF ARGUMENT ERROR - see the description of DSQRT. .


DSQRT
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99991 POINTS. MEAN R.E. -.0910E-29 RMS R.E. .98E-29
+2ULP
.0%
+1ULP
18.2%
EXACT
58.0%
-1ULP
23.8%

ROUTINE : DSQRT.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by value for DSQRT. , calls generated by the use of DSQRT within FORTRAN programs. DSQRT. computes the square root function.

2. METHOD.

The input range is the collection of all double-precision quantities which are zero or positive, and are in-range and definite. Upon entry, the argument x is checked for a zero value. If it is zero, zero is returned. Otherwise the argument is put into the form

$$x = 2 ** (2.e) \cdot y,$$

where e is an integer, and $.25 \le y \le 1.00$. The result returned is 2**e * (y)**(1/2), and y**(1/2) is calculated as follows. A fourth-order Chebyshev approximation p(y) is evaluated to obtain a single-precision initial approximation to y**1/2 , where y is the upper half of the double-precision argument. Heron's rule (z(n+1)=(z(n) + y/z(n))/2) is applied in two stages in single-precision to give a single-precision approximation, and this is followed by an application of Heron's rule in double-precision to give the final double-precision approximation.

The polynomial p(y) is

$$p(y) = .182481834043495 + 1.5462934655996 \cdot y - 1.4758658070997 \cdot y^2 + 1.06285652589999 \cdot y^3 - .3239873450200001 \cdot y^4.$$

This is evaluated as

$$p(y) = c(((y + \underline{e})^2 + (y + \underline{e}) + \underline{b})((y + \underline{e})^2 + \underline{a}) + \underline{c})$$

where

$$\underline{e} = -1.070137377460206$$
$$\underline{a} = -1.391599471253464$$
$$\underline{b} = 2.286166868052419$$
$$\underline{c} = -.00601995587532198$$
$$c = -.3239873450200001$$

The final result is packed with the correct exponent e, and returned to the calling program.

3. ERROR ANALYSIS.

The algorithm error is at most 2.05E-31 (always positive). The round-off error in computing the single-precision approximation x is exactly 1/2 ulp.

Including algorithm error, x may have just over 1/2 ulp error, so x^2 may have just over 1 ulp error; since x is an approximation to the

single part only, the total error in x^2 may exceed 2 ulp when $y>x^2$ (max. 7.55E-15). Then $y-x^2$ may contain 50 significant bits, and the error range for $y-x^2$ is (-1.78E-15,+3.55E-15), and that for $(y-x^2)/(2*x)$ is (-8.88E-15,+3.55E-15). Relative to x, this error is (-6.71E-29,+2.68E-29). In order to get this error, the error in x must be at least 7.11E-15, so the resulting error after the last Heron step is in (2.52E-29,2.85E-29). The total error is in (-4.18E-29,+5.55E-29). The maximum observed error in 100000 points randomly chosen in [1,4) was 3.19E-29; the maximum in 200000 points randomly chosen in [1.0,1.5) was 3.89E-29.

4. EFFECT OF ARGUMENT ERROR.

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

ROUTINE : DTAN

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument and returns a double-precision result.

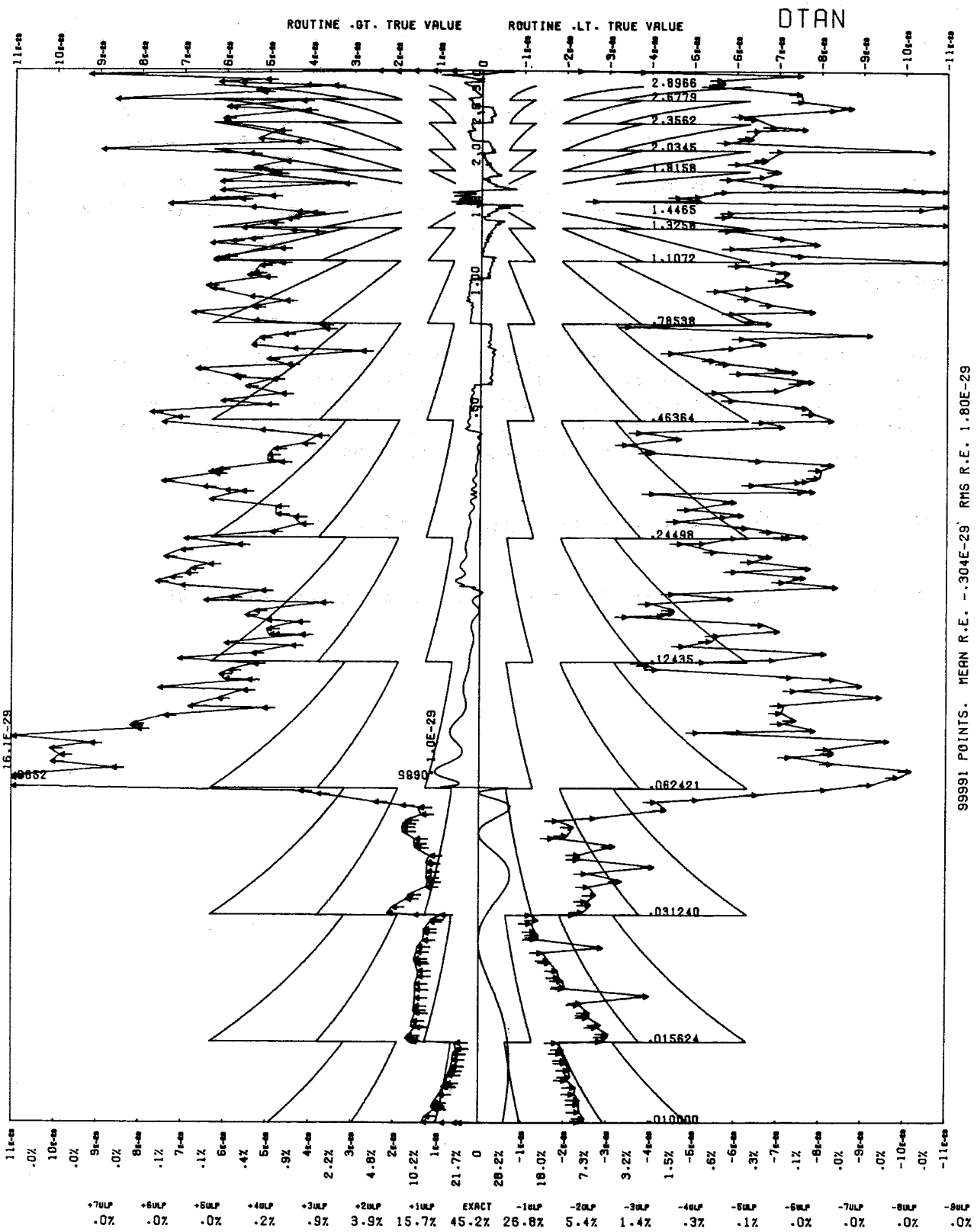
1.2. Purpose. To accept calls by name for DTAN from FORTRAN programs. DTAN computes the trignonmetric tangent function.

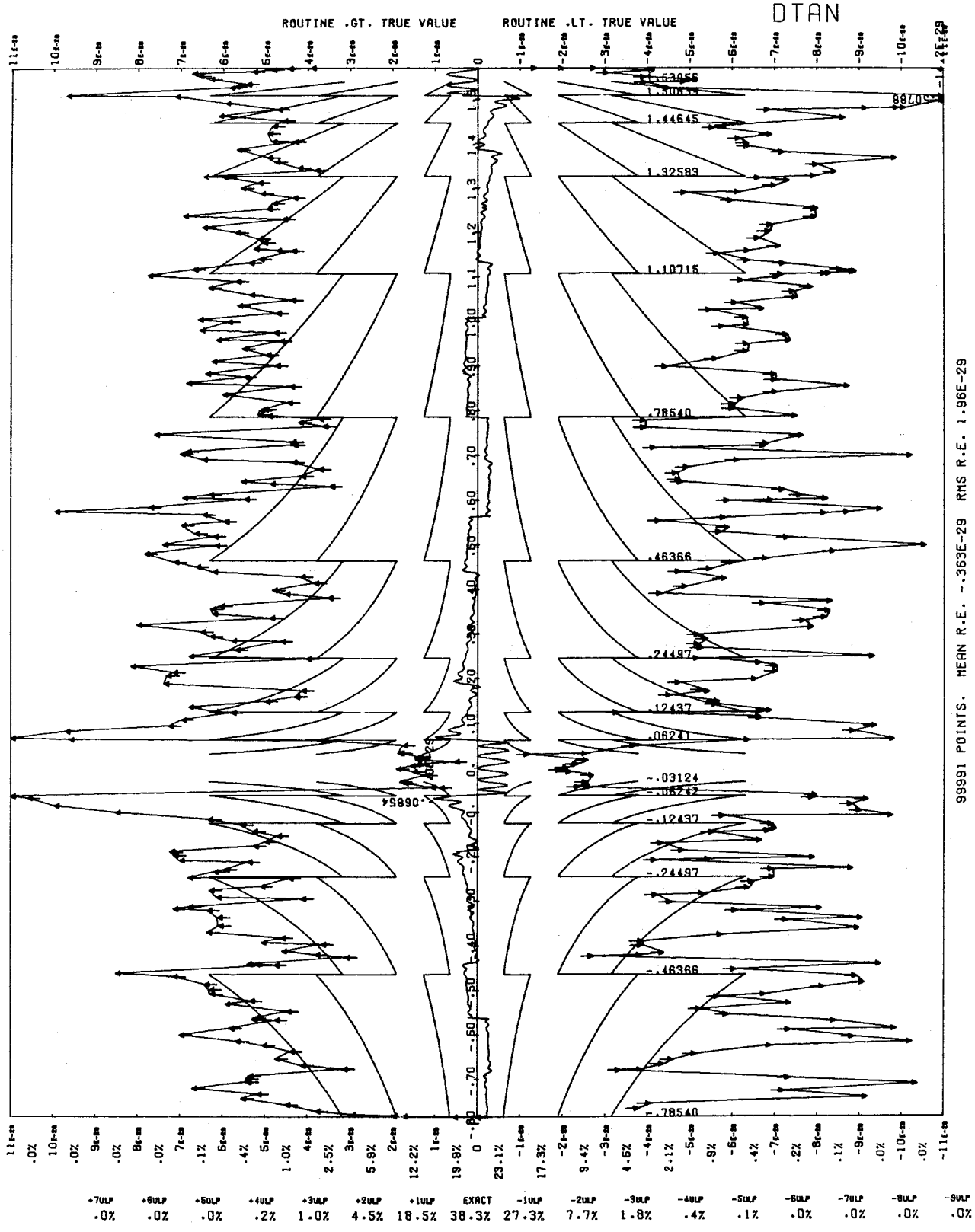
2. METHOD.

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, DTAN= is called at entry point DTAN. for the computation. The result is returned to the calling program.

3. ERROR ANALYSIS - see the description of DTAN. .

4. EFFECT OF ARGUMENT ERROR - see the description of DTAN. .


DTAN
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99991 POINTS. MEAN R.E. -.304E-29 RMS R.E. 1.80E-29
16.1E-29
1.0E-29
2.8966
2.6779
2.3562
2.0345
1.8158
1.4465
1.3258
1.1072
.78538
.46364
.24498
.12435
.062421
.031240
.015624
.010000
+7ULP .0%
+6ULP .0%
+5ULP .0%
+4ULP .2%
+3ULP .9%
+2ULP 3.9%
+1ULP 15.7%
EXACT 45.2%
-1ULP 26.8%
-2ULP 5.4%
-3ULP 1.4%
-4ULP .3%
-5ULP .1%
-6ULP .0%
-7ULP .0%
-8ULP .0%
-9ULP .0%


DTAN
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99991 POINTS. MEAN R.E. -.363E-29 RMS R.E. 1.96E-29
+7ULP .0%
+6ULP .0%
+5ULP .0%
+4ULP .2%
+3ULP 1.0%
+2ULP 4.5%
+1ULP 18.5%
EXACT 38.3%
-1ULP 27.3%
-2ULP 7.7%
-3ULP 1.8%
-4ULP .4%
-5ULP .1%
-6ULP .0%
-7ULP .0%
-8ULP .0%
-9ULP .0%

ROUTINE : DTAN.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by value for DTAN. , calls generated by the use of DTAN within FORTRAN programs. DTAN. computes the trigonometric tangent function.

2. METHOD.

The input range is the collection of all double-precision quantities which are zero or positive, and are in-range and definite. Upon entry, the argument x is checked for a zero value. If it is zero, zero is returned. Otherwise the argument is put into the form

$$x = 2 ** (2.e) \cdot y,$$

where e is an integer, and $.25 \le y \le 1.00$. The result returned is 2**e * (y)**(1/2), and y**(1/2) is calculated as follows. A fourth-order Chebyshev approximation p(y) is evaluated to obtain a single-precision initial approximation to y**1/2, where y is the upper half of the double-precision argument. Heron's rule (z(n+1)=(z(n) + y/z(n))/2) is applied in two stages in single-precision to give a single-precision approximation, and this is followed by an application of Heron's rule in double-precision to give the final double-precision approximation.

The polynomial p(y) is

$$p(y) = .182481834043495 + 1.5462934655996 \cdot y - 1.4758658070997 \cdot y^2 + 1.06285652589999 \cdot y^3 - .323987345020001 \cdot y^4.$$

This is evaluated as

$$p(y) = c(((y + e)^2 + (y + e) + b)((y + e)^2 + a) + c)$$

where

$$e = -1.070137377460206$$
$$a = -1.391599471253464$$
$$b = 2.286166868052419$$
$$c = -.00601995587532198$$
$$c = -.323987345020001$$

The final result is packed with the correct exponent e, and returned to the calling program.

A graph of the relative error in the algorithm of approximation to square root in double-precisior over (.25, 1.00) is given in figure 12. The maximum absolute value of the relative error of approximation of the algorithm is 1.230. 10**(-31). An upper bound for the relative error due to machine round-off and truncation errors has been established at 2.524 . 10**(-28) . Hence the absolute value of the relative error in the routine is less than or equal to 2.524 . 10**(-28) .

3. ERROR ANALYSIS.

4. EFFECT OF ARGUMENT ERROR.

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

ROUTINE: DTANH

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. The routine accepts a double-precision argument, and returns a double-precision result.

   1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic tangent function.

2. METHOD.

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

3. ERROR ANALYSIS.

   See the description of routine DTANH. for the error analysis.

4. EFFECT OF ARGUMENT ERROR.

   See the description of routine DTANH. for the effect of argument error.

ROUTINE : DTANH.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. The routine accepts a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic tangent function.

2. METHOD.

The input domain is the collection of all valid double-precision quantities. Arguments outside the domain which are indefinite will 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) 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) argument in (-1/2*log 2,1/2*log 2)

```
x = <argument>
y = <reduced argument>
y = x
tanh(x) = DS/(2.+DC)
```

(iii) 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. , 1 X2 holds the argument, and on exit, X6 X7 holds the result.

a. Let 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 <-49, direct 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 <-42, direct 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. ,
      B7 = n = integer offset in argument reduction,
               X7 X6 = n*log2 + y
      X4 = (DX)(u)
      X5 = (DX)(1)
      X0 = (DC)(u)
      X1 = (DC)(1)
      X6 = (DS)(u)
      X7 = (DS)(1)
      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
      Go to step g.

e.    u ← DS
      v ← 1.+DC , in double
      Go to step d.

f.    w ← 1. - 2 * ((1.+DC-DS) * 2**-n - ((1.+DC-DS) * 2**-n)2)
      (evaluated in double, although only second word of 1. is
      affected)

