TOPS-10/TOPS-20 Common Math Library Reference Manual

Order No. AA-M400A-TK

September 1983

Abstract

This manual describes the mathematical routines that constitute the *TOPS-10/TOPS-20 Math Library.*

OPERATING SYSTEM:	TOPS-20 Version 5.0 and 5.1 TOPS-10 Version 7.01A
SOFTWARE:	FORTRAN-10/20 Version 7 Pascal-10/20 Version 1

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid-Atlantic Region

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061
Telephone:(603)884-6660

Central Region

Digital Equipment Corporation
Accessories and Supplies Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone:(312)640-5612

Western Region

Digital Equipment Corporation
Accessories and Supplies Center
632 Caribbean Drive
Sunnyvale, California 94086
Telephone:(408)734-4915

digital equipment corporation • marlboro massachusetts

First Printing, September 1983

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

Contents

Chapter 4 Exponential and Exponentiation Routines

Chapter 5 Trigonometric Routines

Chapter 6 Inverse Trigonometric Routines

Chapter 7 Hyperbolic Routines

Chapter 8 Random Number Generating Routines

Chapter 9 Absolute Value Routines

Chapter 10 Data Type Conversion Routines

Chapter 11 Rounding and Truncation Routines

Chapter 12 Product, Remainder, and Positive Difference Routines

Chapter 13 Transfer of Sign Routines

Chapter 14 Maximum/Minimum Routines

Chapter 15 Miscellaneous Complex Routines

Appendix A ELEFUNT Test Results

Appendix B Using the Common Math Library with MACRO Programs

Tables

Preface

This manual describes the TOPS-10/TOPS-20 Common Math Library. At present, the library is included as part of each object-time system of each language that uses it. In the future, the library will be a separate entity as described in this manual. Chapter 1 introduces the library routines and gives information on how they are described. A table of the routines, arranged in alphabetical order, is included for easy reference. Chapters 2 through 15 contain the descriptions of the routines, grouped logically such that all like routines are together (e.g., all the square root routines are in Chapter 2). Appendix A gives the results of the ELEFUNT tests and Appendix B describes error handling for MACRO programs.

Chapter 1
Introduction

1.1 The Math Library

The TOPS-10/TOPS-20 Common Math Library contains a set of routines that perform the following mathematical functions for several types of data.

- square root
- natural and base-10 logarithm
- exponential and exponentiation
- trigonometric
- inverse trigonometric
- hyperbolic
- random number generation
- absolute value
- data type conversion
- rounding and truncation
- product
- remainder
- positive difference
- transfer of sign
- maximum or minimum of a series
- complex conjugate
- complex multiplication or division

Most of the routines are functions; but some, notably the complex double-precision, are subroutines. The difference between the types of routines is the way in which they are called from a program. Consult the applicable language manual for more information.

The routines are listed alphabetically in Table 1–1 with a short description of each and a page reference.

Table 1-1: Math Library Routines

Routine Name	Page	Purpose
ABS	9–4	absolute value
ACOS	6–4	arc cosine
AIMAG	15–4	imaginary part of complex number
AINT	11–9	truncation to integer
ALOG	3–3	natural logarithm
ALOG10	3–5	base-10 logarithm
AMAX0	14–5	largest of a series
AMAX1	14–6	largest of a series
AMIN0	14–11	smallest of a series
AMIN1	14–12	smallest of a series
AMOD	12–6	remainder
ANINT	11–6	nearest whole number
ASIN	6–3	arc sine
ATAN	6–13	arc tangent
ATAN2	6–15	polar angle of a point in the x-y plane
CABS	9–7	complex absolute value
CCOS	5–21	complex cosine
CDABS	9–8	complex, double-precision, D-floating-point absolute value
CDCOS	5–25	complex, double-precision, D-floating-point cosine
CDEXP	4–11	complex, double-precision, D-floating-point exponential
CDLOG	3–17	complex, double-precision, D-floating-point natural logarithm
CDSIN	5–23	complex, double-precision, D-floating-point sine
CDSQRT	2–11	complex, double-precision, D-floating-point square root
CEXP	4–9	complex exponential
CEXP2.	4–22	exponentiation of a complex number to the power of an integer
CEXP3.	4–34	exponentiation of a complex number to the power of another complex number
CFDV	15–7	complex division
CFM	15–6	complex multiplication
CGABS	9–9	complex, double-precision, G-floating-point absolute value
CGCOS	5–29	complex, double-precision, G-floating-point cosine

(continued on next page)

Table Table 1–1 (Cont.): Math Library Routines

Routine Name	Page	Purpose
CGEXP	4–13	complex, double-precision, G-floating-point exponential
CGLOG	3–19	complex, double-precision, G-floating-point natural logarithm
CGSIN	5–27	complex, double-precision, G-floating-point sin
CGSQRT	2–13	complex, double-precision, G-floating-point square root
CLOG	3–15	complex natural logarithm
CMPL.C	10–23	conversion of two complex numbers to one complex number
CMPL.D	10–21	conversion of two double-precision, D-floating-point numbers to complex format
CMPL.G	10–22	conversion of two double-precision, G-floating-point numbers to complex format
CMPL.I	10–19	conversion of two integers to complex format
CMPLX	10–20	conversion of two single-precision numbers to complex format
CONJ	15–5	complex conjugate
COS	5–7	cosine (angle in radians)
COSD	5–9	cosine (angle in degrees)
COSH	7–4	hyperbolic cosine
COTAN	5–33	cotangent
CSIN	5–19	complex sine
CSQRT	2–9	complex square root
DABS	9–5	double-precision, D-floating-point absolute value
DACOS	6–7	double-precision, D-floating-point arc cosine
DASIN	6–5	double-precision, D-floating-point arc sine
DATAN	6–17	double-precision, D-floating-point arc tangent
DATAN2	6–19	double-precision, D-floating-point polar angle of a point in the x-y plane
DBLE	10–12	conversion from single-precision to double-precision, D-floating-point format
DCOS	5–13	double-precision, D-floating-point cosine
DCOSH	7–7	double-precision, D-floating-point hyperbolic cosine
DCOTAN	5–37	double-precision, D-floating-point cotangent
DDIM	12–11	double-precision, D-floating-point positive difference
DEXP	4–5	double-precision, D-floating-point exponential

(continued on next page)

Table 1-1 (cont.): Math Library Routines

Routine Name	Page	Purpose
DEXP	4-5	double-precision, D-floating-point exponential
DEXP2.	4-18	exponentiation of a double-precision, D-floating-point number to the power of an integer
DEXP3.	4-28	exponentiation of a double-precision, D-floating-point number to the power of another double-precision, D-floating-point number
DFLOAT	10-11	conversion of an integer to double-precision, D-floating-point format
DIM	12-10	positive difference
DINT	11-10	double-precision, D-floating,point truncation
DLOG	3-7	double-precision, D-floating-point natural logarithm
DLOG10	3-9	double-precision, D-floating-point base-10 logarithm
DMAX1	14-7	double-precision, D-floating-point largest in a series
DMIN1	14-13	double-precision, D-floating-point smallest in a series
DMOD	12-7	double-precision, D-floating-point remainder
DNINT	11-7	double-precision, D-floating-point nearest whole number
DPROD	12-3	double-precision, D-floating-point product
DSIGN	13-5	double-precision, D-floating-point transfer of sign
DSIN	5-11	double-precision, D-floating-point sine
DSINH	7-5	double-precision, D-floating-point hyperbolic sine
DSQRT	2-5	double-precision, D-floating-point square root
DTAN	5-35	double-precision, D-floating-point tangent
DTANH	7-12	double-precision, D-floating-point hyperbolic tangent
DTOG	10-17	conversion of a double-precision, D-floating-point number to double-precision, G-floating-point format
DTOGA	10-18	conversion of an array of double-precision, D-floating-point numbers to double-precision, G-floating-point format
EXP	4-3	exponential
EXP1.	4-15	exponentiation of an integer to the power of another integer
EXP2.	4-16	exponentiation of a single-precision number to the power of an integer
EXP3.	4-25	exponentiation of a single-precision number to the power of another single-precision number
FLOAT	10-8	conversion of an integer to single-precision format
GABS	9-6	double-precision, G-floating-point absolute value
GACOS	6-11	double-precision, G-floating-point arc cosine
GASIN	6-9	double-precision, G-floating-point arc sine
GATAN	6-21	double-precision, G-floating-point arc tangent

(continued on next page)

Table 1-1 (cont.): Math Library Routines

Routine Name	Page	Purpose
GATAN2	6–23	double-precision, G-floating-point polar angle of a point in the x-y plane
GCOS	5–17	double-precision, G-floating-point cosine
GCOSH	7–10	double-precision, G-floating-point hyperbolic cosine
GCOTAN	5–41	double-precision, G-floating-point cotangent
GDB.n	10–16	conversion of a single-precision number to double-precision, G-floating-point format
GDIM	12–12	double-precision, G-floating-point positive difference
GEXP	4–7	double-precision, G-floating-point exponential
GEXP2.	4–20	exponentiation of a double-precision, G-floating-point number to the power of an integer
GEXP3.	4–31	exponentiation of a double-precision, G-floating-point number to the power of another double-precision, G-floating-point number
GFL.n	10–15	conversion of an integer to double-precision, G-floating-point format
GFX.n	10–6	conversion of a double-precision, G-floating-point number to integer format
GINT.	11–11	double-precision, G-floating-point truncation
GLOG	3–11	double-precision, G-floating-point natural logarithm
GLOG10	3–13	double-precision, G-floating-point base-10 logarithm
GMAX1	14–8	double-precision, G-floating-point largest of a series
GMIN1	14–14	double-precision, G-floating-point smallest of a series
GMOD	12–8	double-precision, G-floating-point remainder
GNINT.	11–8	double-precision, G-floating-point nearest whole number
GPROD.	12–4	double-precision, G-floating-point product
GSIGN	13–6	double-precision, G-floating-point transfer of sign
GSIN	5–15	double-precision, G-floating-point sine
GSINH	7–8	double-precision, G-floating-point hyperbolic sine
GSN.n	10–10	conversion of a double-precision, G-floating-point number to single-precision format
GSQRT	2–7	double-precision, G-floating-point square root
GTAN	5–39	double-precision, G-floating-point tangent
GTANH	7–13	double-precision, G-floating-point hyperbolic tangent
GTOD	10–13	conversion of a double-precision, G-floating-point number to double-precision, D-floating-point format
GTODA	10–14	conversion of an array of double-precision, G-floating-point numbers to double-precision, D-floating-point format

(continued on next page)

Table 1-1 (cont.): Math Library Routines

Routine Name	Page	Purpose
IABS	9-3	integer absolute value
IDIM	12-9	integer positive difference
IDINT	10-5	conversion of a double-precision, D-floating-point number to integer format
IDNINT	11-4	integer nearest whole number for a double-precision, D-floating-point number
IFIX	10-3	conversion of a single-precision number to integer format
IGNIN.	11-5	integer nearest whole number for a double-precision, G-floating-point number
INT	10-4	conversion of a single-precision number to integer format
ISIGN	13-3	integer transfer of sign
MAX0	14-3	largest of a series
MAX1	14-4	largest of a series
MIN0	14-9	smallest of a series
MIN1	14-10	smallest of a series
MOD	12-5	integer remainder
NINT	11-3	integer nearest whole number for a single-precision number
RAN	8-3	random number generator
RANS	8-5	random number generator with shuffling
REAL	10-7	conversion of an integer to single-precision format
REAL.C	15-3	real part of a complex number
SAVRAN	8-7	save the seed for the last random number generated
SETRAN	8-6	set the seed value for the random number generator
SIGN	13-4	transfer of sign
SIN	5-3	sine (angle in radians)
SIND	5-5	sine (angle in degrees)
SINH	7-3	hyperbolic sine
SNGL	10-9	conversion of a double-precision, D-floating-point number to single-precision format
SQRT	2-3	square root
TAN	5-31	tangent
TANH	7-11	hyperbolic tangent

The routines in this library are available to most of the languages available with TOPS-10 and TOPS-20. Consult the applicable language manual for specific information on how to use the Math Library. Although all of the routines listed in Table 1-1 exist in the library, not all of them can be called from all languages. That is, some languages or compilers have restrictions that disallow calling of a particular routine from a user program. For example, the complex data type does not exist in PASCAL, so the routines that perform complex mathematics are never called by a PASCAL program. However, a compiler may itself call a routine because a user program has a statement that necessitates use of a Math Library routine. For example, a FORTRAN program cannot call any of the routines whose names contain a period (.). However, the compiler recognizes when a statement within a program requires use of one of those routines, and the compiler calls the appropriate routine. Similarly, a statement in an APL program may require a mathematical function, so the APL interpreter translates that statement into a call to the appropriate Math Library routine.

1.2 Math Symbols and Names Used in Equations

Throughout this manual, certain mathematical symbols and names are used to indicate values, quantities, actions, or states. These symbols and their meanings are listed below.

$=$	equal to
$+$	plus
$-$	minus
$\bullet$	multiplied by (used in equations)
x	multiplied by (used in numbers)
/	divided by
$>$	greater than
$\geq$	greater than or equal to
$<$	less than
$\leq$	less than or equal to
$\neq$	not equal to
$\sqrt{\ }$	square root
π	Pi (3.14159265358979323846264950338327)
$\pm$	plus or minus
[]	greatest integer in
||	absolute value
$\cong$	equals approximately
x_y	subscript
x^y	superscript or raised to the power
$\log_e$	natural logarithm
$\log_{10}$	base-10 logarithm
i	imaginary number ($\sqrt{-1}$)
e^x	exponential
sin	sine of an angle
cos	cosine of an angle
tan	tangent of an angle
cot	cotangent of an angle
$\sin^{-1}$	arc sine
$\cos^{-1}$	arc cosine
$\tan^{-1}$	arc tangent
sinh	hyperbolic sine
cosh	hyperbolic cosine
tanh	hyperbolic tangent
sgn	sign of
conj	complex conjugate

In addition, some equations use the names of routines to indicate a state or action. These routines and their meanings are as follows.

FLOAT	convert and round from an integer to a single-precison, floating-point number
INT	convert and truncate from a single-precision, floating-point number to an integer
MAX	largest of a series
MIN	smallest of a series
MOD	remainder

Each of these routines is described in detail in this manual.

Also, machine infinity (or infinity) is a term used to indicate the largest or smallest number representable in the machine.

+machine infinity = 377777777777_8 for single-precision
377777777777, 377777777777_8 for double-precision
−machine infinity = 400000000000_8 for single precision
400000000000, 000000000001_8 for double-precision

1.3 Data Types and Their Precision

The Common Math Library routines can handle several data types — integer; single-precision, floating-point (also called real); double-precision, D-floating-point; double-precision, G-floating-point; complex; complex, double-precision, D-floating-point; and complex, double-precision, G-floating-point. Each data type is described in detail in one of the following sections.

1.3.1 Integer

An integer value is a string of one to eleven digits that represents a whole decimal number (a number without a fractional part). Integer values must be within the range of -2^{35} to $+2^{35}-1$ (−34359738368 to +34359738367).

1.3.2 Single-Precision, Floating-Point

Single-precision, floating-point values may be of any size; however, each will be rounded to fit the precision of 27 bits (7 to 9 decimal digits).

Precision for single-precision, floating-point values is maintained to approximately eight significant digits; the absolute precision depends upon the numbers involved.

The range of magnitude permitted a single-precision, floating-point value is from approximately 1.47×10^{-39} to $1.70 \times 10^{+38}$.

1.3.3 Double-Precision, D-Floating-Point

Double-precision, D-floating-point values are similar to single-precision, floating-point values; the differences between these two values are:

- Double-precision, D-floating-point values, depending on their magnitude, have precision of 62 bits, rather than the 27-bit precision obtained for single-precision, floating-point values.
- Each double-precision, D-floating-point value occupies two storage locations.

The range of magnitude permitted a double-precision, D-floating-point value is from approximately 1.47×10^{-39} to $1.70 \times 10^{+38}$.

1.3.4 Double-Precision G-Floating-Point[1]

Double-precision, G-floating-point values are similar to double-precision, D-floating-point values. They differ in:

- the number of bits of exponent
- the number of bits of mantissa
- the range of numbers they can represent
- the digits of precision

Table 1-2 summarizes the differences among single-precision and the two forms of double-precision.

Table 1-2: Comparison of Single-Precision, D-Floating-Point, and G-Floating-Point

	Bits of Exponent	Bits of Mantissa	Range	Digits of Precision
single-precision	8	27	1.47×10^{-39} to $1.70 \times 10^{+38}$	8.1
D-floating-point	8	62	1.47×10^{-39} to $1.70 \times 10^{+38}$	18.7
G-floating-point	11	59	2.78×10^{-309} to $8.99 \times 10^{+307}$	17.8

[1] Double-precision, G-floating-point data type is available only with TOPS-20 Version 5 (or later) on the DECSYSTEM-20 KL10 model B.

1.3.5 Complex

A complex value contains two numbers; it is assumed that the first (leftmost) value of the pair represents the real part of the number and that the second value represents the imaginary part of the number. The values that represent the real and imaginary parts of a complex value occupy two consecutive storage locations.

1.3.6 Complex, Double-Precision

You can use two types of complex, double-precision values — D-floating-point and G-floating-point. Both are assumed to be double-precision arrays with two elements. The first element is the real part, and the second element is the imaginary part.

1.4 Information About the Routines

Each routine described in this manual has the following information provided.

- A short description
- The names of other routines called by the routine
- The data type and range of the argument(s)
- The data type and range of the result
- The accuracy of the result
- The algorithm used to calculate the result
- A reference to any text used for information about the algorithm (where applicable)
- Any error conditions and the messages that result

Some additional information about the routines not included in each write-up is:

- Calling sequence
- Entry points
- Return location(s)
- Register usage

This information is described below. It is not included for each routine because it is identical for most routines and is relevant only for MACRO and BLISS users.

1.4.1 Calling Sequence

Most routines are called by an identical calling sequence. This calling sequence is:

```
XMOVEI   L,ARG
PUSHJ    P, routine-name
```

ARG is the address of the argument block. L is the pointer to the argument list for the routine; it is AC16. P is the stack pointer; it is AC17. Note that the contents of L (AC16) are not preserved.

For example, the SQRT routine is called by:

```
XMOVEI   16,ARG
PUSHJ    17,SQRT
```

Those routines called by a different calling sequence contain the calling sequence in their descriptions.

1.4.2 Entry Points

In most cases each routine has at least two entry points — its name and its name followed by a period. For example, SQRT and SQRT. are entry points for the SQRT routine. The name with the period is the one used by the FORTRAN compiler. Some routines have additional entry points because they perform more than one function. Thus, one routine calculates both sine and cosine, so SIN, SIN., COS, and COS. are all entry points into that routine. If you are calling a routine from a MACRO or BLISS program, you can use the name of the routine as the entry point; it will always work.

1.4.3 Return Location

The result of the calculation of most routines is returned to one or two registers. For integer and single-precision results, the return location is register 0. For double-precision and complex (single-precision) results, the return locations are registers 0 and 1. For complex, double-precision results, the return location must be specified as the second argument included in the call to the routine. The requirements for the arguments included in the call are included with each write-up of the complex, double-precision routines.

1.4.4 Register Usage

All the routines have similar register usage. Some may use more registers than others, however. As stated above, registers 0 and 1 are used for the return locations; therefore the original contents of one or both are lost on return from a routine. These registers are also occasionally used to store the argument initially. Registers 2 through 15 are saved, used, and restored. The number of such registers used depends on the routine.

1.5 Accuracy Tests

Each routine contains a section headed "Accuracy of Result." The accuracy figures were obtained from the tests described below. These tests were run with typical values for arguments. There may be unusual arguments that could cause larger errors; for example, if you get too close to a threshold that could cause overflow or underflow, larger errors can occur. The format of the accuracy section is as follows. Note that the elements are explained with the descriptions of the tests.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	1.55×10^{-8} (25.9 bits)				
RMS:	3.76×10^{-9} (28.0 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	8%	83%	9%	0%

To test a routine, several representative intervals for each routine were chosen. Sample values were then chosen randomly from each interval, approximately 200,000 for single-precision and 20,000 for double-precision. Each routine was then called using these values. The relative error of each result was then obtained by the following equation.

$$\left| \frac{\text{actual exact result - result of routine}}{\text{actual exact result}} \right|$$

For example:

$$\left| \frac{\sin(x) - \mathrm{SIN}(x)}{\sin(x)} \right|$$

The test computed the maximum relative error (MRE) and the average relative error, called the root mean square (RMS). To interpret the MRE and RMS, consider an "exact" routine, one that always returns an exact result rounded to machine precision. Such a routine would show a maximum relative error of 2^{-27} for single-precision; 2^{-62} for double-precision, D-floating-point; and 2^{-59} for double-precision, G-floating-point. To make the MRE and RMS more understandable in terms of bits of accuracy, the tests also give the number of bits of accuracy by finding the negative base-2 logarithm of the MRE and RMS. For the "exact" routine, the negative base-2 logarithm of the MRE would be 27 for single-precision; 62 for double-precision, D-floating-point; and 59 for double-precision, G-floating-point. The negative base-2 logarithm of the RMS error from an "exact" routine would be about 28.3, 63.3, and 60.3, respectively. These numbers are slightly larger than those for the

MRE because they reflect the RMS average of the "worst case" of exactness (only 27 or 62 or 59 bits correct) and the "best case" (infinite bits correct). Therefore, the closer the number of bits of accuracy of a routine approaches that of an "exact" routine, the more accurate the routine. The accuracy figures for "exact" routines for the three levels of precision are as follows.

Single-Precision

test interval:	0.00000 through 8192.0				
MRE:	7.44×10^{-9} (27.0 bits)				
RMS:	3.11×10^{-9} (28.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	100%	0%	0%

Double-precision, D-floating-point

test interval:	−infinity to +infinity				
MRE:	2.17×10^{-19} (62.0 bits)				
RMS:	8.81×10^{-20} (63.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	100%	0%	0%

Double-precision, G-floating-point

test interval:	−infinity to +infinity				
MRE:	1.73×10^{-18} (59.0 bits)				
RMS:	7.05×10^{-19} (60.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	100%	0%	0%

A second test compared the result of the routines with the exact result rounded to single- or double-precision. It counted the number of times the routine's result agreed exactly with the rounded exact result, the number of times they differed by ± 1 bit, ± 2 bits, and so on. The result of these comparisons is expressed as a percent of error distribution for the least significant bit (LSB).

Appendix A shows accuracy results derived from the ELEFUNT tests of W. J. Cody, Argonne National Laboratory. These tests show accuracy derived by testing carefully-chosen identities for each function. This appendix is provided for your information, not for comparison with the test results described above. Such a comparison would not be meaningful.

Chapter 2
Square Root Routines

Description

The SQRT routine calculates the single-precision, floating-point square root of its single-precision, floating-point argument. That is:

$$SQRT(x) = \sqrt{x} = x^{\frac{1}{2}}$$

Routines Called

SQRT calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value greater than or equal to 0.0.

Type of Result

The result returned is a single-precision, floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	0.00000 through 8192.0				
MRE:	8.09×10^{-9} (26.9 bits)				
RMS:	3.21×10^{-9} (28.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	98%	2%	0%

Algorithm Used

SQRT(x) is calculated as follows.

First the routine does a linear, single-precision approximation on the argument to provide an initial guess for $\sqrt{x}$. The routine then does two iterations of the Newton-Raphson method, which results in an answer that is correct to, but not always including, the last bit.

If $x < 0.0$
  $SQRT(x) = SQRT(|x|)$

If $x = 0.0$
  $SQRT(x) = 0.0$

If $x > 0.0$
  Let $x = 2^{2b} \cdot f$ where $.25 \leqslant f < 1.0$
    then $\sqrt{x} = 2^{b} \cdot \sqrt{f}$
    and $z_0 = 2^{b} \cdot (af - b)$

$$\begin{aligned} a &= .82812500 \text{ if } .25 \leqslant f < .5 \\ &= .58593750 \text{ if } .5 \leqslant f < 1.0 \\ b &= .29722518 \text{ if } .25 \leqslant f < .5 \\ &= .42060167 \text{ if } .5 \leqslant f < 1.0 \end{aligned}$$

The Newton-Raphson method, as applied to the SQRT function, yields the following iterative approximation.

$$z_{k+1} = 1/2 \cdot (z_k + x/z_k)$$

z_{k+1} = the next iteration

z_k = the current iteration

x = the number whose square root is being calculated

z_0 = the initial approximation calculated by the linear approximation

Error Conditions

If the argument is negative, the following message is issued and the absolute value of the argument is used.