g.    Clean up w , affix sign in B5 , and leave in X6 X7 .
      Return.
```

3. ERROR ANALYSIS.

10000 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 the figure following this routine's description. In this sample, the maximum absolute value of the relative error is 8.581E-29 . Random samples of 100 arguments were generated in the intervals

listed, and the following statistics on relative error were observed.

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

3.1. ALGORITHM ERROR

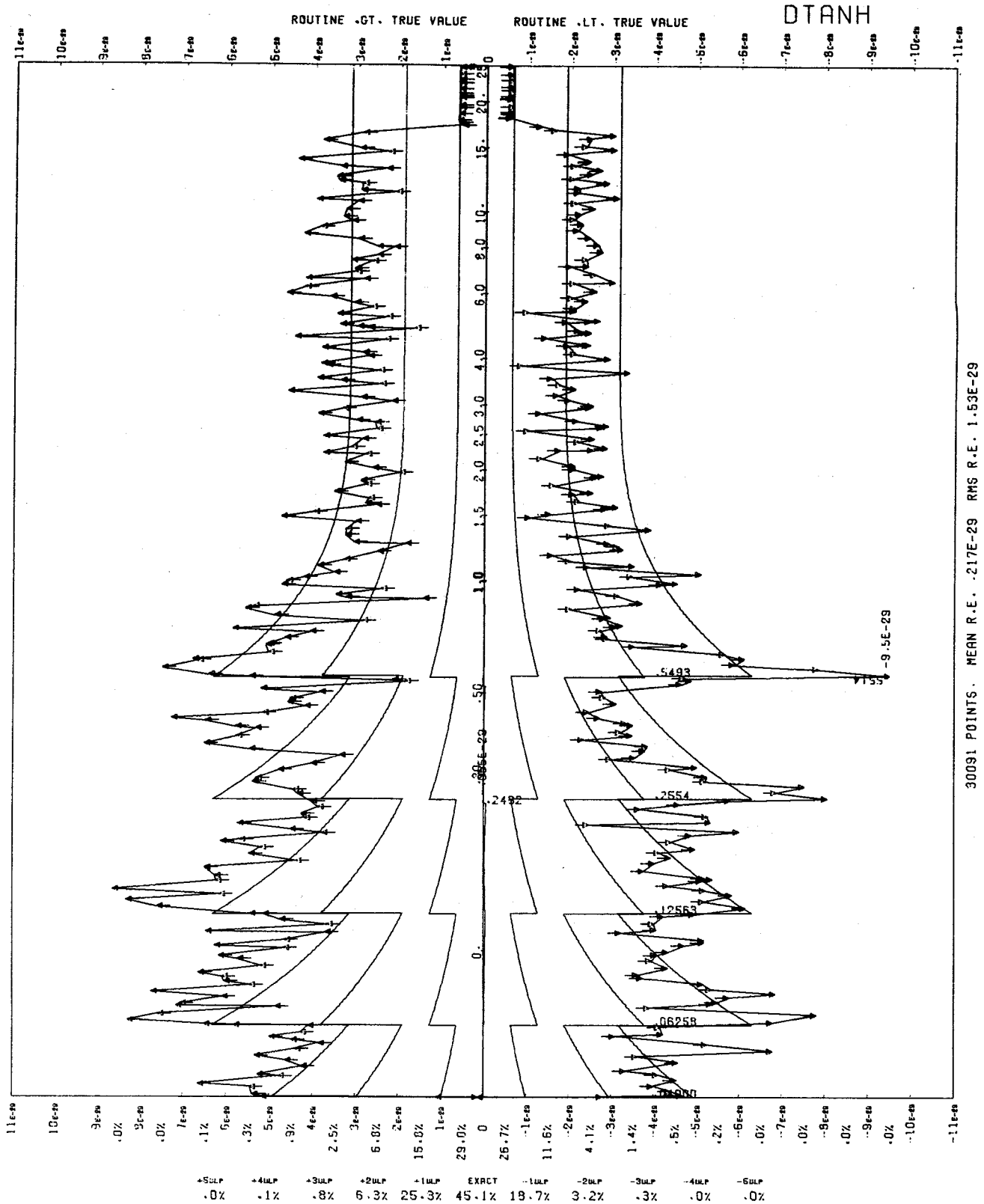
The algorithm error is insignificant. It is predominated by the error in the sinh expression in DEULER. , but by various folding actions, the error is damped even further.

3.2 TOTAL ERROR

The error plot should be symmetric about the origin. In the range (0,.5) the error is dominated by the code to divide s/(1+c): secondarily are the errors in s and in adding 1+c . 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 partially damped and fairly insignificant in comparison.) Up to 16.5 to 16.5 (23.75*log 2), 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 because it is 1-(lower precision stuff) where the was computed in double-precision. 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 2**-96 .

4. EFFECT OF ARGUMENT ERROR.

If a small error e^* occurs in the argument x, the error in the result is given approximately by $e^* * \mathrm{sech}^2(x)$.


DTANH
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
30091 POINTS. MEAN R.E. .217E-29 RMS R.E. 1.53E-29
-9.5E-29
+5ULP +4ULP +3ULP +2ULP +1ULP EXACT -1ULP -2ULP -3ULP -4ULP -5ULP
.0% .1% .8% 6.3% 25.3% 45.1% 18.7% 3.2% .3% .0% .0%

ROUTINE : DTOD*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising two double-precision arguments, and returns a double-precision result.

1.2. Purpose. To accept calls by name for DTOD* , generated by FORTRAN programs which raise double-precision quantities to double-precision exponents.

2. 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, or if the base is zero and the exponent is not greater than zero, or if floating 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 above. The result is returned to the calling program.

3. ERROR ANALYSIS.

The algorithm used in routine DTOD* is the same as that used in routine DTOD. , the call-by-value counterpart. See section 3 of the description of DTOD. for the error analysis.

4. 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.

ROUTINE : DTOD.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising two double-precision arguments, and returns a double-precision result.

1.2. Purpose. To accept calls by value for DTOD. , calls generated by FORTRAN programs which raise double-precision bases to double-precision exponents.

2. 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, 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, DLNLOG. computes log b, and DEXP. computes exp(p.log b) . The result is returned to the calling program.

3. ERROR ANALYSIS.

10,000 pairs of double-precision random numbers were generated, with distribution 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 found to be $2.977 * 10^{**}(-25)$.

4. 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:

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

ROUTINE : DTOI*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a double-precision argument and a fixed-point argument, and returns a double-precision result.

1.2. Purpose. To accept calls by name for DTOI* from FORTRAN programs, generated when the programs raise double-precision quantities to fixed-point exponents.

2. 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.

3. ERROR ANALYSIS - not applicable.

4. 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.

ROUTINE : DTOI.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a double-precision base and a fixed-point exponent, and returns a double-precision result.

1.2. Purpose. To accept calls by value for DTOI. , generated by FORTRAN programs which raise double-precision quantities to fixed-point exponents.

2. METHOD.

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

$$p = i(0).2^0 + i(1).2^1 + \ldots + i(n).2 \text{ ** } n$$

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

$$b \text{ ** } p = \text{Prod}\{b \text{ ** } 2 \text{ ** } j : 0 \leq j \leq n \;\&\; i(j) = 1\} .$$

The numbers $b = \text{** } 2 \text{ ** } 0$, b^2, b^4,...., b ** 2 ** n 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, say 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 formulae below, where n is the normalization operation and the subscript l on one of the operations +, -, and . 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 coeffient's exponent.

$$(1/b)(u) = n(1/x(u) + (((n(1-(1/x(u)).x(u) + (1 -(l) (1/x(u).x(u)) - (1/x(u) .(l) x(u)) - (1/x(u) . x(l)/x(u))) + (1/x(u) +(l) (((n(1-(1/x(u)) . x(u)) + (1 -(l) (1/x(u)) . x(u))) - (1/x(u)) .(l) x(u) - (1/x(u)) . x(l))/x(u))$$

and

$$(1/b)(l) = n(\ldots) +(l) (\ldots) .$$

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

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

where

$$(x.y)(u) = (((x(u).y(l)) + (x(l).y(u))) + (x(u) .(l) y(u))) + (x(u).y(u)) .$$

and

$$(x \cdot y)(1) = (((x(u) \cdot y(1)) + (x(1) \cdot y(u))) + (x(u) \cdot (1)\ y(u))) + (1)\ (x(u) \cdot y(u)) \cdot$$

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.

3. ERROR ANALYSIS - not applicable.

4. 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

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

ROUTINE : DTOX*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a double-precision argument and a floating point argument, and returns a double-precision result.

1.2. Purpose. To accept calls by name for DTOX* generated by FORTRAN programs raising double-precision quantities to floating- point exponents.

2. 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 from

base ** exponent = exp(exponent * log(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.

3. ERROR ANALYSIS.

The algorithm used in DTOX* is the same as that used in DTOX. . See section 3 of the description of routine DTOX. for an error analysis.

4. 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.

ROUTINE : DTOX.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a double-precision quantity and a floating-point quantity, and returns a double-precision result.

1.2. Purpose. To accept calls by value for DTOX. , calls generated by FORTRAN programs which raise double-precision bases to floating-point exponents.

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

3. ERROR ANALYSIS.

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

4. 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).

ROUTINE : DTOZ*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a double-precision argument and a complex argument, and returns a complex result.

1.2. Purpose. To accept calls by name generated by FORTRAN programs raising double-precision quantities to complex quantities.

2. METHOD.

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

base ** exponent = X + i.Y,

where

X = exp(re(exponent).log(base)).cos(im(exponent).log(base))

and

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, if the base is zero and the real part of the exponent is not positive, if the base is negative, if arithmetic overflow occurs during any stage of the computation, or if precision is lost through the arguments' being 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.

3. ERROR ANALYSIS.

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

4. 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.

ROUTINE : DTOZ.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a double-precision and a complex argument, and returns a complex result.

1.2. Purpose. To accept calls by value for DTOZ. , calls generated by FORTRAN programs which raise double-precision bases to complex exponents.

2. 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 complexquantity, b is greater than zero, and b ** z and |b ** 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.

3. ERROR ANALYSIS.

10,000 pairs (b,z) (where b is double-precision and z is complex) were generated with distribution the product of uniform distributions over (.5, 1.5) and (-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).

4. 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) .

ROUTINE : ERF.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for ERF and ERFC from FORTRAN programs. ERF. computes the error function; ERFC. computes the complementary error function, 1-ERF.

2. 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: (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	underflow
+INF	+1.0	+0.0

where the constants .477 and 25.9 are inverse erf(0.5) and inverse erfc(2^{-975}), which are approximately 0.4769362762044? 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.

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

3. 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, while stable, do not enjoy the high accuracy of most exponential-type approximations, which, when evaluated using Horner's rule, sum the smallest terms first. Inverting x and reversing coefficients does not help due to the high error in divide.

The maximum error in the approximations p1, p2, p3, scaled by 10^{15}, is:

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

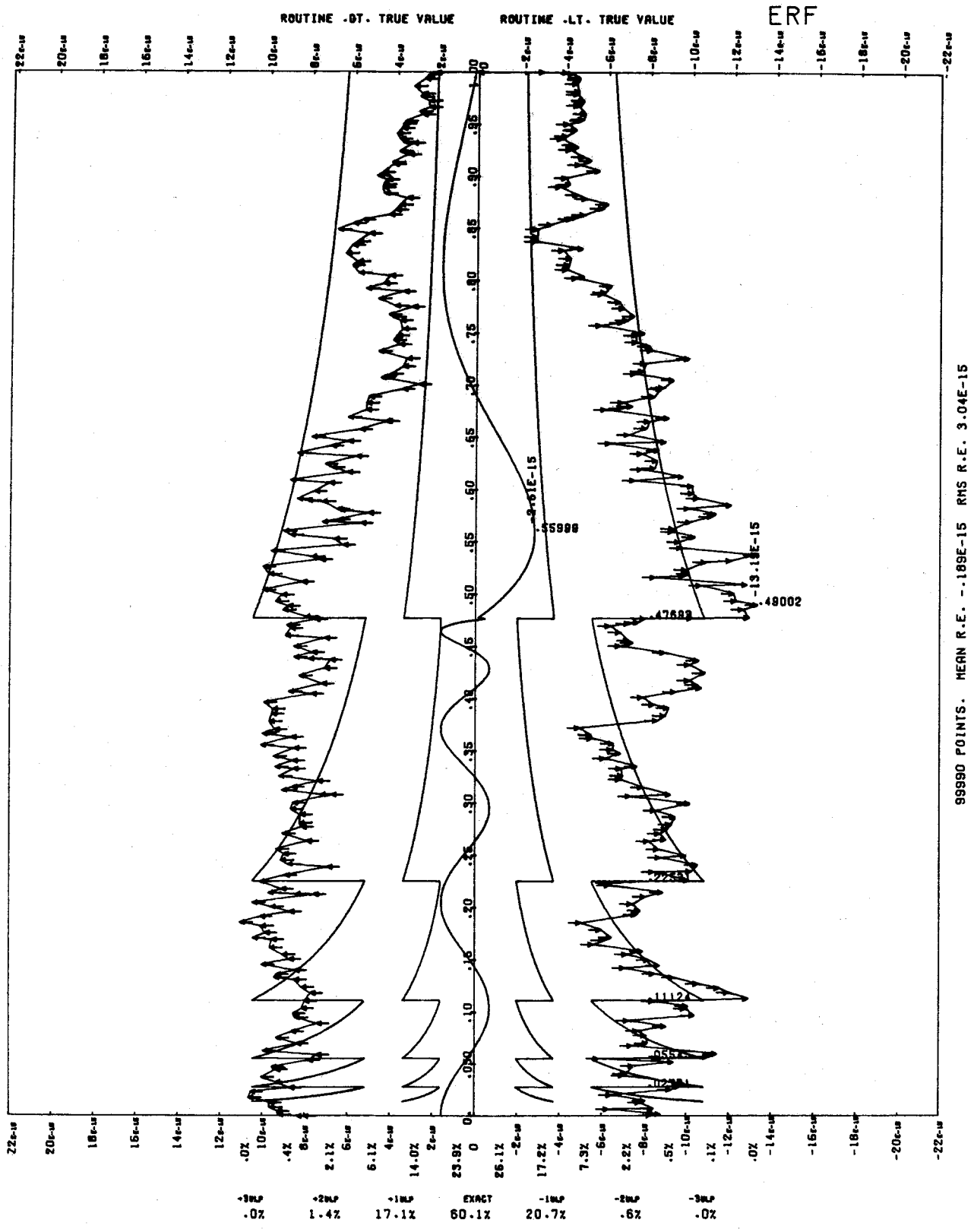
In regions where a constant is added, that constant dominates and the error is less than that shown.

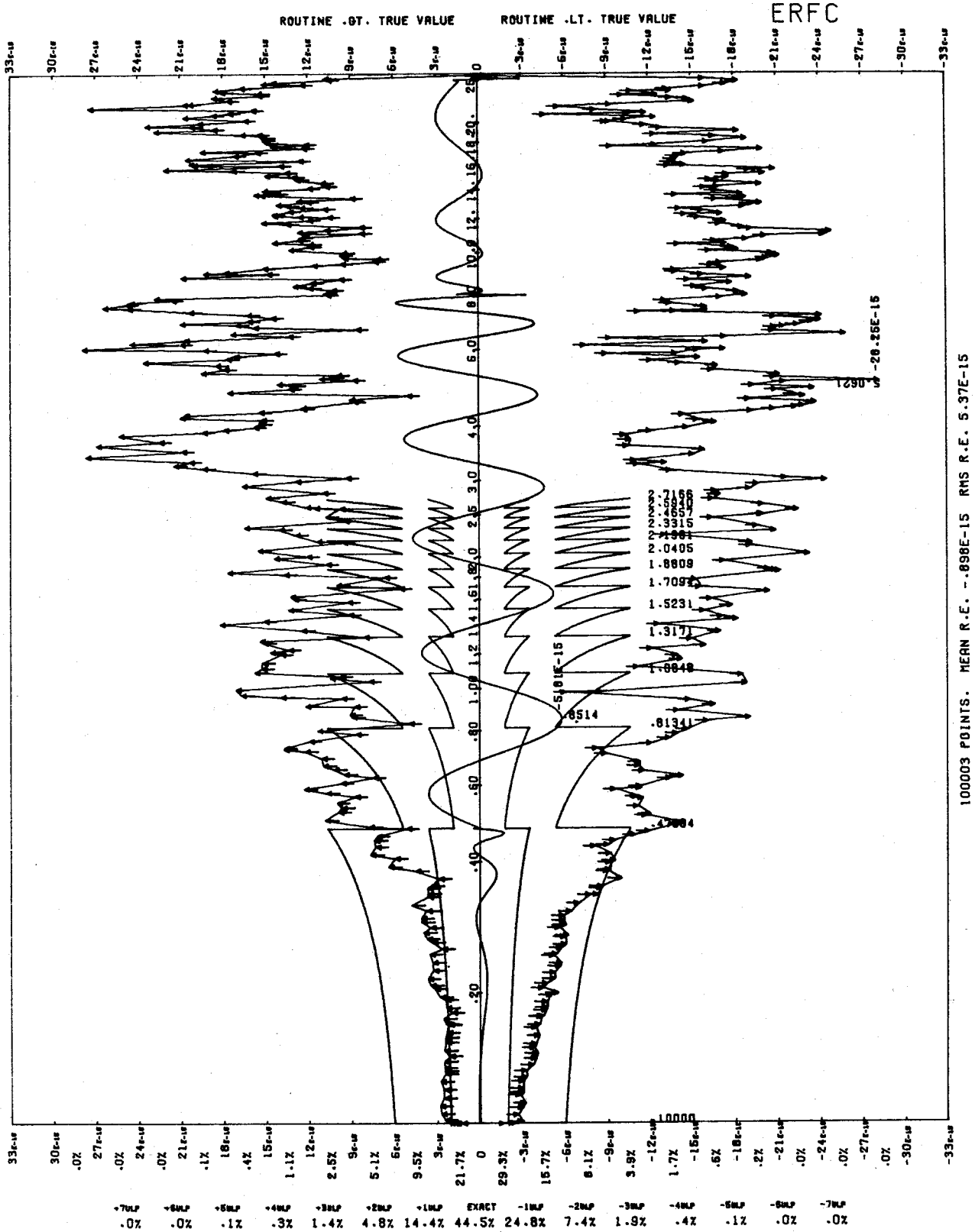
4. EFFECT OF ARGUMENT ERROR.

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(-x²) and that of relative error is (2/sqrt(pi))*x*exp(-x²)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC of 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 sequence of FORTRAN may be used to compute a good 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))
```


ERF
ROUTINE -GT. TRUE VALUE
ROUTINE -LT. TRUE VALUE
99990 POINTS. MEAN R.E. -.189E-15 RMS R.E. 3.04E-15
-3.61E-15
.55988
-13.18E-15
.48002
+3MLP .0%
+2MLP 1.4%
+1MLP 17.1%
EXACT 60.1%
-1MLP 20.7%
-2MLP .6%
-3MLP .0%


ERFC
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
100003 POINTS. MEAN R.E. -.898E-15 RMS R.E. 5.37E-15
+7ULP .0%
+6ULP .0%
+5ULP .1%
+4ULP .3%
+3ULP 1.4%
+2ULP 4.8%
+1ULP 14.4%
EXACT 44.5%
-1ULP 24.8%
-2ULP 7.4%
-3ULP 1.9%
-4ULP .4%
-5ULP .1%
-6ULP .0%
-7ULP .0%

ROUTINE : EXP

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by name and by value for EXP from FORTRAN programs. EXP computes the exponential function.

2. 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 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

```
exp(x) = 2 ** (x/log2)
       = (2 ** 1/16) ** (16 * x / log 2)
       = (2 ** 1/16) ** n * (2 ** 1/16) ** u.
```

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

```
exp(x) = 2 ** q * (2 ** 1/16) ** r * (2 ** 1/16) ** u .
```

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

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

where the constants are given by

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

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

3. 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 9. 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 may 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 10000 arguments chosen randomly from the following intervals, the following statistics on relative error were observed.

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

4. 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* .

ROUTINE : HYP. (SINH & COSH)

1. ROUTINE'S FUNCTION.

1.1. Type. FORTRAN external functions. The routine accepts a floating-point argument, and returns a floating-point result.

1.2. Purpose. To accept calls from FTN compiled code for computation of the hyperbolic cosine and sine functions.

2. 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)] = [-742.3606303797,742.3606303797].

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

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

where

$$s(3) = .16666666666693558$$
$$s(5) = -.00597299565652368$$
$$\underline{b} = 1.031539921161$$
$$\underline{a} = 72.10374670722$$
$$c(4) = .041666666666488081$$
$$c(6) = .0013888888952318045$$
$$c(8) = 89.75473897315022$$
$$c(10) = 2.763250805803 * 10^{**}-7$$

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|≥1071*log(2), go to step).

b. u ← |x|
v ← +0 if x≥0
-0 if x<0

d. n ← [u/log2+.5] = nearest integer to u/log2
w ← u - n*log 2, where the right-hand expression is evaluated in double-precision.

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

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

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

h. $c \leftarrow (1+b)*2**(n-1) + a$
X6 ← c

i. Return.

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

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

l. Initiate error processing.

m. X6 ← +IND. if x is indefinite.
← +INF. if x is infinite or too big, and positive, or COSH .
← -INF. if x is infinite or too big, and negative, and SINH .

n. Go to step i.

3. ERROR ANALYSIS.

The maximum absolute value of relative error in the approximation of sinh over [-log2/2,log2/2] is 1.282 * 10**-15 and of cosh over [-log2/2,log2/2] 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 alogrithms used to approximate sinh and cosh over [-log 2/2, log 2/2] are given in figures 17 and 18.

4. EFFECT OF ARGUMENT ERROR.

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 17 and 18. 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 formulae 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)
```

ROUTINE : HYPERB.

1. ROUTINE'S FUNCTION.

1.1. Type. Auxiliary functions from the FORTRAN Common Library. The routine accepts a floating-point argument and returns two floating-point results.

1.2. Purpose. To accept calls by value from CCOS* , CCOS. , CSIN* and CSIN. for incidental computation of cosh and sinh.

2. 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

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

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

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

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

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

3. 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 (-log2, log2). 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 is 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 10. 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 (-log2, log2).

4. 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*.

ROUTINE : ITOD*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a fixed-point and a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by name for ITOD* generated by FORTRAN programs raising fixed-point bases to double-precision quantities.

2. METHOD.

The computation uses

base ** exponent = exp(exponent * log(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, if the base is negative, if either argument is irfinite or indefinite, or if 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.

3. ERROR ANALYSIS.

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

4. EFFECT OF ARGUMENT ERROR.

If a small error occurs in the double precision exponent, the resulting error in the result is given approximately 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.

ROUTINE : ITOD.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a fixed-point argument and a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by value for ITOD. , calls generated by FORTRAN programs which raise fixed-point bases to double-precision exponents.

2. 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 floated, 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.

3. ERROR ANALYSIS.

10,000 random argument sets (b,p) were generated, with distribution 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 found to be 2.466 * 10**(-28).

4. 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.

ROUTINE : IIOJ*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of two fixed-point arguments, and returns a fixed-point result.

1.2. Purpose. To accept calls by name for ITOJ* from FORTRAN programs which raise fixed-point quantities to fixed-point exponents.

2. METHOD.

Let b be the base and p the exponent. If p has binary representation 000....000i(n)i(n-1)...i(i)i(0) where each i(j) (0≤j≤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 ** = Prod {b**(2**j) : 0≤j≤n & i(j) = 1}.

The numbers $1 = b^0$, $b = b^1$, b^2, b^4,...., 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 may 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 p 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 non-zero and the exponent is negative, 1, -1 or 0 will be returned according as the base is 1 (or -1 with the exponent even), -1 (with the exponent odd), or other. The result of the computation is returned to the calling program.

3. ERROR ANALYSIS - not applicable.

4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : ITOJ.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts two fixed-point arguments, and returns a fixed-point result.

1.2. Purpose. To accept calls by value for ITOJ. generated by FORTRAN programs which raise fixed-point quantities to fixed-point exponents.

2. METHOD.

Special case

```
 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≥2 and J≥64
 I ** J = error if I≥2^16 and J≥3
```

Let b be the base and p (≥0) the exponent. If p has binary representation 000...00i(n)i(n-1)...i(1)i(0) where each i(j)(0≤j≤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 do

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

Let r = b.

While p > 1 do

$r = r^2$,

if p is odd then r = r * b,

$p = p/2$.

Now r contains the result. Floating point was used for r so that the remaining overflows could be caught by looking at the final exponent.

3. ERROR ANALYSIS - not applicable.

4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : ITOX*

1. ROUTINE'S FUNCTION

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a fixed-point base and a floating-point exponent, and returns a floating-point result.

1.2. Purpose. To accept calls by name for ITOX* from FORTRAN programs which are generated when fixed-point bases are raised to floating-point exponents.

2. 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, if the base is negative, if the base is zero and the exponent is not greater than zero, or if 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.

3. ERROR ANALYSIS.

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

4. EFFECT OF ARGUMENT ERROR.

If a small error occurs in the floating-point exponent, the error in the result is given approximately 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 will be large.

ROUTINE : ITOX.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiation routine. It accepts an argument set (n,x) comprising a fixed-point argument n and a floating-point argument x, and returns a floating-point result.

1.2. Purpose. To accept calls by value for ITOX. , calls which are generated by FORTRAN programs which raise fixed-point bases to floating-point exponents.

2. METHOD.

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

3. ERROR ANALYSIS.

500,000 pairs (n,x) of random numbers were generated with distribution 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 usingthe routine, and then using the double-precision routine. The maximum absolutevalue of the relative error in the routine was 3.929 * 10**(-12) for the 500,000 pairs.

4. 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.

ROUTINE : ITOZ*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a real and a complex argument, and returns a floating-point result.

1.2. Purpose. To accept calls by name for ITOZ* generated by FORTRAN programs which raise fixed-point bases to complex exponents.

2. 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, if either argument is infinite or indefinite, or if floating overflow occurs during the calculation, or if x*log r is greater than 741.67. If the argument set is invalid, then a diagnostic message is issued, and POS. INDEF. is returned. Otherwise, the computation proceeds as outlined above and the result is returned to the calling program.

3. ERROR ANALYSIS.

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

4. EFFECT OF ARGUMENT ERROR.

If a small error occurs in the argument, the error in the result is given approximately 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 will be 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.

ROUTINE : ITOZ.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiation routine. It accepts an argument set comprising a fixed-point quantity and a complex quantity, and returns a complex result.

1.2. Purpose. To accept calls by value for ITOZ. , generated by FORTRAN programs which raise fixed-point bases to complex exponents.

2. METHOD.

The input range to this routine is the collection of all argument sets (n,z) comprising a fixed-point quantity n and a complex quantity z such that z is definite and in-range, and such that: if n is zero then z is a positive non-zero real, im(z) . log n does not exceed pi.2**6 (where n>0 and im(z) is the imaginary part of z), and the real number n ** re(z) is in-range.
Upon entry, the fixed-point argument is packed and normalized, and then routine XTOZ. is called at entry XTOZ. to compute the result. The result is returned to the calling program.

3. ERROR ANALYSIS.

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

4. 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) .

ROUTINE : RANF

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a dummy argument and returns a floating-point result.

1.2. Purpose. To accept calls by name for RANF and RANGET from FORTRAN programs. RANF computes pseudo-random numbers.

2. METHOD.

RANF uses the multiplicative congruential method modulo 2^{48}, i.e.
$x(n+1) = a * x(n) \pmod{2^{48}}$
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 the multiplier to generate a 96 bit product, and the lower 48 bits become the new random seed and is used to generate subsequent random numbers. RANDOM. has a default initial value of 1717 1274 3214 7741 3155B ($241^{463} \bmod 2^{47}$) . This new random seed is normalized and is returned as the random number.

The multiplier RANMLT. is constant, and has a value of 2000 1207 2642 7173 0565B . This multiplier can be shown to pass the Coveyou-Macpherson test as well as other statistical tests for randomness, including the auto-correlation test with lag≤100 and the pair triplet test (Reference: D. E. Knuth, The Art of Computer Programming, vol. 2).
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 .

3. ERROR ANALYSIS - not applicable.

4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : RANSEI

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external routine. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by name for RANSET from FORTRAN programs. RANSET resets the seed of the random number generator.

2. METHOD.

The call supplied the new address of a (suggested) new seed value in X1 . If the new seed is 0. , the new seed value is made 171712743214774131558 (= .170998394044023172008). Otherwise, the coefficient of the new seed is made odd if necessary (by adding 1B), and the exponent of the new seed value is set equal to -48 (1717(8)).

3. ERROR ANALYSIS - not applicable.

4. EFFECT OF ARGUMENT ERROR - not applicable.

ROUTINE : SINCOS.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by name and by value for SIN (at entry points SIN and SIN. respectively), and to accept calls by name and by value for COS (at entry points COS and COS. respectively). SINCOS. computes the trigonometric sine and cosine functions.

2. 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

$$y = 2/pi*x - n$$

is performed in double-precision, where x is the argument, and n is an integer, and y is in [-1/2, 1/2]. Depending upon the sign of x and n(mod 4), the result will be complemented or not, and a polynomial approximation (p(y) or q(y)) will be chosen to give the result. The polynomial approximations p(y) and q(y) are

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

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) = 8.0371891697670? * 10**-1
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) = 1.11072073453935
c(1) = 1.1419139843400? * 10**-1
c(2) = -3.5219497139982?5 * 10**-3
c(3) = 5.1726060692765?8 * 10**-5
c(4) = -4.4132825283871?1 * 10**-7.

The polynomial approximations p(y) and q(y) are minimax approximations to their corresponding functions over [-pi/4,pi/4]. (The algorithm and constants are copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455, and are employed under licence. Coding is by Larry Liddiard, University of Minnesota.)

3. ERROR ANALYSIS.

A graph of the error of approximation in the algorithm for sin(x) over [-pi/4, pi/4] is given in figure 3, and for cos(x) over [-pi/4, pi/4] in figure 4. 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 and 4.444 * 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 the following intervals for the entry points shown, the associated statistics on absolute or relative error were observed.

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

4. 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).

ROUTINE : SINCSD.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for SIND and COSD, the trigonometric sine and cosine function with argument in degrees.

2. METHOD.

Argument range: $(-2^{48}, +2^{48})$.

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. The appropriate sign is copied to the value of the appropriate function (sine, cosine) as determined by these identities:

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

3. ERROR ANALYSIS.

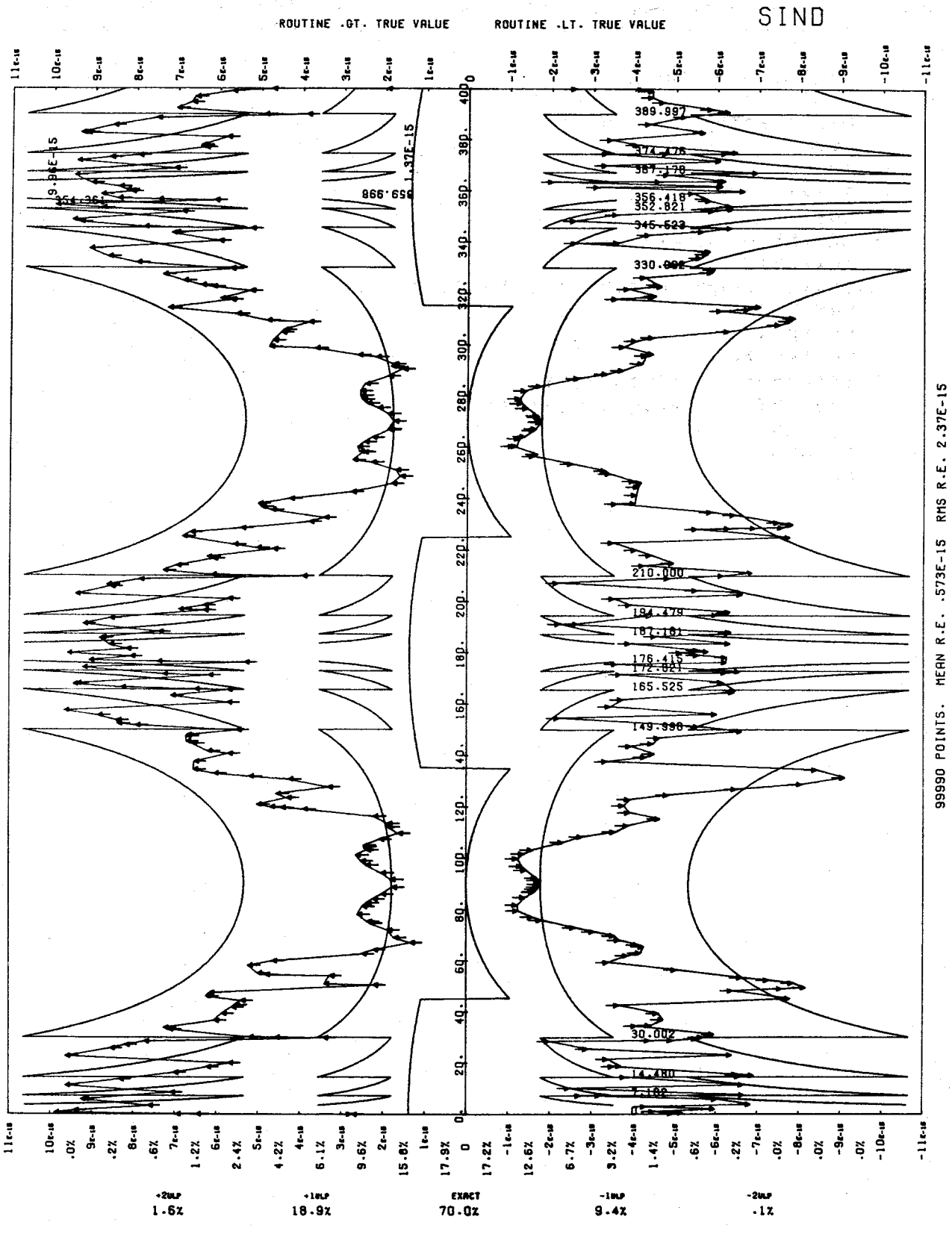
The reduction to [-45, +45) is exact; the constant pi/180 has relative error 1.37E-15, and the multiply by this constant has relative error 5.33E-15, for a total error of 6.7E-15. Since errors in the argument of SIN and COS contribute only (pi/4) of their value to the result, the error due to the reduction and conversion is at most 5.26E-15. The total error in SIND and COSD is at most this value plus the maximum error in SINCOS. on [-pi/4, +pi4), namely 7.31E-15, for a total of 12.57E-15. The maximum observed error in 100000 points in the interval [0,360) was 9.96E-15 for SIND and 9.95E-15 for COSD.

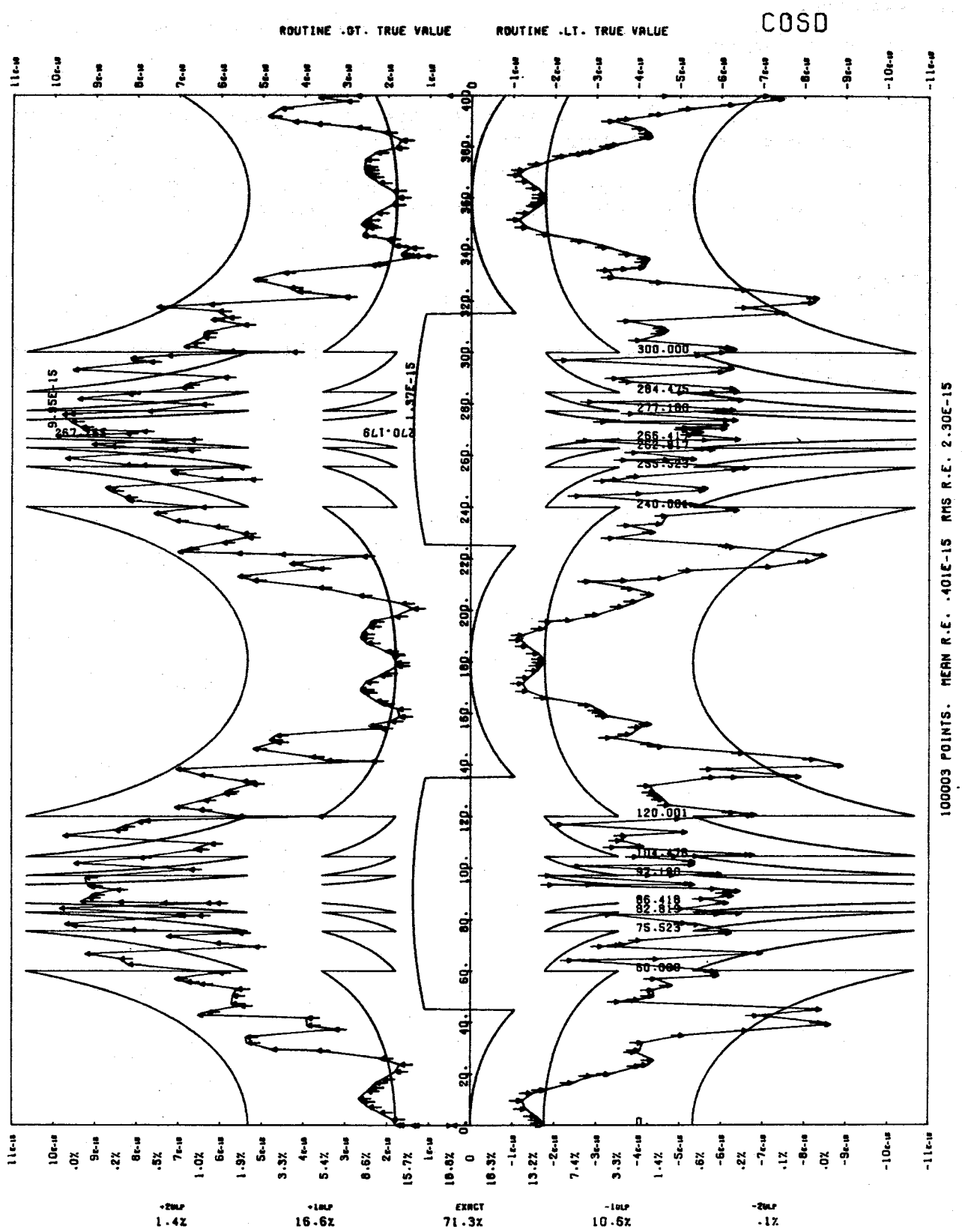
4. EFFECT OF ARGUMENT ERROR.

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 [-45°, +45°] but have poles at even (SIND) or odd (COSD) multiples

of 90°, and are large between multiples of 90° if X is large. When X is known to 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(SNGL(X)/90)
      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
```


SIND
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99990 POINTS. MEAN R.E. .573E-15 RMS R.E. 2.37E-15
+2ULP
1.6%
+1ULP
18.9%
EXACT
70.0%
-1ULP
9.4%
-2ULP
.1%


COSD
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
100003 POINTS. MEAN R.E. .401E-15 RMS R.E. 2.30E-15
+2WLP
1.4%
+1WLP
16.6%
EXACT
71.3%
-1WLP
10.6%
-2WLP
.1%

ROUTINE : SQRT

1. ROUTINE'S FUNCTION

1.1. Type. A FORTRAN external function. It accepts a real argument and returns a real result.

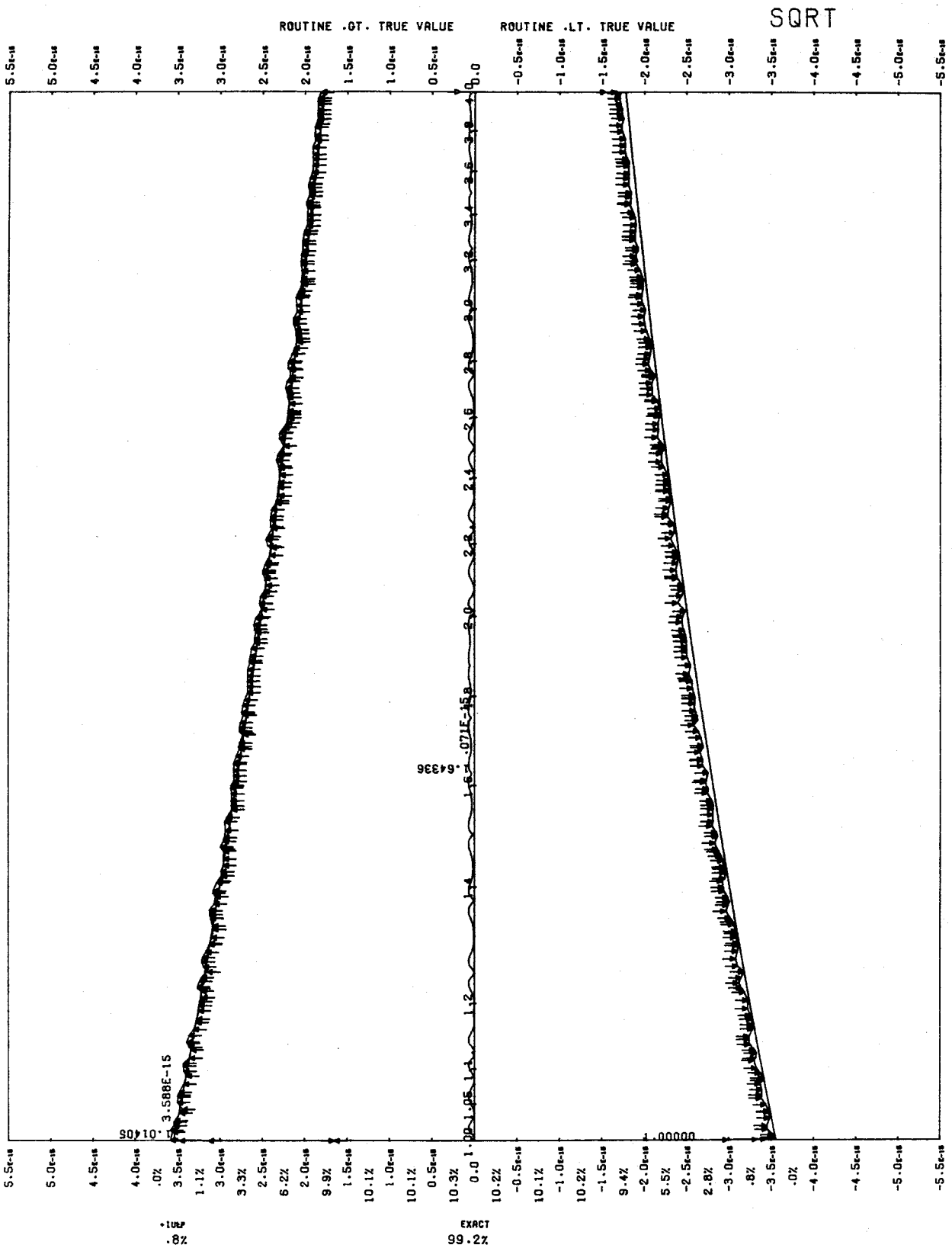
1.2. Purpose. To accept calls by reference for SQRT from FORTRAN programs. SQRT computes the square root function.

2. METHOD.

The argument is loaded into X1 and the call is converted to a SQRT. call.

3. ERROR ANALYSIS - see SQRT.

4. EFFECT OF ARGUMENT ERROR - see SQRT.


SQRT
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99991 POINTS. MEAN R.E. .0494E-15 RMS R.E. 1.51E-15
EXACT
99.2%

ROUTINE : SQRT.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a real argument and returns a real result.

1.2. Purpose. To accept calls by value for SQRT. from FORTRAN programs. SQRT. computes the square root function.

2. METHOD.

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

```
sqrt(y*2↑p)=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 sqrt on this interval, giving at least 12 bits of accuracy. Two Heron's rule iterations are made to obtain 48 bits.

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 so as not to affect 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 accomplishes the multiply 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.

The sqrt (2) scaling is fudged slightly so that the error is centered after this scaling, picking up one bit at that point.

3. 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 divide is in (-7.1E-15,0) and in the rounded odd is in [0,+3.55E-15), so the total round-off error is less than 3.55E-15 in absolute value. (Error in divide is halved because x=y/x approx.)

The upper bound on relative error is then 3.62E-15. The maximum observed relative error in 100000 randomly chosen point in the interval [0.5,2) was 3.59E-15.

4. 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.

ROUTINE 1 SYS=AID

1. ROUTINE'S FUNCTION.

   1.1. Type. An auxiliary routine.

   1.2. Purpose. To provide a link between routines in the math library, and the system error processor.

2. METHOD.

   Execution proceeds as follows.

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

   b. Read up 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.

3. ERROR ANALYSIS.

   Not applicable.

4. EFFECT OF ARGUMENT ERROR.

   Not applicable.

ROUTINE : SYS=1ST

1. ROUTINE'S FUNCTION.

1.1. Type. An auxiliary routine.

1.2. Purpose. To provide a link between routines in the math library and a system error processor.

2. METHOD.

Execution proceeds as follows at MORGUE. :

a. Enter SYS1ST and save registers X1 , X2 , X6 and 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 entering contents of X6 into register X5 .
g. Set X6 and X7 to +IND. .
h. Return to the calling program.

3. ERROR ANALYSIS.

Not applicable.

4. EFFECT OF ARGUMENT ERROR.

Not applicable.

ROUTINE : TAN

1. ROUTINE'S FUNCTION.

   1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

   1.2. Purpose. To accept calls by value for TAN computes the trigonometric tangent.

2. NAMES.

   2.1. Ident name - TAN

   2.2. UPDATE deck name - TAN

   2.3. Entry point name - TAN

3. CALLS.

   3.1. Source of calls. From FTN compiled code mentioning TAN and compiled under control card option T, D, or OPT=0 or mentioning TAN in an EXTERNAL statement.

   3.2. Format of calls. Call by reference. Entry is made by return jump to TAN.

   3.3. Format of return. The result is returned in X6.

4. CALLED ROUTINES.

   TAN. at entry point TAN. to compute the result.

5. METHOD.

2. METHOD.

   The argument is loaded into X1 and the call is converted to a TAN. call.

3. ERROR ANALYSIS - see TAN.

4. EFFECT OF ARGUMENT ERROR - see TAN.

ROUTINE : TAN.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by name for TAN. from FORTRAN programs. TAN. computes the trigonometric tangent functions.

2. METHOD.

The input range is the collection of all definite, in-range floating-point quantities in the interval (-2**7, +2**7).

The identities

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

are used in the form

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

to reduce the evaluation to the interval [-0.5,+0.5] using an approximation for tan((pi/2*y). The reduction is done by multiplying x by 2/pi and subtracting the nearest integer, rounding the result to single.

The function tan((pi/2)*y) is approximated with a rational form, (7th order odd)/(6th order even), which has minimax relative error on the interval [-0.5,+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.

If identity (iv) is used, i.e., if the integer subtracted is odd, the result is negated and inverted by dividing -Q/P instead of P/Q.

3. ERROR ANALYSIS.

The range reduction, the final add in each part of the rational form, the final multiply in P and the divide dominate the error. Each of these operations contributes directly to the final error, and each is accurate to about 1/2 ulp (unit in the last place). The maximum relative errors are

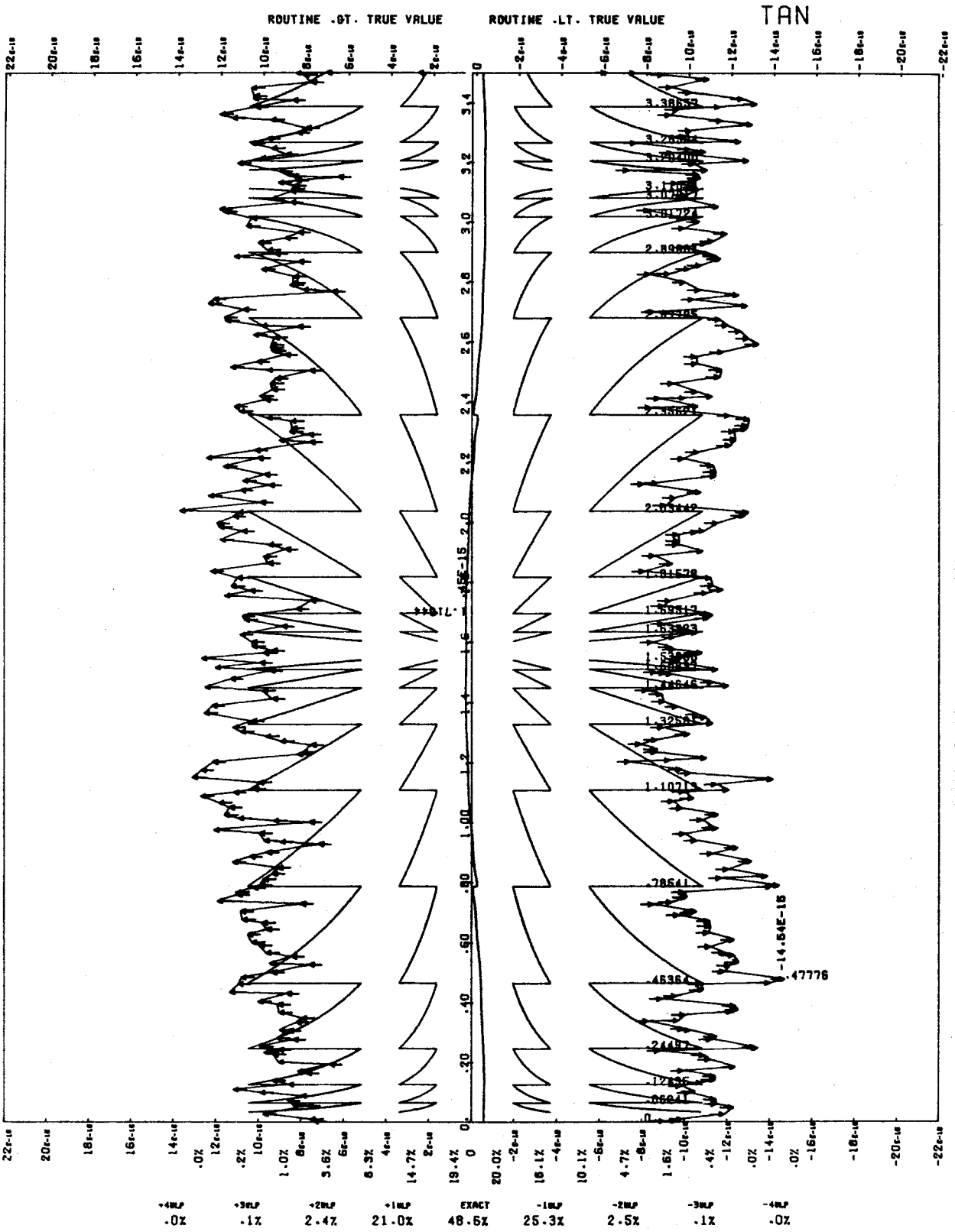
$source of error$	$amount*10^{15}$
range reduction	3.6
rational form	.02
coefficient rounding	<.08
round-off	14.2
upper bound	18.0
maximum observed	14.5

4. 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 may be arbitrarily large near a multiple of pi/2. If x is known to more than double precision, the tangent addition formula may be used if x is less than 3E7:

```
DOUBLE X

(compute X)

T=TAN(SNGL(X))

S=SNGL(X-SNGL(X))

Y=T+S*(1+T**2)/(1-S*T)
```

(S=TAN(S) if X<3E7). This approximation may give less than single precision when S*T is near 1.0, where it is more accurate than TAN(SNGL(X)) but less accurate than SNGL(DTAN(X)).


TAN
ROUTINE -GT- TRUE VALUE
ROUTINE -LT- TRUE VALUE
99990 POINTS. MEAN R.E. -.30OE-15 RMS R.E. 3.78E-15
-14.64E-15
.47776
+4ULP .0%
+3ULP .1%
+2ULP 2.4%
+1ULP 21.0%
EXACT 48.6%
-1ULP 25.3%
-2ULP 2.5%
-3ULP .1%
-4ULP .0%

ROUTINE 1 TAND.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for TAND, the trigonometric tangent function with argument in degrees.

2. METHOD.

Argument range: $(-2^{48}, +2^{48})$ except 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:

$$\tan(X \pm 180^\circ) = \tan(X)$$

$$\tan(X \pm 90^\circ) = -1/\tan(X)$$

3. ERROR ANALYSIS.

The reduction to [-45, +45) is exact: the constant pi/180 has relative error 1.37E-15, and the multiply by this constant has relative error 5.33E-15, for a total error of 6.7E-15. Since errors in the argument of TAN are amplified by at most pi/2, the error due to reduction and conversion is at most 10.52E-15. The error in the final divide 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 100000 points in the interval [0,360) was 17.72E-15.

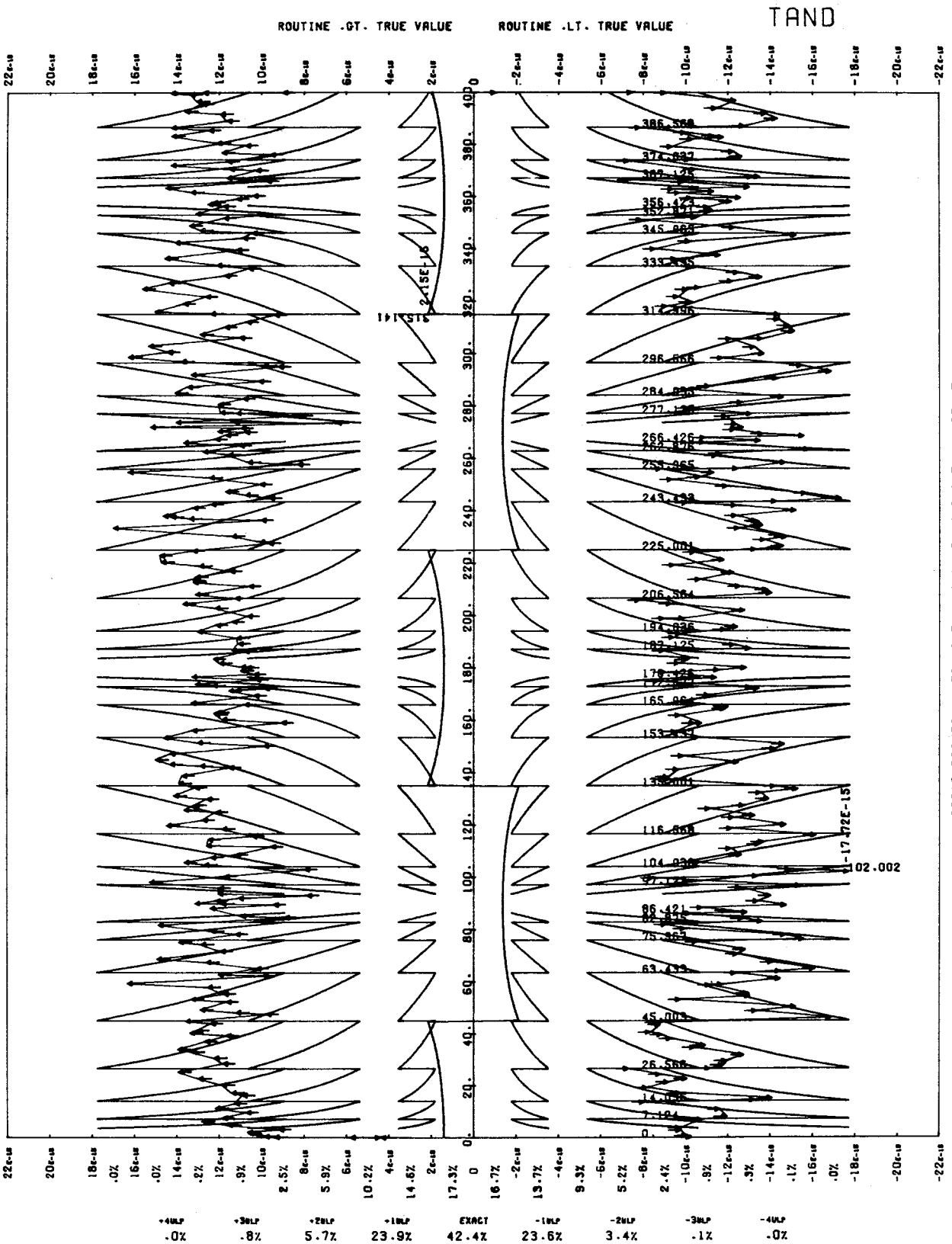
4. EFFECT OF ARGUMENT ERROR.

Errors in the argument X are amplified by at most X/(sin(X)*cos(X)). This function has a maximum of pi/2 within [-45° +45°] but has poles at all multiples of 90° except zero and is at least 2*X elsewhere. When X is known to 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
```

which always returns an accurate value since the range reduction is exact.
(Note: NINT(X) = IFIX(X+SIGN(0.5,X)), the nearest integer.)


TAND
ROUTINE .GT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
100016 POINTS. MEAN R.E. .131E-15 RMS R.E. 4.48E-15
+4ULP .0%
+3ULP .8%
+2ULP 5.7%
+1ULP 23.9%
EXACT 42.4%
-1ULP 23.6%
-2ULP 3.4%
-3ULP .1%
-4ULP .0%

ROUTINE : IANH

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for TANH from FORTRAN programs. TANH computes the hyperbolic tangent function.

2. METHOD.

The argument is loaded into X1 and the call is converted to a TANH. call.

3. ERROR ANALYSIS - see TANH.

4. EFFECT OF ARGUMENT ERROR - see TANH.

ROUTINE : TANH.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts a floating-point argument and returns a floating-point result.

1.2. Purpose. To accept calls by value for computation of the hyperbolic tangent, including converted calls from TANH .

2. METHOD.

The input range is the collection of all definite floating-point quantities in the range [-INF,+INF].

The identity tanh(-x)=-tanh(x) is used to reduce the range to [0,+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,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.

3. ERROR ANALYSIS.

The algorithm error (due to finite approximation, coefficient truncation) is 1.7E-15. For abs(x)<0.55 the form p(x)/q(x) is used, and 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 subtract, 1.0-small, dominates and an upper bound on the error is 4.2E-15; the maximum observed was 3.8E-15.

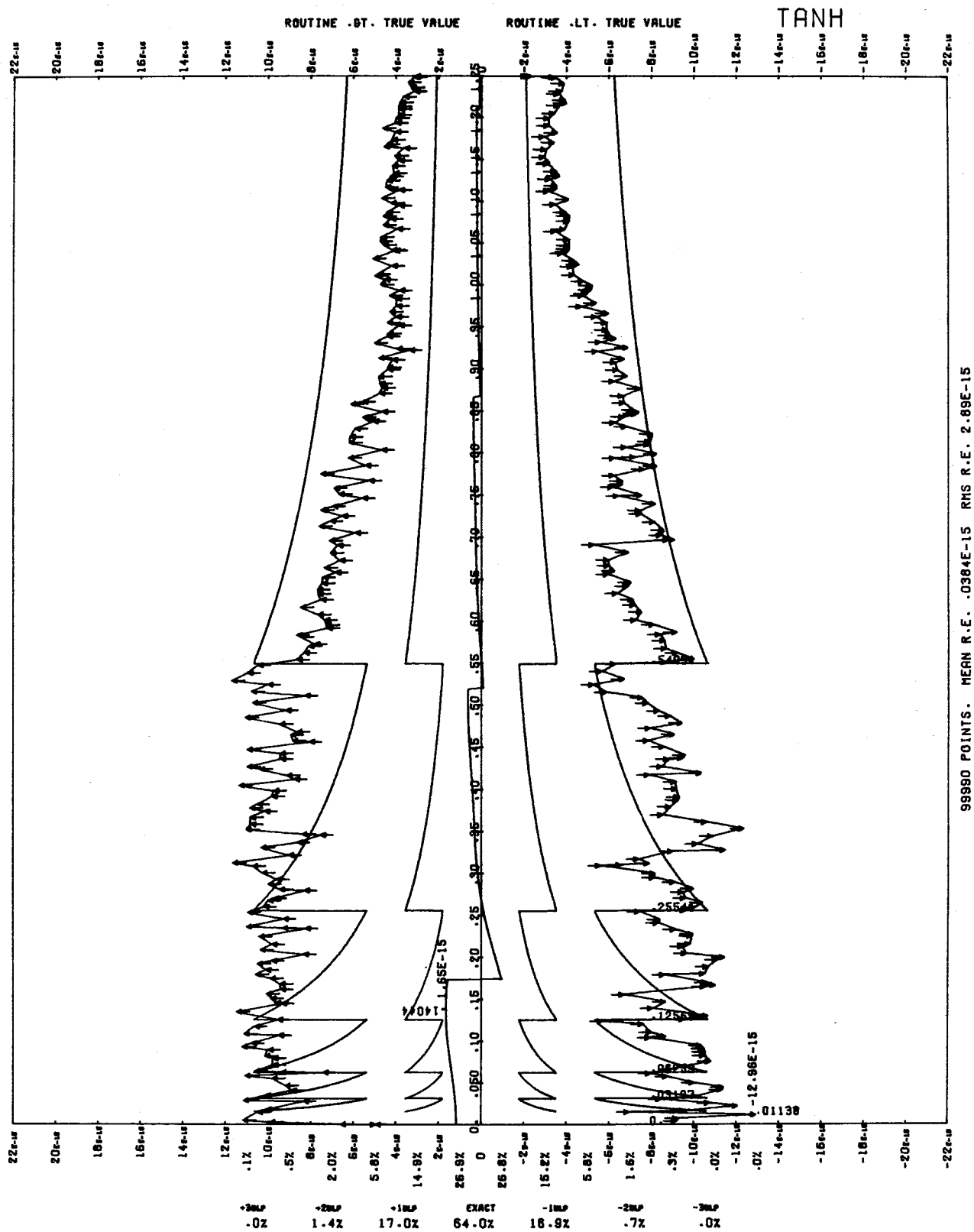
For 0.55≤abs(x)≤1.25 the final operation is 1-R where R becomes smaller as x approaches 1.25, so the worst relative error is near 0.55, namely (contribution from R)+(error in final sum), where R=2*(q-p)/((q-p)+4*(q+p)). An upper bound: 16.7E-15; maximum observed: 10.0E-15.

Relative Error:

$source of error$	$error*10^{15}$
rational form	0.5
coefficient rounding	1.2
round-off	16.5
upper bound	18.2
maximum observed	13.0

4. 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.


TANH
ROUTINE .BT. TRUE VALUE
ROUTINE .LT. TRUE VALUE
99990 POINTS. MEAN R.E. .0384E-15 RMS R.E. 2.89E-15
1.65E-15
-12.96E-15
.01138
+3ULP .0%
+2ULP 1.4%
+1ULP 17.0%
EXACT 64.0%
-1ULP 16.9%
-2ULP .7%
-3ULP .0%

ROUTINE : XTOD*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a floating-point and a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by name for XTOD* generated by FORTRAN programs which raise floating-point bases to double-precision exponents.

2. METHOD.

The formula used is:

base ** exponent = 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 overflow will occur 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.

3. ERROR ANALYSIS.

The algorithm used in XTOD* is the same as that used in XTOD. . See the description of routine XTOD. for ar error analysis.

4. 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.

ROUTINE : XTOD.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN external function. It accepts an argument set comprising a floating-point and a double-precision argument, and returns a double-precision result.

1.2. Purpose. To accept calls by value for XTOD. , calls generated by FORTRAN programs which raise floating-point bases to double-precision exponents.

2. 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 is in-range. The result is computed according to 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.

3. ERROR ANALYSIS.

10,000 argument sets (b,p) were randomly generated, with distribution a product of uniform distribution on (.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 found to be 1.163 . 10**(-25).

4. 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) .

ROUTINE : XIOI*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a floating-point and a fixed-point argument, and returns a floating-point result.

1.2. Purpose. To accept calls by name for XTOI* generated by FORTRAN programs which raise floating-point bases to fixed-point exponents.

2. METHOD.

Load arguments and call XTOI.

3. ERROR ANALYSIS.

Not applicable, since the only errors are round-off errors. See the description of XTOI. .

4. EFFECT OF ARGUMENT ERROR.

If a small error e' occurs in the base b, 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.

ROUTINE : XIOI.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a floating-point quantity and a fixed-point quantity, and returns a floating-point result.

1.2. Purpose. To accept calls by value for XTOI. , generated by FORTRAN programs which raise floating-point quantities to fixed-point exponents.

2. METHOD.

Special case

X indefinite : error
X infinite : error
0. ** 0 : error
X ** 0 = 1.0
(number of bits in I)+(number of bits in scale of X)>8 : use careful code
X ** I = 1.0 /(X **(-I)) if I<0

Quick version: Walk through the binary representation of I, starting with the most significant bit. For each bit, square the result (which was initialized to X); if the next bit is on, also multiply by X.

Careful version: Scale X to be between 0.75 and 1.5, remembering the exponent. Walk through 10 bits of I in the quick-version way. Then repeat (scale+walk) until I is used up. Invert if necessary. Carefully decide if the exponent is too big. If OK, exit.

3. ERROR ANALYSIS - not applicable.

4. 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, we can only say that the absolute error in the result is bounded above by
|p| * max(|b| , |b + e*|))**(p-1) * |e*| .

ROUTINE : XTOY*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of two floating-point arguments and returns a floating-point result.

1.2. Purpose. To accept calls by name for XTOY* generated by FORTRAN programs which raise floating-point bases to floating-point exponents.

2. METHOD.

The formula used is

base ** exponent = exp (exponent.log(base)).

The argument set is checked upon entry. It is invalid if either base or exponent 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 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.

3. ERROR ANALYSIS.

The algorithm used in XTOY* is the same as that used in XTOY. . See the description of routine XTOY. for an error analysis.

4. 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.

ROUTINE : XTOY.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiation routine. It accepts an argument set comprising two floating-point arguments, and returns a floating-point result.

1.2. Purpose. To accept calls by value for XTOY, , generated by FORTRAN programs which raise floating-point bases to floating-point exponents.

2. 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 non-zero, and b**e is in-range.
The formula used is:

$$b^{**}p = \exp(p * \log b),$$

where b > 0.
Upon entry, ALOG. computes log b, and then EXP. computes

$$\exp(p * \log b).$$

The result is returned.

3. ERROR ANALYSIS.

500,000 pairs (b,p) of random numbers were generated with distribution 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 $4.583 * 10^{**}(-12)$ for these 500,000 pairs.

4. 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

$$r * (\log b * e^{**}p + p * (e(b))/b) .$$

ROUTINE: XTOZ*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a floating-point base and a complex exponent, and returns a complex result.

1.2. Purpose. To accept calls by name for XTOZ* generated by FORTRAN programs raising floating-point quantities to complex exponents.

2. METHOD.

If the base b is real and the exponent z = 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))

and

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, if the base b is negative, if the base is zero and the real part of exponent z is greater than zero, if y * log (b) is so large that precision is lost in the computation, or if floating 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.

3. ERROR ANALYSIS.

The algorithm used in XTOZ* is the same as that used in XTOZ. . See the description of routine XTOZ. for an error analysis.

4. 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.

ROUTINE : XTOZ.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set (x,z) where x is a floating-point quantity and z a complex quantity, and returns a complex result.

1.2. Purpose. To accept calls by value for XTOZ., calls which are generated by FORTRAN programs which raise floating-point bases to complex exponents.

2. 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 non-zero, both x and z are definite and in-range, floating overflow does not occur during the computation of x ** u (i.e., |u.log(x)| ≤ 741.67 , and |v.log(x)| ≤ 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 then 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.

3. ERROR ANALYSIS.

400,000 pairs (x,z) of random numbers were generated with distribution the product of a right half of a Cauchy distribution, and the product of two Cauchy distributions. x**z was computed for each of these pairs, first using the routine, and then using double-precision operations. The maximum absolute value of the relative error in the routine was found to be 7.196 * 10**(-10) for these pairs.