SQRT: Negative arg; result = SQRT(ABS(arg))

Description

The DSQRT routine calculates the double-precision, D-floating-point square root of its double-precision, D-floating-point argument. That is:

$$DSQRT(x) = \sqrt{x} = x^{\frac{1}{2}}$$

Routines Called

DSQRT calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, D-floating-point value greater than or equal to 0.0.

Type of Result

The result returned is a double-precision, D-floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	0.00000 through 8192.0				
MRE:	3.25×10^{-19} (61.4 bits)				
RMS:	1.23×10^{-19} (62.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	75%	25%	0%

Algorithm Used

DSQRT(x) is calculated as follows.

First the routine does a linear, single-precision approximation on the high-order word. Then the routine does two single-precision iterations of the Newton-Raphson method, followed by two double-precision iterations of the Newton-Raphson method using a value derived from the linear approximation.

The linear approximation is as follows.

If $x < 0.0$
 DSQRT(x) = DSQRT(|x|)

If $x = 0.0$
 DSQRT(x) = 0.0

If $x > 0.0$
 Let $x = 2^{2b} \cdot f$ where $.25 \leqslant f < 1.0$
  then $\sqrt{x} = 2^{b} \cdot \sqrt{f}$
  and $z_0 = 2^{b} \cdot (af - b)$

$$a = .82812500 \text{ if } .25 \leqslant f < .5$$
$$= .58593750 \text{ if } .5 \leqslant f < 1.0$$
$$b = .29722518 \text{ if } .25 \leqslant f < .5$$
$$= .42060167 \text{ if } .5 \leqslant f < 1.0$$

The Newton-Raphson method yields the following iterative approximation.

$z_{k+1} = 1/2 \bullet (z_k + x/z_k)$

z_{k+1} = the next iteration

z_k = the current iteration

x = the number whose square root is being calculated

z_0 = the initial approximation calculated by the linear approximation

For the single-precision approximations, x is truncated to single-precision and all calculations are done in single-precision. For the double-precision iterations, the full double-precision value of x is used, the current value of z_2 is zero-extended to double-precision, and all remaining calculations are done in double-precision.

Error Conditions

If the argument is negative, the following message is issued and the absolute value of the argument is used.

```
DSQRT: Negative arg; result = DSQRT(ABS(arg))
```

Description

The GSQRT routine calculates the double-precision, G-floating-point square root of its double-precision, G-floating-point argument. That is:

$$GSQRT(x) = \sqrt{x} = x^{\frac{1}{2}}$$

Routines Called

GSQRT calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value greater than or equal to 0.0.

Type of Result

The result returned is a double-precision, G-floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	0.00000 through 8192.0				
MRE:	2.60×10^{-18} (58.4 bits)				
RMS:	9.87×10^{-19} (59.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	75%	25%	0%

Algorithm Used

GSQRT(x) is calculated as follows.

First the routine does a linear, single-precision approximation on the high-order word. Then the routine does two single-precision iterations of the Newton-Raphson method, followed by two double-precision iterations of the Newton-Raphson method using a value derived from the linear approximation.

The linear approximation is as follows.

If $x < 0.0$
$GSQRT(x) = GSQRT(|x|)$

If $x = 0.0$
$GSQRT(x) = 0.0$

If $x > 0.0$
Let $x = 2^{2b} \cdot f$ where $.25 \leq f < 1.0$
then $\sqrt{x} = 2^{b} \cdot \sqrt{f}$
and $z_0 = 2^{b} \cdot (af - b)$

$a = .82812500$ if $.25 \leq f < .5$
$a = .58593750$ if $.5 \leq f < 1.0$
$b = .29722518$ if $.25 \leq f < .5$
$b = .42060167$ if $.5 \leq f < 1.0$

The Newton-Raphson method yields the following iterative approximation.

$$z_{k+1} = 1/2 \cdot (z_k + x/z_k)$$

z_{k+1} = the next iteration

z_k = the current iteration

x = the number whose square root is being calculated

z_0 = the initial approximation calculated by the linear approximation

For the single-precision approximations, x is truncated to single-precision and all calculations are done in single-precision. For the double-precision iterations, the full double-precision value of x is used, the current value of z_2 is zero-extended to double-precision, and all remaining calculations are done in double-precision.

Error Conditions

If the argument is negative, the following message is issued and the absolute value of the argument is used.

```
GSQRT: Negative arg; result = GSQRT(ABS(arg))
```

Description

The CSQRT routine calculates the complex, single-precision square root of its complex, single-precision argument. That is:

$$CSQRT(z) = \sqrt{z} = z^{\frac{1}{2}}$$

Routines Called

CSQRT calls the SQRT and MTHERR routines.

Type of Argument

The argument must be a complex, single-precision, floating-point value; it can be any such value.

Type of Result

The result returned is a complex, single-precision, floating-point value, the real part of which is greater than or equal to 0.0.

Accuracy of Result

test interval:	-1000.0 through 1000.0 real -1000.0 through 1000.0 imaginary					
MRE:	3.07×10^{-8} (25.0 bits) real 3.05×10^{-8} (25.0 bits) imaginary					
RMS:	7.05×10^{-9} (27.1 bits) real 7.33×10^{-9} (27.0 bits) imaginary					
	-2	-1	0	+1	+2	
LSB error distribution:	2%	16%	59%	20%	2%	real
	2%	19%	55%	20%	3%	imaginary

Algorithm Used

CSQRT(z) is calculated as follows.

Let $z = x+i \cdot y$
then $CSQRT(z) = u+i \cdot v$, which is defined as follows.

If $x \geqslant 0.0$

$$u = \sqrt{(|x|+|z|)/2.0}$$
$$v = y/(2.0 \cdot u)$$

If $x < 0.0$ and $y \geqslant 0.0$

$$u = y/(2.0 \cdot v)$$
$$v = \sqrt{(|x|+|z|)/2.0}$$

If x and y are both < 0.0

$$u = y/(2.0 \cdot v)$$
$$v = -\sqrt{(|x|+|z|)/2.0}$$

The result is in the right half plane; that is, the polar angle of the result lies in the closed interval $[-\pi/2,+\pi/2]$. That is, the real part of the result is greater than or equal to 0.0.

Error Conditions

If the imaginary part of the input value is too small, underflow can occur on y/(2.0•u) or y/(2.0•v). If such underflow occurs, one of the following messages is issued and the relevant part of the result is set to 0.0.

```
CSQRT: Real part underflow
CSQRT: Imaginary part underflow
```

Description

The CDSQRT subroutine calculates the complex, double-precision, D-floating-point square root of its complex, double-precision, D-floating-point argument. That is:

$$CDSQRT(z,r) = \sqrt{z} = z^{\frac{1}{2}}$$

z = location of input value
r = location of result

Routines Called

CDSQRT calls the DSQRT and MTHERR routines.

Type of Arguments

CDSQRT is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, D-floating-point value; it can be any such value.

Type of Result

The result returned is a complex, double-precision, D-floating-point value, the real part of which is greater than or equal to 0.0. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-1000.0 through 1000.0 real -1000.0 through 1000.0 imaginary
MRE:	1.10×10^{-18} (59.7 bits) real 1.04×10^{-18} (59.7 bits) imaginary
RMS:	2.69×10^{-19} (61.7 bits) real 2.75×10^{-19} (61.7 bits) imaginary

	-2	-1	0	+1	+2	
LSB error distribution:	4%	17%	43%	32%	5%	real
	5%	24%	41%	25%	5%	imaginary

Algorithm Used

CDSQRT is calculated as follows.

Let $z = x+i \cdot y$

then $CDSQRT(z) = u+i \cdot v$, which is defined as follows.

If $x \geqslant 0.0$

$u = \sqrt{(|x|+|z|)/2.0}$

$v = y/(2.0 \cdot u)$

If $x < 0.0$ and $y \geqslant 0.0$

$u = y/(2.0 \cdot v)$

$v = \sqrt{(|x|+|z|)/2.0}$

If x and y are both < 0.0

$u = y/(2.0 \cdot v)$

$v = -\sqrt{(|x|+|z|)/2.0}$

The result is in the right half plane; that is, the polar angle of the result lies in the closed interval $[-\pi/2,+\pi/2]$. That is, the real part of the result is greater than or equal to 0.0.

Error Conditions

If the imaginary part of the input value is too small, underflow can occur on $y/(2.0 \cdot u)$ or $y/(2.0 \cdot v)$. If such underflow occurs, one of the following messages is issued and the relevant part of the result is set to 0.0.

CDSQRT: Real part underflow
CDSQRT: Imaginary part underflow

Description

The CGSQRT subroutine calculates the complex, double-precision, G-floating-point square root of its complex, double-precision, G-floating-point argument. That is:

CGSQRT(z,r) = $\sqrt{z} = z^{\frac{1}{2}}$
z = location of input value
r = location of result

Routines Called

CGSQRT calls the GSQRT and MTHERR routines.

Type of Argument

CGSQRT is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, G-floating-point value; it can be any such value.

Type of Result

The result returned is a complex, double-precision, G-floating-point value; it may be any such value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval: -1000.0 through 1000.0 real
-1000.0 through 1000.0 imaginary

MRE: 8.61×10^{-18} (56.7 bits) real
8.78×10^{-18} (56.7 bits) imaginary

RMS: 2.16×10^{-18} (58.7 bits) real
2.21×10^{-18} (58.7 bits) imaginary

LSB error distribution:	-2	-1	0	+1	+2	
	5%	16%	41%	32%	5%	real
	5%	25%	40%	25%	5%	imaginary

Algorithm Used

CGSQRT(z) is calculated as follows.

Let $z = x+i \cdot y$
then CGSQRT(z) = $u+i \cdot v$ is defined as follows.

If $x \geqslant 0.0$

$u = \sqrt{(|x|+|z|)/2.0}$

$v = y/(2.0 \cdot u)$

If $x < 0.0$ and $y \geqslant 0.0$

$u = y/(2.0 \cdot v)$

$v = \sqrt{(|x|+|z|)/2.0}$

If x and y are both < 0.0

$u = y/(2.0 \cdot v)$

$v = -\sqrt{(|x|+|z|)/2.0}$

The result is in the right half plane; that is, the polar angle of the result lies in the closed interval $[-\pi/2, +\pi/2]$.

Error Conditions

If the imaginary part of the argument is too small, underflow can occur on $y/(2.0 \cdot u)$ or $y/(2.0 \cdot v)$. If this occurs, one of the following messages is issued and the relevant part of the result is set to 0.0.

```
CGSQRT: Real part underflow
CGSQRT: Imaginary part underflow
```

Chapter 3
Logarithm Routines

Description

The ALOG routine calculates the single-precision, floating-point natural logarithm of its argument. That is:

$$ALOG(x) = \log_e(x)$$

Routines Called

ALOG calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value greater than 0.0.

Type of Result

The result returned is a single-precision, floating-point value in the range -89.415 to 88.029.

Accuracy of Result

test interval: 1.46937×10^{-39} through 256.00

MRE: 1.84×10^{-8} (25.7 bits)

RMS: 5.21×10^{-9} (27.5 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	1%	81%	18%	0%

Algorithm Used

ALOG(x) is calculated as follows.

If $x = 0.0$

$$ALOG(x) = -\text{machine infinity}$$

If $x < 0.0$

$$ALOG(x) = ALOG(|x|)$$

If x is close to 1.0

$$ALOG(x) = L3 \cdot z^7 + L4 \cdot z^5 + L5 \cdot z^3 + L6 \cdot z$$

$$z = (x-1)/(x+1)$$

$$L3 = .301003281$$

$$L4 = .39965794919$$

$$L5 = .666669484507$$

$$L6 = 2.0$$

If x is not close to 1.0

$$ALOG(x) = (k-.5) \cdot \log_e(2) + \log_e(f \cdot \sqrt{2})$$

$$x = 2^k \cdot f$$

$$\log_e(f \cdot \sqrt{2}) = L3 \cdot z^7 + L4 \cdot z^5 + L5 \cdot z^3 + L6 \cdot z$$

$$z = (f - \sqrt{.5})/(f + \sqrt{.5})$$

Reference

Hart et. al., *Computer Approximations*, (New York, N.Y.: John Wiley and Sons, 1968).
The algorithm used is #2662, the coefficients are listed on page 193, and the range of validity is on page 111.

Error Conditions

1. If the argument is equal to 0.0, the following message is issued and the result is set to -machine infinity.

   ALOG: Arg is zero; result = -infinity.

2. If the argument is less than 0.0, the following message is issued and the absolute value of the argument is used.

   ALOG: Negative arg, result = ALOG(ABS(arg))

Description

The ALOG10 routine calculates the single-precision, floating-point base-10 logarithm of its single-precision, floating-point argument. That is:

$$ALOG10(x) = \log_{10}(x)$$

Routines Called

ALOG10 calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value greater than 0.0.

Type of Result

The result returned is a single-precision, floating-point value in the range -38.832 to 38.230.

Accuracy of Result

test interval:	1.46937×10^{-39} through 256.00				
MRE:	2.52×10^{-8} (25.2 bits)				
RMS:	5.99×10^{-9} (27.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	1%	19%	64%	15%	0%

Algorithm Used

ALOG10(x) is calculated as follows.

If $x = 0.0$

$ALOG10(x) = -$machine infinity

If $x < 0.0$

$ALOG10(x) = ALOG10(|x|)$

If x is close to 1.0

$$ALOG10(x) = \log_e(x) \cdot \log_{10}(e)$$
$$\log_e(x) = L3 \cdot z^7 + L4 \cdot z^5 + L5 \cdot z^3 + L6 \cdot z$$
$$z = (x-1)/(x+1)$$
$$L3 = .301003281$$
$$L4 = .39965794919$$
$$L5 = .666669484507$$
$$L6 = 2.0$$

If x is not close to 1.0

$$ALOG10(x) = \log_e(x) \cdot \log_{10}(e)$$
$$x = 2^k \cdot f$$
$$\log_e(x) = (k-.5) \cdot \log_e(2) + \log_e(f \cdot \sqrt{2})$$
$$\log_e(f \cdot \sqrt{2}) = L3 \cdot z^7 + L4 \cdot z^5 + L5 \cdot z^3 + L6 \cdot z$$
$$z = (f-\sqrt{.5})/(f+\sqrt{.5})$$

Reference

Hart et. al, *Computer Approximations*, (New York, N.Y.: John Wiley and Sons, 1968). The algorithm used is #2662, the coefficients are listed on page 193, and the range of validity is on page 111.

Error Conditions

1. If the argument is 0.0, the following message is issued and the result is set to -machine infinity.

   ALOG10: Arg is zero; result = -infinity

2. If the argument is less than 0.0, the following message is issued and the absolute value of the argument is used.

   ALOG10: Negative arg; result = ALOG10(ABS(arg))

Description
The DLOG routine calculates the double-precision, D-floating-point natural logarithm of its double-precision, D-floating-point argument. That is:

$DLOG(x) = \log_e(x)$

Routines Called
DLOG calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value greater than 0.0.

Type of Result
The result returned is a double-precision, D-floating-point value in the range -89.415 to 88.029.

Accuracy of Result
test interval: 1.46937×10^{-39} through 256.00

MRE: 9.78×10^{-19} (59.8 bits)

RMS: 3.03×10^{-19} (61.5 bits)

LSB error distribution:	-2	-1	0	+1	+2
	1%	12%	51%	23%	13%

Algorithm Used
DLOG(x) is calculated as follows.

If $x = 0.0$
  DLOG(x) = -machine infinity

If $x < 0.0$
  DLOG(x) = DLOG(|x|)

If $x > 0.0$
  $x = 2^k \cdot f$ where $.5 < f < 1.0$
  and g and n are defined so that
    $f = 2^{-n} \cdot g$ where $1/\sqrt{2} \leq g < \sqrt{2}$
  Then $DLOG(x) = (k-n) \cdot \log_e(2) + \log_e(g)$
    $\log_e(g)$ is evaluated by defining
      $s = (g-1)/(g+1)$ and
      $z = 2 \cdot s$
    and then calculating
      $\log_e(g) = \log_e((1+z/2)/(1-z/2))$ using a minimax rational approximation.

Error Conditions

1. If the argument is equal to 0.0, the following message is issued and the result is set to -machine infinity.

   DLOG: Arg is zero; result = -infinity

2. If the argument is less than 0.0, the following message is issued and the absolute value of the argument is used.

   DLOG: Negative arg; result = DLOG(ABS(arg))

Description

The DLOG10 routine calculates the double-precision, D-floating-point base-10 logarithm of its double-precision D-floating-point argument. That is:

$DLOG10(x) = \log_{10}(x)$

Routines Called

DLOG10 calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, D-floating-point value greater than 0.0.

Type of Result

The result returned is a double-precision, D-floating-point value in the range –38.832 to 38.320.

Accuracy of Result

test interval:	1.46937×10^{-39} through 256.00
MRE:	1.20×10^{-18} (59.5 bits)
RMS:	3.65×10^{-19} (61.2 bits)

LSB error distribution:	–2	–1	0	+1	+2	+3
	3%	17%	38%	26%	14%	2%

Algorithm Used

DLOG10(x) is calculated as follows.

If $x = 0.0$
    DLOG10(x) = –machine infinity

If $x < 0.0$
    DLOG10(x) = DLOG10(|x|)

If $x > 0.0$
    $x = 2^k \cdot f$ where $.5 < f < 1.0$
    and g and n are defined so that
        $f = 2^{-n} \cdot g$ where $1/\sqrt{2} \leq g < \sqrt{2}$
    Then $DLOG10(x) = \log_{10}(e) \cdot \log_e(x) = \log_e(x)/\log_e(10)$
        $\log_e(g)$ is evaluated by defining
            $s = (g - 1)/(g+1)$ and
            $z = 2 \cdot s$
        and then calculating
            $\log_e(g) = \log_e((1+z/2)/(1 - z/2))$ using a minimax rational approximation.

Error Conditions

1. If the argument is equal to 0.0, the following message is issued and the result is set to -machine infinity.

   DLOG10: Arg is zero; result = -infinity

2. If the argument is less than 0.0, the following message is issued and the absolute value of the argument is used.

   DLOG10: Negative arg; result = DLOG10(ABS(arg))

Description

The GLOG routine calculates the double-precision, G-floating-point natural logarithm of its double-precision, G-floating-point argument. That is:

$GLOG(x) = \log_e(x)$

Routines Called

GLOG calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value greater than 0.0.

Type of Result

The result returned is a double-precision, G-floating-point value in the range -710.475 to 709.089.

Accuracy of Result

test interval:	0.00000 through 256.00				
MRE:	5.13×10^{-18} (57.4 bits)				
RMS:	1.26×10^{-18} (59.5 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	10%	74%	16%	0%

Algorithm Used

GLOG(x) is calculated as follows.

If $x = 0.0$
  GLOG(x) = machine infinity

If $x < 0.0$
  GLOG(x) = GLOG(|x|)

If $x > 0.0$
  $x = 2^k \cdot f$ where $.5 < f < 1.0$
  and g and n are defined so that
    $f = 2^{-n} \cdot g$ where $1/\sqrt{2} \leq g < \sqrt{2}$

  Then $GLOG(x) = (k-n) \cdot \log_e(2) + \log_e(g)$
    $\log_e(g)$ is evaluated by defining
      $s = (g-1)/(g+1)$ and
      $z = 2 \cdot s$
    and then calculating
      $\log_e(g) = \log_e((1+z/2)/(1-z/2))$
      using a minimax rational approximation.

Error Conditions

1. If the argument is equal to 0.0, the following message is issued and the result is set to -machine infinity.

   GLOG: Arg is zero; result = -infinity

2. If the argument is negative, the following message is issued and the absolute value of the argument is used.

   GLOG: Negative arg; result = GLOG(ABS(arg))

Description

The GLOG10 routine calculates the double-precision, G-floating-point base-10 logarithm of its double-precision, G-floating-point argument. That is:

GLOG10(x) = $\log_{10}(x)$

Routines Called

GLOG10 calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value greater than 0.0.

Type of Result

The result returned is a double-precision, G-floating-point value in the range -308.555 to 307.953.

Accuracy of Result

test interval:	2.78134×10^{-309} through 256.00				
MRE:	6.05×10^{-18} (57.2 bits)				
RMS:	1.42×10^{-18} (59.3 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	1%	18%	62%	18%	0%

Algorithm Used

GLOG10(x) is calculated as follows.

If $x = 0.0$
GLOG10(x) = -machine infinity

If $x < 0.0$
GLOG10(x) = GLOG10(|x|)

If $x > 0.0$
$x = 2^k \cdot f$ where $.5 < f < 1.0$
and g and n are defined so that
$f = 2^{-n} \cdot g$ where $1/\sqrt{2} \leq g < \sqrt{2}$

Then GLOG10(x) = $\log_{10}(e) \cdot \log_e(x) = \log_e(x)/\log_e(10)$
$\log_e(g)$ is evaluated by defining
$s = (g-1)/g+1)$ and
$z = 2 \cdot s$
and then calculating
$\log_e(g) = \log_e((1+z/2)/(1-z/2))$
using a minimax rational approximation.

Error Conditions

1. If the argument is equal to 0.0, the following message is issued and the result is set to -machine infinity.

   GLOG10: Arg is zero; result = -infinity

2. If the argument is negative, the following message is issued and the absolute value of the argument is used.

   GLOG10: Negative arg; result = GLOG10(ABS(arg))

Description

The CLOG routine calculates the complex, single-precision, floating-point natural logarithm of its complex, single-precision, floating-point argument. That is:

$CLOG(z) = log_e(z)$

Routines Called

CLOG calls the ALOG, ATAN, ATAN2, and MTHERR routines.

Type of Argument

The argument must be a complex, single-precision, floating-point value, both parts of which cannot be equal to 0.0, although either can be equal to 0.0.

Type of Result

The result returned is a complex, single-precision, floating-point value. The real part of the result is in the range -89.415 to 88.029; the imaginary part is in the range $-\pi$ to π.

Accuracy of Result

test interval:	-1000.0 through 1000.0 real -100.00 through 100.00 imaginary
MRE:	5.30×10^{-5} (14.2 bits) real 1.49×10^{-8} (26.0 bits) imaginary
RMS:	1.06×10^{-7} (23.2 bits) real 3.44×10^{-9} (28.1 bits) imaginary

LSB error distribution:

	-4^{+}	-3	-2	-1	0	+1	+2
real	1%	1%	1%	6%	82%	7%	1%
imaginary	0%	0%	0%	3%	94%	3%	0%

Algorithm Used

CLOG(z) is calculated as follows.

Let $z = x+i \cdot y$

If $x = 0.0$ and $y = 0.0$
  CLOG(z) = (+infinity, 0.0)

If $x = 0.0$ and $y \neq 0.0$
  $CLOG(z) = log_e(|y|)+i \cdot sgn(y) \cdot \pi/2$

If $x \neq 0.0$ and $y = 0.0$

If $x > 0.0$
$CLOG(z) = \log_e(x)+i \cdot 0.0$

If $x < 0.0$
$CLOG(z) = \log_e(|x|)+i \cdot \pi$

If $x \neq 0.0$ and $y \neq 0.0$
$CLOG(z) = u+i \cdot v$
$u = .5 \cdot \log_e(x^2+y^2)$
$v = \tan^{-1}(y/x)$
Scaled values are calculated on occurences of overflow/underflow for (x^2,y^2) or (x^2+y^2) and propagated to give a valid in-range result for u.

Error Conditions

1. If both parts of the argument equal 0.0, the following message is issued and the result is set to (+infinity, 0.0).

   CLOG; Arg is zero; result = (+infinity, zero)

2. If either part of the result underflows, one or both of the following messages are issued and the relevant part of the result is set to 0.0.

   CLOG: Real part underflow
   CLOG: Imaginary part underflow

Description

The CDLOG subroutine calculates the complex, double-precision, D-floating-point natural logarithm of its complex, double-precision, D-floating-point argument. That is:

CDLOG(z,r) = $\log_e(z)$
z = location of input value
r = location of result

Routines Called

CDLOG calls the DLOG, DATAN, DATAN2, and MTHERR routines.

Type of Argument

CDLOG is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, D-floating-point value, both parts of which cannot be equal to 0.0, although either can be equal to 0.0.