4. EFFECT OF ARGUMENT ERROR.

If a small error e(x) occurs in the base x, and 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 x * e(z) + z * e(x)/x) .
```

ROUTINE : ZTOI*

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set consisting of a complex base and a fixed-point exponent, and returns a complex result.

1.2. Purpose. To accept calls by name for ZTOI* generated by FORTRAN programs raising complex quantities to fixed-point exponents.

2. 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 if 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.

3. ERROR ANALYSIS.

Not applicable, since the only errors are round-off errors.

4. 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.

ROUTINE : ZIOI.

1. ROUTINE'S FUNCTION.

1.1. Type. A FORTRAN exponentiating routine. It accepts an argument set comprising a complex and a fixed-point argument, and returns a complex result.

1.2. Purpose. To accept calls by value for ZTOI. , generated by FORTRAN programs which raise complex quantities to fixed-point exponents.

2. METHOD.

Let b be the base and p(≥0) the exponent. If p has a binary representation 000...0i(n)i(n-1)...i(1)i(0) where each i(j)(0≤j≤n) is 0 or 1, then

$$p = i(0) \cdot 2^0 + i(1) \cdot 2^1 + \ldots + i(n) \cdot 2^{**} n$$

and $n = [\log(2)p]$ = greatest integer not exceeding log(2)p. Then

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

The numbers $b = b^{2**0}$, b^2 , b^4 ,...., b^{**2**n} 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 $p \ge 0$ and p is even

$b^{**} p = b.(b^2)^{**} (p-1)/2$ if $p \ge 0$ 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 non-zero if p is negative, both arguments are definite and in-range, and the result is in-range.

3. ERROR ANALYSIS - not applicable.

4. 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

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

APPENDIX A - CLASSIFICATION OF ROUTINES.

The Mathematical Library routines are classified according to the following criteria.

Routine: the name of the SCOPE deck concerned.
Entry: the name of the routine's entry point.
Calls: by name or by value.
Checking: of arguments by the routine.
Argument: the type of arguments to the routine.
FL = floating-point
FI = fixed-point
D = double-precision
C = complex
A = any
Result: the type of the result (or results).
Function: whether an external (Ext) or intrinsic (Int) function.

Routine.	Entry.	Calls.	Checking.	Argument.	Result.	Function.
ALOG	ALOG	Name	Yes	FL	FL	Ext
	ALOG10	Name	Yes	FL	FL	Ext
ATAN	ATAN	Name	Yes	FL	FL	Ext
ATAN2	ATAN2	Name	Yes	(FL,FL)	FL	Ext
	ATAN2.	Value	Yes	(FL,FL)	FL	Ext
ATANH	ATANH	Name	Yes	FL	FL	Ext
CCOS	CCOS	Name	Yes	C	C	Ext
CEXP	CEXP	Name	Yes	C	C	Ext
CLOG	CLOG	Name	Yes	C	C	Ext
COS	COS	Name	Yes	FL	FL	Ext
COSD	COSD	Name	Yes	FL	FL	Ext
COSH	COSH	Name	Yes	FL	FL	Ext
CSIN	CSIN	Name	Yes	C	C	Ext
CSQRT	CSQRT	Name	Yes	C	C	Ext
DACOS	DACOS	Name	Yes	D	D	Ext
DASIN	DASIN	Name	Yes	D	D	Ext
DATAN	DATAN	Name	Yes	D	D	Ext
DATAN2	DATAN2	Name	Yes	(D,D)	D	Ext
DCOSH	DCOSH	Name	Yes	D	D	Ext
DEXP	DEXP	Name	Yes	D	D	Ext
DLOG	DLOG	Name	Yes	D	D	Ext
DLOG10	DLOG10	Name	Yes	D	D	Ext
DMOD	DMOD	Name	Yes	(D,D)	D	Ext
DMOD.	DMOD.	Value	No	(D,D)	D	Ext
DCOS	DCOS	Name	Yes	D	D	Ext

Routine.	Entry.	Calls.	Checking.	Argument.	Result.	Function.
DSIN	DSIN	Name	Yes	D	D	Ext
DSINH	DSINH	Name	Yes	D	D	Ext
DSQRT	DSQRT	Name	Yes	D	D	Ext
DTAN	DTAN	Name	Yes	D	D	Ext
DTANH	DTANH	Name	Yes	D	D	Ext
DTOD*	DTOD$	Name	Yes	(D,D)	D	-
DTOI*	DTOI$	Name	Yes	(D,FI)	D	-
DTOX*	DTOX$	Name	Yes	(D,FL)	D	-
DTOZ*	DTOZ$	Name	Yes	(D,C)	C	-
ERF	ERF	Name	Yes	FL	FL	Ext
ERFC	ERFC	Name	Yes	FL	FL	Ext
EXP	EXP	Name	Yes	FL	FL	Ext
ITOD*	ITOD$	Name	Yes	(FI,D)	D	-
ITOJ*	ITOJ$	Name	Yes	(FI,FI)	FI	-
ITOX*	ITOX$	Name	Yes	(FI,FL)	FL	-
ITOZ*	ITOZ$	Name	Yes	(FI,C)	C	-
SINCOS.	SIN	Name	Yes	FL	FL	Ext
	COS	Name	Yes	FL	FL	Ext
	SIN.	Value	Yes	FL	FL	Ext
	COS.	Value	Yes	FL	FL	Ext
SIND	SIND	Name	Yes	FL	FL	Ext
SINH	SINH	Name	Yes	FL	FL	Ext
SQRT	SQRT	Name	Yes	FL	FL	Ext
	SQRT.	Value	Yes	FL	FL	Ext
TAN	TAN	Name	Yes	FL	FL	Ext
TAND	TAND	Name	Yes	FL	FL	Ext
TANH	TANH	Name	Yes	FL	FL	Ext
XTOD*	XTOD$	Name	Yes	(FL,D)	D	-
XTOI*	XTOI$	Name	Yes	(FL,FI)	FL	-
XTOY*	XTOY$	Name	Yes	(FL,FL)	FL	-
XTOZ*	XTOZ$	Name	Yes	(FL,C)	C	-
ZTOI*	ZTOI$	Name	Yes	(C,FI)	C	-
RANF	RANF	Name	No	A	FL	Ext
	RANGET		No	FL	-	Subroutine
RANSET	RANSET	Name	No	FL	-	Subroutine
AND	AND	Name	No	(A,A,...)	A	Int
COMPL	COMPL	Name	No	A	A	Int
LOCF	LOCF	Name	No	A	FI	Int
MASK	MASK	Name	Yes	FI	A	Int
OR	OR	Name	No	(A,A,...)	A	Int
SHIFT	SHIFT	Name	No	(A,FI)	A	Int
XOR	XOR	Name	No	(A,A,...)	A	Ext
COUNT	COUNT	Name	No	A	FI	Int
ABS	ABS	Name	No	FL	FL	Int
	IABS			FI	FI	

Routine.	Entry.	Calls.	Checking.	Argument.	Result.	Function.
AIMAG	AIMAG	Name	No	C	FL	Int
AINT	AINT	Name	No	FL	FL	Int
AMAX0	AMAX0	Name	No	(FI,FI,...)	FL	Int
AMAX1	AMAX1	Name	No	(FL,FL,...)	FL	Int
AMIN0	AMIN0	Name	No	(FI,FI,...)	FL	Int
AMIN1	AMIN1	Name	No	(FL,FL,...)	FL	Int
AMOD	AMOD	Name	No	(FL,FL)	FL	Int
CMPLX	CMPLX	Name	No	(FL,FL)	C	Int
CONJG	CONJG	Name	No	C	C	Int
DABS	DABS	Name	No	D	D	Int
DBLE	DBLE	Name	No	FL	D	Int
DIM	DIM	Name	No	(FL,FL)	FL	Int
DMAX1	DMAX1	Name	No	(D,D,...)	D	Int
DMIN1	DMIN1	Name	No	(D,D,...)	D	Int
DSIGN	DSIGN	Name	No	(D,D)	D	Int
FLOAT	FLOAT	Name	No	FI	FL	Int
IDIM	IDIM	Name	No	(FI,FI)	FI	Int
INT	INT	Name	No	FL	FI	Int
	IFIX					
	IDINT					
ISIGN	ISIGN	Name	No	(FI,FI)	FI	Int
	SIGN			(FL,FL)	FL	
MAX0	MAX0	Name	No	(FI,FI,...)	FI	Int
MAX1	MAX1	Name	No	(FL,FL,...)	FI	Int
MIN0	MIN0	Name	No	(FI,FI,...)	FI	Int
MIN1	MIN1	Name	No	(FL,FL,...)	FI	Int
MOD	MOD	Name	No	(FI,FI)	FI	Int
REAL	REAL	Name	No	C	FL	Int
	SNGL			D	FL	
ACOSIN.	ACOS	Name	Yes	FL	FL	Ext
	ASIN	Name	Yes	FL	FL	Ext
	ACOS.	Value	Yes	FL	FL	Ext
	ASIN.	Value	Yes	FL	FL	Ext
ALOG.	ALOG.	Value	Yes	FL	FL	Ext
	ALOG10.	Value	Yes	FL	FL	Ext
ATAN.	ATAN.	Value	Yes	FL	FL	Ext
ATANH.	ATANH.	Value	Yes	FL	FL	Ext
CABS.	CABS	Name	Yes	C	FL	Ext
	CABS.	Value	Yes	C	FL	Ext
CCOS.	CCOS.	Value	Yes	C	C	Ext
CEXP.	CEXP.	Value	Yes	C	C	Ext
CLOG.	CLOG.	Value	No	C	C	Ext
COS=SIN	COS.SIN	Value	No	FL	(FL,FL)	Helper
CSIN.	CSIN.	Value	Yes	C	C	Ext
CSQRT.	CSQRT.	Value	No	C	C	Ext

Routine.	Entry.	Calls.	Checking.	Argument.	Result.	Function.
DASNCS.	DACOS.	Value	Yes	D	D	Ext
	DASIN.	Value	Yes	D	D	Ext
DATAN.	DATAN.	Value	Yes	D	D	Ext
DATAN2.	DATAN2.	Value	Yes	(D,D)	D	Ext
DEXP.	DEXP.	Value	Yes	D	D	Ext
DHYP.	DCOSH.	Value	Yes	D	D	Ext
	DSINH.	Value	Yes	D	D	Ext
DLNLOG.	DLOG.	Value	No	D	D	Ext
	DLOG10.	Value	No	D	D	Ext
DSNCOS.	DSIN.	Value	No	D	D	Ext
	DCOS.	Value	No	D	D	Ext
DSQRT.	DSQRT.	Value	Yes	D	D	Ext
DTAN.	DTAN.	Value	Yes	D	D	Ext
DTANH.	DTANH.	Value	Yes	D	D	Ext
DTOD.	DTOD.	Value	No	(D,D)	D	-
DTOI.	DTOI.	Value	No	(D,FI)	D	-
DTOX.	DTOX.	Value	No	(D,FL)	D	-
DTOZ.	DTOZ.	Value	No	(D,C)	C	-
ERF.	ERF	Value	Yes	FL	FL	Ext
	ERFC.	Value	Yes	FL	FL	Ext
EXP.	EXP.	Value	Yes	FL	FL	Ext
HYP.	COSH.	Value	Yes	FL	FL	Ext
	SINH.	Value	Yes	FL	FL	Ext
HYPERB.	HYPERB.	Value	No	FL	(FL,FL)	Helper
ITOD.	ITOD.	Value	No	(FI,D)	D	-
ITOJ.	ITOJ.	Value	Yes	(FI,FI)	FI	-
ITOX.	ITOX.	Value	No	(FI,FL)	FL	-
ITOZ.	ITOZ.	Value	No	(FI,C)	C	-
SINCSD.	COSD.	Value	Yes	FL	FL	Ext
	SIND.	Value	Yes	FL	FL	Ext
TAN.	TAN.	Value	Yes	FL	FL	Ext
TAND.	TAND.	Value	Yes	FL	FL	Ext
TANH.	TANH.	Value	Yes	FL	FL	Ext
XTOD.	XTOD.	Value	No	(FL,D)	D	-
XTOI.	XTOI.	Value	Yes	(FL,FI)	FL	-
XTOY.	XTOY.	Value	No	(FL,FL)	FL	-
XTOZ.	XTOZ.	Value	No	(FL,C)	C	-
ZTOI.	ZTOI.	Value	No	(C,FI)	C	-

APPENDIX B - ERROR RECOVERY.

All routines in the FORTRAN common library checking arguments and issuing error messages allow for standard and non-standard error recovery, as described in the FORTRAN Extended Version 4 Reference Manual. Routine: the name of the loader deck concerned. 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, and the first word address of the parameter list in the previous call, otherwise. 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 will return 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 will result in the printing of 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 contains 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 FWA 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 or indefinite (or 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 Extended Reference Manual.) (Step (f.) may precede step (d.).)

(g.) Return jump 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 Extended Reference Manual.

As the first routines to be rewritten in a project to implement full error checking in all routines, some routines (listed in Appendix A) now detect errors and issue messages for all bad arguments passed to them. These routines call new 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 section III of the Introduction.

(e.) Return jump to 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, set X2 to the first word address of the constructed message, then execute the communication cell SYSAID. , after traceback linkage information has been inserted in its lower 18 bits.

(g.) Return +IND. in registers X6 and X7 , and restore registers A0 , X1 , X2 (and X3 and X4 , if entry was to SYS=AID).

```
s = .746926199335419 * 10**-3
```

APPENDIX C - TIMING OF ROUTINES

The times listed below were determined empirically, and arguments to routines were chosen as many as practicable to cover all the possibilities for times to each routine. CYBER 76 times were obtained through the machine instruction 01610 which accesses a hardware clock, while CYBER 72, 73 and 74 times were obtained 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 through use of a system-maintained real-time clock which is synchronized with a hardware clock on one of the data channels. These times do not include time for setting up arguments and parameter lists, but measure from the time a return-jump to the routine is issued, to the time that the next instruction in sequence is issued. All times given are in minor cycles (or clock-periods). On CYBER 72, 73 and 74, 1 minor cycle = 100 nanoseconds, while on CYBER 76, 1 clock-period = 27.5 nanoseconds. On CYBER 171, 172 and 173, 1 minor cycle = 50 nanoseconds.

Certain facts should be noted. On CYBER 76, a return jump may be delayed in execution if the instruction stack control has requested one or more instruction words that have not arrived at the instruction stack. Thus, CYBER 76 routine times depend on how the routine is called. On CYBER 72 and 73, a floating instruction executes at least 48 minor cycles faster if either of the operands is zero, infinite or indefinite. If in the course of evaluating an algorithm for computation of a function, a routine happens to produce an intermediate zero result, it will execute faster by at least 48 minor cycles if this intermediate result is combined arithmetically with anything else. The number of possibilities for this case is too large for enumeration in this appendix.

Some routines will naturally call others, but the time listed under each routine only the time spent in that routine, and does not include time spent in return jumps to and execution of other routines. To find total execution time in a routine, one must add times for execution at entry points with arguments listed after an "&".

Timings are supplied here for valid argument sets only. Take the time for the first alternative listed which covers the argument concerned.

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)					
x valid & ACOS. (x)			56	35	47
ASIN (x)					
x valid & ASIN. (x)			59	35	41
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 |x| 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
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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
ALOG					
ALOG10 (x)			67	42	46
& ALOG10. (x)					
ALOG (x)			67	40	49
& ALOG. (x)					
ALOG10. (x)					
x infinite or indefinite			253	192	110
& SYSAID. (Append. B)					
0.			266	199	120
& SYSAID. (Append. B)					
x valid, x<0.			285	213	129
& SYSAID. (Append. B)					
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			298	176	97
& SYSAID. (Append. B)					
0.			311	183	112
& SYSAID. (Append. B)					
x valid, x>0			330	192	117
& SYSAID. (Append. B)					
x valid, x= y*2**n, n integral					
1.≤y<1.8572	941		814	197	119
1.8572<2	1072		933	218	143

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
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	
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	

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
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)					
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. (Append. B)					

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
CCOS.					
CCOS. (z)					
z valid		327	279	78	71
& HYPERB. (im(z))					
& COS.SIN (re(z))					
CEXP					
CEXP (z)					
|re(z)|> 741.67		356	273	158	120
& SYSERR. (Append. B)					
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. (Append. B)					
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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
CONJG					
CONJG (z)					
z valid		128	101	58	68
-					
COS. -- see SINCOS.					
-					
COSH. -- see HYP.					
COS=SIN					
COS.SIN (x)					
|x| > pi*2**6		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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
COUNT					
COUNT (x)		148	133	49	62
CSIN					
CSIN (z)					
|re(z)| > pi*2**46 & COS.SIN (re(z)) & SYSERR. (Append. B)		386	295	162	121
|im(z)| > 741.67 & COS.SIN (re(z)) & SYSERR. (Append. B)		470	362	221	136
z valid & COS.SIN (re(z)) & HYPERB. (im(z))		551	436	181	123
CSIN.					
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

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DASNCS.					
DACOS.					
0≤x≤.09375	3344			529	
.09375<x<.7071	4844			853	
.701<x<.9956	4823			841	
.9956<x<1	4228			756	
DASIN.					
0≤x≤.09375	3260			492	
.09375<x<.7071	4756			814	
.701<x<.9956	4779			820	
.9956<x<1	4197			736	

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DATAN					
DATAN (x)					
x valid			130	42	143
& DATAN. (x)					
DATAN.					
DATAN.					
x valid, and:					
|x| < 1.			144	74	40
& DTN. (see routine' description)					
|x| ≥ 1.			320	134	73
& DATCOM. (see routine' description)					
DATAN2					
DATAN2 (y,x)					
y,x valid, and (y,x)≠(0,0)			124	46	66
& DATAN2. ((y,x))					
DATAN2.					
DATAN2 (y,x)					
where both are valid, and					
(y,x)≠(0,0), and:					
|y| ≤ |x|			276	144	65
& DATCOM. (see routine' description)					
|y| > |x|			283	175	71
& DATCOM. (see routine' description)					
DATCOM.					
DATCOM. (y,x) (from DATAN2.)					
argument set validated. If n					
is nearest integer to					
8*min(|x|,|y|)/max(|x|,|y|),					
then:					
n=0			3150	521	337
n≠0 and min(|x|,|y|)-n/8*max(|x|,|y|)≠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≠0 and (y - n/8)≠0			3356	587	367
otherwise			1212	307	200

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
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<-643.240583559629247139191409:			515	163	107
& DEULER.					
x otherwise:			378	147	100
& DEULER.					

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DHYP					
DCOSH. (x)					
x valid and:					
abs(x)>42.360630379701426385855602079: & DEULER.			560	215	127
abs(x)/log 2 ≥ 48: & DEULER.			546	182	101
abs(x)/log 2 ≥ 24: & DEULER.			658	203	125
x in [-1/2 log 2,1/2 log 2]: & DEULER.			233	136	86
x otherwise & DEULER.			719	229	132
DSINH. (x)					
x valid and:					
abs(x)>42.360630379701426385855602079: & DEULER.			575	202	119
abs(x)/log 2 ≥ 48: & DEULER.			515	160	93
abs(x)/log 2 ≥ 24: & DEULER.			625	206	136
x in [-1/2 log 2,1/2 log 2]: & DEULER.			155	94	64
x otherwise & DEULER.			720	226	134
DIM					
DIM (x,y)					
x,y valid	191	150	84	96	

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DNLOG.					
DLOG10. (x)					
x=(2**n)*y					
1/2≤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≤y< 1/2**.5	6797	7576	6631	1158	731
1/2**.5≤y<1	6636	7444	6487	1144	731
DLOG					
DLOG (x)					
x=0.		284	215	136	83
& SYSARG. SYSERR. (Append. B)					
x<0		332	251	142	105
& SYSARG. SYSERR. (Append. B)					
x valid		150	96	101	68
& DLOG. (x)					
DLOG10					
DLOG10 (x)					
x=0.		284	216	130	89
& SYSARG. SYSERR. (Append. B)					
x, x<0		333	255	216	105
& SYSARG. SYSERR. (Append. B)					
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. (Append. B)					
(x,y) valid		266	203	34	97
& DMOD. (x,y)					
DMOD.					
DMOD. (x,y)					
x,y valid,y≠0 |x/y|≥2^{96}	2007			582	
|x/y|≥2^{48}	1426			431	
|x/y|<2^{48}	841			281	

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)					
|x|>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)					
|x|>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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DSQRT					
DSQRT (x)					
x<0.		282	234	125	85
& SYSARG. SYSERR. (Append. B)					
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.					
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	
DTANH					
DTANH (x)					
x valid			124	120	42
& DTANH. (x)					
DTANH.					
DTANH. (x)					
x valid and:			765	217	134
|x|<1/8:					
& DEULER.(x)					
|x|≥ 32:			214	103	62
If x (or 2x)=y+n*log(2), n>47:			619	163	122
& DEULER.(2x)					
otherwise:			1055	311	171
& DEULER.(2x)					

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DTOD*					
DTOD$ (x,y)					
(0.,0.)		441	341	192	158
& SYSERR. (Append. B)					
(0,y), to y>0		352	387	153	208
(0,y), to y<0		439	340	195	152
& SYSERR. (Append. B)					
x<0		410	318	169	130
& SYSERR. (Append. B)					
(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. (Append. B)					
(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 n with -n		467	415	114	73
(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)≤ log(2)n≤t≤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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
DTOX*					
DTOX$ (x,y)					
(0.,0) & SYSERR. (Append. B)		426	337	199	184
(0.,y),y<0		299	239	245	228
(0.,y),y<0 & SYSERR.		426	335	107	178
x<0 & SYSERR. (Append. B)		383	299	176	145
(x,y) valid, x>0 & DLOG. (x) & DEXP. (y*log 2)		708	606	236	158
DTOX.					
DTOX. (x,y)					
(0.,y)		95	74	59	54
(x,y) valid & DLOG. (x) & DEXP. (y*log 2)		460	415	85	63
DTOZ*					
DTOZ$ (x,z)					
(0.,0.+i.0.) & SYSERR. (Append. B)		403	311	189	148
x<0 & SYSERR. (Append. B)		342	263	168	117
(0.,z), Re(z)≥0 & SYSERR. (Append. B)		277	211	102	136
(0.,z), re(z)<0 im(z)≠0 & SYSERR. (Append. B)		432	312	221	136
(0.,z), re(z)<0 im(z)=0 & SYSERR. (Append. B)		432	312	223	136
(x,z) valid & ALOG. (X) & EXP. (re(z)*log x) & COS.SIN (im(z)*log x)		762	636	231	90
DTOZ.					
DTOZ. (x,z)					
x=0.		103	81	63	59
x,z valid, x≠0 & ALOG. (x) & EXP. (re(z)*log x) & COS.SIN (im(z)*log x)		480	426	149	85

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
ERF.					
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					

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
EXP					
EXP (x)			34	57	38
& EXP. (x)					
EXP. (x)					
x infinite			268	140	89
& SYSAID. (Append. B)					
x indefinite			201	103	58
& SYSAID. (Append. B)					
x valid, x>741.67			304	155	97
& SYSAID. (Append. B)					
x valid x≥512.	932		864	184	130
x valid, x<-675.84			298	157	119
& SYSAID. (Append. B)					
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

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
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. (Append. B)					
(0,x),x<0	582	483	166	123	
& SYSERR. (Append. B)					
(0,x),x>0	496	451	109	173	
n<0	322	153	128	92	
& SYSERR. (Append. B)					
n>0, x*log n overflows	695	584	238	914	
& SYSERR. (Append. B)					
& DLOG. (n)					
(n,x) valid, n>0	598	613	264	125	
& DLOG. (n)					
& DEXP. (x*log n)					
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)					

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	 72	 73	 74	 76
ITOJ*					
ITOJ$ (m,n) & ITOJ. (m,n)					
ITOJ.					
ITOJ. (m,n) m**n <2**8					
(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					

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
ITOX*					
ITOX$ (n,x)					
(0,0.)			389	267	175	158
& SYSERR. (Append. B)					
(0,x),x>0			352	313	114	208
(0,x),x<0			346	268	178	149
& SYSERR. (Append. B)					
n<0			289	223	136	102
& SYSERR. (Append. B)					
n>0, |x*logn|≥741.67			459	354	246	122
& ALOG. (n)					
& SYSERR. (Append. B)					
(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.+i0.)			376	291	165	129
& SYSERR. (Append. B)					
(0,z), re(z) <0,im(z)=0			395	287	199	120
& SYSERR. (Append. B)					
(0,z),re(z)>0			238	187	210	91
im(z)=0 (0,z),im(z)≠0,			376	291	165	120
& SYSERR. (Append. B)					
re(z)<0 (n,z),n<0			316	241	144	104
& SYSERR. (Append. B)					
(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)					

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. (Append. B)					
n<0		274	210	127	83
& SYSERR. (Append. B)					
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

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
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
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. (Append. B)						
x=0.	888			821	193	141
x valid, |x|>pi*2**46				166	109	79
& SYSAID. (Append. B)						
x valid, |x|≤pi*2**46	1283			1256	194	141
COS. (x)						
x infinite or indefinite				169	115	75
& SYSAID. (Append. B)						
0.	831			757	188	165
x valid |x|≤pi*2**46	1190			1230	188	178
x valid |x|>pi*2**46					220	165
& SYSAID. (Append. B)						

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
SQRT					
SQRT (x)			78	40	37
& SQRT. (x)					
SQRT. (x)					
x infinite, indefinite or negative			222	180	279
& SYSAID. (Append. B)					
x valid, x≠0	527		523	119	101
0.	244		393	196	97
SYS=AID					
, SYSAID.					
(1 in lower half of RJ word)			359		133
& SYSERR. (Append. B)					
(other than 1 in lower half of RJ word)			986	423	267
& SYSERR. (Append. B)					
SYS=1ST					
SYS1ST.					
(1 in lower half of RJ word			299		106
& SYSERR. (Append. B)					
(other than 1 in lower half of RJ word)			892	377	239
& SYSERR. (Append. B)					
TAN					
TAN (x)					
x valid, not an odd mutiple of pi/2		216	175	142	109
& TAN. (x)					
TAN.					
TAN. (x)					
x=0	617			155	
|x|<2^{47}, x=n(pi/2)+y, pi/4<x<pi/4					
n=0	1065			156	
n odd	1061			147	
n even	1061			157	

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
TANH					
TANH (x)					
x valid		98	75	65	58
& TANH. (x)					
TANH.					
TANH. (x)					
x valid and:					
|x|≤.55	812			153	
.55<|x|≤17.1	970			210	
|x|>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. (Append. B)					
(0.,x),x valid x>0		476	389	158	204
x<0		454	343	199	147
& SYSERR. (Append. B)					
(x,y)x<0, x valid		403	304	167	132
& SYSERR. (Append. B)					
x,y valid,x<0, y*logx>741.67		753	606	280	188
& DLOG. (x)					
& SYSERR. (Append. B)					
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)					

```
Routine
    Entry Points
        Arguments
            &Times at Entry Points (argument)      Times for CYBER
                                                  173   72    73    74    76
-------------------------------------------------------------------------------

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
```

Routine / Entry Points / Arguments / &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
XTOY*					
XTOY$ (x,y)					
(0.,0.)		283	179	186	155
& SYSERR. (Append. B)					
(0.,x),x valid,x>0		396	330	92	198
(0.,x), x valid,x<0		368	284	189	155
& SYSERR. (Append. B)					
x,y valid, x<0		309	243	157	114
& SYSERR. (Append. B)					
x,y valid, x>0 valid, x≠0		399	315	201	141
& ALOG. (x)					
& EXP. (y*log x)					
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. (Append. B)					
z valid,re(z)=0		355	294	180	121
& SYSERR. (Append. B)					
x,z valid, x<0		312	240	156	104
& SYSERR. (Append. B)					
x,z valid,					
x>0 re(z)*log x >741.67		632	469	251	130
& ALOG. (x)					
& SYSERR. (Append. B)					
(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)					

Routine Entry Points Arguments &Times at Entry Points (argument)	Times for CYBER 173	72	73	74	76
ZTOI*					
ZTOI$ (z,n)					
(0.,0.)	379	303	189	140	
& SYSERR. (Append. B)					
(0,x), x>0	232	178	137	111	
(0.,x),x<0	369	287	182	61	
& SYSERR. (Append. B)					
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≤t≤					
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	

FIGURES INDEX