Type of Result

The result returned is a complex, double-precision, D-floating-point value. The real part of the result is in the range –89.415 to 88.376; the imaginary part is in the range $-\pi$ to π. The result is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	–1000.0 through 1000.0 real –100.00 through 100.00 imaginary
MRE:	$9.07\text{x}10^{-16}$ (50.0 bits) real $5.09\text{x}10^{-19}$ (60.8 bits) imaginary
RMS:	$1.59\text{x}10^{-18}$ (59.1 bits) real $1.04\text{x}10^{-19}$ (63.1 bits) imaginary

LSB error distribution:

–4⁺	–3	–2	–1	0	+1	+2	
1%	1%	1%	5%	84%	6%	1%	real
0%	0%	0%	4%	92%	4%	0%	imaginary

Algorithm Used

CDLOG is calculated as follows.

Let $z = x+i \cdot y$

If $x = 0.0$ and $y = 0.0$
    $CDLOG(z) = (+infinity, 0.0)$

If $x = 0.0$ and $y \neq 0.0$
    $CDLOG(z) = \log_e(|y|)+i \cdot sgn(y) \cdot \pi/2$

If $x \neq 0.0$ and $y = 0.0$
    If $x > 0.0$
        $CDLOG(z) = \log_e(x)+i \cdot 0.0$
    If $x < 0.0$
        $CDLOG(z) = \log_e(|x|)+i \cdot \pi$

If $x \neq 0.0$ and $y \neq 0.0$
    $CDLOG(z) = u+i \cdot v$
        $u = .5 \cdot \log_e(x^2+y^2)$
        $v = \tan^{-1}(y,x)$
        Scaled values are calculated on occurrences of overflow/underflow for (x^2, y^2) or (x^2+y^2) and progagated to give a valid in-range result for u.

Error Conditions

1. If both parts of the argument equal 0.0, the following message is issued and the result is set to (+infinity, 0.0).

    CDLOG: Arg is zero; result = (+infinity, zero)

2. If either part of the result underflows, one or both of the following messages are issued and the relevant part of the result is set to 0.0.

    CDLOG: Imaginary part underflow
    CDLOG: Real part underflow

Description

The CGLOG subroutine calculates the complex, double-precision, G-floating-point natural logarithm of its complex, double-precision, G-floating-point argument. That is:

CGLOG(z,r) = $\log_e(z)$
z = location of input value
r = location of result

Routines Called

CGLOG calls the GLOG, GATAN, GATAN2, and MTHERR routines.

Type of Argument

CGLOG is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, G-floating-point value, both parts of which cannot be equal to 0.0, although either can be equal to 0.0.

Type of Result

The result returned is a complex, double-precision, G-floating-point value. The real part of the result is in the range -710.475 to 709.436; the imaginary part is in the range $-\pi$ to π. The result is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval: -1000.0 through 1000.0 real
-100.00 through 100.00 imaginary

MRE: 7.15×10^{-15} (47.0 bits) real
3.54×10^{-18} (58.0 bits) imaginary

RMS: 1.77×10^{-17} (55.7 bits) real
8.19×10^{-19} (60.1 bits) imaginary

LSB error distribution:

	-4^+	-3	-2	-1	0	+1	+2	
	1%	0%	1%	5%	86%	6%	1%	real
	0%	0%	0%	4%	92%	4%	0%	imaginary

Algorithm Used

CGLOG(z) is calculated as follows.

Let $z = x+i \cdot y$

If $x = 0.0$ and $y = 0.0$
CGLOG(z) = +machine infinity

If $x = 0.0$ and $y \neq 0.0$
$CGLOG(g) = \log_e(|y|)+i \cdot sgn(y) \cdot \pi/2$

If $x \neq 0.0$ and $y = 0.0$

If $x > 0.0$
$CGLOG(z) = \log_e(x)+i \cdot 0.0$

If $x < 0.0$
$CGLOG(z) = \log_e(|x|)+i \cdot \pi$

If $x \neq 0.0$ and $y \neq 0.0$
$CGLOG(z) = u+i \cdot v$
$u = .5 \cdot \log_e(x^2+ y^2)$
$v = \tan^{-1}(y/x)$
Scaled values are calculated on occurrence of overflow/underflow for (x^2, y^2) or (x^2+y^2) and propagated to give a valid in-range result for u.

Error Conditions

1. If both parts of the argument equal 0.0, the following message is issued and the result is set to (+machine infinity, 0.0).

   CGLOG: Arg is zero; result = (+infinity, zero)

2. If either part of the result underflows, one or both of the following messages are issued and the relevant part of the result is set to 0.0.

   CGLOG: Real part underflow
   CGLOG: Imaginary part underflow

Chapter 4
Exponential and Exponentiation Routines

Description

The EXP routine calculates the single-precision, floating-point exponential function of its single-precision, floating-point argument. That is:

$EXP(x) = e^x$

Routines Called

EXP calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value in the range -89.4159863 to 88.0296919.

Type of Result

The result returned is a single-precision, floating-point value greater than zero.

Accuracy of Result

test interval:	-89.000 through 88.000				
MRE:	1.74×10^{-8} (25.8 bits)				
RMS:	3.98×10^{-9} (27.9 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	2%	86%	12%	0%

Algorithm Used

EXP(x) is calculated as follows.

If $x < -89.4159863$
  EXP(x) = 0.0

If $x > 88.0296919$
  EXP(x) = +machine infinity

Otherwise, the argument is reduced as follows:
  Let n = the nearest integer to $x/\log_e(2)$
  The reduced argument is:
    $g = x - n \cdot \log_e(2)$

The calculation is:
  $EXP(x) = R(g) \cdot 2^{(n+1)}$
    $R(g) = .5 + g \cdot p/(q - g \cdot p)$
      $p = p1 \cdot g^2 + .25$
      $q = q1 \cdot g^2 + .5$
        $p1 = .00416028863$
        $q1 = .0499871789$

Error Conditions

1. If the argument is less than -89.4159863, the following message is issued and the result is set to 0.0.

   EXP: Result underflow

2. If the argument is greater than 88.0296919, the following message is issued and the result is set to +machine infinity.

   EXP: Result overflow

Description
The DEXP routine calculates the double-precision, D-floating-point exponential function of its double-precision, D-floating-point argument. That is:

$DEXP(x) = e^x$

Routines Called
DEXP calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value in the range -89.415986292232944914 to 88.029691931113054295.

Type of Result
The result returned is a double-precision, D-floating-point value greater than zero.

Accuracy of Result

test interval:	-89.000 through 88.000				
MRE:	4.89×10^{-19} (60.8 bits)				
RMS:	1.17×10^{-19} (62.9 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	2%	86%	12%	0%

Algorithm Used
DEXP(x) is calculated as follows.

If $x < -89.415986292232944914$
$DEXP(x) = 0.0$

If $x > 88.029691931113054295$
$DEXP(x) = +$machine infinity

Otherwise, the argument is reduced as follows:
Let $x1 = [x]$, the greatest integer in x
$x2 = x-x1$
$n =$ the nearest integer to $x/\log_e(2)$

The reduced argument is:
$g = x1-n \cdot c1+x2+n \cdot c2$
$c1 = .543_8$
$c2 = \log_e(2)-.543_8$

The calculation is:

$$DEXP(x) = R(g) \cdot 2^{(n+1)}$$

$$R(g) = .5+g \cdot p/(q-g \cdot p)$$

$$p = (((p2 \cdot g^2+p1) \cdot g^2)+p0) \cdot g^2$$

$$q = ((((q3 \cdot g^2+q2) \cdot g^2)+q1) \cdot g^2)+q0$$

$$p0 = .250$$

$$p1 = .757531801594227767 \times 10^{-2}$$

$$p2 = .315551927656846464 \times 10^{-4}$$

$$q0 = .5$$

$$q1 = .568173026985512218 \times 10^{-1}$$

$$q2 = .631218943743985036 \times 10^{-3}$$

$$q3 = .751040283998700461 \times 10^{-6}$$

Error Conditions

1. If the argument is less than –89.415986292232944914, the following message is issued and the result is set to 0.0.

   DEXP: Result underflow

2. If the argument is greater than 88.029691931113054295, the following message is issued and the result is set to +machine infinity.

   DEXP: Result overflow

Description
The GEXP routine calculates the double-precision, G-floating-point exponential function of its double-precision, G-floating-point argument. That is:

GEXP(x) = e^x

Routines Called
GEXP calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, G-floating-point value in the range -710.475860073943942 to 709.089565712824051.

Type of Result
The result returned is a double-precision, G-floating-point value greater than or equal to zero.

Accuracy of Result

test interval:	-89.000 through 88.000				
MRE:	3.99×10^{-18} (57.8 bits)				
RMS:	9.40×10^{-19} (59.9 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	2%	85%	13%	0%

Algorithm Used
GEXP(x) is calculated as follows.

If x ≤ -710.475860073943942
GEXP(x) = 0.0

If x > 709.089565712824051
GEXP(x) = +machine infinity

Otherwise, the argument is reduced as follows:
Let x1 = [x], the greatest integer in x
x2 = x−x1
n = the nearest integer to $x/\log_e(2)$

The reduced argument is:
g = x1−n•c1+x2+n•c2
c1 = $.543_8$
c2 = $\log_e(2)-.543_8$

The calculation is:

$$GEXP(x) = R(g) \cdot 2^{(n+1)}$$

$$R(g) = .5+g \cdot p/(q-g \cdot p)$$

$$p = (((p2 \cdot g^2+p1) \cdot g^2)+p0) \cdot g^2$$

$$q = ((((q3 \cdot g^2+q2) \cdot g^2)+q1) \cdot g^2)+q0$$

$$p0 = .250$$

$$p1 = .757531801594227767 \times 10^{-2}$$

$$p2 = .315551927656846464 \times 10^{-4}$$

$$q0 = .5$$

$$q1 = .568173026985512218 \times 10^{-1}$$

$$q2 = .631218943743985036 \times 10^{-3}$$

$$q3 = .751040283998700461 \times 10^{-6}$$

Error Conditions

1. If the argument is less than or equal to −710.475860073943942, the following message is issued and the result is set to 0.0.

   GEXP: Result underflow

2. If the argument is greater than 709.089565712824051, the following message is issued and the result is set to +machine infinity.

   GEXP: Result overflow

Description

The CEXP routine calculates the complex, single-precision, floating-point exponential function of its complex, single-precision, floating-point argument. That is:

CEXP(z) = e^z

Routines Called

CEXP calls the EXP, COS, SIN, and MTHERR routines.

Type of Argument

The argument must be a complex, single-precision, floating-point value in the range -89.4159863 to 176.0593838 for the real part and less than 823549.66 for the imaginary part.

Type of Result

The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	-40.000 through 12.000 real				
	-10.000 through 157.08 imaginary				
MRE:	2.77×10^{-8} (25.1 bits) real				
	2.88×10^{-8} (25.0 bits) imaginary				
RMS:	6.51×10^{-9} (27.2 bits) real				
	6.38×10^{-9} (27.2 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	1%	19%	58%	21%	1% real
	1%	17%	59%	23%	1% imaginary

Algorithm Used

CEXP(z) is calculated as follows.

Let z = x+i•y

If |y| > 823549.66
  CEXP(z) = (0.0,0.0)

If x < -89.4159863
  CEXP(z) = (0.0,0.0)

If x > 88.0296919 and y = 0.0
  CEXP(z) = (+infinity, 0.0)

If 88.0296919 < x < 176.0593838
  and a component of the result is out of range,
  that component is set to +infinity.

If x > 176.0593838 and y ≠ 0.0
  CEXP(z) = (± infinity, ± infinity)

Otherwise
  CEXP(z) = e^x•(cos(y)+i•sin(y))

Error Conditions

The following table gives the possible error conditions and the resulting error messages.

Error Conditions for CEXP

Real Part of Argument	Imaginary Part of Argument	Result	Error Message(s)
Any Value	> 823549.66	(0.0,0.0)	#1
< -89.4159863	0.0	(0.0,0.0)	#2
	Not 0.0 and ≤ 823549.66	(0.0,0.0)	#2 and #3
Between -89.41598663 and 88.0296919	Not 0.0 and ≤ 823549.66	Underflow may occur on neither, either, or both parts	None or #2 or #3 or #2 and #3
> 88.0296919	0.0	(+infinity, 0.0)	#4
> 176.0593838	Not 0.0 and ≤ 823549.66	(± infinity, ± infinity)	#4 and #5
Between 88.0296919 and 176.0593838	Not 0.0 and ≤ 823549.66	Overflow may occur on neither, either, or both parts	None or #4 or #5 or #4 and #5

Error Messages:

1. CEXP:ABS(IMAG(arg)) too large; result = zero
2. CEXP: Real part underflow
3. CEXP: Imaginary part underflow
4. CEXP: Real part overflow
5. CEXP: Imaginary part overflow

Description

The CDEXP subroutine calculates the complex, double-precision, D-floating-point exponential function of its complex, double-precision, D-floating-point argument. That is:

CDEXP(z,r) = e^z
z = location of input value
r = location of result

Routines Called

CDEXP calls the DEXP, DSIN, DCOS, and MTHERR routines.

Type of Argument

CDEXP is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex double-precision, D-floating-point value in the range -89.415986292232944914 to 176.059383862226109 for the real part and less than 6746518850.429 for the imaginary part.

Type of Result

The result returned is a complex, double-precision, D-floating-point value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-40.000 through 12.000 real				
	-10.000 through 157.08 imaginary				
MRE:	8.78×10^{-19} (60.0 bits) real				
	9.49×10^{-19} (59.9 bits) imaginary				
RMS:	1.90×10^{-19} (62.2 bits) real				
	1.87×10^{-19} (62.2 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	1%	23%	57%	18%	1% real
	1%	20%	59%	19%	1% imaginary

Algorithm Used

CDEXP is calculated as follows.

Let z = x+i•y

If |y| > 6746518850.429
CDEXP(z) = (0.0,0.0)

If x < -89.415986292232944914
CDEXP(z) = (0.0,0.0)

If x > 88.029691931113054295 and y = 0.0
CDEXP(z) = (+infinity, 0.0)

If 88.029691931113054295 < x < 176.059383862226109
and a component of the result is out of range,
that component is set to +infinity.

If x > 176.059383862226109 and y ≠ 0.0
CDEXP(z) = (± infinity, ± infinity).

Otherwise
CDEXP(z) = e^x•(cos(y)+i•sin(y))

Error Conditions

The following table gives the possible error conditions and the resulting error messages.

Error Conditions for CDEXP

Real Part of Argument	Imaginary Part of Argument	Result	Error Message(s)
Any Value	> 6746518850.429	(0.0,0.0)	#1
< -89.415986292232944914	0.0	(0.0,0.0)	#2
	Not 0.0 and ≤ 6746518850.429	(0.0,0.0)	#2 and #3
Between -89.415986292232944914 and 88.029691931113054295	Not 0.0 and ≤ 6746518850.429	Underflow may occur on neither, either, or both parts	None or #2 or #3 or #2 and #3
> 88.029691931113054295	0.0	(+infinity, 0.0)	#4
> 176.059383862226109	Not 0.0 and ≤ 6746518850.429	(± infinity, ± infinity)	#4 and #5
Between 88.029691931113054295 and 176.059383862226109	Not 0.0 and ≤ 6746518850.429	Overflow may occur on neither, either, or both parts	None or #4 or #5 or #4 and #5

Error Messages:

1. CDEXP:ABS(IMAG(arg)) too large; result = zero
2. CDEXP: Real part underflow
3. CDEXP: Imaginary part underflow
4. CDEXP: REAL(arg) too large; REAL(result) = +infinity
5. CDEXP: REAL(arg) too large; IMAG(result) = +infinity

Description

The CGEXP subroutine calculates the complex, double-precision, G-floating-point exponential function of its complex, double-precision, G-floating-point argument. That is:

CGEXP(z,r) $= e^z$
z = location of input value
r = location of result

Routines Called

CGEXP calls the GEXP, GSIN, GCOS, and the MTHERR routines.

Type of Argument

CGEXP is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, G-floating-point value in the range –710.475860073943942 to 1418.179131425648102 for the real part and less than 1686629713.065 for the imaginary part.

Type of Result

The result returned is a complex, double-precision, G-floating-point value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	–40.000 through 12.000 real –10.000 through 157.08 imaginary
MRE:	6.50×10^{-18} (57.1 bits) real 6.67×10^{-18} (57.1 bits) imaginary
RMS:	1.53×10^{-18} (59.2 bits) real 1.44×10^{-18} (59.3 bits) imaginary

	–2	–1	0	+1	+2	
LSB error distribution:	1%	19%	57%	22%	1%	real
	0%	16%	60%	22%	1%	imaginary

Algorithm Used

CGEXP(z) is calculated as follows.

> Let z = x+i•y
> If |y| > 1686629713.065
> CGEXP(z) = (0.0,0.0)
>
> If x < -710.475860073943942
> CGEXP(z) = (0.0,0.0)
>
> If x > 709.089565 and y = 0.0
> CGEXP(z) = (+infinity, 0.0)
>
> If 709.089565 < x < 1418.179131425648102
> and a component of the result is out of range,
> that component is set to +infinity.
>
> If x > 1418.179131425648102 and y ≠ 0.0
> CGEXP(z) = (±infinity, ±infinity)
>
> Otherwise
> CGEXP(z) = e^x•(cos(y)+i•sin(y))

Error Conditions

The table below shows the possible values of the argument that could cause error conditions.

Error Conditions for CGEXP

Real Part of Argument	Imaginary Part of Argument	Result	Error Messages
Any value	> 1686629713.065	(0.0,0.0)	#1
< -710.475860073943942	0.0	(0.0,0.0)	#2
	Not 0.0 and ≤ 1686629713.065	(0.0,0.0)	#2 and #3
Between -710.475860073943942 and 709.089565	Not 0.0 and ≤ 1686629713.065	Underflow may occur on neither, either, or both parts	None or #2 or #3 or #2 and #3
> 709.089565	0.0	(infinity, 0.0)	#4
> 1418.179131425648102	Not 0.0 and ≤ 1686629713.065	(± infinity, ± infinity)	#4 and #5
Between 709.089565 and 1418.179131425648102	Not 0.0 and ≤ 1686629713.065	Overflow may occur on neither, either, or both parts	None or #4 or #5 or #4 and #5

Error Messages:

1. CGEXP: ABS(IMAG(arg)) too large; result = zero
2. CGEXP: Real part underflow
3. CGEXP: Imaginary part underflow
4. CGEXP: REAL(arg) too large; REAL(result) = +infinity
5. CGEXP: REAL(arg) too large; IMAG(result) = +infinity

Description
The EXP1. routine raises one integer to the power of another integer. That is:

EXP1.(m,n) = m^n

Routines Called
EXP1. calls the MTHERR routine.

Type of Arguments
The two arguments must be integer values; they can be any such values.

Type of Result
The result returned is an integer value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
EXP1.(m,n) is calculated as shown in the following table.

Calculations for EXP1.

Value of m	Value of n	Result
$\neq 0$	0	1
0	0	0
0	>0	0
0	<0	+infinity
+1	any value	1
-1	even	1
-1	odd	-1
$\neq \pm 1$	<0	0
$\neq \pm 1$	>0	m^n

Error Conditions

1. If the exponent is too large a number, the following message is issued and the result is set to ± infinity.

   EXP1.: Result overflow

2. If both the base and the exponent are 0, the following message is issued and the result is set to 0.

   EXP1.: Zero**zero is indeterminate, result = zero

EXP2.

Description
The EXP2. routine raises a single-precision, floating-point number to the power of an integer. That is:

EXP2.(x,n) = x^n

Routines Called
EXP2. calls the MTHERR routine.

Types of Arguments
There are two arguments. The base must be a single-precision, floating-point value, and the exponent must be an integer value. They can be any such values.

Type of Result
The result returned is a single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval x	n	MRE	RMS
.50000 through 1.0000	2	$7.45\text{x}10^{-9}$ (27.0 bits)	$3.48\text{x}10^{-9}$ (28.1 bits)
.50000 through 1.0000	-5	$3.07\text{x}10^{-8}$ (25.0 bits)	$8.88\text{x}10^{-9}$ (26.7 bits)
.50000 through 1.0000	9	$5.53\text{x}10^{-8}$ (24.1 bits)	$1.61\text{x}10^{-8}$ (25.9 bits)
.50000 through 1.0000	-12	$7.91\text{x}10^{-8}$ (23.6 bits)	$2.37\text{x}10^{-8}$ (25.3 bits)
.50000 through 1.0000	15	$9.08\text{x}10^{-8}$ (23.4 bits)	$2.70\text{x}10^{-8}$ (25.1 bits)
.50000 through 1.0000	-20	$1.27\text{x}10^{-7}$ (22.9 bits)	$3.95\text{x}10^{-8}$ (24.6 bits)
.50000 through 1.0000	40	$2.65\text{x}10^{-7}$ (21.8 bits)	$7.87\text{x}10^{-8}$ (23.6 bits)
total		$2.65\text{x}10^{-7}$ (21.8 bits)	$3.67\text{x}10^{-8}$ (24.7 bits)

LSB error distribution according to the value of n

	-4^+	-3	-2	-1	0	+1	+2	+3	$+4^+$
n = 2	0%	0%	0%	0%	100%	0%	0%	0%	0%
n = -5	0%	0%	5%	24%	41%	25%	5%	0%	0%
n = 9	1%	4%	13%	21%	23%	21%	13%	4%	1%
n = -12	7%	8%	13%	15%	15%	15%	12%	8%	7%
n = 15	9%	9%	12%	13%	13%	13%	12%	9%	9%
n = -20	20%	8%	9%	9%	9%	9%	9%	8%	20%
n = 40	34%	4%	5%	5%	5%	5%	5%	5%	34%
total	10%	5%	8%	12%	29%	12%	8%	5%	10%

Algorithm Used

EXP2.(x,n) is calculated as shown in the following table.

Calculations for EXP2.

Value of x	Value of n	Result
≠ 0.0	0	1.0
0.0	0	0.0
0.0	> 0	0.0
0.0	< 0	+infinity
> 0.0	> 0	x^n

Error Conditions

1. If the exponent has sufficiently large magnitude, overflow occurs in one of the following ways:

Base	Exponent	Result
> 1.0	positive	+infinity
< -1.0	positive, even positive, odd	+infinity -infinity
0.0 to 1.0	negative	+infinity
-1.0 to 0.0	negative, even negative, odd	+infinity -infinity

   and the following message is issued.

   EXP2.: Result overflow

2. If the exponent has sufficiently large magnitude, underflow occurs in one of the following ways:

Magnitude of Base	Exponent	Result
> 1.0	negative	0.0
< 1.0	positive	0.0

   and the following message is issued.

   EXP2.: Result underflow

3. If both the exponent and the base are zero, the following message is issued and a result of zero is returned.

   EXP2.: Zero**zero is indeterminate, result = zero

DEXP2.

Description

The DEXP2. routine raises a double-precision, D-floating-point number to the power of an integer. That is:

DEXP2.(x,n) = x^n

Routines Called

DEXP2. calls the MTHERR routine.

Type of Arguments

There are two arguments. The base must be a double-precision, D-floating-point value, and the exponent must be an integer value. They can be any such values.

Type of Result

The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result

test interval x	n	MRE	RMS
.50000 through 1.0000	2	2.16×10^{-19} (62.0 bits)	1.01×10^{-19} (63.1 bits)
.50000 through 1.0000	-9	1.62×10^{-18} (59.1 bits)	4.72×10^{-19} (60.9 bits)
.50000 through 1.0000	12	2.27×10^{-18} (58.6 bits)	6.79×10^{-19} (60.4 bits)
.50000 through 1.0000	15	2.73×10^{-18} (58.3 bits)	7.89×10^{-19} (60.1 bits)
.50000 through 1.0000	-40	7.50×10^{-18} (56.9 bits)	2.31×10^{-18} (58.6 bits)
total		7.50×10^{-18} (56.9 bits)	1.15×10^{-18} (59.6 bits)

LSB error distribution according to the value of n

	-4^+	-3	-2	-1	0	+1	+2	+3	$+4^+$
n = 2	0%	0%	0%	0%	100%	0%	0%	0%	0%
n = -9	1%	4%	12%	20%	23%	20%	12%	5%	2%
n = 12	6%	8%	12%	15%	16%	15%	13%	9%	6%
n = 15	9%	9%	12%	13%	13%	13%	12%	9%	9%
n = -40	34%	4%	5%	4%	5%	5%	4%	4%	34%
total	10%	5%	8%	11%	31%	11%	8%	5%	10%

Algorithm Used

DEXP2.(x,n) is calculated as shown in the following table.

Calculations for DEXP2.

Value of x	Value of n	Result
≠ 0.0	0	1.0
0.0	0	0.0
0.0	> 0	0.0
0.0	< 0	+infinity
> 0.0	> 0	x^n

Error Conditions

1. If the exponent has sufficiently large magnitude, overflow occurs in one of the following ways:

Base	Exponent	Result
> 1.0	positive	+infinity
< -1.0	positive, even positive, odd	+infinity -infinity
0.0 to 1.0	negative	+infinity
-1.0 to 0.0	negative, even negative, odd	+infinity -infinity

   and the following error message is issued.

   DEXP2.: Result overflow

2. If the exponent has sufficiently large magnitude, underflow occurs in one of the following ways:

Magnitude of Base	Exponent	Result
> 1.0	negative	0.0
< 1.0	positive	0.0

   and the following message is issued.

   DEXP2.: Result underflow

3. If both the exponent and the base are zero, the following message is issued and the result is set to zero.

   DEXP2.: Zero**zero is indeterminate, result = zero

GEXP2.

Description

The GEXP2. routine raise a double-precision, G-floating-point number to the power of an integer. That is:

GEXP2.(x,n) = x^n

Routines Called

GEXP2. calls the MTHERR routine.

Type of Arguments

There are two arguments. The base must be a double-precision, G-floating-point value; it can be any such value. The exponent must be an integer value; it can be any such value.

Type of Result

The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result

test interval x	n	MRE	RMS
.50000 through 1.0000	2	$1.72x10^{-18}$ (59.0 bits)	$8.11x10^{-19}$ (60.1 bits)
.50000 through 1.0000	-9	$1.26x10^{-17}$ (56.1 bits)	$3.79x10^{-18}$ (57.9 bits)
.50000 through 1.0000	12	$1.69x10^{-17}$ (55.7 bits)	$5.45x10^{-18}$ (57.3 bits)
.50000 through 1.0000	15	$2.13x10^{-17}$ (55.4 bits)	$6.27x10^{-18}$ (57.1 bits)
.50000 through 1.0000	-40	$5.64x10^{-17}$ (54.0 bits)	$1.85x10^{-17}$ (55.6 bits)
total		$5.64x10^{-17}$ (54.0 bits)	$9.25x10^{-18}$ (56.6 bits)

LSB error distribution according to the value of n

	-4^+	-3	-2	-1	0	+1	+2	+3	$+4^+$
n = 2	0%	0%	0%	0%	100%	0%	0%	0%	0%
n = -9	2%	5%	12%	21%	23%	20%	12%	4%	1%
n = 12	6%	8%	13%	16%	15%	15%	13%	8%	6%
n = 15	9%	9%	12%	13%	14%	13%	12%	9%	9%
n = -40	34%	4%	4%	5%	4%	5%	5%	4%	34%
total	10%	5%	8%	11%	31%	10%	8%	5%	10%

Algorithm Used

GEXP2.(x,n) is calculated as shown in the following table.

Calculations for GEXP2.

Value of x	Value of n	Result
$\neq 0.0$	0	1.0
0.0	0	0.0
0.0	> 0	0.0
0.0	< 0	+infinity
> 0.0	> 0	x^n

Error Conditions

1. If the exponent has sufficiently large magnitude, overflow occurs in one of the following ways:

Base	Exponent	Result
> 1.0	positive	+infinity
<-1.0	positive, even	+infinity
	positive, odd	-infinity
0.0 to 1.0	negative	+infinity
-1.0 to 0.0	negative, even	+infinity
	negative, odd	-infinity

   and the following error message is issued:

   GEXP2.: Result overflow

2. If the exponent has sufficiently large magnitude, underflow occurs in one of the following ways:

Magnitude of Base	Exponent	Result
> 1.0	negative	0.0
< 1.0	positive	0.0

   and the following message is issued:

   GEXP2.: Result underflow

3. If both the exponent and the base are zero, the following message is issued and the result is set to zero.

   GEXP2.: Zero**zero is indeterminate, result = zero

CEXP2.

Description

The CEXP2. routine raises a complex, single-precision, floating-point number to the power of an integer. That is:

$$\text{CEXP2.}(z,n) = z^n$$

Routines Called

CEXP2. calls the CDLOG, DLOG, DSIN, DCOS, DEXP, and MTHERR routines.

Type of Arguments

There are two arguments. The base must be a complex, single-precision, floating-point value, and the exponent must be an integer. They can be any such values.

Type of Result

The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	.50000 through 1.0000 for z (real) .50000 through 1.0000 for z (imaginary) -10 through 20 for n				
MRE:	7.45×10^{-9} (27.0 bits) real 7.45×10^{-9} (27.0 bits) imaginary				
RMS:	3.17×10^{-9} (28.2 bits) real 3.16×10^{-9} (28.2 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	0%	0%	100%	0%	0% real
	0%	0%	100%	0%	0% imaginary

When the ratio of the imaginary part of the base to the real part is less than -10^{10}, one part of the result is less accurate. Which part is less accurate depends on the exponent. For example:

test interval:	-1.00000×10^{-10} through -1.00000×10^{-15} for z (real) -2.0000 through -1.0000 for z (imaginary) -1 for n				
	-2	-1	0	+1	+2
LSB error distribution:	0%	6%	65%	28%	2% real
	0%	0%	100%	0%	0% imaginary

test interval:	-1.00000×10^{-10} through -1.00000×10^{-15} for z (real) -2.0000 through -1.0000 for z (imaginary) 2 for n				
	-2	-1	0	+1	+2
LSB error distribution:	0%	0%	100%	0%	0% real
	6%	27%	60%	8%	0% imaginary

Algorithm Used

CEXP2.(z,n) is calculated as follows.

Let z = x+i•y

First the routine checks for the special cases shown in the following table.

Special Cases for CEXP2.

Value of x	Value of y	Value of n	Result
any value	any value	1	x+i•y
0.0	0.0	< 0	(+infinity, +infinity)
0.0	0.0	0	(0.0,0.0)
0.0	0.0	> 0	(0.0,0.0)
not both 0.0		0	(1.0,0.0)

If none of the special cases applies, the routine continues calculations as follows.

The CEXP2. function is evaluated as the complex exponential of n•(LNRHO+i•THETA).

LNRHO is the real part of:

$\log_e(x+i \cdot y)$

THETA is the imaginary part of:

$\log_e(x+i \cdot y)$

The real part of n•(LNRHO+i•THETA) is:

ALPHA = n•LNRHO

and the imaginary part is:

PHI = n•THETA

Since it is ultimately $e^{i \cdot PHI}$ that is needed, it would appear that sin(PHI) and cos(PHI) are needed. However, these functions will be multiplied by e^{ALPHA}, and the handling of exception boundaries on the product will be expedited by use of $\log_e$(sin(PHI)) and $\log_e$(cos(PHI)), which will be added to ALPHA before the call to the DEXP function. The absolute values of sin(PHI) and cos(PHI) are used as arguments of the CDLOG function; the signs of sin(PHI) and cos(PHI) are stored for use in determining the signs for the real and imaginary parts of the complex exponential, CEXP.

The real part of the final result is:

$\text{sgn}(\cos(PHI)) \cdot e^{ALPHA+\log_e(|\cos(PHI)|)}$

The imaginary part of the final result is:

$\text{sgn}(\sin(PHI)) \cdot e^{ALPHA+\log_e(|\sin(PHI)|)}$

Error Conditions

The following error messages are returned for error conditions detected during the check for the special cases shown above. Other errors detected will result in error messages relating to the CEXP3. routine because CEXP2. is part of the CEXP3. routine.

1. If both the real and imaginary parts of the argument are zero and the exponent is also zero, the following message is issued and the result is set to (0.0,0.0).

   CEXP2.: Zero**zero is indeterminate, result = zero

2. If both the real and imaginary parts of the argument are zero and the exponent is negative, the following message is issued and the result is set to (infinity, infinity).

   CEXP2.: Zero** negative exponent, result = infinity

3. If PHI ≥ 6746518852, argument reduction for sin/cos is impossible so the following message is issued and the result is set to (+infinity, +infinity).

   CEXP2.: Both parts indeterminate

4. If the base and/or the exponent are such that one or both parts of the result overflow, one of the following messages is issued and the corresponding result is set to ± infinity.

   CEXP2.: Real part overflow
   CEXP2.: Imaginary part overflow
   CEXP2.: Both parts overflow

5. If the base and/or the exponent are such that one or both parts of the result underflows, one of the following messages is issued and the corresponding result is set to 0.0.

   CEXP2.: Real part underflow
   CEXP2.: Imaginary part underflow
   CEXP2.: Both parts underflow

Description

The EXP3. routine raises a single-precision, floating-point number to the power of another single-precision, floating-point number. That is:

EXP3.(x,y) = x^y

Routines Called

EXP3. calls the MTHERR routine.

Type of Arguments

There are two arguments; both must be single-precision, floating-point values. The base must not be less than zero unless the exponent is an integer. The base must not be equal to zero unless the exponent is greater than zero.

Type of Result

The result returned is a single-precision, floating-point value in the range 2^{-129} to 2^{127}.

Accuracy of Result

test interval x	y	MRE	RMS
.50000 through 1.0000	5.1	$1.52x10^{-8}$ (26.0 bits)	$4.70x10^{-9}$ (27.7 bits)
.50000 through 1.0000	-10.1	$1.86x10^{-8}$ (25.7 bits)	$4.92x10^{-9}$ (27.6 bits)
.50000 through 1.0000	15.1	$2.27x10^{-8}$ (25.4 bits)	$5.42x10^{-9}$ (27.5 bits)
.50000 through 1.0000	-20.1	$3.14x10^{-8}$ (24.9 bits)	$6.05x10^{-9}$ (27.3 bits)
.50000 through 1.0000	30.1	$3.90x10^{-8}$ (24.6 bits)	$7.32x10^{-9}$ (27.0 bits)
.50000 through 1.0000	-50.1	$6.18x10^{-8}$ (23.9 bits)	$1.07x10^{-8}$ (26.5 bits)
.50000 through 1.0000	80.1	$9.04x10^{-8}$ (23.4 bits)	$1.60x10^{-8}$ (25.9 bits)
	total	$9.04x10^{-8}$ (23.4 bits)	$8.74x10^{-9}$ (26.8 bits)

LSB error distribution according to the value of y

	-4+	-3	-2	-1	0	+1	+2	+3	+4+
y = 5.1	0%	0%	0%	12%	74%	14%	0%	0%	0%
y = -10.1	0%	0%	0%	11%	70%	19%	0%	0%	0%
y = 15.1	0%	0%	0%	18%	66%	16%	0%	0%	0%
y = -20.1	0%	0%	0%	14%	61%	24%	1%	0%	0%
y = 30.1	0%	0%	3%	21%	56%	18%	1%	0%	0%
y = -50.1	0%	0%	3%	17%	46%	23%	7%	2%	1%
y = 80.1	4%	4%	9%	19%	36%	19%	6%	2%	1%
total	1%	1%	2%	16%	58%	19%	2%	1%	0%

Algorithm Used

EXP3.(x,y) is calculated as follows.

First the routine checks for the special cases shown in the following table.

Special Cases for EXP3.

Value of x	Value of y	Result
0.0	> 0.0	0.0
0.0	0.0	0.0
0.0	< 0.0	infinity
≠ 0.0	0.0	1.0
< 0.0	odd integer	< 0.0
< 0.0	even integer	> 0.0
< 0.0	not integer	$(-x)^y$

Otherwise

$x^y = 2^w$

$w = y \cdot \log_2(x)$

$\log_2(x)$ is calculated as follows:

$x = 2^m \cdot f$ where $.5 \le f < 1.0$

Let p be an odd integer < 16 and

let $a = 2^{-p/16}$

Then select p to minimize $|a-f|$

now $x = 2^m \cdot a \cdot (f/a)$

Then $\log_2(x) = m + \log_2(a) + \log_2(f/a)$ or

$\log_2(x) = m - p/16 + \log_2(f/a)$

Let $u1 = m-p/16$ and

$u2 = \log_2(f/a) = \log_2((1+s)/(1-s))$

Then $\log_2(x) = u1+u2$ and

$s = (f-a)/(f+a)$

A rational approximation is used to evaluate u2; u1 and u2 are then used to determine w1 and w2.

$w = y \cdot \log_2(x) = w1+w2$ and

w1 = FLOAT(INT(w•16.0))/16.0 = m1+p1/16

m1 and p1 are integers with $0 \le p1 \le 15$

Finally

If $-129 \le w < 127$

EXP3.(x,y) = $x^y = 2^w$ is reconstructed as:

EXP3.(x,y) = $2^{w1} \cdot 2^{w2}$

2^{w1} is evaluated by table lookup and 2^{w2} is evaluated from another rational approximation.

Error Conditions

1. If the base is a negative value and the exponent is not an integer, the following message is issued and the calculation proceeds using the absolute value of the base.

   EXP3.: Negative base**non-integer; ABS(base) used

2. If the base is 0.0 and the exponent is negative, the following message is issued and the result is set to infinity.

   EXP3.: Zero**negative exponent; result = infinity

3. If both the base and the exponent are 0.0, the following message is issued and the result is set to 0.0.

   EXP3.: Zero**zero is indeterminate; result = zero

4. If $y \cdot \log_2(x) \geq 127$, the result overflows. Then the following message is issued and the result is set to –infinity if x is less than 0.0 and y is an odd integer. Otherwise, the result is set to +infinity.

   EXP3.: Result overflow

5. If $y \cdot \log_2(x) < -129$, the result underflows. Then the following message is issued and the result is set to 0.0.

   EXP3.: Result underflow

DEXP3.

Description

The DEXP3. routine raises a double-precision, D-floating-point number to the power of another double-precision, D-floating-point number. That is:

DEXP3.(x,y) = x^y

Routines Called

DEXP3. calls the MTHERR routine.

Type of Argument

There are two arguments; both must be double-precision, D-floating-point values. The base must not be less than zero unless the exponent is an integer. The base must not be equal to zero unless the exponent is greater than zero.

Type of Result

The result returned is a double-precision, D-floating-point value greater than or equal to 2^{-129} and less than or equal to 2^{127}.

Accuracy of Result

test interval x	y	MRE	RMS
.50000 through 1.0000	5.1	5.23×10^{-19} (60.7 bits)	1.45×10^{-19} (62.6 bits)
.50000 through 1.0000	-10.1	5.50×10^{-19} (60.7 bits)	1.46×10^{-19} (62.6 bits)
.50000 through 1.0000	20.1	9.07×10^{-19} (59.9 bits)	1.84×10^{-19} (62.2 bits)
.50000 through 1.0000	-50.1	1.97×10^{-18} (58.8 bits)	3.27×10^{-19} (61.4 bits)
.50000 through 1.0000	80.1	3.02×10^{-18} (58.2 bits)	5.10×10^{-19} (60.8 bits)
total		3.02×10^{-18} (58.2 bits)	2.98×10^{-19} (61.5 bits)

LSB error distribution according to the value of y

	-4^+	-3	-2	-1	0	+1	+2	+3	$+4^+$
y = 5.1	0%	0%	0%	7%	73%	20%	0%	0%	0%
y = -10.1	0%	0%	0%	13%	70%	17%	0%	0%	0%
y = 20.1	0%	0%	0%	11%	63%	25%	1%	0%	0%
y = -50.1	1%	2%	6%	19%	46%	21%	4%	1%	0%
y = -80.1	1%	2%	5%	16%	35%	22%	10%	5%	5%
total	0%	1%	2%	13%	57%	21%	3%	1%	1%

Algorithm Used

DEXP3.(x,y) is calculated as follows.

First the routine checks for the special cases shown in the following table.

Special Cases for DEXP3.

Value of x	Value of y	Result
0.0	> 0.0	0.0
0.0	0.0	0.0
0.0	< 0.0	infinity
≠ 0.0	0.0	1.0
< 0.0	odd integer	< 0.0
< 0.0	even integer	> 0.0
< 0.0	not integer	$(-x)^y$

Otherwise

$x^y = 2^w$

$w = y \cdot \log_2(x)$

$\log_2(x)$ is calculated as follows:

$x = 2^m \cdot f$ where $.5 \leq f < 1.0$

Let p be an odd integer < 16 and

let $a = 2^{-p/16}$

Then select p to minimize $|a-f|$

now $x = 2^m \cdot a \cdot (f/a)$

Then $\log_2(x) = m+\log_2(a)+\log_2(f/a)$ or

$\log_2(x) = m-p/16+\log_2(f/a)$

Let $u1 = m-p/16$ and

$u2 = \log_2(f/a) = \log_2((1+s)/(1-s))$

Then $\log_2(x) = u1+u2$ and

$s = (f-a)/(f+a)$

A rational approximation is used to evaluate u2; u1 and u2 are then used to determine w1 and w2.

$w = y \cdot \log_2(x) = w1+ w2$ and

$w1 = \text{FLOAT}(\text{INT}(w \cdot 16.0))/16.0 = m1+p1/16$

m1 and p1 are integers with $0 \leq p1 \leq 15$

Finally

If $-129 \leq w < 127$

DEXP3.(x,y) = $x^y = 2^w$ is reconstructed as:

DEXP3.(x,y) = $2^{w1} \cdot 2^{w2}$

2^{w1} is evaluated by table lookup and 2^{w2} is evaluated from another rational approximation.

Error Conditions

1. If the base is a negative value and the exponent is not an integer, the following message is issued and the calculation proceeds using the absolute value of the base.

   DEXP3.: Negative base**non-integer; ABS(base) used

2. If the base is 0.0 and the exponent is negative, the following message is issued and the result is set to infinity.

   DEXP3.: Zero**negative exponent; result = infinity

3. If both the base and the exponent are 0.0, the following message is issued and the result is set to 0.0.

   DEXP3.: Zero**zero is indeterminate; result = zero

4. If $y \cdot \log_2(x) \geq 127$, the result overflows. Then the following message is issued and the result is set to –infinity if x is less than 0.0 and y is an odd integer. Otherwise, the result is set to +infinity.

   DEXP3.: Result overflow

5. If $y \cdot \log_2(x) < -129$, the result underflows. Then the following message is issued and the result is set to 0.0.

   DEXP3.: Result underflow

Description

The GEXP3. routine raises a double-precision, G-floating-point number to the power of another double-precision, G-floating-point number. That is:

GEXP3.(x,y) = x^y

Routines Called

GEXP3. calls the MTHERR routine.

Type of Arguments

There are two arguments; both must be double-precision, G-floating-point values. The base must not be less than zero unless the exponent is an integer. The base must not be equal to zero unless the exponent is greater than zero.

Type of Result

The result returned is a double-precision, G-floating-point value in the range 2^{-1025} to 2^{1023}.

Accuracy of Result

test interval x	y	MRE	RMS
.50000 through 1.0000	5.10	3.69×10^{-18} (57.9 bits)	1.18×10^{-18} (59.6 bits)
.50000 through 1.0000	-10.10	4.91×10^{-18} (57.5 bits)	1.22×10^{-18} (59.5 bits)
.50000 through 1.0000	20.10	7.92×10^{-18} (56.8 bits)	1.49×10^{-18} (59.2 bits)
.50000 through 1.0000	-50.10	1.46×10^{-17} (55.9 bits)	2.70×10^{-18} (58.4 bits)
.50000 through 1.0000	80.10	2.17×10^{-17} (55.4 bits)	4.13×10^{-18} (57.7 bits)
total		2.17×10^{-17} (55.4 bits)	2.43×10^{-18} (58.5 bits)

LSB error distribution according to the value of y

	-4^+	-3	-2	-1	0	+1	+2	+3	$+4^+$
y = 5.10	0%	0%	0%	14%	70%	16%	0%	0%	0%
y = -10.10	0%	0%	0%	12%	68%	20%	0%	0%	0%
y = 20.10	0%	0%	1%	19%	60%	19%	1%	0%	0%
y = -50.10	0%	1%	4%	17%	43%	24%	7%	2%	1%
y = 80.10	4%	5%	8%	18%	34%	19%	7%	3%	2%
total	1%	1%	3%	16%	55%	20%	3%	1%	1%

Algorithm Used

GEXP3.(x,y) is calculated as follows.

First the routine checks for the special cases shown in the following table.

Special Cases for GEXP3.

Value of x	Value of y	Result
0.0	> 0.0	0.0
0.0	0.0	0.0
0.0	< 0.0	infinity
≠ 0.0	0.0	1.0
< 0.0	odd integer	< 0.0
< 0.0	even integer	> 0.0
< 0.0	not integer	$(-x)^y$

Otherwise
$x^y = 2^w$
$w = y \cdot \log_2(x)$
$\log_2(x)$ is calculated as follows:
$x = 2^m \cdot f$ where $.5 \leq f < 1.0$
Let p be an odd integer < 16 and
let $a = 2^{-p/16}$
Then select p to minimize $|a-f|$
now $x = 2^m \cdot a \cdot (f/a)$
Then $\log_2(x) = m+\log_2(a)+\log_2(f/a)$ or
$\log_2(x) = m-p/16+\log_2(f/a)$

Let $u1 = m-p/16$ and
$u2 = \log_2(f/a) = \log_2((1+s)/(1-s))$
Then $\log_2(x) = u1+u2$ and
$s = (f-a)/(f+a)$

A rational approximation is used to evaluate u2; u1 and u2 are then used to determine w1 and w2.
$w = y \cdot \log_2(x) = w1+w2$ and
$w1 = \mathrm{FLOAT}(\mathrm{INT}(w \cdot 16.0))/16.0 = m1+p1/16$
m1 and p1 are integers with $0 \leq p1 \leq 15$

Finally
If $-1025 \leq w < 1023$
GEXP3.(x,y) = $x^y = 2^w$ is reconstructed as:
GEXP3.(x,y) = $2^{w1} \cdot 2^{w2}$
2^{w1} is evaluated by table lookup and 2^{w2} is evaluated from another rational approximation.

Error Conditions

1. If the base is a negative value and the exponent is not an integer, the following message is issued and the calculation proceeds using the absolute value of the base.

   GEXP3.: Negative base**non-integer; ABS(base) used

2. If the base is 0.0 and the exponent is negative, the following message is issued and the result is set to infinity.

   GEXP3.: Zero**negative exponent; result = infinity

3. If both the base and the exponent are 0.0, the following message is issued and the result is set to 0.0.

   GEXP3.: Zero**zero is indeterminate, result = zero

4. If $y \cdot \log_2(x) \geqslant 1023$, the result overflows, the following message is issued, and the result is set to –infinity if x less than 0.0 and y is an odd integer. Otherwise, the result is set to +infinity.

   GEXP3.: Result overflow

5. If $y \cdot \log_2(x) < -1025$, the result underflows, the following message is issued, and the result is set to 0.0.

   GEXP3.: Result underflow

CEXP3.

Description

The CEXP3. routine raises a complex, single-precision, floating-point number to the power of another complex, single-precision, floating-point number. That is:

CEXP3.(z,g) = z^g

Routines Called

CEXP3. calls the CDLOG, DLOG, DSIN, DCOS, DEXP, and MTHERR routines.

Type of Arguments

There are two arguments; both must be complex, single-precision, floating-point values. They can be any such values.

Type of Result

The result returned is a complex, single-precision, floating-point value. It may be any such value.

Accuracy of Result

test interval:	.50000 through 1.0000 for z (real) .50000 through 1.0000 for z (imaginary) -100.00 through 207.00 for g (real) -163.00 through 7.00 for g (imaginary)					
MRE:	7.45×10^{-9} (27.0 bits) real 7.45×10^{-9} (27.0 bits) imaginary					
RMS:	3.17×10^{-9} (28.2 bits) real 3.17×10^{-9} (28.2 bits) imaginary					
	-2	-1	0	+1	+2	
LSB error distribution:	0%	0%	100%	0%	0%	real
	0%	0%	100%	0%	0%	imaginary

When the ratio of the imaginary part of the base to the real part is less than -10^{10}, one part of the result is less accurate. Which part is less accurate depends on the exponent. For example:

test interval:	-1.00000×10^{-10} through -1.00000×10^{-15} for z (real) -2.0000 through -1.0000 for z (imaginary) (-1,0) for g					
	-2	-1	0	+1	+2	
LSB error distribution:	0%	6%	65%	28%	2%	real
	0%	0%	100%	0%	0%	imaginary
test interval:	-1.00000×10^{-10} through -1.00000×10^{-15} for z (real) -2.0000 through -1.0000 for z (imaginary) (2,0) for g					
	-2	-1	0	+1	+2	
LSB error distribution:	0%	0%	100%	0%	0%	real
	6%	27%	60%	8%	0%	imaginary

Algorithm Used

CEXP3.(z,g) is calculated as follows.

Let $z = x+i \cdot y$
$g = a+i \cdot b$

First the routine checks for the special cases shown in the following table.

Special Cases for CEXP3.

Value of x	Value of y	Value of a	Result
0.0	0.0	> 0.0	(0.0,0.0)
0.0	0.0	≤ 0.0	(+infinity, +infinity)
0.0	0.0	0.0	(0.0,0.0)

If none of the special cases applies, the routine continues calculation as follows.

If x and $y \neq 0$
$x+i \cdot y$ is rewritten as
$e^{\log_e(x+i \cdot y)}$

The CEXP3. function is evaluated as the complex exponential of $(a+i \cdot b) \cdot (LNRHO+i \cdot THETA)$.
LNRHO is the real part of:
$\log_e(x+i \cdot y)$
THETA is the imaginary part of:
$\log_e(x+i \cdot y)$
The real part of $(a+i \cdot b) \cdot (LNRHO+i \cdot THETA)$ is:
$ALPHA = a \cdot LNRHO-b \cdot THETA$
and the imaginary part is:
$PHI = a \cdot THETA+b \cdot LNRHO$

Since it is ultimately $e^{i \cdot PHI}$ that is needed, it would appear that sin(PHI) and cos(PHI) are needed. However, these functions will be multiplied by e^{ALPHA}, and the handling of exception boundaries on the product will be expedited by use of $\log_e$(sin(PHI)) and $\log_e$(cos(PHI)), which will be added to ALPHA before the call to the DEXP function. The absolute values of sin(PHI) and cos(PHI) are used as arguments of the CDLOG function; the signs of sin(PHI) and cos(PHI) are stored for use in determining the signs for the real and imaginary parts of the complex exponential, CEXP.

The real part of the final result is:
$\text{sgn}(\cos(PHI)) \cdot e^{ALPHA+\log_e(|\cos(PHI)|)}$

The imaginary part of the final result is:
$\text{sgn}(\sin(PHI)) \cdot e^{ALPHA+\log_e(|\sin(PHI)|)}$

Error Conditions

1. If both the real and imaginary parts of both arguments are 0.0, the following message is issued and the result is set to (0.0,0.0).

   CEXP3.: Zero**zero is indeterminate; result = zero

2. If both the real and imaginary parts of the base are zero and the real part of the exponent is negative, the following message is issued and the result is set to (+infinity,+infinity).

   CEXP3.: Zero**(negative,non-zero) is indeterminate,
   result = (infinity,infinity)

3. If PHI $\geq$ 6746518852, argument reduction for sin/cos is impossible so the following message is issued and the result is set to (+infinity,+infinity).

   CEXP3.: Both parts indeterminate

4. If the base and/or the exponent are such that one or both parts of the result overflow, one of the following messages is issued and the corresponding result is set to $\pm$ infinity.

   CEXP3.: Real part overflow
   CEXP3.: Imaginary part overflow
   CEXP3.: Both parts overflow

5. If the base and/or the exponent are such that one or both parts of the result underflows, one of the following messages is issued and the corresponding result is set to (0.0).

   CEXP3.: Real part underflow
   CEXP3.: Imaginary part underflow
   CEXP3.: Real and imaginary parts underflow

Chapter 5
Trigonometric Routines

Description

The SIN routine calculates the single-precision, floating-point sine of the single-precision, floating-point angle given in radians as the argument. That is:

SIN(x) = sin(x)

Routines Called

SIN calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value less than or equal to 210828714.

Type of Result

The result returned is a single-precision, floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	1.95×10^{-8} (25.6 bits)				
RMS:	3.87×10^{-9} (27.9 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	12%	78%	10%	0%

Algorithm Used

SIN(x) is calculated as follows. Note that SIN(x) = -SIN(-x).

Let $|x| = \pi \cdot n + f$
$|f| < \pi/2$

The argument reduction is as follows.
n = the nearest integer to $|x|/\pi$
Then the reduced argument is:
$f = |x| - \pi \cdot n$

If $|f| < 863167530\times10^{-4}$
sin(f) = f

Otherwise
$\sin(f) = f + f \cdot R(g)$
$g = f^2$
$R(g) = ((((r5 \cdot g+r4) \cdot g+r3) \cdot g+r2) \cdot g+r1) \cdot g$
r1 = -.166666666
$r2 = .833333072\times10^{-2}$
$r3 = -.198408328\times10^{-3}$
$r4 = .275239711\times10^{-5}$
$r5 = -.238683464\times10^{-7}$

Finally
$SIN(x) = sgn(x) \cdot (-1)^n \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than 210828714, the following message is issued and the result is set to 0.0.

```
SIN: ABS(arg) too large; result = zero
```

Description

The SIND routine calculates the single-precision, floating-point sine of the single-precision, floating-point angle given in degrees as the argument. That is:

SIND(x) = sin(x)

Routines Called

SIND calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value less than or equal to 47185919.

Type of Result

The result returned is a single-precision, floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-1000.0 through 3600.0				
MRE:	1.95×10^{-8} (25.6 bits)				
RMS:	4.11×10^{-9} (27.9 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	13%	73%	14%	0%

Algorithm Used

SIND(x) is calculated as follows. Note that SIND(x) = -SIND(-x).

Let $|x| = 180 \cdot n + f$
$|f| \leq 90$

The argument reduction is as follows.
n = the nearest integer to $|x|/180$
Then the reduced argument, converted to radians is:
$f = (|x| - 180 \cdot n) \cdot (\pi/180)$

If $|f| < 863167530 \times 10^{-4}$
$\sin(f) = f$

Otherwise
$\sin(f) = f + f \cdot R(g)$
$g = f^2$
$R(g) = ((((r5 \cdot g + r4) \cdot g + r3) \cdot g + r2) \cdot g + r1) \cdot g$
$r1 = -.166666666$
$r2 = .833333072 \times 10^{-2}$
$r3 = -.198408328 \times 10^{-3}$
$r4 = .275239711 \times 10^{-5}$
$r5 = -.238683464 \times 10^{-7}$

Finally
$SIND(x) = sgn(x) \cdot (-1)^n \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than 47185919, the following message is issued and the result is set to 0.0.

```
SIND: ABS(arg) too large; result = zero
```

Description

The COS routine calculates the single-precision, floating-point cosine of the single-precision, floating-point angle given in radians as the argument. That is:

COS(x) = cos(x)

Routines Called

COS calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value less than 210828714.

Type of Result

The result returned is a single-precision, floating-point value in the range −1.0 to 1.0.

Accuracy of Result

test interval:	−10.000 through 201.06				
MRE:	$1.86\text{x}10^{-8}$ (25.7 bits)				
RMS:	$4.26\text{x}10^{-9}$ (27.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	12%	70%	17%	0%

Algorithm Used

COS(x) is calculated as follows. Note that COS(x) = COS(−x).

Let $|x| = \pi \cdot n + f$
$|f| < \pi/2$

The argument reduction is as follows.
$n = .5 +$ the nearest integer to $|x|/\pi$
Then the reduced argument is:
$f = |x| - \pi \cdot n$

If $|f| < .863167530\text{x}10^{-4}$
$\sin(f) = f$

Otherwise
$\sin(f) = f + f \cdot R(g)$
$g = f^2$
$R(g) = ((((r5 \cdot g + r4) \cdot g + r3) \cdot g + r2) \cdot g + r1) \cdot g$
$r1 = -.166666666$
$r2 = .833333072\text{x}10^{-2}$
$r3 = -.198408328\text{x}10^{-3}$
$r4 = .275239711\text{x}10^{-5}$
$r5 = -.238683464\text{x}10^{-7}$

Finally
$COS(x) = (-1)^{n+1} \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than or equal to 210828714, the following message is issued and the result is set to 0.0.

```
COS: ABS(arg) too large; result = zero
```

Description

The COSD routine calculates the single-precision, floating-point cosine of the single-precision, floating-point angle given in degrees as the argument. That is:

$$COSD(x) = \cos(x)$$

Routines Called

COSD calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value less than 47185919.

Type of Result

The result returned is a single-precision, floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-1000.0 through 3600.0				
MRE:	1.75×10^{-8} (25.8 bits)				
RMS:	4.20×10^{-9} (27.8 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	12%	72%	16%	0%

Algorithm Used

COSD(x) is calculated as follows. Note that COSD(x) = COSD(-x).

Let $|x| = 180 \cdot n + f$

$|f| \leq 90$

The argument reduction is:

$n = .5 +$ the nearest integer to $|x|/180$

Then the reduced argument, converted to radians, is:

$f = (|x| - 180 \cdot n) \cdot (\pi/180)$

If $|f| < .863167530 \times 10^{-4}$

$\sin(f) = f$

Otherwise

$\sin(f) = f + f \cdot R(g)$

$g = f^2$

$R(g) = ((((r5 \cdot g + r4) \cdot g + r3) \cdot g + r2) \cdot g + r1) \cdot g$

$r1 = -.166666666$

$r2 = .833333072 \times 10^{-2}$

$r3 = -.198408328 \times 10^{-3}$

$r4 = .275239711 \times 10^{-5}$

$r5 = -.238683464 \times 10^{-7}$

Finally

$COSD(x) = (-1)^{n+1} \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than or equal to 47185919, the following message is issued and the result is set to 0.0.

```
COSD: ABS(arg) too large; result = zero
```

Description

The DSIN routine calculates the double-precision, D-floating-point sine of the double-precision, D-floating-point angle given in radians as the argument. That is:

DSIN(x) = sin(x)

Routines Called

DSIN calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, D-floating-point value less than or equal to 6746518852 (or $2^{31} \cdot \pi$).

Type of Result

The result returned is a double-precision, D-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	6.06×10^{-19} (60.5 bits)				
RMS:	1.35×10^{-19} (62.7 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	22%	68%	10%	0%

Algorithm Used

DSIN(x) is calculated as follows. Note that DSIN(x) = -DSIN(-x).

Let $|x| = \pi \cdot n + f$
$|f| < \pi/2$

The argument reduction is as follows.
$f = ((|x| - n \cdot c1) - n \cdot c2) - n \cdot c3$
c1 = high-order 34 bits of π
c2 = next 31 bits of π
c3 = next 62 bits of π

If $|f| < 2^{-31}$
sin(f) = f

Otherwise

$\sin(f) = f+f \cdot R(g)$

$g = f^2$

$R(g) = (g \cdot XNUM/XDEN+rp1) \cdot g$

$XNUM = ((rp5 \cdot g+rp4) \cdot g+rp3) \cdot g+rp2$

$XDEN = ((g \cdot q2) \cdot g+q1) \cdot g+q0$

$rp1 = -.16666666666666667$

$rp2 = .45145690470446199 0\text{x}10^5$

$rp3 = -.48948715196946379 7\text{x}10^3$

$rp4 = .42818307589777826 5\text{x}10$

$rp5 = -.12156074059671019 0\text{x}10^1$

$q0 = .54174828564535185 3\text{x}10^7$

$q1 = .70249228822184251 8\text{x}10^5$

$q2 = .39492472352045014 1\text{x}10^3$

Finally

$DSIN(x) = sgn(x) \cdot (-1)^n \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than 6746518850, the following message is issued and the result is set to 0.0.

DSIN: ABS(arg) too large; result = zero

Description
The DCOS routine calculates the double-precision, D-floating-point cosine of the double-precision, D-floating-point angle given in radians as the argument. That is:

DCOS(x) = cos(x)

Routines Called
DCOS calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value less than 6746518852 (or $2^{31} \cdot \pi$).

Type of Result
The result returned is a double-precision, D-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	4.96×10^{-19} (60.8 bits)				
RMS:	1.41×10^{-19} (62.6 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	16%	66%	18%	0%

Algorithm Used
DCOS(x) is calculated as follows. Note that DCOS(x) = DCOS(-x).

Let $|x| = \pi \cdot n+f$
$|f| < \pi/2$

The argument reduction is as follows.
$f = (|x|-n \cdot c1)-n \cdot c2)-n \cdot c3$
c1 = high-order 34 bits of π
c2 = next 31 bits of π
c3 = next 62 bits of π

If $|f| < 2^{-31}$
sin(f) = f

Otherwise

$\sin(f) = f+f \cdot R(g)$

$g = f^2$

$R(g) = (g \cdot XNUM/XDEN+rp1) \cdot g$

$XNUM = ((rp5 \cdot g+rp4) \cdot g+rp3) \cdot g+rp2$

$XDEN = ((g \cdot q2) \cdot g+q1) \cdot g+q0$

$rp1 = .16666666666666667$

$rp2 = .45145690470446199 0x10^5$

$rp3 = -.48948715196946379 7x10^3$

$rp4 = .42818307589777826 5x10$

$rp5 = -.12156074059671019 0x10^{-1}$

$q0 = .54174828564535185 3x10^7$

$q1 = .70249228822184251 8x10^5$

$q2 = .39492472352045014 1x10^3$

Finally

$DCOS(x) = (-1)^{n+1} \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than or equal to 6746518852, the following message is issued and the result is set to 0.0.

DCOS: ABS(arg) too large; result = zero

Description

The GSIN routine calculates the double-precision, G-floating-point sine of the double-precision, G-floating-point angle given in radians as the argument. That is,

GSIN(x) = sin(x)

Routines Called

GSIN calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value less than or equal to 1686629713 (or $2^{29} \cdot \pi$).

Type of Result

The result returned is a double-precision, G-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	–10.000 through 201.06				
MRE:	3.30×10^{-18} (58.1 bits)				
RMS:	8.85×10^{-19} (60.0 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	13%	78%	9%	0%

Algorithm Used

GSIN(x) is calculated as follows. Note that GSIN(x) = –GSIN(–x).

Let $|x| = \pi \cdot n + f$
$|f| < \pi/2$

The argument reduction is as follows.
$f = ((|x| - n \cdot c1) - n \cdot c2) - n \cdot c3$
c1 = high-order 30 bits of π
c2 = next 28 bits of π
c3 = next 62 bits of π

If $|f| < 2^{-30}$
sin(f) = f

Otherwise

$\sin(f) = f+f\cdot R(g)$

$g = f^2$

$R(g) = (g\cdot XNUM/XDEN+rp1)\cdot g$

$XNUM = ((rp5\cdot g+rp4)\cdot g+rp3)\cdot g+rp2$

$XDEN = ((g\cdot q2)\cdot g+q1)\cdot g-q0$

$rp1 = -.16666666666666667$

$rp2 = .45145690470446199O\text{x}10^5$

$rp3 = -.48948715196946379 7\text{x}10^3$

$rp4 = .42818307589777826 5\text{x}10^1$

$rp5 = -.12156074059671019 0\text{x}10^{-1}$

$q0 = .54174828564535185 3\text{x}10^7$

$q1 = .70249228822184251 8\text{x}10^5$

$q2 = .39492472352045014 1\text{x}10^3$

Finally

$GSIN(x) = sgn(x)\cdot(-1)^n\cdot\sin(f)$

Error Conditions

If the absolute value of the argument is greater than 1686629713, the following message is issued and the result is set to 0.0.

GSIN: ABS(arg) too large; result = zero

Description

The GCOS routine calculates the double-precision, G-floating-point cosine of the double-precision, G-floating-point angle given in radians as the argument. That is:

GCOS(x) = cos(x)

Routine Called

GCOS calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value less than 1686629713 (or $2^{29} \cdot \pi$).

Type of Result

The result returned is a double-precision, G-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	3.44×10^{-18} (58.0 bits)				
RMS:	9.84×10^{-19} (59.8 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	14%	72%	15%	0%

Algorithm Used

GCOS(x) is calculated as follows. Note that GCOS(x) = GCOS(-x).

Let $|x| = \pi \cdot n + f$
$|f| < \pi/2$

The argument reduction is as follows.
$f = ((|x| - n \cdot c1) - n \cdot c2) - n \cdot c3$
c1 = high-order 30 bits of π
c2 = next 28 bits of π
c3 = next 62 bits of π

If $|f| < 2^{-30}$
sin(f) = f

Otherwise

$\sin(f) = f+f \cdot R(g)$

$g = f^2$

$R(g) = (g \cdot XNUM/XDEN+rp1) \cdot g$

$XNUM = ((rp5 \cdot g+rp4) \cdot g+rp3) \cdot g+rp2$

$XDEN = ((g \cdot q2) \cdot g+q1) \cdot g+q0$

$rp1 = -.166666666666666667$

$rp2 = .451456904704461990 \times 10^5$

$rp3 = -.489487151969463797 \times 10^3$

$rp4 = .428183075897778265 \times 10^1$

$rp5 = -.121560740596710190 \times 10^{-1}$

$q0 = .541748285645351853 \times 10^7$

$q1 = .702492288221842518 \times 10^5$

$q2 = .394924723520450141 \times 10^3$

Finally

$GCOS(x) = (-1)^{n+1} \cdot \sin(f)$

Error Conditions

If the absolute value of the argument is greater than or equal to 1686629713, the following message is issued and the result is set to 0.0.

GCOS: ABS(arg) too large; result = zero

Description
The CSIN routine calculates the complex, single-precision, floating-point sine of the complex, single-precision, floating-point angle given in radians as the argument. That is:

CSIN(z) = sin(z)

Routines Called
CSIN calls the SIN, COS, EXP, ALOG, and MTHERR routines.

Type of Argument
The argument must be a complex, single-precision, floating-point value, the real part of which must be less than 210828714 (or $2^{26} \cdot \pi$).

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	-200.00 through 200.00 real					
	-10.000 through 10.000 imaginary					
MRE:	3.30×10^{-8} (24.9 bits) real					
	3.44×10^{-8} (24.8 bits) imaginary					
RMS:	7.68×10^{-9} (27.0 bits) real					
	6.75×10^{-9} (27.1 bits) imaginary					
	-2	-1	0	+1	+2	
LSB error distribution:	2%	23%	51%	22%	2%	real
	1%	19%	57%	22%	1%	imaginary

Algorithm Used
CSIN(z) is calculated as follows.

Let $z = x+i \cdot y$

If $|x| > 210828714$
 CSIN(z) = (0.0,0.0)

If $|y| > 88.029692$, calculation proceeds as follows.

For the real part of the result:
 Let $t = |\sin(x)|$

 If $t = 0.0$
  $x = 0.0$

 If $\log_e(t)+|y| > 88.722839$
  $x = \pm$machine infinity
  $(88.722839 = 88.029692+\log_e(2))$

For the imaginary part of the result:
Let $t = |\cos(x)| \neq 0$

If $\log_e(t)+|y| < 88.722839$
$y = \pm$ infinity

Otherwise
$CSIN(z) = \sin(x) \cdot \cosh(y) + i \cdot \cos(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than 210828714, the following message is issued and the result is set to (0.0,0.0).

   CSIN: ABS(REAL(arg)) too large; result = zero

2. If $|y|+\log_e(|\sin(x)|) > 88.722839$, the real part overflows. If $|y|+\log_e(|\cos(x)|) > 88.722839$, the imaginary part overflows. If either part overflows, one of the following messages is issued and the relevant part of the result is set to $\pm$ machine infinity.

   CSIN: Imaginary part overflow
   CSIN: Real part overflow

3. If the imaginary part of the result is too small a number, the following message is issued and the imaginary part of the result is set to 0.0.

   CSIN: Imaginary part underflow

Description
The CCOS routine calculates the complex, single-precision, floating-point cosine of the complex, single-precision, floating-point angle given in radians as the argument. That is:

CCOS(z) = cos(z)

Routines Called
CCOS calls the SIN, COS, EXP, ALOG, and MTHERR routines.

Type of Argument
The argument must be a complex, single-precision, floating-point value, the real part of which must be less than 210828714 (or $2^{26} \cdot \pi$).

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	-200.00 through 200.00 real				
	-10.000 through 10.000 imaginary				
MRE:	3.35×10^{-8} (24.8 bits) real				
	3.57×10^{-8} (24.7 bits) imaginary				
RMS:	7.76×10^{-9} (26.9 bits) real				
	6.68×10^{-9} (27.2 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	2%	20%	50%	25%	3% real
	1%	20%	57%	20%	1% imaginary

Algorithm Used
CCOS(z) is calculated as follows.

Let z = x+i•y

If |x| > 210828714
CCOS(z) = (0.0,0.0)

If |y| > 88.029692 calculation proceeds as follows.

For the real part of the result:
Let t = |cos(x)|≠0

If $\log_e(t)+|y| > 88.722839$
x = ± machine infinity
($88.722839 = 88.029692+\log_e(2)$)

For the imaginary part of the result:
Let $t = |\sin(x)|$

If $t = 0.0$
$y = 0.0$

If $\log_e(t)+|y| > 88.722839$
$y = \pm$ machine infinity

Otherwise
$CCOS(z) = \cos(x) \cdot \cosh(y) - i \cdot \sin(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than 210828714, the following message is issued and the result is set to (0.0,0.0).

   CCOS: ABS(REAL(arg)) too large: result = zero

2. If $|y|+\log_e(|\cos(x)|) > 88.722839$, the real part overflows. If $|y|+\log_e(|\sin(x)|) > 88.722839$, the imaginary part overflows. If either part overflows, one of the following messages is issued and the relevant part of the result is set to ± machine infinity.

   CCOS: Imaginary part overflow
   CCOS: Real part overflow

3. If the imaginary part of the result is too small a number, the following message is issued and the imaginary part of the result is set to 0.0.

   CCOS: Imaginary part underflow

Description

The CDSIN subroutine calculates the complex, double-precision, D-floating-point sine of the complex, double-precision, D-floating-point angle given in radians as the argument. That is:

CDSIN(z,r) = sin(z)
z = location of input value
r = location of result

Routines Called

CDSIN calls the DSIN, DCOS, DEXP, DLOG, and MTHERR routines.

Type of Argument

CDSIN is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, D-floating-point value, the real part of which must be less than $2^{31} \cdot \pi - \pi/2$.

Type of Result

The result returned is a complex, double-precision, D-floating-point value; it may be any such value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-200.00 through 200.00 real				
	-10.000 through 10.000 imaginary				
MRE:	1.09×10^{-18} (59.7 bits) real				
	9.86×10^{-19} (59.8 bits) imaginary				
RMS:	2.22×10^{-19} (62.0 bits) real				
	2.08×10^{-19} (62.1 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	2%	22%	51%	23%	2% real
	2%	26%	54%	17%	1% imaginary

Algorithm Used

CDSIN(z) is calculated as follows.

Let $z = x+i \cdot y$

If $|x| > 2^{31} \cdot \pi - \pi/2$
CDSIN(z) = (0.0,0.0)

If $|y| > 88.029692$, calculations proceed as follows.

For the real part of the result:
Let $t = |\sin(x)|$

If $t = 0.0$
$x = 0.0$

If $\log_e(t)+|y| > 88.722839$
$x = \pm$ infinity
$(88.722839 = 88.029692+\log_e(2))$

For the imaginary part of the result:
Let $t = |\cos(x)| \neq 0$

If $\log_e(t)+|y| > 88.722839$
$y = \pm$ infinity

Otherwise
$CDSIN(z) = \sin(x) \cdot \cosh(y)+i \cdot \cos(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than $2^{31} \cdot \pi - \pi/2$, the following message is issued and the result is set to (0.0,0.0).

   CDSIN: ABS(REAL(arg)) too large; result = zero

2. If $|y|+\log_e(|\sin(x)|) > 88.722839$, the real part overflows. If $|y|+\log_e(|\cos(x)|) > 88.722839$, the imaginary part overflows. If either part overflows, one of the following messages is issued and the relevant part of the result is set to ± machine infinity.

   CDSIN: ABS(IMAG(arg)) too large; REAL(result) = infinity
   CDSIN: ABS(IMAG(arg)) too large; IMAG(result) = infinity

3. If the imaginary part of the result is too small a number, the following message is issued and the imaginary part of the result is set to 0.0.

   CDSIN: Imaginary part underflow

Description

The CDCOS subroutine calculates the complex, double-precision, D-floating-point cosine of the complex, double-precision, D-floating-point angle given in radians as the argument. That is:

CDCOS(z) = cos(z)
z = location of input value
r = location of result

Routines Called

CDCOS calls the DSIN, DCOS, DEXP, DLOG, and MTHERR routines.

Type of Argument

CDCOS is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, D-floating-point value, the real part of which must be less than $2^{31} \cdot \pi - \pi/2$.

Type of Result

The result returned is a complex, double-precision, D-floating-point value; it may be any such value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-200.00 through 200.00 real -10.000 through 10.000 imaginary
MRE:	9.89×10^{-19} (59.8 bits) real 9.98×10^{-19} (59.8 bits) imaginary
RMS:	2.25×10^{-19} (61.9 bits) real 2.03×10^{-19} (62.1 bits) imaginary

	-2	-1	0	+1	+2	
LSB error distribution:	3%	24%	50%	21%	2%	real
	1%	21%	55%	21%	1%	imaginary

Algorithm Used

CDCOS(z) is calculated as follows.

Let z = x+i•y

If $|x| > 2^{31} \cdot \pi - \pi/2$
CDCOS(z) = (0.0,0.0)

If $|y| > 88.029692$, calculation proceeds as follows.

For the real part of the result:
Let $t = |\cos(x)| \neq 0$

If $\log_e(t)+|y| > 88.722839$
x = ± infinity
$(88.722839 = 88.029692+\log_e(2))$

For the imaginary part of the result:
Let $t = |\sin(x)|$

If t = 0.0
y = 0.0

If $\log_e(t)+|y| > 88.722839$
y = ± infinity

Otherwise
$CDCOS(z) = \cos(x) \cdot \cosh(y) - i \cdot \sin(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than $2^{31} \cdot \pi - \pi/2$, the following message is issued and the result is set to (0.0,0.0).

   CDCOS: ABS(REAL(arg)) too large; result = zero

2. If $|y|+\log_e(|\cos(x)|) > 88.722839$, the real part overflows. If $|y|+\log_e(|\sin(x)|) > 88.722839$, the imaginary part overflows. If either part overflows, one of the following messages is issued and the relevant part of the result is set to ± machine infinity.

   CDCOS: ABS(IMAG(arg)) too large; REAL(result) = infinity
   CDCOS: ABS(IMAG(arg)) too large; IMAG(result) = infinity

3. If the imaginary part of the result is too small a number, the following message is issued and the imaginary part of the result is set to 0.0

   CDCOS: Imaginary part underflow

Description
The CGSIN subroutine calculates the complex, double-precision, G-floating-point sine of the complex, double-precision, G-floating-point angle given in radians as the argument. That is,

CGSIN(z,r) = sin(z)
z = location of input value
r = location of result

Routines Called
CGSIN calls the GSIN, GCOS, GEXP, GLOG, and MTHERR routines.

Type of Argument
CGSIN is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, G-floating-point value, the real part of which must be less than $2^{29} \cdot \pi - \pi/2$.

Type of Result
The result returned is a complex, double-precision, G-floating-point value; it may be any such value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-200.00 through 200.00 real -10.000 through 10.000 imaginary
MRE:	7.35×10^{-18} (56.9 bits) real 7.01×10^{-18} (57.0 bits) imaginary
RMS:	1.76×10^{-18} (59.0 bits) real 1.61×10^{-18} (59.1 bits) imaginary

	-2	-1	0	+1	+2	
LSB error distribution:	2%	22%	51%	23%	2%	real
	1%	20%	55%	22%	2%	imaginary

Algorithm Used

CGSIN(z) is calculated as follows.

Let $z = x+i \cdot y$

If $|x| > 2^{29} \cdot \pi - \pi/2$
$CGSIN(z) = (0.0,0.0)$

If $|y| > 709.089565712824$, calculation proceeds as follows.

For the real part of the result:
Let $t = |\sin(x)|$

If $t = 0.0$
$x = 0.0$

If $\log_e(t)+|y| > 709.782712893384$
$x = \pm$machine infinity
$(709.782712893384 = 709.089565712824+\log_e(2))$

For the imaginary part of the result:
Let $t = |\cos(x)| \neq 0.0$

If $\log_e(t)+|y| > 709.782712893384$
$y = \pm$machine infinity

Otherwise
$CGSIN(z) = \sin(x) \cdot \cosh(x)+i \cdot \cos(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than $2^{29} \cdot \pi - \pi/2$, the following message is issued and the result is set to (0.0,0.0).

   CGSIN: ABS(REAL(arg)) too large; result = zero

2. If $|y|+\log_e(|\sin(x)|) > 709.782712893384$, the real part of the result will overflow. If $|y|+\log_e(|\cos(x)|) > 709.782712893384$, the imaginary part of the result will overflow. Any overflowed result is set to ±machine infinity and one of the following messages is issued.

   CGSIN: ABS(IMAG(arg)) too large; REAL(result) = infinity

   CGSIN: AGS(IMAG(arg)) too large; IMAG(result) = infinity

3. If the imaginary part of the result underflows, the following message is issued and the imaginary part of the result is set to 0.0.

   CGSIN: Imaginary part underflow

Description

The CGCOS subroutine calculates the complex, double-precision, G-floating-point cosine of the complex, double-precision, G-floating-point angle given in radians as the argument. That is:

CGCOS(z,r) = cos(z)
z = location of input value
r = location of result

Routines Called

CGCOS calls the GSIN, GCOS, GEXP, GLOG, and MTHERR routines.

Type of Argument

CGCOS is a subroutine that is called with two arguments. Both arguments must be two-element, double-precision vectors. The first vector (z) contains the input value; the second vector (r) will contain the result. The real part of the input value must be stored in the first element of z; the imaginary part must be stored in the second element of z. The input value must be a complex, double-precision, G-floating-point value, the real part of which must be less than $2^{29} \cdot \pi - \pi/2$.

Type of Result

The result returned is a complex, double-precision, G-floating-point value; it may be any such value. It is returned in the second vector (r) supplied in the call. The real part of the result is returned in the first element of r; the imaginary part is returned in the second element of r.

Accuracy of Result

test interval:	-200.00 through 200.00 real				
	-10.000 through 10.000 imaginary				
MRE:	8.31×10^{-18} (56.7 bits) real				
	7.00×10^{-18} (57.0 bits) imaginary				
RMS:	1.83×10^{-18} (58.9 bits) real				
	1.53×10^{-18} (59.2 bits) imaginary				
	-2	-1	0	+1	+2
LSB error distribution:	2%	20%	50%	25%	3% real
	2%	20%	58%	20%	1% imaginary

Algorithm Used

CGCOS(z) is calculated as follows.

Let $z = x+i \cdot y$

If $|x|>2^{29} \cdot \pi-\pi/2$
CGCOS(z) = (0.0,0.0)

If $|y| > 709.089565712824$, calculation proceeds as follows.

For the real part of the result:
Let $t = |\cos(x)| \neq 0.0$

If $\log_e(t)+|y| > 709.782712893384$
$x = \pm$machine infinity
$(709.782712893384 = 709.089565712824+\log_e(2))$

For the imaginary part of the result:
Let $t = |\sin(x)|$

If $t = 0.0$
$y = 0.0$

If $\log_e(t)+|y| > 709.782712893384$
$y = \pm$machine infinity

Otherwise
$CGCOS(z) = \cos(x) \cdot \cosh(y)-i \cdot \sin(x) \cdot \sinh(y)$

Error Conditions

1. If the absolute value of the real part of the argument is greater than $2^{29} \cdot \pi-\pi/2$, the following message is issued and the result is set to (0.0,0.0).

   CGCOS: ABS(REAL(arg)) too large; result = zero

2. If $|y|+\log_e(|\cos(x)|) > 709.782712893384$, the real part of the result will overflow. If $|y|+\log_e(|\sin(x)|) > 709.782712893384$, the imaginary part of the result will overflow. Any overflowed result is set to $\pm$machine infinity and one of the following messages is issued.

   CGCOS: ABS(IMAG(arg)) too large; REAL(result) = infinity

   CGCOS: ABS(IMAG(arg)) too large; IMAG(result) = infinity

3. If the imaginary part of the result underflows, the following message is issued and the imaginary part is set to 0.0.

   CGCOS: Imaginary part underflow

Description

The TAN routine calculates the single-precision, floating-point tangent of the single-precision, floating-point angle given in radians as the argument. That is:

$TAN(x) = tan(x)$

Routines Called

TAN calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value less than or equal to $2^{26} \cdot \pi/2$.

Type of Result

The result returned is a single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	−10.000 through 201.06				
MRE:	2.35×10^{-8} (25.3 bits)				
RMS:	5.28×10^{-9} (27.5 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	13%	70%	16%	0%

Algorithm Used

TAN(x) is calculated as follows.

If $|x| > 2^{26} \cdot \pi/2$
$TAN(x) = 0.0$

Otherwise, the identities:
$\tan(\pi/2.0-g) = 1.0/\tan(g)$
$\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
$\tan(-x) = -\tan(x)$
are used to reduce TAN(x) to a problem with
$-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:
$x = n \cdot \pi/4.0+f$ where $0.0 \leq f \leq \pi/4.0$

If $f < 2^{-14}$
$\tan(f) = f$

Otherwise

$\tan(f) = f \cdot R(f^2)$

$R(f^2) = (p0+f^2 \cdot (p1+f^2 \cdot p2))/(q0+f^2 \cdot (q1+f^2))$

$p0 = 62.604$

$p1 = -6.9716$

$p2 = 6.7309$

$q0 = p0$

$q1 = -27.839$

Then, TAN(x) can be derived if L is an integer and n has the values shown in the following table.

Deriving TAN(x)

Value of n	Low-order two bits of n	TAN(x)
4L	00	sgn(x)•tan(f)
4L+1	01	sgn(x)•(1/tan(f))
4L+2	10	sgn(x)•(-1/tan(f))
4L+3	11	sgn(x)•-tan(f)

Reference

Coefficients are derived flom those given in Cody and Waite, *Software Manual for Elementary Functions* (Englewood Cliffs, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

If the absolute value of the argument is greater than $2^{26} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

```
TAN: ABS(arg) too large; result = zero
```

Description
The COTAN routine calculates the single-precision, floating-point cotangent of the single-precision, floating-point angle given in radians as the argument. That is:

COTAN(x) = cot(x)

Routines Called
COTAN calls the MTHERR routine.

Type of Argument
The argument must be a single-precision, floating-point value less than or equal to $2^{26} \cdot \pi/2$ and greater than $2^{-126} \cdot (1/2+2^{-27})$.

Type of Result
The result returned is a single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	2.42×10^{-8} (25.3 bits)				
RMS:	5.29×10^{-9} (27.5 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	18%	66%	16%	0%

Algorithm Used
COTAN(x) is calculated as follows.

If $|x| > 2^{26} \cdot \pi/2$
COTAN(x) = 0.0

If $|x| < 2^{-126} \cdot (1/2+2^{-27})$
COTAN(x) = +machine infinity

Otherwise, the identities:
$\tan(\pi/2.0-g) = 1.0/\tan(g)$
$\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
$\tan(-x) = -\tan(x)$
$\cot(x) = 1.0/\tan(x)$
$\cot(-x) = -\cot(x)$
are used to reduce COTAN(x) to a problem with
$-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:

$x = n \cdot \pi/4.0 + f$ where $0.0 \le f \le \pi/4.0$

If $f < 2^{-14}$

$\tan(f) = f$

Otherwise

$\tan(f) = f \cdot R(f^2)$

$R(f^2) = (p0+f^2 \cdot (p1+f^2 \cdot p2))/(q0+f^2 \cdot (q1+f^2))$

p0 = 62.604
p1 = -6.9716
p2 = 6.7309
q0 = p0
q1 = -27.839

Then COTAN(x) can be derived if L is an integer and n has the value shown in the following table.

Deriving COTAN(x)

Value of n	Low-order two bits of n	COTAN(x)
4L	00	sgn(x)•(1/tan(f))
4L+1	01	sgn(x)•tan(f)
4L+2	10	sgn(x)•-tan(f)
4L+3	11	sgn(x)•-(1/tan(f))

Reference

Coefficients are derived from those given in Cody and Waite, *Software Manual for Elementary Functions* (Englewood Cliffs, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

1. If the absolute value of the argument is less than $2^{-126} \cdot (1/2+2^{-27})$, the following message is issued and the result is set to +machine infinity.

   COTAN: result overflow

2. If the absolute value of the argument is greater than $2^{26} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

   COTAN: ABS(arg) too large; result = zero

Description
The DTAN routine calculates the double-precision, D-floating-point tangent of the double-precision, D-floating-point angle given in radians as the argument. That is:

DTAN(x) = tan(x)

Routines Called
DTAN calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value less than or equal to $2^{31} \cdot \pi/2$.

Type of Result
The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result

test interval:	–10.000 through 201.06				
MRE:	9.60×10^{-19} (59.9 bits)				
RMS:	2.08×10^{-19} (62.1 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	1%	18%	55%	22%	3%

Algorithm Used
DTAN(x) is calculated as follows.

If $|x| > 2^{31} \cdot \pi/2$
DTAN(x) = 0.0

Otherwise, the identities:
$\tan(\pi/2.0-g) = 1.0/\tan(g)$
$\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
$\tan(-x) = -\tan(x)$
are used to reduce DTAN(x) to a problem with
$-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:
$x = n \cdot \pi/2.0+f$ where $-\pi/4.0 \leq f \leq \pi/4.0$

If $f < 2^{-31}$
$\tan(f) = f$

Otherwise

$\tan(f) = R(f)$

$R(f) = (((((xp4 \cdot g+xp3) \cdot g+xp2) \cdot g+xp1) \cdot g) \cdot f+f)/$
$((((q4 \cdot g+q3) \cdot g+q2) \cdot g+q1) \cdot g+1.0)$

$g = f \cdot f$

$xp1 = -.1372889460941120802$

$xp2 = .3925934686364577602 \cdot 10^{-2}$

$xp3 = -.2882482747560198194 \cdot 10^{-4}$

$xp4 = .2927308283322907641 \cdot 10^{-7}$

$q1 = -.4706222794274454135$

$q2 = .2746669449551304872 \cdot 10^{-1}$

$q3 = -.4030063705745304384 \cdot 10^{-3}$

$q4 = .1312960309685759549 \cdot 10^{-5}$

If n is even

DTAN(x) = tan(f)

If n is odd

DTAN(x) = -1/tan(f)

Reference

Coefficients are derived from those given in Cody and Waite, *Software Manual for Elementary Functions,* (Englewood Cliffs, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

If the absolute value of the argument is greater than $2^{31} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

DTAN: ABS(arg) too large; result = zero

Description
The DCOTAN routine calculates the double-precision, D-floating-point cotangent of the double-precision, D-floating-point angle given in radians as the argument. That is:

DCOTAN(x) = cot(x)

Routines Called
DCOTAN calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value less than or equal to $2^{31} \cdot \pi/2$ and greater than $2^{-127} \cdot (1+2^{-61})$.

Type of Result
The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result

test interval:	−10.000 through 201.06				
MRE:	9.09×10^{-19} (59.9 bits)				
RMS:	2.08×10^{-19} (62.1 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	2%	23%	55%	19%	1%

Algorithm Used
DCOTAN(x) is calculated as follows.

If $|x| > 2^{31} \cdot \pi/2$
  DCOTAN(x) = 0.0

If $|x| < 2^{-127} \cdot (1+2^{-61})$
  DCOTAN(x) = +machine infinity

Otherwise, the identities:
  $\tan(\pi/2.0-g) = 1.0/\tan(g)$
  $\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
  $\tan(-x) = -\tan(x)$
  $\cot(x) = 1.0/\tan(x)$
  $\cot(-x) = -\cot(x)$
are used to reduce DCOTAN(x) to a problem with
  $-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:
  $x = n \cdot \pi/2.0+f$ where $-\pi/4.0 \leq f \leq \pi/4.0$

If $f < 2^{-31}$
  $\tan(f) = f$

Otherwise

$\tan(f) = R(f)$

$R(f) = (((((xp4 \cdot g+xp3) \cdot g+xp2) \cdot g+xp1) \cdot g) \cdot f+f)/$
$((((q4 \cdot g+q3) \cdot g+q2) \cdot g+q1) \cdot g+1.0)$

$g = f \cdot f$

$xp1 = -.1372889460941120802$

$xp2 = .3925934686364577602 \cdot 10^{-2}$

$xp3 = -.2882482747560198194 \cdot 10^{-4}$

$xp4 = .2927308283322907641 \cdot 10^{-7}$

$q1 = -.4706222794274454135$

$q2 = .2746669449551304872 \cdot 10^{-1}$

$q3 = -.4030063705745304384 \cdot 10^{-3}$

$q4 = .1312960309685759549 \cdot 10-5$

If n is even

DCOTAN(x) = 1/tan(f)

If n is odd

DCOTAN(x) = -tan(f)

References

Coefficients are derived from those given in Cody and Waite, *Software Manual for Elementary Functions,* (Englewood Cliffs, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

1. If the absolute value of the argument is greater than $2^{31} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

   DCOTAN: ABS(arg) too large; result = zero

2. If the absolute value of the argument is less than $2^{-127} \cdot (1+(2^{-61}))$, the following message is issued and the result is set to +machine infinity.

   DCOTAN: Result overflow

Description

The GTAN routine calculates the double-precision, G-floating-point tangent of the double-precision, G-floating-point angle given in radians as the argument. That is:

GTAN(x) = tan(x)

Routines Called

GTAN calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value less than or equal to $2^{29} \cdot \pi/2$.

Type of Result

The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	5.95×10^{-18} (57.2 bits)				
RMS:	1.43×10^{-18} (59.3 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	1%	20%	60%	18%	0%

Algorithm Used

GTAN(x) is calculated as follows.

If $|x| > 2^{29} \cdot \pi/2$
  GTAN(x) = 0.0

Otherwise, the identities:
  $\tan(\pi/2.0-g) = 1.0/\tan(g)$
  $\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
  $\tan(-x) = -\tan(x)$
are used to reduce GTAN(x) to a problem with
  $-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:
  $x = n \cdot \pi/2.0+f$ where $-\pi/4.0 \leq f \leq \pi/4.0$

If $f < 2^{-30}$
  $\tan(f) = f$

Otherwise

$\tan(f) = R(f)$

$R(f) = (((((xp4 \cdot g+xp3) \cdot g+xp2) \cdot g+xp1) \cdot g) \cdot f+f)/ ((((q4 \cdot g+q3) \cdot g+q2) \cdot g+q1) \cdot g+1.0)$

$g = f \cdot f$

$xp1 = -.1372889460941120802$

$xp2 = .3925934686364577602 \cdot 10^{-2}$

$xp3 = -.2882482747560198194 \cdot 10^{-4}$

$xp4 = .2927308283322907641 \cdot 10^{-7}$

$q1 = -.4706222794274454135$

$q2 = .2746669449551304872 \cdot 10^{-1}$

$q3 = -.4030063705745304384 \cdot 10^{-3}$

$q4 = .1312960309685759549 \cdot 10-5$

If n is even

$GTAN(x) = \tan(f)$

If n is odd

$GTAN(x) = -1/\tan(f)$

Reference

Coefficients are derived from those given in Cody and Waite, *Software Manual for the Elementary Functions*, (Englewood, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

If the absolute value of the argument is greater than $2^{29} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

GTAN: ABS(arg) too large; result = zero

Description
The GCOTAN routine calculates the double-precision, G-floating-point cotangent of the double-precision, G-floating-point angle given in radians as the argument. That is:

GCOTAN(x) = cot(x)

Routines Called
GCOTAN calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, G-floating-point value less than or equal to $2^{29} \cdot \pi/2$ and greater than $2^{-1023} \cdot (1+2^{-58})$.

Type of Result
The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result

test interval:	-10.000 through 201.06				
MRE:	6.46×10^{-18} (57.1 bits)				
RMS:	1.43×10^{-18} (59.3 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	1%	18%	60%	20%	1%

Algorithm Used
GCOTAN(x) is calculated as follows.

If $|x| > 2^{29} \cdot \pi/2$
GCOTAN(x) = 0.0

If $|x| < 2^{-1023} \cdot (1+2^{-58})$
GCOTAN(x) = +machine infinity

Otherwise, the identities
$\tan(\pi/2.0-g) = 1.0/\tan(g)$
$\tan(n \cdot \pi+h) = \tan(h)$ where $-\pi/2.0 < h \leq \pi/2.0$
$\tan(-x) = -\tan(x)$
$\cot(x) = 1.0/\tan(x)$
$\cot(-x) = -\cot(x)$
are used to reduce GCOTAN(x) to a problem with
$-\pi/2.0 < x \leq \pi/2.0$

Then n and f are defined so that:
$x = n \cdot \pi/2.0+f$ where $-\pi/4.0 \leq f \leq \pi/4.0$

If $f < 2^{-30}$
$\tan(f) = f$

Otherwise

$$\tan(f) = R(f)$$

$$R(f) = ((((xp4 \cdot g+xp3) \cdot g+xp2) \cdot g+xp1) \cdot g) \cdot f+f)/ ((((q4 \cdot g+q3) \cdot g+q2) \cdot g+q1) \cdot g+1.0)$$

$$g = f \cdot f$$

$$xp1 = -.1372889460941120802$$

$$xp2 = .3925934686364577602 \cdot 10^{-2}$$

$$xp3 = -.2882482747560198194 \cdot 10^{-4}$$

$$xp4 = .2927308283322907641 \cdot 10^{-7}$$

$$q1 = -.4706222794274454135$$

$$q2 = .2746669449551304872 \cdot 10^{-1}$$

$$q3 = -.4030063705745304384 \cdot 10^{-3}$$

$$q4 = .1312960309685759549 \cdot 10^{-5}$$

If n is even

$$GCOTAN(x) = 1/\tan(f)$$

If n is odd

$$GCOTAN(x) = -\tan(f)$$

Reference

Coefficients are derived from those given in Cody and Waite, *Software Manual for Elementary Functions,* (Englewood Cliffs, N.J.: Prentice Hall, 1980) for machines with 25–32 bit precision.

Error Conditions

1. If the absolute value of the argument is greater than $2^{29} \cdot \pi/2$, the following message is issued and the result is set to 0.0.

   GCOTAN:ABS(arg) to large; result = zero

2. If the absolute value of the argument is less than $2^{-1023} \cdot (1+2^{-58})$, the following message is issued and the result is set to + machine infinity.

   GCOTAN: Result overflow

Chapter 6
Inverse Trigonometric Routines

Description

The ASIN routine calculates, in radians, the single-precision, floating-point arc sine of its single-precision, floating-point argument. That is:

$ASIN(x) = \sin^{-1}(x)$

Routines Called

ASIN calls the SQRT and MTHERR routines.

Type of Argument

The argument must be a single-precision, floating-point value in the range -1.0 to 1.0.

Type of Result

The result returned is a single-precision, floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval:	0.00000 through 1.0000
MRE:	2.56×10^{-8} (25.2 bits)
RMS:	5.34×10^{-9} (27.5 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	10%	83%	7%	0%

Algorithm Used

ASIN(x) is calculated as follows.

Let $R(z) = z \cdot (p0+z \cdot (p1+z \cdot p2))/(q0+z \cdot (q1+z))$

$p0 = .564915737$
$p1 = -.409490163$
$p2 = 1.93496723 \times 10^{-2}$
$q0 = 3.38949412$
$q1 = -3.98220081$

Let $s = y+y \cdot R(z)$

Then, the following table gives the value of ASIN(x) depending on the values of x, z, and y.

range of x	z	y	ASIN(x)
-1.0 to -.5	$(1+x)/2$	$-2\sqrt{z}$	$-(\pi/2+s)$
-.5 to 0.0	x^2	$-x$	$-s$
0.0 to .5	x^2	x	s
.5 to 1.0	$(1-x)/2$	$-2\sqrt{z}$	$\pi/2+s$

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to +machine infinity.

ASIN: ABS(arg) greater than 1.0; result = +infinity

Description

The ACOS routine calculates, in radians, the single-precision, floating-point arc cosine of its single-precision, floating-point argument. That is:

$$ACOS(x) = \cos^{-1}(x)$$

Routines Called

ACOS calls the SQRT and MTHERR routines.

Type of Argument

The argument must be a single-precision, floating-point value in the range -1.0 to 1.0.

Type of Result

The result returned is a single-precision, floating-point value in the range 0.0 to π.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	$1.55\text{x}10^{-8}$ (25.9 bits)				
RMS:	$3.76\text{x}10^{-9}$ (28.0 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	8%	83%	9%	0%

Algorithm Used

ACOS(x) is calculated as follows.

Let $R(z) = z\cdot(p0+z\cdot(p1+z\cdot p2))/(q0+z\cdot(q1+z))$

$p0 = .564915737$
$p1 = -.409490163$
$p2 = .93496723\text{x}10^{-2}$
$q0 = 3.38949412$
$q1 = -3.98220081$

Let $s = y+y\cdot R(z)$

Then, the following table gives the values of ACOS(x) depending on the values of x, z, and y.

range of x	z	y	ACOS(x)
-1.0 to -.5	(1+x)/2	$-2\sqrt{z}$	π+s
-.5 to 0.0	x^2	−x	π/2+s
0.0 to .5	x^2	x	π/2 −s
.5 to 1.0	(1−x)/2	$-2\sqrt{z}$	−s

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to +machine infinity.

ACOS: ABS(arg) greater than 1.0; result = +infinity

Description
The DASIN routine calculates, in radians, the double-precision, D-floating-point arc sine of its double-precision, D-floating-point argument. That is:

$$DASIN(x) = \sin^{-1}(x)$$

Routines Called
DASIN calls the DSQRT and MTHERR routines.

Type of Argument
The argument must be a double-precision, D-floating-point value in the range -1.0 to 1.0.

Type of Result
The result returned is a double-precision, D-floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	8.96×10^{-19} (60.0 bits)				
RMS:	1.88×10^{-19} (62.2 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	1%	25%	69%	5%	0%

Algorithm Used
DASIN(x) is calculated as follows.

$$\text{Let } R(g) = (g\cdot(rp1+g\cdot(rp2+g\cdot(rp3+g\cdot(rp4+g\cdot rp5)))))/(q0+g\cdot(q1+g\cdot(q2+g\cdot(q3+g\cdot(q4+g)))))$$

$rp1 = -.27368494524164255994\times10^{2}$
$rp2 = .57208227877891731407\times10^{2}$
$rp3 = -.39688862997504877339\times10^{2}$
$rp4 = .10152522233806463645\times10^{2}$
$rp5 = -.69674573447350646411$
$q0 = -.16421096714498560795\times10^{3}$
$q1 = .41714430248260412556\times10^{3}$
$q2 = -.38186303361750149284\times10^{3}$
$q3 = .15095270841030604719\times10^{3}$
$q4 = -.23823859153670238830\times10^{2}$

Let $s = y+y\cdot R(g)$
Then, the following table gives the values of DASIN(x) depending on the values of x, z, and y.

range of x	z	y	DASIN(x)
-1.0 to -.5	(1+x)/2	$-2\sqrt{z}$	$-(\pi/2+s)$
-.5 to 0.0	x^2	-x	-s
0.0 to .5	x^2	x	s
.5 to 1.0	(1-x)/2	$-2\sqrt{z}$	$\pi/2+s$

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to +machine infinity.

DASIN: ABS(arg) greater than 1.0; result = +infinity

Description

The DACOS routine calculates, in radians, the double-precision, D-floating-point arc cosine of its double-precision, D-floating-point argument. That is:

$$DACOS(x) = \cos^{-1}(x)$$

Routines Called

DACOS calls the DSQRT and MTHERR routines.

Type of Argument

The argument must be a double-precision, D-floating-point value in the range -1.0 to 1.0.

Type of Result

The result returned is a double-precision, D-floating-point value in the range 0.0 to π.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	4.48×10^{-19} (61.0 bits)				
RMS:	1.25×10^{-19} (62.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	19%	75%	6%	0%

Algorithm Used

DACOS(x) is calculated as follows.

Let $R(g) = (g\cdot(rp1+g\cdot(rp2+g\cdot(rp3+g\cdot(rp4+g\cdot rp5)))))/(q0+g\cdot(q1+g\cdot(q2+g\cdot(q3+g\cdot(q4+g)))))$

$rp1 = -.27368494524164255994\times10^{2}$
$rp2 = .57208227877891731407\times10^{2}$
$rp3 = -.39688862997504877339\times10^{2}$
$rp4 = .10152522233806463645\times10^{2}$
$rp5 = -.69674573447350646411$
$q0 = -.16421096714498560795\times10^{3}$
$q1 = .41714430248260412556\times10^{3}$
$q2 = -.38186303361750149284\times10^{3}$
$q3 = .15095270841030604719\times10^{3}$
$q4 = -.23823859153670238830\times10^{2}$

Let $s = y+y\cdot R(g)$

Then, the following table gives the values of DACOS(x) depending on the values of x, z, and y.

range of x	z	y	ACOS(x)
-1.0 to -.5	(1+x)/2	$-2\sqrt{z}$	π+s
-.5 to 0.0	x^2	-x	π/2+s
0.0 to .5	x^2	x	π/2-s
.5 to 1.0	(1-x)/2	$-2\sqrt{z}$	-s

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to +machine infinity.

DACOS: ABS(arg) greater than 1.0; result = +infinity

Description

The GASIN routine calculates, in radians, the double-precision, G-floating-point arc sine of its double-precision, G-floating-point argument. That is:

$GASIN(x) = \sin^{-1}(x)$

Routines Called

GASIN calls the GSQRT and MTHERR routines.

Type of Argument

The argument must be a double-precision, G-floating-point value in the range -1.0 to 1.0.

Type of Result

The result returned is a double-precision, G-floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	6.69×10^{-18} (57.1 bits)				
RMS:	1.54×10^{-18} (59.2 bits				
LSB error distribution:	-2	-1	0	+1	+2
	1%	26%	72%	2%	0%

Algorithm Used

GASIN(x) is calculated as follows.

Let $R(g) = (g \cdot (rp1+g \cdot (rp2+g \cdot (rp3+g \cdot (rp4+g \cdot rp5))))) / (q0+g \cdot (q1+g \cdot (q2+g \cdot (q3+g \cdot (q4+g)))))$

$rp1 = -.27368494524164255994 \times 10^{2}$
$rp2 = .57208227877891731407 \times 10^{2}$
$rp3 = -.39688862997504877339 \times 10^{2}$
$rp4 = .10152522233806463645 \times 10^{2}$
$rp5 = -.69674573447350646411$
$q0 = -.16421096714498560795 \times 10^{3}$
$q1 = .41714430248260412556 \times 10^{3}$
$q2 = -.38186303361750149284 \times 10^{3}$
$q3 = .15095270841030604719 \times 10^{3}$
$q4 = -.23823859153670238830 \times 10^{2}$

Let $s = y+y \cdot R(g)$

Then, the following table gives the value of GASIN(x) depending on the values of x, z, and y.

range of x	z	y	GASIN(x)
-1.0 to -.5	$(1+x)/2$	$-2\sqrt{z}$	$-(\pi/2+s)$
-.5 to 0.0	x^2	$-x$	$-s$
0.0 to .5	x^2	x	s
.5 to 1.0	$(1-x)/2$	$-2\sqrt{z}$	$\pi/2+s$

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to +machine infinity.

GASIN: ABS(arg) greater than 1.0; result = +infinity

Description

The GACOS routine calculates, in radians, the double-precision, G-floating-point arc cosine of its double-precision, G-floating-point argument. That is:

$$GACOS(x) = \cos^{-1}(x)$$

Routines Called

GACOS calls the GSQRT and MTHERR routines.

Type of Argument

The argument must be a double-precision, G-floating-point value in the range -1.0 to 1.0.

Type of Result

The result returned is a double-precision, G-floating-point value in the range 0.0 to π.

Accuracy of Result

test interval:	0.00000 through 1.0000				
MRE:	4.18×10^{-18} (57.7 bits)				
RMS:	1.03×10^{-18} (59.8 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	14%	72%	15%	0%

Algorithm Used

GACOS(x) is calculated as follows.

$$\text{Let } R(g) = (g \cdot (rp1+g \cdot (rp2+g \cdot (rp3+g \cdot (rp4+g \cdot rp5)))))/(q0+g \cdot (q1+g \cdot (q2+g \cdot (q3+g \cdot (q4+g)))))$$

$rp1 = -.27368494524164255994 \times 10^2$
$rp2 = .57208227877891731407 \times 10^2$
$rp3 = -.39688862997504877339 \times 10^2$
$rp4 = .10152522233806463645 \times 10^2$
$rp5 = -.69674573447350646411$
$q0 = -.16421096714498560795 \times 10^3$
$q1 = .41714430248260412556 \times 10^3$
$q2 = -.38186303361750149284 \times 10^3$
$q3 = .15095270841030604719 \times 10^3$
$q4 = -.23823859153670238830 \times 10^2$

Let $s = y+y \cdot R(g)$

Then the following table gives the value of GACOS(x) depending on the values of x, z, and y.

range of x	z	y	GACOS(x)
-1.0 to -.5	(1+x)/2	$-2\sqrt{z}$	π+s
-.5 to 0.0	x^2	-x	π/2+s
0.0 to .5	x^2	x	π/2-s
.5 to 1.0	(1-x)/2	$-2\sqrt{z}$	-s

Error Conditions

If the absolute value of the argument is greater than 1.0, the following message is issued and the result is set to machine infinity.

GACOS: ABS(arg) greater than 1.0; result = +infinity

Description
The ATAN routine calculates, in radians, the single-precision, floating-point arc tangent of its single-precision, floating-point argument. That is:

$$ATAN(x) = \tan^{-1}(x)$$

Routines Called
None

Type of Argument
The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval:	–80.000 through 80.000				
MRE:	8.07×10^{-9} (26.9 bits)				
RMS:	2.99×10^{-9} (28.3 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	1%	98%	1%	0%

Algorithm Used
ATAN(x) is calculated as follows.

If $x < 0.0$
$ATAN(x) = -ATAN(|x|)$

If $x > 0.0$
$ATAN(x) = \tan^{-1}(XHI) + \tan^{-1}(z)$
$z = (x-XHI)/(1+x \cdot XHI)$
XHI is chosen so that
$|z| \le \tan(\pi/32)$

$\tan^{-1}(XHI)$ is found by table lookup. It is stored as ATANHI and ATANLO to provide guard bits for improved accuracy.
$\tan^{-1}(z)$ is evaluated by means of a polynomial approximation (see "Reference" below).

If $x < \tan(\pi/32)$
$z = x$
$ATAN(x) = \tan^{-1}(z)$

If $x > 1/\tan(\pi/32)$
$z = 1/x$
$ATAN(x) = \pi/2 - \tan^{-1}(z)$

If $\tan(\pi/32) < x < 1/\tan(\pi/32)$
an appropriate XHI is obtained from a table. The table contains values for XHI for various ranges of x.

Reference

The polynomial approximation used in the algorithm is formula #4901 from Hart et al., *Computer Approximations*, (New York, N.Y.: John Wiley and Sons, 1968).

Error Conditions

None

Description
The ATAN2 routine calculates, in radians, the single-precision, floating-point polar angle for the two single-precision, floating-point coordinates of a point in the x–y plane that are included as the arguments. That is:

$$ATAN2(y,x) = \tan^{-1}(y/x)$$

Routines Called
ATAN2 calls the ATAN and MTHERR routines.

Type of Arguments
The arguments must be single-precision, floating-point values; they can be any such values provided both arguments are not zero.

Type of Result
The result returned is a single-precision, floating-point value in the range $-\pi$ to π.

Accuracy of Result

test interval:	–80.000 through 1.0000 for x –80.000 through 1.0000 for y				
MRE:	1.46×10^{-8} (26.0 bits)				
RMS:	3.08×10^{-9} (28.3 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	1%	98%	1%	0%

Algorithm Used
ATAN2 (y,x) is calculated as follows.

Let $u = |y|$ and
$v = |x|$ and compute $\tan^{-1}(u,v)$

Then find ATAN2(y,x) based on the signs of y and x as follows.

x	**y**	**ATAN2(y,x)**
+	+	$\tan^{-1}(u,v)$
+	–	$-\tan^{-1}(u,v)$
–	+	$-(\tan^{-1}(u,v)-\pi)$
–	–	$\tan^{-1}(u,v)-\pi$

The reduced argument for ATAN2 is:

z = (u/v-XHI)/(1+u/v•XHI)

This is rewritten as:

z = (u-v•XHI)/(v+u•XHI)

The numerator is calculated to be:

u-v•XHI = u-VHI•XHI-VLO•XHI

v = VHI+VLO

VHI has, at most, 27 significant bits

VLO has, at most, 35 significant bits

XHI is tabulated with, at most, 13 significant bits

This guarantees that the numerator of z is calculated exactly.

Error Conditions

1. If both arguments are 0.0, the following message is issued and the result is set to 0.0.

   ATAN2: Both arguments are zero, result = zero

2. If y/x underflows and x is greater than 0.0, the following message is issued and the result is set to 0.0.

   ATAN2: Result underflow

Description

The DATAN routine calculates, in radians, the double-precision D-floating-point arc tangent of its double-precision, D-floating-point argument. That is:

$$DATAN(x) = \tan^{-1}(x)$$

Routines Called

None

Type of Argument

The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result

The result returned is a double-precision, D-floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval:	–80.000 through 80.000				
MRE:	3.40×10^{-19} (61.3 bits)				
RMS:	9.37×10^{-20} (63.2 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	1%	94%	5%	0%

Algorithm Used

DATAN(x) is calculated as follows.

If $x < 0.0$
 $DATAN(x) = -DATAN(|x|)$

If $x > 0.0$
 $DATAN(x) = \tan^{-1}(XHI)+\tan^{-1}(z)$
 $z = (x-XHI)/(1+x \cdot XHI)$
 XHI is chosen so that
 $|z| \leq \tan(\pi/32)$

$\tan^{-1}(XHI)$ is found by table lookup. It is stored as ATANHI and ATANLO to provide guard bits for improved accuracy.
$\tan^{-1}(z)$ is evaluated by means of a polynomial approximation (see"Reference" below).

If $x < \tan(\pi/32)$
 $z = x$
 $DATAN(x) = \tan^{-1}(z)$

If $x > 1/\tan(\pi/32)$
 $z = 1/x$
 $DATAN(x) = \pi/2-\tan^{-1}(z)$

If $\tan(\pi/32) < x < 1/\tan(\pi/32)$
 an appropriate XHI is obtained from a table. The table contains values for XHI for various ranges of x.

Reference

The polynomial approximation used in the algorithm is formula #4904 from Hart et al., *Computer Approximations*, (New York, N.Y.: John Wiley and Sons, 1968).

Error Conditions

None

Description

The DATAN2 routine calculates, in radians, the double-precision, D-floating-point polar angle for the two double-precision, D-floating-point coordinates of a point in the x–y plane that are included as the arguments. That is:

$$DATAN2(y,x) = \tan^{-1}(y/x)$$

Routines Called

DATAN2 calls the DATAN and MTHERR routines.

Type of Arguments

The arguments must be double-precision, D-floating-point values; they can be any such values provided both arguments are not zero.

Type of Result

The result returned is a double-precision, D-floating-point value in the range $-\pi$ to π.

Accuracy of Result

test interval:	–80.000 through 1.0000 for x –80.000 through 1.0000 for y
MRE:	5.27×10^{-19} (60.7 bits)
RMS:	9.09×10^{-9} (63.3 bits)

LSB error distribution:	–2	–1	0	+1	+2
	0%	1%	97%	2%	0%

Algorithm Used

DATAN2(y,x) is calculated as follows.

Let $u = |y|$ and
$v = |x|$ and compute $\tan^{-1}(u/v)$

Then find DATAN2(y,x) based on the signs of y and x as follows.

x	y	DATAN2(y,x)
+	+	$\tan^{-1}(u/v)$
+	–	$-\tan^{-1}(u/v)$
–	+	$-(\tan^{-1}(u/v)-\pi)$
–	–	$\tan^{-1}(u/v)-\pi$

The reduced argument for DATAN2 is:

z = (u/v–XHI)/(1+u/v•XHI)

This is rewritten as:

z = (u–v•XHI)/(v+u•XHI)

The numerator is calculated to be:

u–v•XHI = u–VHI•XHI–VLO•XHI

v = VHI+VLO

VHI has, at most, 27 significant bits

VLO has, at most, 35 significant bits

XHI is tabulated with, at most, 13 significant bits

This guarantees that the numerator of z is calculated exactly.

Error Conditions

1. If both arguments are 0.0, the following message is issued and the result is set to 0.0.

   DATAN2: Both arguments are zero, result = zero

2. If y/x underflows and x is greater than 0.0, the following message is issued and the result is set to 0.0.

   DATAN2: Result underflow

Description
The GATAN routine calculates, in radians, the double-precision, G-floating-point arc tangent of its double-precision, G-floating-point argument. That is:

GATAN(x) = $\tan^{-1}(x)$

Routines Called
None

Type of Argument
The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, G-floating-point value in the range $-\pi/2$ to $\pi/2$.

Accuracy of Result

test interval: -80.000 through 80.000

MRE: 2.04×10^{-18} (58.8 bits)

RMS: 7.03×10^{-19} (60.3 bits)

LSB error distribution:	−2	−1	0	+1	+2
	0%	1%	97%	2%	0%

Algorithm Used
GATAN(x) is calculated as follows.

If $x < 0.0$
GATAN(x) = −GATAN(|x|)

If $x > 0.0$
GATAN(x) = $\tan^{-1}$(XHI)+$\tan^{-1}$(z)
z = (x−XHI)/(1+x•XHI)
XHI is chosen so that
$|z| \le \tan(\pi/32)$

$\tan^{-1}$(XHI) is found by table lookup. It is stored as ATANHI and ATANLO to provide guard bits for improved accuracy.
$\tan^{-1}(z)$ is evaluated by means of a polynomial approximation (see "Reference" below).

If $x < \tan(\pi/32)$
z = x
GATAN(x) = $\tan^{-1}(z)$

If $x > \tan(\pi/32)$
z = 1/x
GATAN(x) = $\pi/2-\tan^{-1}(z)$

If $\tan(\pi/32) < x < 1/\tan(\pi/32)$
an appropriate XHI is obtained from a table. The table contains values for XHI for various ranges of x.

Reference

The polynomial approximation used in the algorithm is formula 4904 from Hart et al., *Computer Approximations*, (New York, N.Y.: John Wiley and Sons, 1968).

Error Conditions

None

Description

The GATAN2 routine calculates, in radians, the double-precision, G-floating-point polar angle for the two double-precision, G-floating-point coordinates of a point in the x–y plane that are included as the arguments. That is:

$$GATAN2(y,x) = \tan^{-1}(y/x)$$

Routines Called

GATAN2 calls the GATAN and MTHERR routines.

Type of Arguments

The arguments must be double-precision, G-floating-point values; they can be any such values provided both arguments are not zero.

Type of Result

The result returned is a double-precision, G-floating-point value in the range $-\pi$ to π.

Accuracy of Result

test interval:	–80.000 through 1.0000 for x –80.000 through 1.0000 for y				
MRE:	3.28×10^{-18} (58.1 bits)				
RMS:	7.15×10^{-19} (60.3 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	1%	98%	2%	0%

Algorithm Used

GATAN2(y,x) is calculated as follows.

Let $u = |y|$ and
$v = |x|$ and compute $\tan^{-1}(u/v)$

Then find GATAN2(y,x) based on the signs of y and x as follows.

x	**y**	**GATAN2(y,x)**
+	+	$\tan^{-1}(u/v)$
+	–	$-\tan^{-1}(u/v)$
–	+	$-(\tan^{-1}(u/v)-\pi)$
–	–	$\tan^{-1}(u/v)-\pi$

The reduced argument for GATAN2 is:

z = (u/v–XHI)/(1+u/v•XHI)

This is rewritten as:

z = (u–v•XHI)/(v+u•XHI)

The numerator is calculated to be:

u–v•XHI = u–VHI•XHI–VLO•XHI

v = VHI+VLO

VHI has, at most, 27 significant bits

VLO has, at most, 35 significant bits

XHI is tabulated with, at most, 13 significant bits

This guarantees that the numerator of z is calculated exactly.

Error Conditions

1. If both arguments are 0.0, the following message is issued and the result is set to 0.0.

   GATAN2: Both arguments are zero, result = zero

2. If y/x underflows and x is greater than 0.0, the following message is issued and the result is set to 0.0.

   GATAN2: Result underflow

Chapter 7
Hyperbolic Routines

Description

The SINH routine calculates the single-precision, floating-point hyperbolic sine of its single-precision, floating-point argument. That is:

SINH(x) = sinh(x)

Routines Called

SINH calls the EXP and MTHERR routines.

Type of Argument

The argument must be a single-precision, floating-point value in the range –88.722 to 88.722.

Type of Result

The result returned is a single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	0.00000 through 88.721				
MRE:	2.61×10^{-8} (25.2 bits)				
RMS:	4.24×10^{-9} (27.8 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	4%	85%	11%	0%

Algorithm Used

SINH(x) is calculated as follows.

The table below gives the value of SINH(x) depending upon the range of values for |x|.

range of |x|	SINH(x)
0.0 to 2^{-13}	x
2^{-13} to 1.0	$x \cdot p4(x^2)$
1.0 to 9.7 = $14 \cdot \log_e(2)$	$(e^x - e^{-x})/2 \cdot sgn(x)$
9.7 to 88.03 = $127 \cdot \log_e(2)$	$e^x/2 \cdot sgn(x)$
88.03 to 88.722 = $128 \cdot \log_e(2)$	$e^{x-\log_e(2)} \cdot sgn(x)$
88.722 to infinity	infinity $\cdot$ sgn(x)

If $z = x^2$

$p4(z) = 1+z \cdot (c1+z \cdot (c2+z \cdot (c3+c4 \cdot z)))$

$c1 = 1.666666643 \times 10^{-1}$

$c2 = 8.333352593 \times 10^{-3}$

$c3 = 1.983581245 \times 10^{-4}$

$c4 = 2.818523951 \times 10^{-6}$

Error Conditions

If the absolute value of the argument is greater than 88.722, the following message is issued and the result is set to ± machine infinity using the sign of the argument.

SINH: Result overflow

COSH

Description

The COSH routine calculates the single-precision, floating-point hyperbolic cosine of its single-precision, floating-point argument. That is:

COSH(x) = cosh(x)

Routines Called

COSH calls the EXP and MTHERR routines.

Type of Argument

The argument must be a single-precision, floating-point value in the range –88.722 to 88.722.

Type of Result

The result returned is a single-precision, floating-point value greater than or equal to 1.0.

Accuracy of Result

test interval:	0.00000 through 88.721				
MRE:	2.12×10^{-8} (25.5 bits)				
RMS:	4.49×10^{-9} (27.7 bits)				
LSB error distribution:	–2	–1	0	+1	+2
	0%	4%	82%	14%	0%

Algorithm Used

COSH(x) is calculated as follows.

The table below gives the value of COSH(x) depending upon the range of values for |x|.

range of |x|	COSH(x)
0.0 to 2^{-14}	1.0
2^{-14} to 9.7 = $14 \cdot \log_e(2)$	$(e^x+e^{-x})/2$
9.7 to 88.03 = $127 \cdot \log_e(2)$	$e^x/2$
88.03 to 88.722 = $128 \cdot \log_e(2)$	$e^{x-\log_e(2)}$
88.722 to infinity	infinity

Error Conditions

If the absolute value of the argument is greater than 88.722, the following message is issued and the result is set to ± machine infinity using the sign of the argument.

COSH: Result overflow

Description
The DSINH routine calculates the double-precision, D-floating-point hyperbolic sine of its double-precision, D-floating-point argument. That is:

DSINH(x) = sinh(x)

Routines Called
DSINH calls the DEXP and MTHERR routines.

Type of Argument
The argument must be a double-precision, D-floating-point value in the range -88.722 to 88.722.

Type of Result
The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result

test interval:	0.00000 through 88.721				
MRE:	$6.82x10^{-8}$ (60.3 bits)				
RMS:	$1.27x10^{-9}$ (62.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	6%	83%	11%	0%

Algorithm Used
DSINH(x) is calculated as follows.

The table below gives the value of DSINH(x) depending upon the range of values for |x|.

range of |x|	DSINH(x)
0.0 to 2^{-31}	x
2^{-31} to 1.0	$x+x \bullet R(x^2)$
1.0 to 22.0 = $32 \bullet \log_e(2)$	$(e^x-e^{-x})/2 \bullet sgn(x)$
22.0 to 88.03 = $127 \bullet \log_e(2)$	$e^x/2 \bullet sgn(x)$
88.03 to 88.722 = $128 \bullet \log_e(2)$	$e^{x-\log_e(2)} \bullet sgn(x)$
88.722 to infinity	infinity•sgn(x)

If $z = x^2$

$$R(z) = (rp0+z\cdot(rp1+z\cdot(rp2+z\cdot rp3)))/(q0+z\cdot(q1+z\cdot(q2+z)))$$

rp0 =.35181283430177117881x10^6

rp1 = .11563521196851768270x10^5

rp2 = .16375798202630751372x10^3

rp3 = .78966127417357099479

q0 = −.21108770058106271242x10^7

q1 = .36162723109421836460x10^5

q2 = −.27773523119650701667x10^3

Error Conditions

If the absolute value of the argument is greater than 88.722, the following message is issued and the result is set to ± machine infinity using the sign of the argument.

DSINH: Result overflow

Description

The DCOSH routine calculates the double-precision, D-floating-point hyperbolic cosine of its double-precision, D-floating-point argument. That is:

DCOSH(x) = cosh(x)

Routines Called

DCOSH calls the DEXP and MTHERR routines.

Type of Argument

The argument must be a double-precision, D-floating-point value in the range -88.722 to 88.722.

Type of Result

The result returned is a double-precision, D-floating-point value greater than or equal to 1.0.

Accuracy of Result

test interval:	0.00000 through 88.721
MRE:	5.90×10^{-19} (60.6 bits)
RMS:	1.34×10^{-19} (62.7 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	5%	81%	14%	0%

Algorithm Used

DCOSH(x) is calculated as follows.

The table below gives the value of DCOSH(x) depending upon the range of values for |x|.

range of |x|	DCOSH(x)
0.0 to 2^{-32}	1.0
2^{-32} to 22.0 = $32 \cdot \log_e(2)$	$(e^x+e^{-x})/2$
22.0 to 88.03 = $127 \cdot \log_e(2)$	$e^x/2$
88.03 to 88.722 = $128 \cdot \log_e(2)$	$e^{x-\log_e(2)}$
88.722 to infinity	infinity

Error Conditions

If the absolute value of the argument is greater than 88.722, the following message is issued and the result is set to ± machine infinity using the sign of the argument.

DCOSH: Result overflow

GSINH

Description

The GSINH routine calculates the double-precision, G-floating-point hyperbolic sine of its double-precision, G-floating-point argument. That is:

GSINH(x) = sinh(x)

Routines Called

GSINH calls the GEXP and MTHERR routines.

Type of Argument

The argument must be a double-precision, G-floating-point value in the range -709.782713 to 709.782713.

Type of Result

The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result

test interval:	0.00000 through 88.721				
MRE:	6.40×10^{-18} (57.1 bits)				
RMS:	9.44×10^{-19} (59.9 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	3%	87%	10%	0%

Algorithm Used

GSINH(x) is calculated as follows.

The table below gives the value of GSINH(x) depending upon the range of values for |x|.

range of |x|	GSINH(x)
0.0 to 2^{-30}	x
2^{-30} to 1.0	$x+x \cdot R(x^2)$
1.0 to 22.0 = $32 \cdot \log_e(2)$	$(e^x-e^{-x})/2 \cdot sgn(x)$
22.0 to 709.089565	$e^x/2 \cdot sgn(x)$
709.089565 to 709.782713	$e^{x-\log_e(2)} \cdot sgn(x)$
709.782713 to infinity	infinity $\cdot$ sgn(x)

If $z = x^2$

$$R(z) = (rp0+z \cdot (rp1+z \cdot (rp2+z \cdot rp3)))/(q0+z \cdot (q1+z \cdot (q2+z)))$$

$$rp0 = .35181283430177117881 \cdot 10^6$$
$$rp1 = .11563521196851768270 \cdot 10^5$$
$$rp2 = .16375798202630751372 \cdot 10^3$$
$$rp3 = .78966127417357099479$$
$$q0 = -.21108770058106271242 \cdot 10^7$$
$$q1 = .36162723109421836460 \cdot 10^5$$
$$q2 = -.27773523119650701667 \cdot 10^3$$

Error Conditions

If the absolute value of the argument is greater than 709.782713, the following message is issued and the result is set to ± machine infinity, using the sign of the argument.

GSINH: Result overflow

Description
The GCOSH routine calculates the double-precision, G-floating-point hyperbolic cosine of its double-precision, G-floating-point argument. That is:

GCOSH(x) = cosh(x)

Routines Called
GCOSH calls the GEXP and MTHERR routines.

Type of Argument
The argument must be a double-precision, G-floating-point value in the range -709.782713 to 709.782713.

Type of Result
The result returned is a double-precision, G-floating-point value greater than or equal to 1.0.

Accuracy of Result

test interval:	0.00000 through 88.721				
MRE:	4.84×10^{-18} (57.5 bits)				
RMS:	1.00×10^{-18} (59.8 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	3%	84%	13%	0%

Algorithm Used
GCOSH(x) is calculated as follows.

The table below gives the value of GCOSH(x) depending upon the range of values for |x|.

range of |x|	GCOSH(x)
0.0 to 2^{-30}	1.0
2^{-30} to 22.0 = $32 \cdot \log_e(2)$	$(e^x+e^{-x})/2$
22.0 to 709.089565	$e^x/2$
709.089565 to 709.782713	$e^{x-\log_e(2)}$
709.782713 to infinity	infinity

Error Conditions
If the absolute value of the argument is greater than 709.782713, the following message is issued and the result is set to ± machine infinity, using the sign of the argument.

GCOSH: Result overflow

Description
The TANH routine calculates the single-precision, floating-point hyperbolic tangent of its single-precision, floating-point argument. That is:

TANH(x) = tanh(x)

Routines Called
TANH calls the EXP routine.

Type of Argument
The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	0.00000 through 90.000				
MRE:	2.69×10^{-8} (25.1 bits)				
RMS:	5.53×10^{-9} (27.4 bits)				
LSB error distribution:	-2	-1	0	+1	+2
	0%	0%	79%	21%	0%

Algorithm Used
TANH(x) is calculated as follows.

The table below gives the value of TANH(x) depending upon the range of values for |x|.

range of |x|	TANH(x)
0.0 to 2^{-15}	x
2^{-15} to $\log_e(3)/2$	$x+x \cdot R(x^2)$
$\log_e(3)/2$ to 9.8479016	$(1-2/(e^{2 \cdot \lvert x \rvert}+1)) \cdot sgn(x)$
9.8479016 to infinity	$1.0 \cdot sgn(x)$

If $g = x^2$
$R(g) = g \cdot (a+b \cdot g)/(c+g)$
$a = -.823772813$
$b = -.383101067 \times 10^{-2}$
$c = 2.47131965$

Error Conditions
None

Description

The DTANH routine calculates the double-precision, D-floating-point hyperbolic tangent of its double-precision, D-floating-point argument. That is:

DTANH(x) = tanh(x)

Routines Called

DTANH calls the EXP routine.

Type of Argument

The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result

The result returned is a double-precision, D-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	0.00000 through 90.000				
MRE:	$7.17\text{x}10^{19}$ (60.3 bits)				
RMS:	$1.75\text{x}10^{19}$ (62.3 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	70%	30%	0%

Algorithm Used

DTANH(x) is calculated as follows.

The table below gives the value of DTANH(x) depending upon the range of values for |x|.

range of |x|	DTANH(x)
0.0 to $2^{-32} \cdot \sqrt{3}$	x
$2^{-32} \cdot \sqrt{3}$ to $\log_e(3)/2$	$x + x \cdot R(x^2)$
$\log_e(3)/2$ to 22.1807100	$(1 - 2/(e^{2 \cdot \|x\|} + 1)) \cdot \text{sgn}(x)$
22.1807100 to infinity	$1.0 \cdot \text{sgn}(x)$

If $g = x^2$

$R(g) = g \cdot (rp0 + g \cdot (rp1 + rp2 \cdot g)) / (q0 + g \cdot (q1 + g \cdot (q2 + g)))$

$rp0 = -.161341190239962281\text{x}10^4$

$rp1 = -.992259296722360833\text{x}10^2$

$rp2 = -.964374927772254698$

$q0 = .484023570719886887\text{x}10^4$

$q1 = .22337720718962312926\text{x}10^4$

$q2 = .112744743805349493\text{x}10^3$

Error Conditions

None

Description
The GTANH routine calculates the double-precision, G-floating-point hyperbolic tangent of its double-precision, G-floating-point argument. That is:

GTANH(x) = tanh(x)

Routines Called
GTANH calls the GEXP routine.

Type of Argument
The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, G-floating-point value in the range -1.0 to 1.0.

Accuracy of Result

test interval:	0.00000 through 90.000				
MRE:	6.44×10^{-18} (57.1 bits)				
RMS:	1.33×10^{-18} (59.4 bits)				
LSB error distribution:	−2	−1	0	+1	+2
	0%	0%	80%	20%	0%

Algorithm Used
GTANH(x) is calculated as follows.

The table below gives the value of GTANH(x) depending upon the range of values for |x|.

range of |x|	GTANH(X)
0.0 to $2^{-32} \cdot \sqrt{3}$	x
$2^{-32} \cdot \sqrt{3}$ to $\log_e(3)/2$	$x+x \cdot R(x^2)$
$\log_e(3)/2$ to 22.1807100	$(1-2/(e^{2 \cdot \|x\|}+1)) \cdot sgn(x)$
22.1807100 to infinity	$1.0 \cdot sgn(x)$

If $g = x^2$

$R(g) = g \cdot (rp0+g \cdot (rp1+rp2 \cdot g))/(q0+g \cdot (q1+g \cdot (q2+g)))$

$rp0 = -.161341190239962281 \times 10^4$

$rp1 = -.992259296722360833 \times 10^2$

$rp2 = -.964374927772254698$

$q0 = .484023570719886887 \times 10^4$

$q1 = .223377207189623129 26 \times 10^4$

$q2 = .112744743805349493 \times 10^3$

Error Conditions
None

Chapter 8
Random Number Generating Routines

Description

The RAN routine returns pseudo random numbers between 0.0 and 1.0, but not including 0.0 or 1.0. The period of the sequence is 2147483647; that is, the numbers repeat every 2147483647 calls.

RAN uses a pure multiplicative congruential random number generator with prime modulus. The seed value can be supplied by the system or supplied by a call to the SETRAN subroutine. (See SETRAN, p. 8-6).

Routines Called

RAN does not call any routines; but you can call the SETRAN subroutine to provide a seed value and the SAVRAN subroutine (see SAVRAN, p. 8–7) to determine the last seed used by RAN.

Type of Argument

The argument is a dummy value that is not used.

Type of Result

The result returned is a single-precision, floating-point value that is greater than 0.0 and less than 1.0.

Accuracy of Result

The independence of successive random numbers generated by multiplicative congruential methods can be measured by the spectral test. For this generator, with seed 630360016 and modulus 2147483647, the spectral test yields the following results.

n	mu(n)	bits
2	2.446	15
3	.4766	9
4	3.715	8
5	4.944	6
6	.8183	5

mu(n) measures how densely n-tuples of random numbers cover an n-dimensional square.

bits is the number of independent bits in successive n-tuples of numbers returned by RAN.

For example, successive pairs of random numbers can be considered to be independent in their first 15 bits. The remaining 12 bits are not independent.

Algorithm Used

RAN(n) is calculated as follows.

> Using a seed value supplied from a call to the SETRAN subroutine or the default seed value 524287($=2^{19}-1$), the seed value is calculated by:
>
> RAN(n) = seed/2^{31}, truncated
>
> On subsequent calls to RAN, a new seed is calculated from the previous seed value by:
>
> seed = seed•630360016 mod ($2^{31}-1$)
>
> and the random number is then generated.

References

A full description of the spectral test is given in R.R. Coveyan and R.D. MacPherson, Journal of the ACM 14 (1967), pp. 100–119 and in D.E. Knuth, Seminumerical Algorithms (Reading, Mass.: Addison-Wesley, 1981), Section 3.3.4.

Error Conditions

None

Description

The RANS routine returns pseudo random numbers between 0.0 and 1.0, but not including 0.0 or 1.0. The period of the sequence 2484877906816; that is, the numbers repeat every 2484877906816 calls.

RANS is based on the same multiplicative random number generator as RAN (p. 8–3). In addition, it shuffles the numbers using a 128-word table.

Routines Called

RANS calls the RAN and SAVRAN routines.

Type of Argument

The argument is a dummy value that is not used.

Type of Result

The result returned is a single-precision, floating-point value that is greater than 0.0 and less than 1.0.

Accuracy of Result

Not applicable

Algorithm Used

RANS(n) is calculated as follows.

On the initial reference to RANS, RAN is called 128 times to generate S_1, $S_2,\ldots,S_{128}$ (uniform random deviates in (0,1)) and a new seed x_0. x_0 is obtained from a call to the SAVRAN subroutine (see SAVRAN, p.8–7) after S_{128} has been generated. Then:

$$x_{i+1} = 630360016 \cdot x_i \bmod(2^{31}-1)$$
$$j = (x_{i+1} \bmod(128))+1$$
$$s_j = x_{i+1}/2^{31}$$
$$t = s_j$$
$$\text{RANS(n)} = t$$

Error Conditions

None

SETRAN

Description

The SETRAN subroutine provides the internal integer seed value for the RAN routine.

SETRAN is used to reset RAN to return the same sequence of random numbers again, or to set RAN to an arbitrary value (such as the time of day) so that it will return an entirely new sequence.

Routines Called

SETRAN does not call any routines; but you can call the SAVRAN subroutine to save and return the last seed value used by RAN.

Type of Argument

The argument must be an integer value in the range 0 to 2^{31}. If the argument is 0, the default seed value for RAN is used.

Type of Result

Not applicable

Accuracy of Result

Not applicable

Algorithm Used

SETRAN(n) is calculated as follows.

Using the value supplied, SETRAN computes:

$$seed = |seed| \bmod (2147483647)$$

Error Conditions

None

Description
The SAVRAN subroutine saves and returns the last seed used by the RAN routine.

Routines Called
None

Type of Argument
The argument must be an integer variable in which the seed value will be stored.

Type of Result
The result returned is an integer value between 1 and 2147483647.

Accuracy of Result
Not applicable

Algorithm Used
Not applicable

Error Conditions
None

Chapter 9
Absolute Value Routines

Description
The IABS routine returns the integer absolute value of its integer argument. That is:

$$IABS(n) = |n|$$

Routines Called
None

Type of Argument
The argument must be an integer value; it can be any such value.

Type of Result
The result returned is an integer value greater than or equal to 0.

Accuracy of Result
The result is exact.

Algorithm Used
IABS(n) is calculated as follows.

If $n \geq 0$
$ABS(n) = n$

If $n < 0$
$ABS(n) = -n$

Error Conditions
If the argument is the "most negative integer" (400000000000_8), overflow occurs and the result is set to machine infinity.

ABS

Description

The ABS routine returns the single-precision, floating-point absolute value of its single-precision, floating-point argument. That is:

$$ABS(x) = |x|$$

Routines Called

None

Type of Argument

The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result

The result returned is a single-precision, floating-point value greater than or equal to 0.0.

Accuracy of Result

The result is exact.

Algorithm Used

ABS(x) is calculated as follows.

If $x \geq 0.0$
    $ABS(x) = x$

If $x < 0.0$
    $ABS(x) = -x$

Error Conditions

None

Description
The DABS routine returns the double-precision, D-floating-point absolute value of its double-precision, D-floating-point argument. That is:

$$DABS(x) = |x|$$

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point value greater than or equal to 0.0.

Accuracy of Result
The result is exact.

Algorithm Used
DABS(x) is calculated as follows.

If $x \geq 0.0$
$DABS(x) = x$

If $x < 0.0$
$DABS(x) = -x$

Error Conditions
None

GABS

Description

The GABS routine returns the double-precision, G-floating-point absolute value of its double-precision, G-floating-point argument. That is:

$$GABS(x) = |x|$$

Routines Called

None

Type of Argument

The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result

The result returned is a double-precision, G-floating-point value greater than or equal to 0.0.

Accuracy of Result

The result is exact.

Algorithm Used

GABS(x) is calculated as follows.

If $x \geq 0.0$
    $GABS(x) = x$

If $x < 0.0$
    $GABS(x) = -x$

Error Conditions

None

Description

The CABS routine returns the single-precision, floating-point absolute value of its complex, single-precision, floating-point argument. That is:

CABS(z) = |z|

Routines Called

CABS calls the SQRT and MTHERR routines.

Type of Argument

The argument must be a complex, single-precision, floating-point value; it can be any such value.

Type of Result

The result returned is a single-precision, floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	-1.00000×10^{18} through 1.00000×10^{18} real -1.00000×10^{18} through 1.00000×10^{18} imaginary
MRE:	1.84×10^{-8} (25.7 bits)
RMS:	5.36×10^{-9} (27.5 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	14%	65%	21%	0%

Algorithm Used

CABS(z) is calculated as follows.

Let z = x+i•y
v = MAX(|x|,|y|)
w = MIN(|x|,|y|)

Then CABS(z) = $v \cdot \sqrt{1.0+(w/v)^2}$

Error Conditions

If the argument is so large that it causes an overflow, the following message is issued and the result is set to +machine infinity.

CABS: Result overflow

CDABS

Description

The CDABS routine calculates the double-precision, D-floating-point absolute value of its complex, double-precision, D-floating-point argument. That is:

CDABS(z) = |z|
z = location of input value

Routines Called

CDABS calls the DSQRT and MTHERR routines.

Type of Argument

The argument must be a two-element, double-precision vector that contains the input value, (z). Z must be a complex, double-precision, D-floating-point value; it can be any such value.

Type of Result

The result returned is a double-precision, D-floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	-1.00000×10^{18} through 1.00000×10^{18} real -1.00000×10^{18} through 1.00000×10^{18} imaginary
MRE:	6.32×10^{-19} (60.5 bits)
RMS:	1.89×10^{-19} (62.2 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	4%	56%	38%	2%

Algorithm Used

CDABS(z) is calculated as follows.

Let $z = x+i \cdot y$
$v = MAX(|x|,|y|)$
$w = MIN(|x|,|y|)$

Then $CDABS(z) = v \cdot \sqrt{1.0+(w/v)^2}$

Error Conditions

If the argument is so large that overflow occurs, the following message is issued and the result is set to +machine infinity.

CDABS: Result overflow

Description

The CGABS routine calculates the double-precision, G-floating-point absolute value of its complex, double-precision, G-floating argument. That is:

CGABS(z) = |z|
z = location of input value

Routines Called

CGABS calls the GSQRT and MTHERR routines.

Type of Argument

The argument must be a two-element, double-precision vector that contains the input value (z). Z must be a complex, double-precision, G-floating-point value; it can be any such value.

Type of Result

The result returned is a double-precision, G-floating-point value greater than or equal to 0.0.

Accuracy of Result

test interval:	-1.00000×10^{18} through 1.00000×10^{18} real -1.00000×10^{18} through 1.00000×10^{18} imaginary
MRE:	4.88×10^{-18} (57.5 bits)
RMS:	1.51×10^{-18} (59.2 bits)

LSB error distribution:	-2	-1	0	+1	+2
	0%	4%	56%	38%	2%

Algorithm Used

CGABS(z) is calculated as follows.

Let z = x+i•y
v = MAX(|x|,|y|)
w = MIN(|x|,|y|)

Then $CGABS(z) = v \cdot \sqrt{1.0+(w/v)^2}$

Error Conditions

If the argument is so large that overflow occurs, the following message is issued and the result is set to +machine infinity.

CGABS: Result overflow

Chapter 10
Data Type Conversion Routines

Description
The IFIX routine converts and truncates its single-precision, floating-point argument to an integer value.

Routines Called
None

Type of Argument
The argument must be a single-precision, floating-point value less than 2^{35}.

Type of Result
The result returned is an integer value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
IFIX(x) is calculated by means of the FIX machine instruction. This instruction converts and truncates the argument to an integer.

Error Conditions
If the argument is greater than 2^{35}, an overflow occurs and the result is set to machine infinity.

INT

Description
The INT routine converts and truncates its single-precision, floating-point argument to an integer value.

Routines Called
None

Type of Argument
The argument must be a single-precision, floating-point value less than 2^{35}.

Type of Result
The result returned is an integer value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
INT(x) is calculated by means of the FIX machine instruction. This instruction converts and truncates the argument to an integer.

Error Conditions
If the argument is greater than 2^{35}, an overflow occurs and the result is set to machine infinity.

Description
The IDINT routine converts and truncates its double-precision, D-floating-point argument to an integer value.

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is an integer value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
IDINT(x) is calculated as follows.

> The routine, working on the magnitude of the argument, copies the exponent field to a scratch register. It then clears the exponent field of the magnitude of the argument, and uses the copy of the exponent to control a shift to leave the integer in the location of the result. If necessary, the routine negates the result.

Error Conditions
If the shift results in a loss of significant bits on the left, an overflow occurs and the result is set to machine infinity.

Description

The GFX.n routine converts and truncates its double-precision, G-floating-point argument to an integer value. n is an even octal number from 0 through 14 that designates a register (AC).

Routines Called

None

Calling Sequence

GFX.n is not called like most of the other routines in the library (see Section 1.4.1). It is called by:

```
EXTEND n, GFX.n
```

Type of Argument

The argument must be a double-precision, G-floating-point value less than 2^{35}. It must be stored in the AC specified in the routine name.

Type of Result

The result returned is an integer value; it may be any such value. It is returned in the AC specified in the routine name.

Accuracy of Result

The result is exact.

Algorithm Used

GFX.n(x) is calculated by means of the GFIX machine instruction. This instruction converts and truncates the argument to an integer.

Error Conditions

If the argument is greater than 2^{35}, an overflow occurs and the result is set to machine infinity.

Description
The REAL routine converts and rounds its integer argument into a single-precision, floating-point value.

Routines Called
None

Type of Argument
The argument must be an integer value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value less than 2^{35}.

Accuracy of Result
The result is rounded with an error bound of half a least significant bit.

Algorithm Used
REAL(n) is calculated by means of the FLTR machine instruction. This instruction converts and rounds the argument to a single-precision, floating-point value.

Error Conditions
None

FLOAT

Description
The FLOAT routine converts and rounds its integer argument to a single-precision, floating-point value.

Routines Called
None

Type of Argument
The argument must be an integer value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value less than 2^{35}.

Accuracy of Result
The result is rounded with an error bound of half a least significant bit.

Algorithm Used
FLOAT(n) is calculated by means of the FLTR machine instruction. This instruction converts and rounds the argument to a single-precision floating-point value.

Error Conditions
None

Description
The SNGL routine converts and rounds its double-precision, D-floating-point argument to a single-precision, floating-point value.

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value; it may be any such value.

Accuracy of Result
The result is accurate to half a least significant bit because of rounding.

Algorithm Used
SNGL(x) is calculated as follows.

The routine tests the most significant bit of the low word of the magnitude of the argument.

- If it is 0, the high word is returned.
- If it is 1, the low bit of the high word of the magnitude is tested.
  - If it is 0, it is made 1 and negated if necessary.
  - If it is 1, the high word of the magnitude is incremented and negated if necessary.

Error Conditions
If overflow occurs, the result is set to machine infinity.

Description

The GSN.n routine converts and rounds its double-precision, G-floating-point argument to a single-precision, floating-point value. n is an even octal number from 0 through 14 that designates a register (AC).

Routines Called

None

Calling Sequence

GSN.n is not called like most of the other routines in the library (see Section 1.4.1). It is called by:

EXTEND n GSN.n

Type of Argument

The argument must be a double-precision, G-floating-point value; it can be any such value. It must be stored in the AC specified in the routine name.

Type of Result

The result returned is a single-precision, floating-point value; it may be any such value. It is returned in the AC specified in the routine name.

Accuracy of Result

The result is exact to half a least significant bit because of rounding.

Algorithm Used

GSN.n(x) is calculated as follows.

The routine tests the most significant bit of the low word of the magnitude of the argument.

If it is 0, the high word is returned.
If it is 1, the low bit of the high word of the magnitude is tested.

> If it is 0, it is made 1 and negated if necessary.
> If it is 1, the high word of the magnitude is incremented and negated if necessary.

Error Conditions

1. If overflow occurs, the result is set to machine infinity.
2. If underflow occurs, the result is set to 0.0.

Description
The DFLOAT routine converts its integer argument to a double-precision, D-floating-point value.

Routines Called
None

Type of Argument
The argument must be an integer value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point value less than 2^{35}.

Accuracy of Result
The result is exact.

Algorithm Used
DFLOAT(n) is calculated by moving the value of the argument to the locations used by a double-precision result. See Chapter 1 for a discussion of the location of the result.

Error Conditions
None

Description
The DBLE routine converts its single-precision floating-point argument to a double-precision, D-floating-point value.

Routines Called
None

Type of Argument
The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
DBLE(x) is calculated by moving the value of the argument to the locations used by a double-precision result. (See Chapter 1 for a discussion of the location of the result.) The low order word is set to 0.

Error Conditions
None

Description
The GTOD routine converts its double-precision, G-floating point argument to a double-precision, D-floating-point value.

Routines Called
GTOD calls the MTHERR routine.

Type of Argument
The argument must be a double-precision G-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
GTOD(x) is calculated by converting the double-precision, G-floating-point value to double-precision, D-floating point and setting the low-order three bits to 0.

Error Conditions

1. If the resulting exponent is too small to be represented as a double-precision, D-floating-point number, the following message is issued and the result is set to 0.0.

   GTOD: Result underflow

2. If the resulting exponent is too large to be represented as a double-precision, D-floating-point number, the following message is issued and the result is set to +machine infinity.

   GTOD: Result overflow

GTODA

Description

The GTODA subroutine converts an array of double-precision, G-floating-point values to an array of double-precision, D-floating-point values. It is called as:

GTODA (x,y,i)
  x = input array