Figure 1
Relative Error in SQRT Algorithm over [.5, 1.]

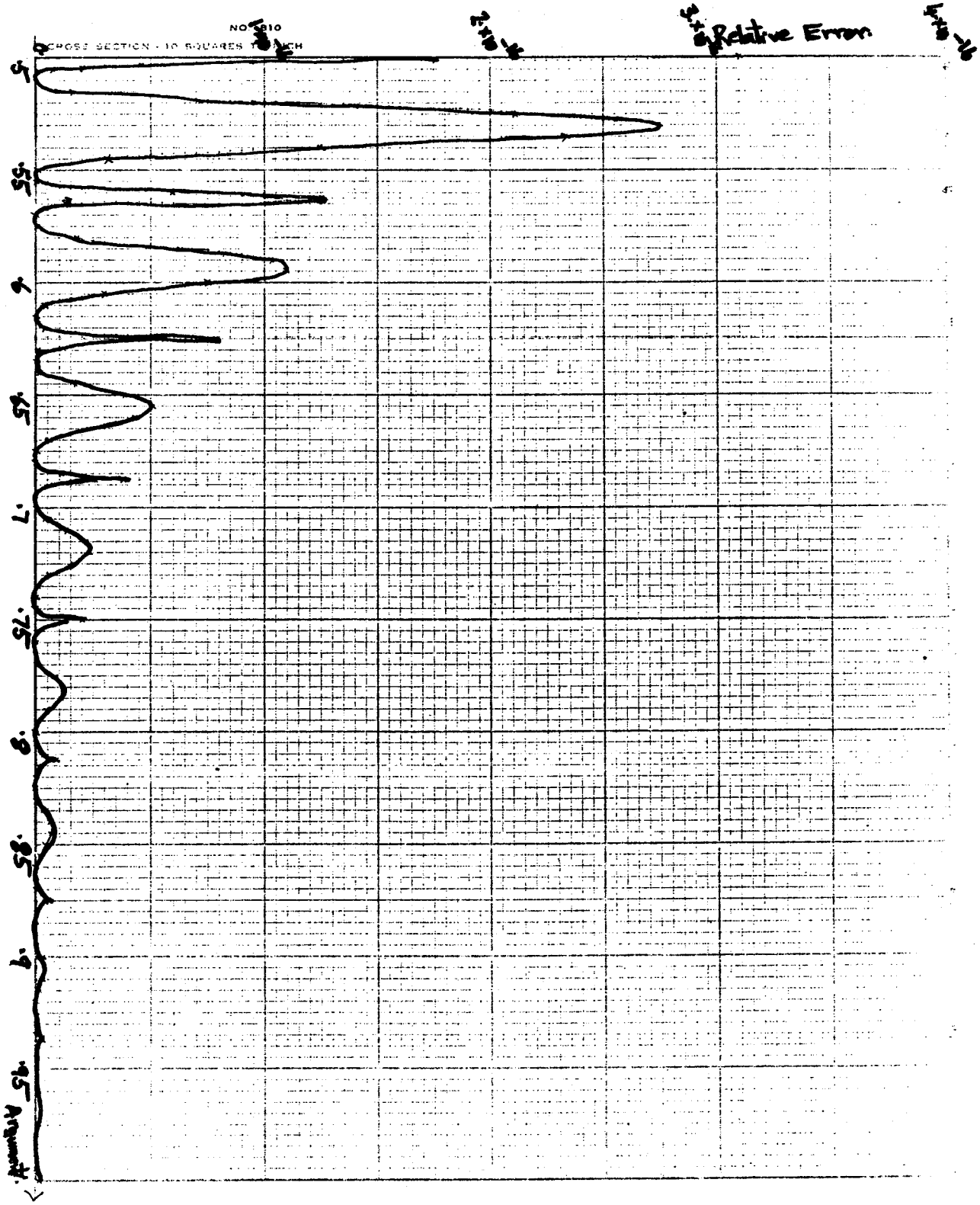


FIGURE 2

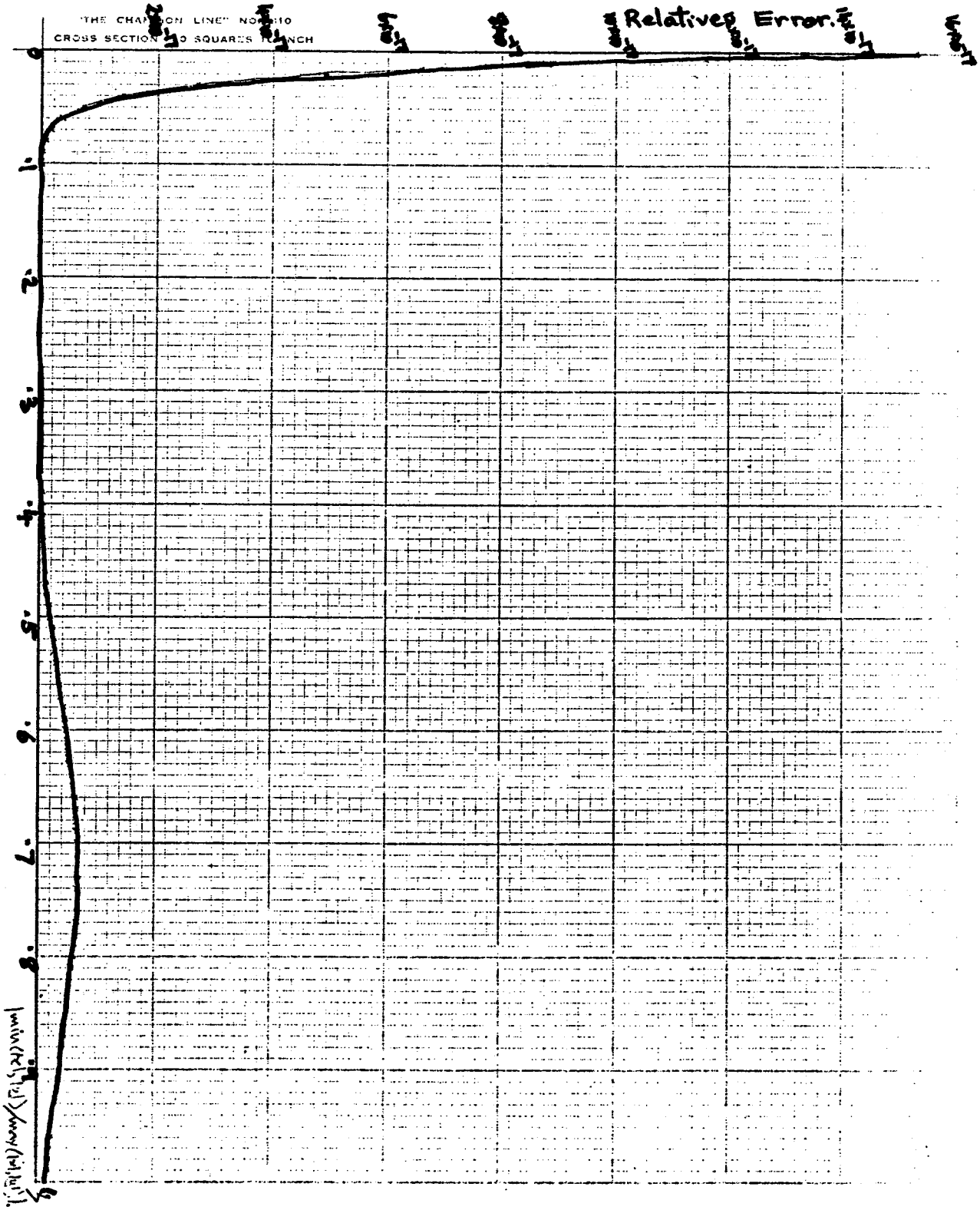


FIGURE 3

Error in the algorithm used to approximate sin(x) over [-pi/4, pi/4]

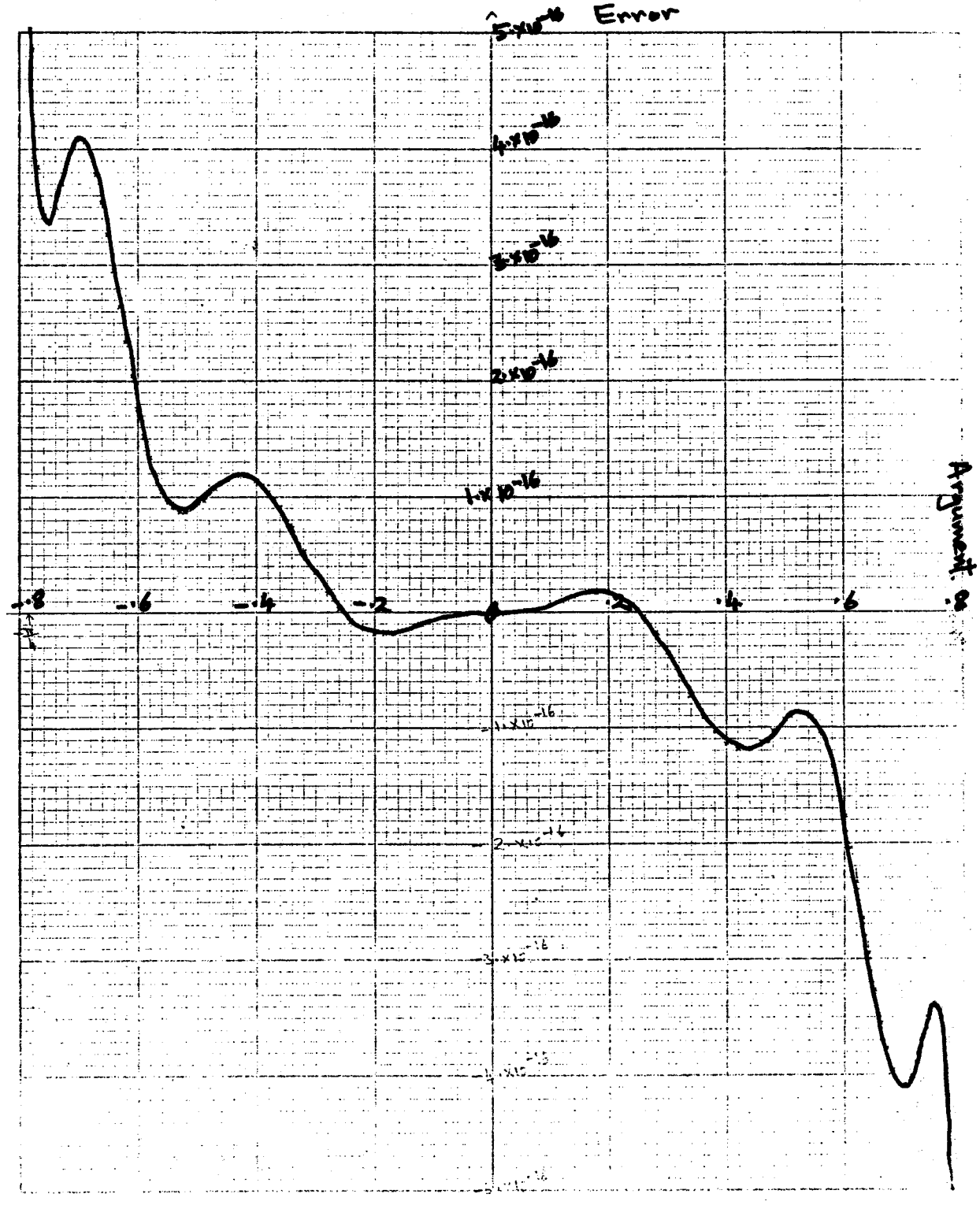


FIGURE 4

Error in the algorithm used to approximate cos(x) over [-pi/4, pi/4]

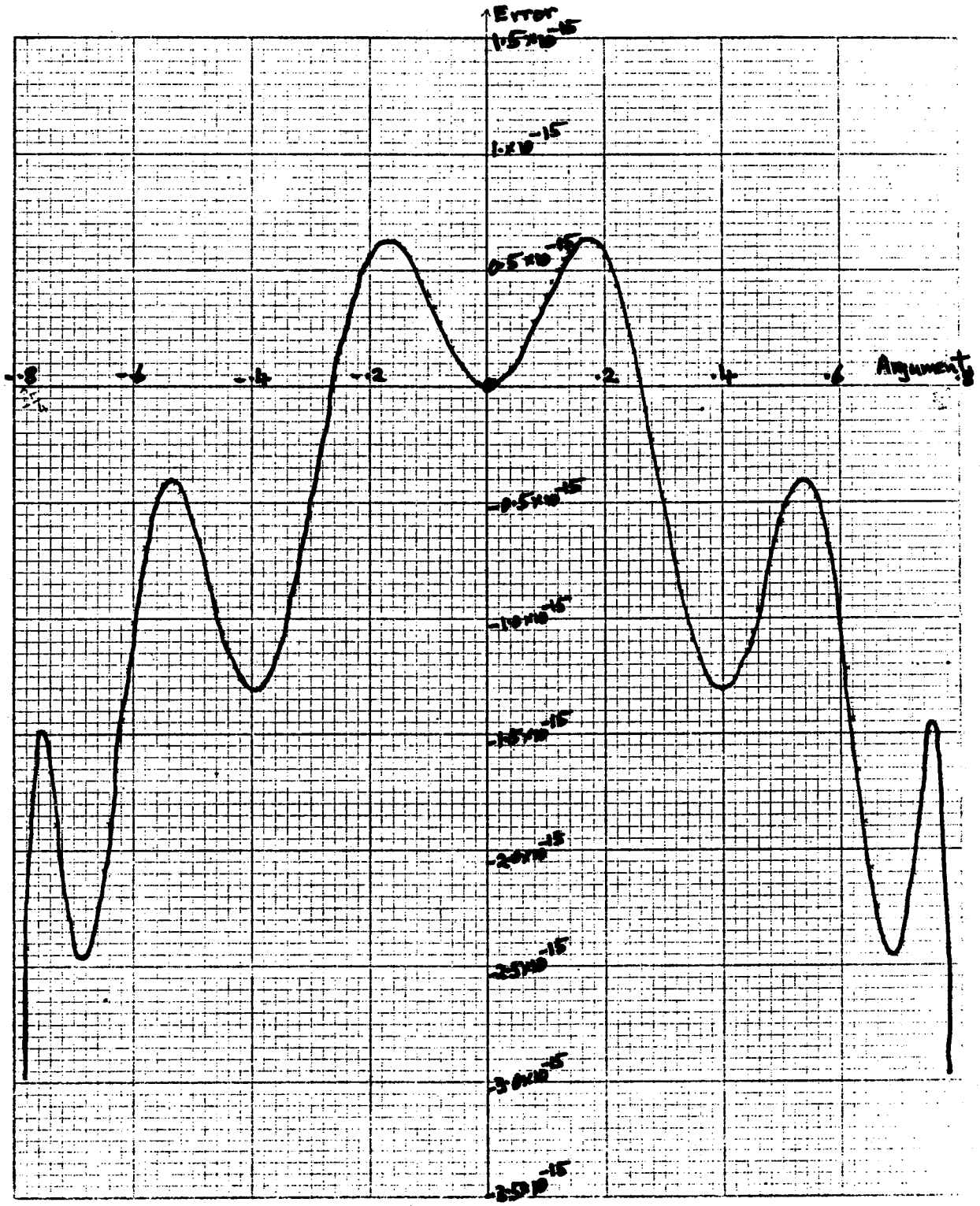


FIGURE 6
Graph of error in the algorithm used in ARCSIN. for arcsin

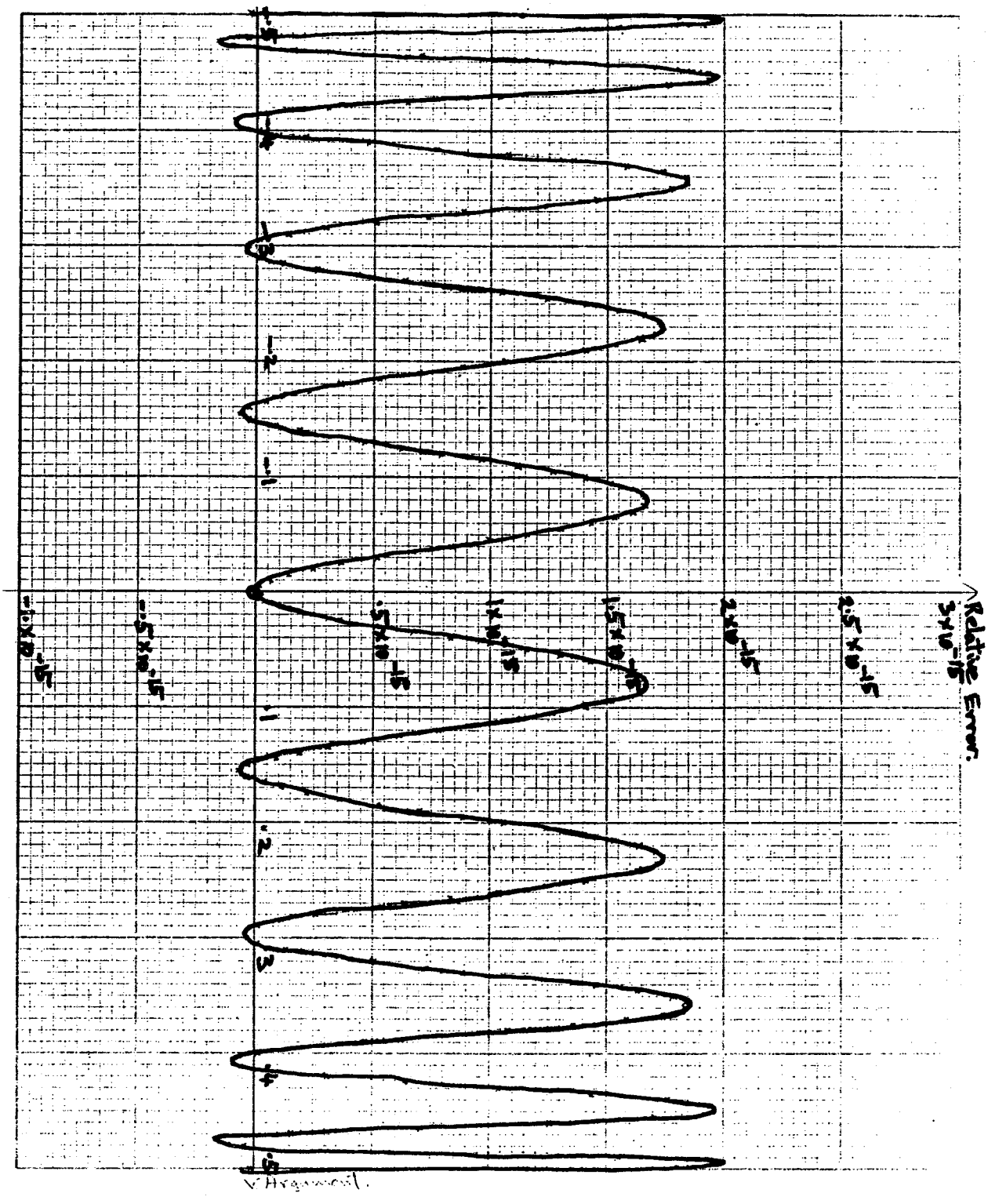


FIGURE 7
Graph of Relative Error in the Algorithm Used For ATAN. over [0, 1/16]

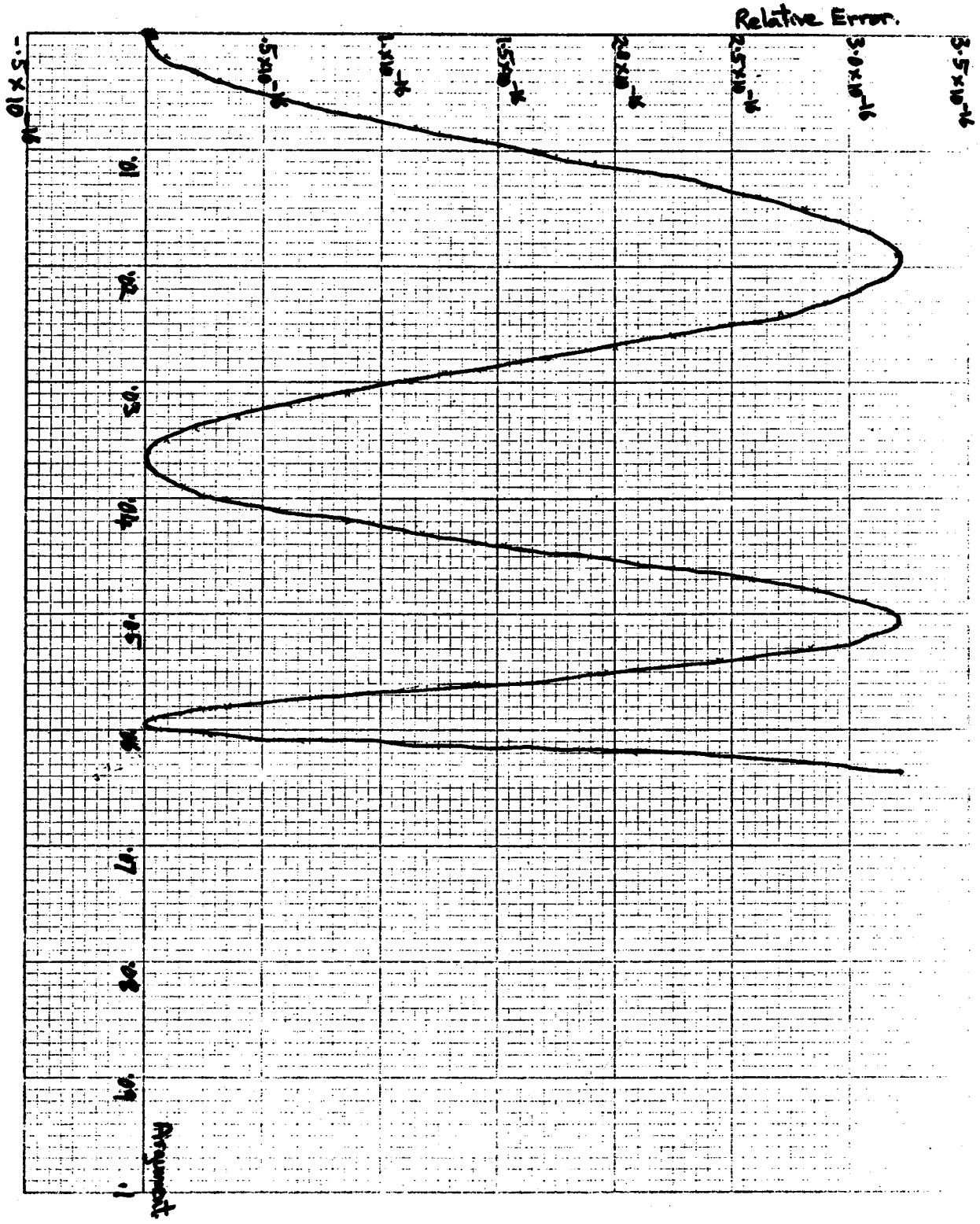


FIGURE 8
Error in the Algorithm Used in ALOG over [1., 2.]

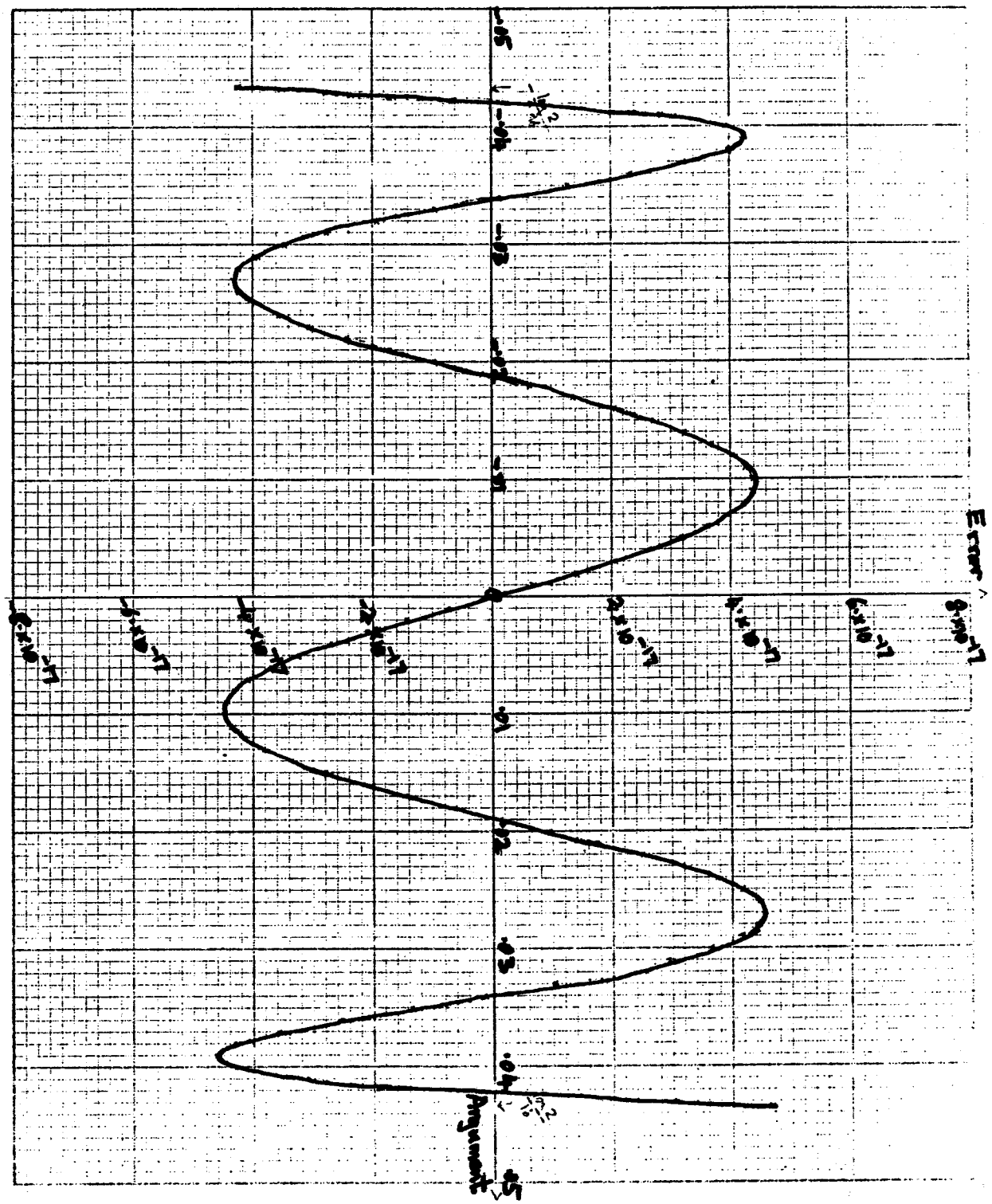


FIGURE 9
Error in the algorithm used in EXP.
over (-log 2/16, log 2/16)

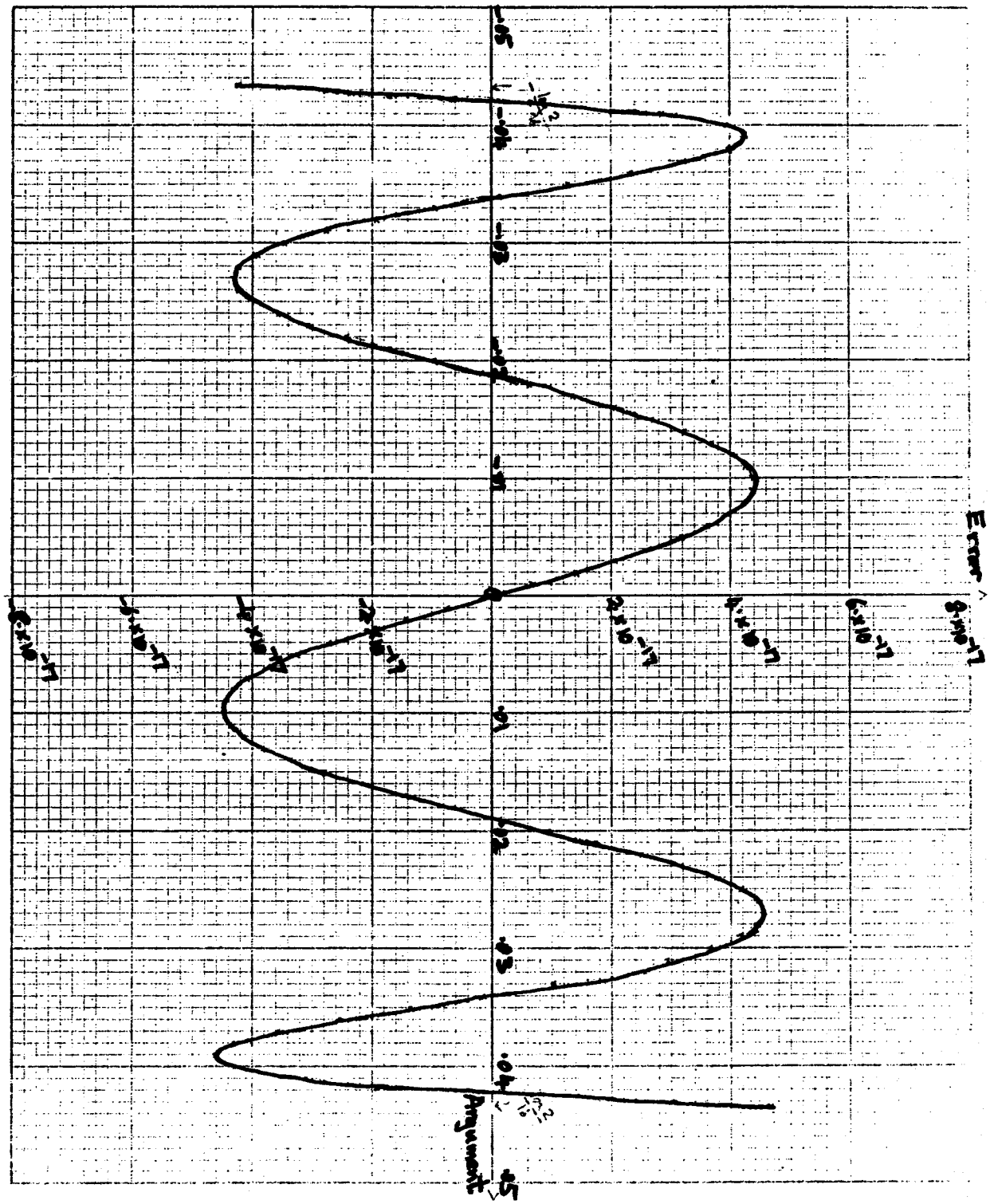


FIGURE 10
Error of approximation in the series (truncated)
for sinh over (-22,.22)

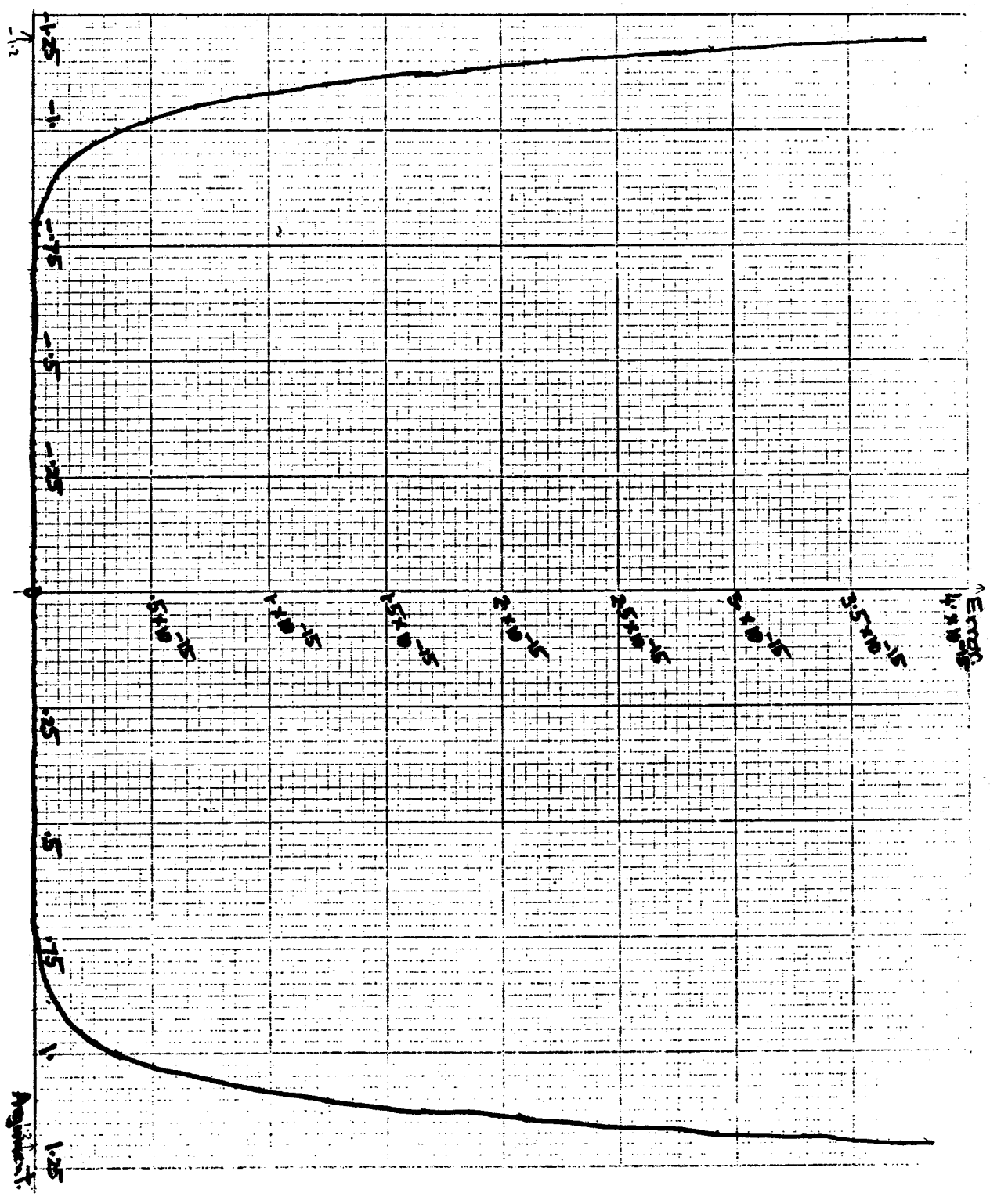


FIGURE 11
Error in the polynomial approximation to tanh, over (-12,.12)

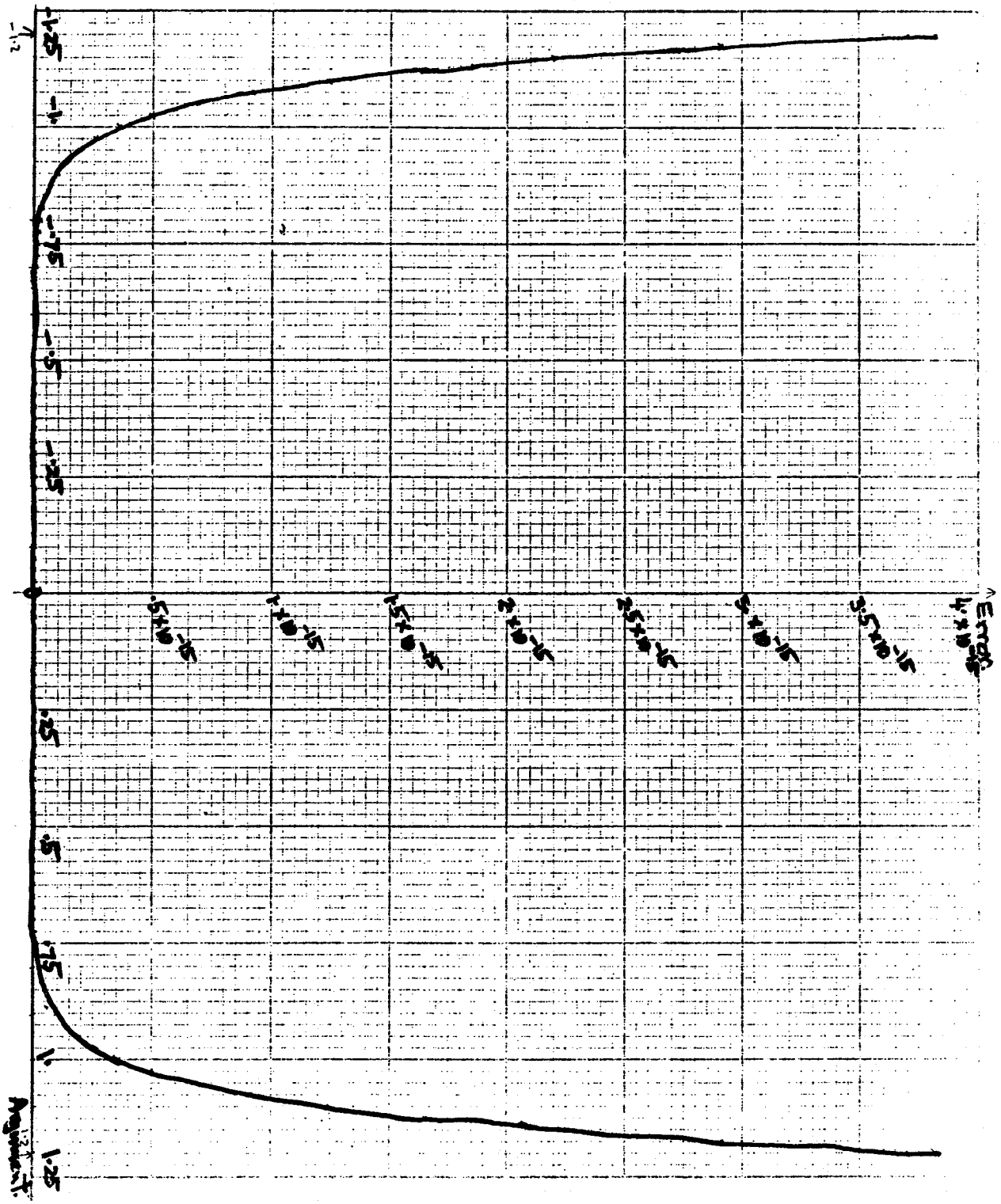


FIGURE 12
Relative error in the approximation
to double square root over (.25,1.0)

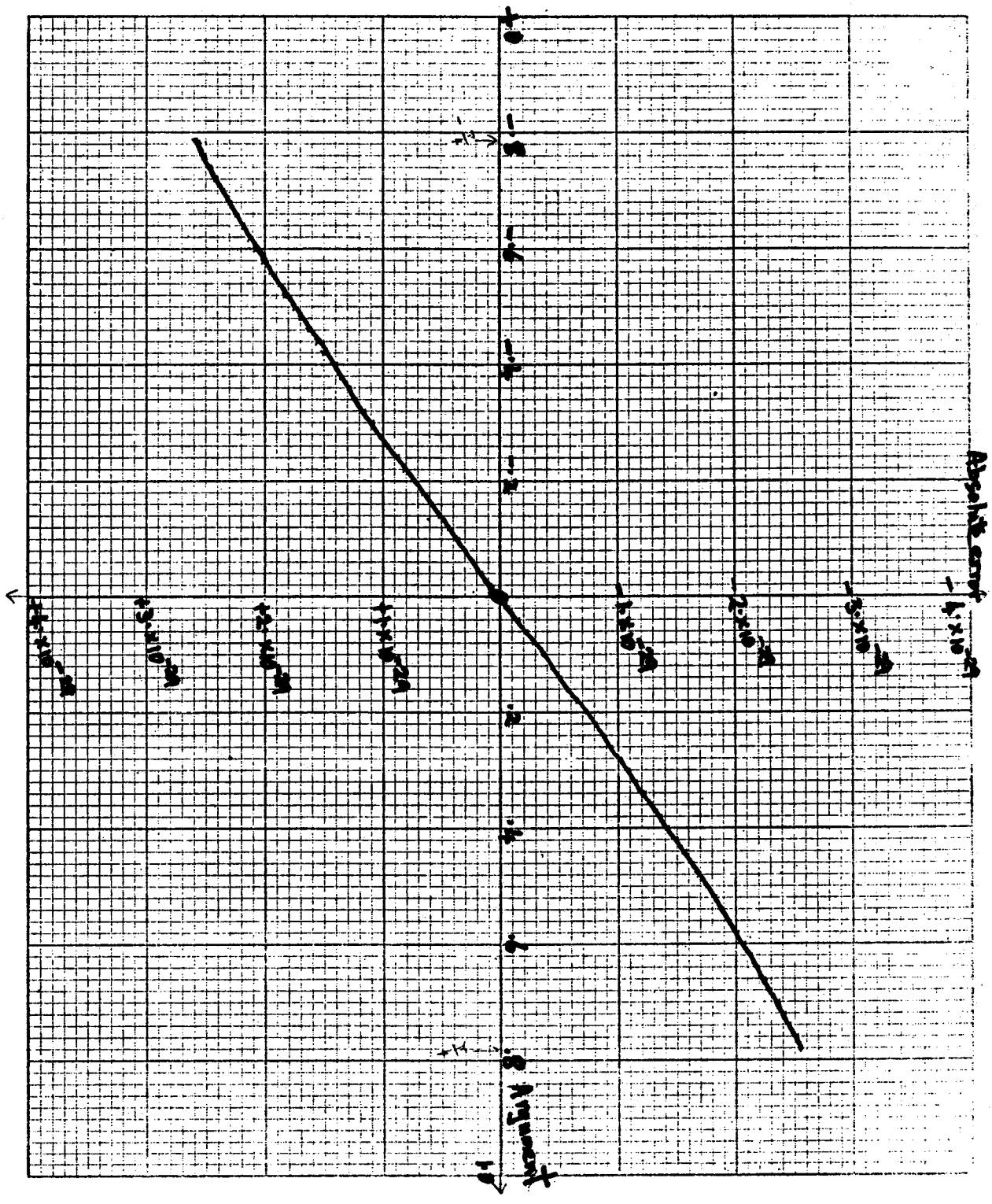


FIGURE 13
Absolute error in the algorithm used in DSNCOS.
for the computation of sine

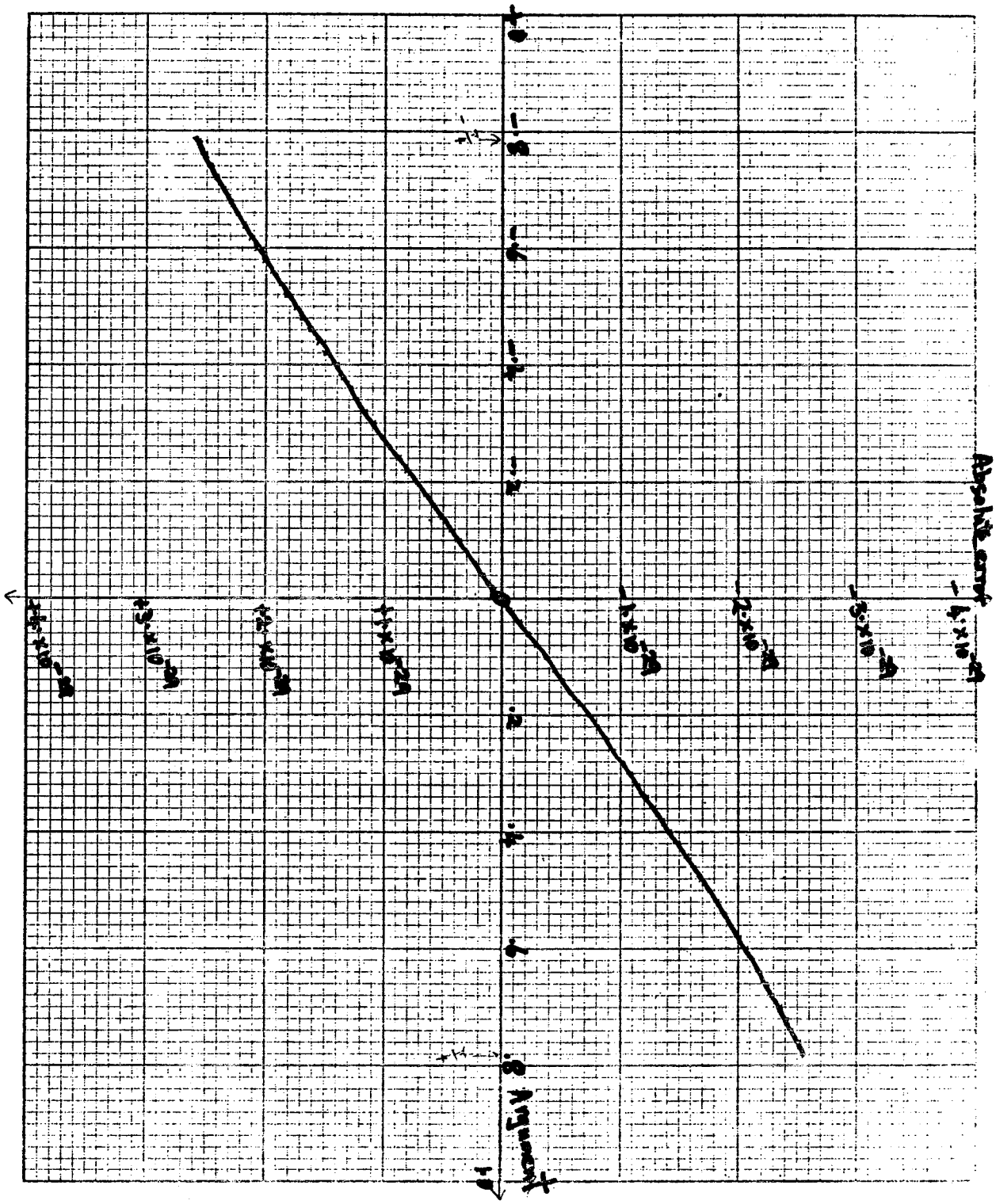


FIGURE 14
Absolute error in the algorithm used in DSNCOS.
for the cosine

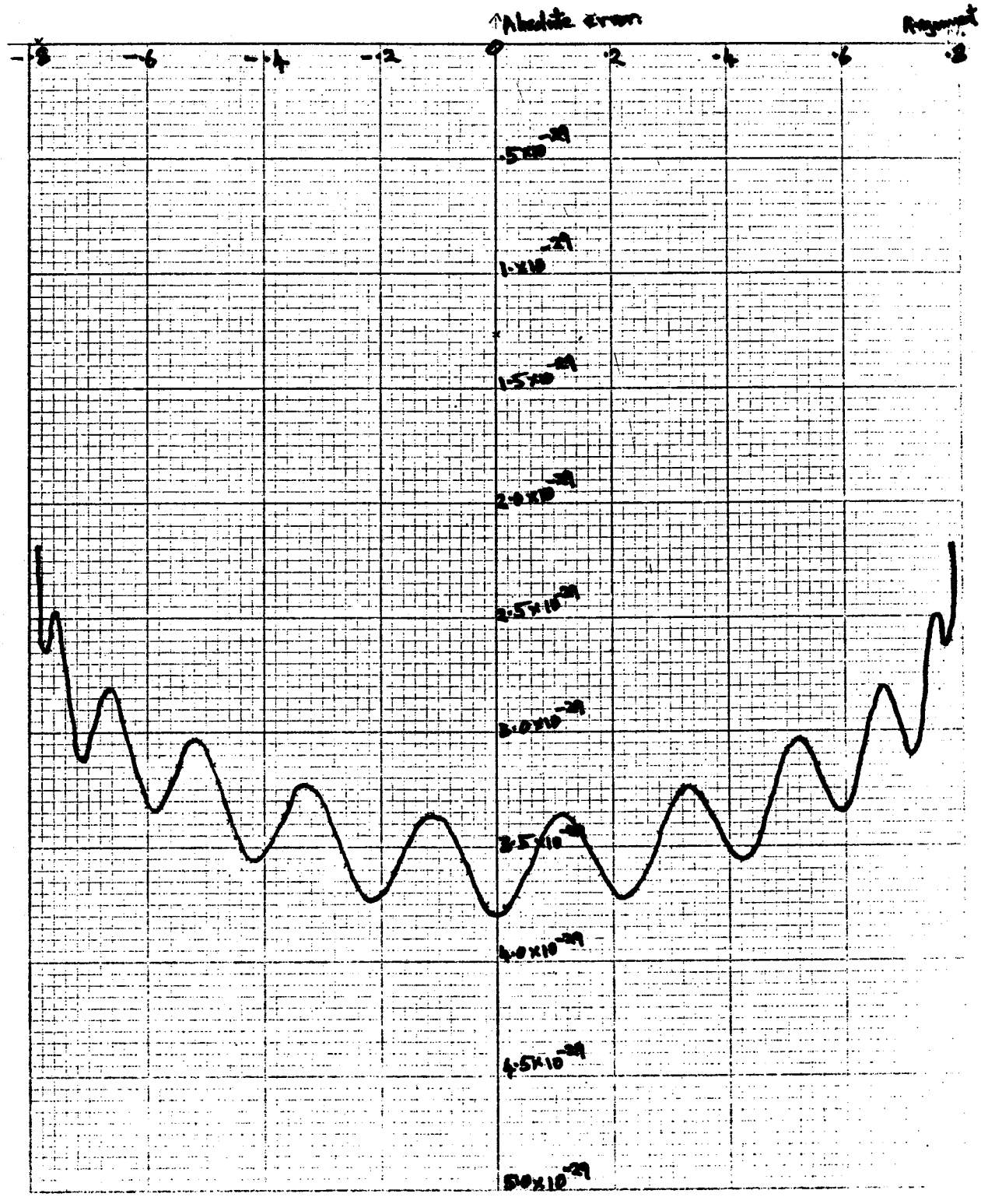


FIGURE 15
Relative error in the algorithm used for approximation of exp in DEXP.

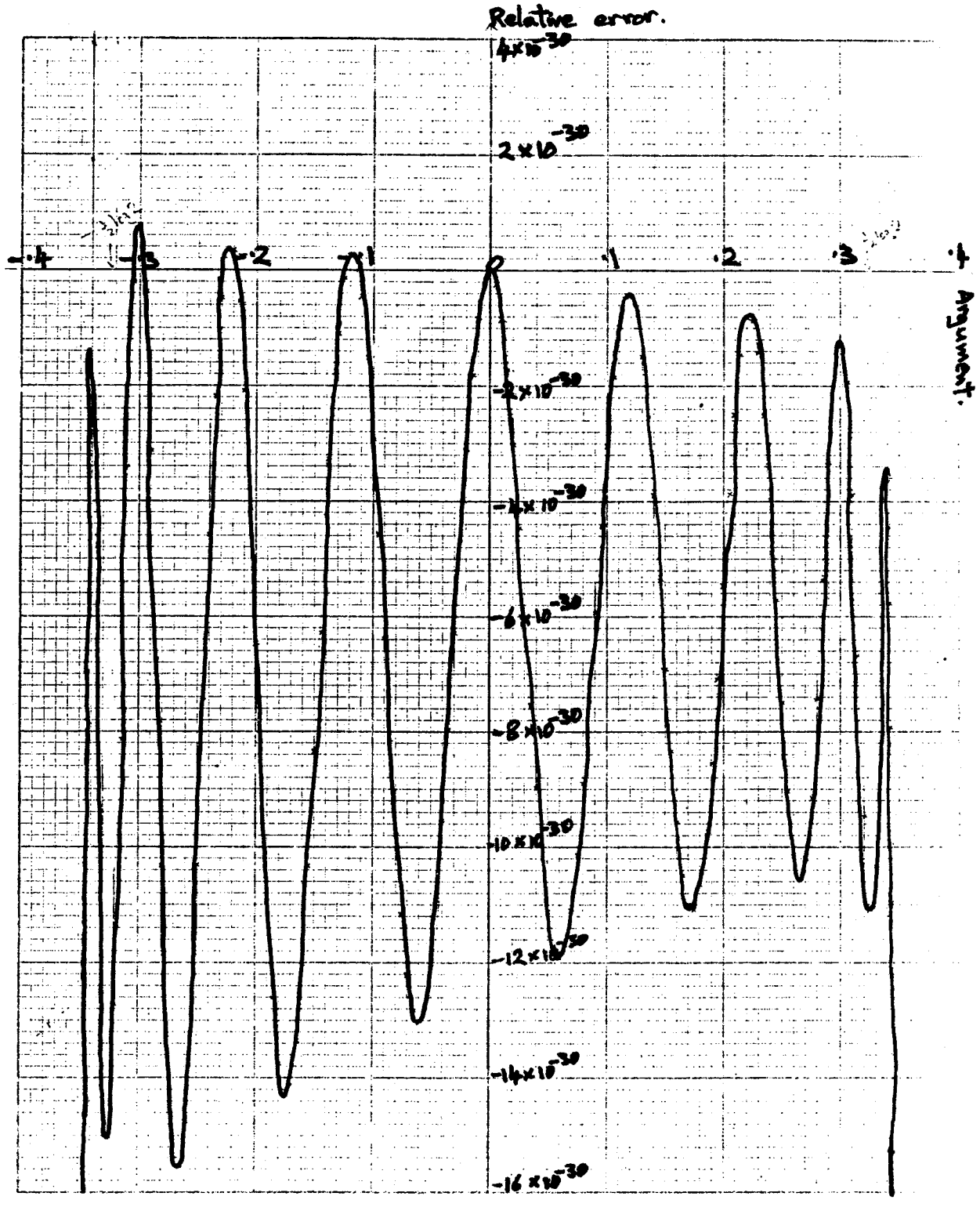


FIGURE 16
Graph of the error of approximation of the algorithm used in DLOG.

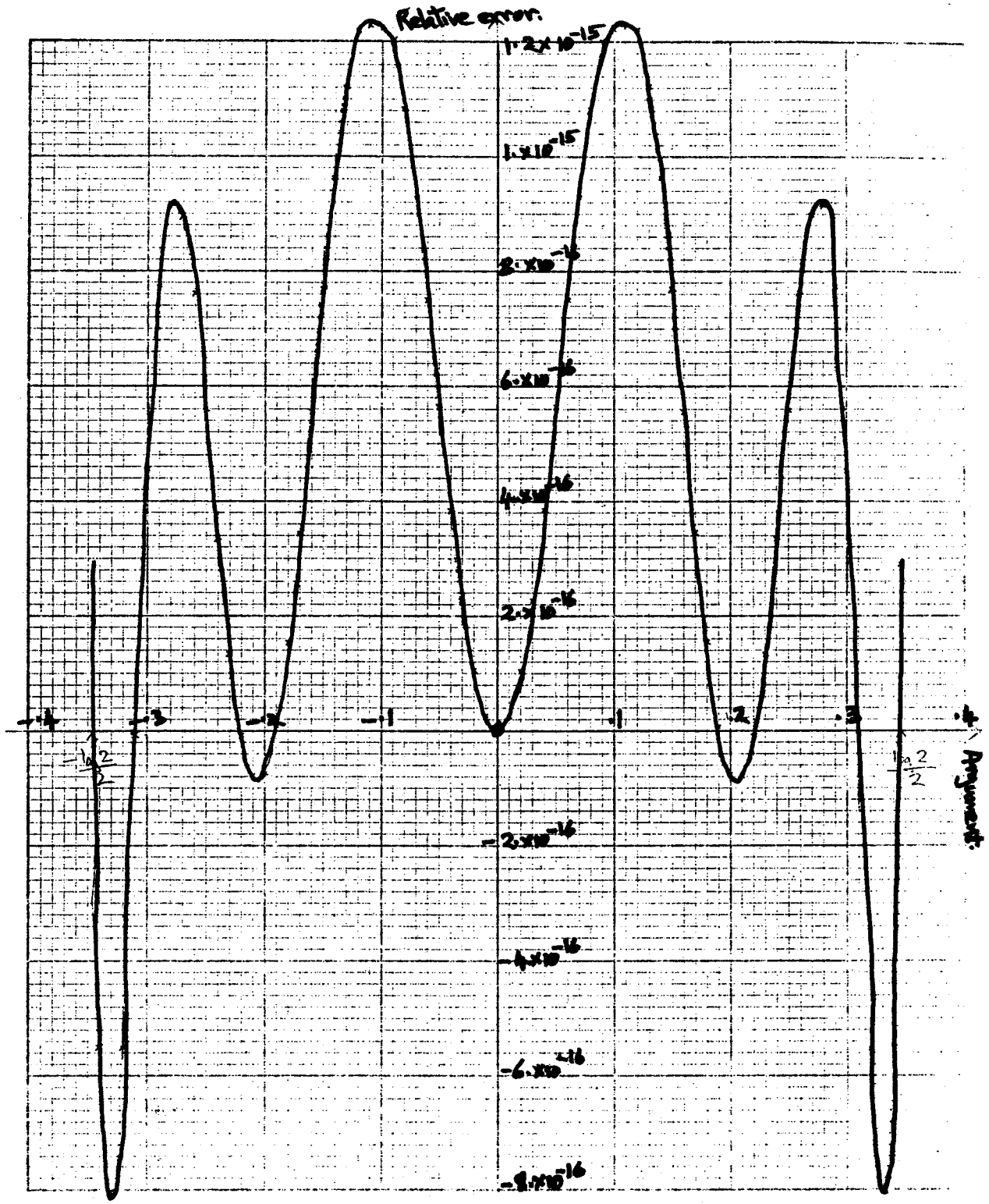


FIGURE 17
Graph of relative error in approximation to sinh(x)
over [-(log 2)/2, (log 2)/2]

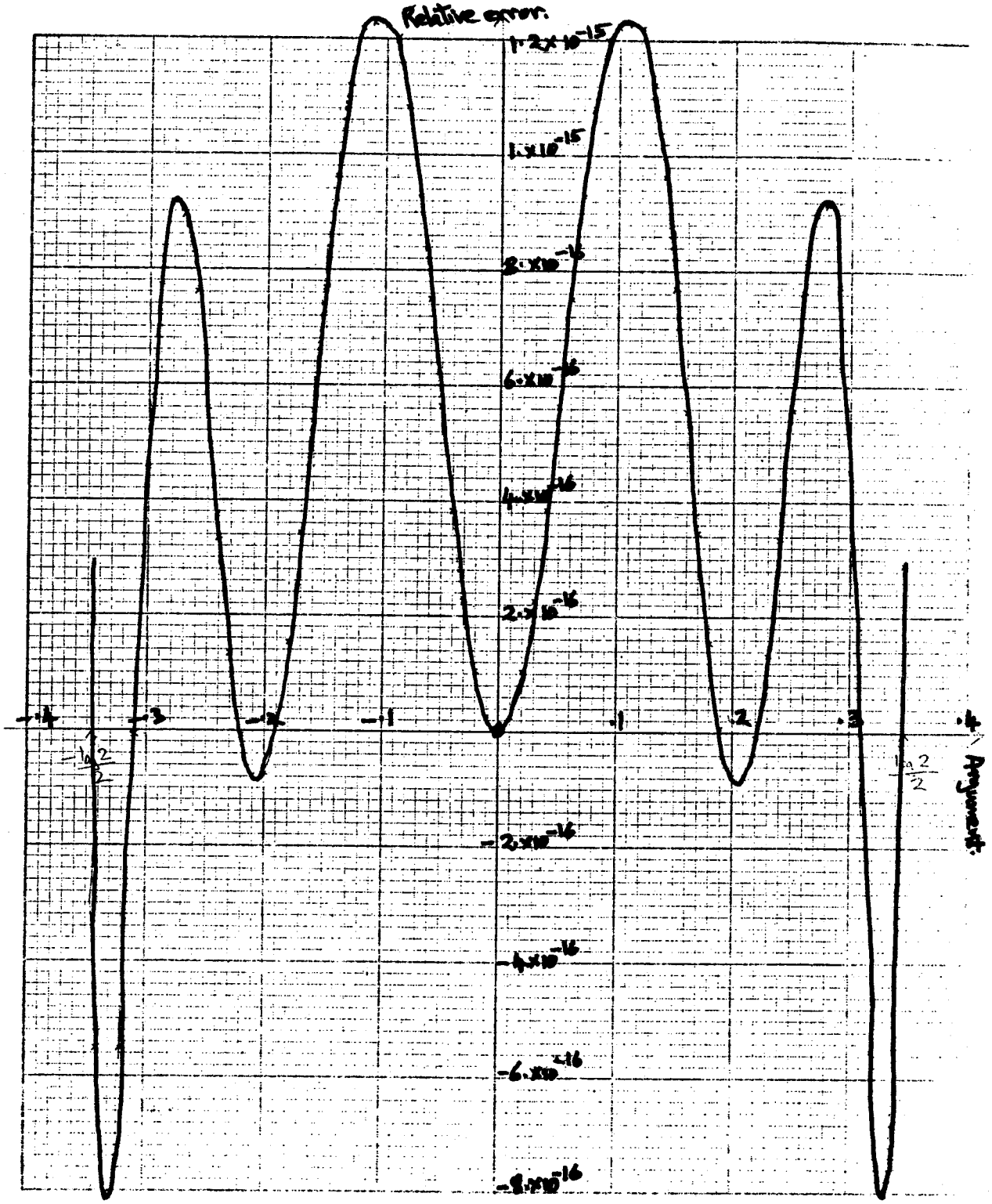
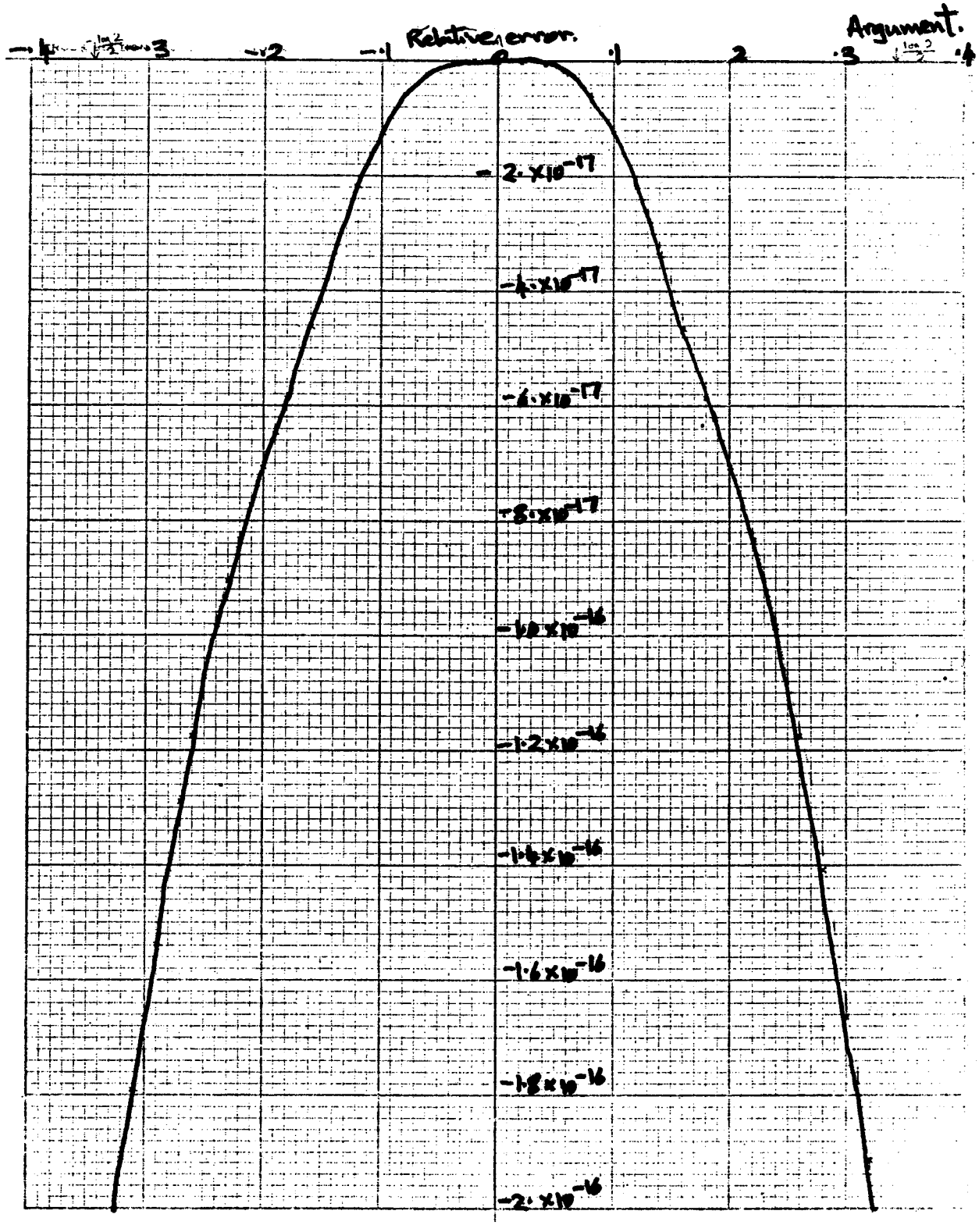


FIGURE 18
Graph of relative error in approximation to cosh(x)
over [-(log 2)/2,(log 2)/2]



COMMENT SHEET

FORTRAN Common Library
TITLE: Mathematical Routines Reference Manual

PUBLICATION NO. 60498200 REVISION C

This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: _____ POSITION: _____

COMPANY NAME: _____

ADDRESS: _____

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

CUT ON THIS LINE

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Publications and Graphics Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION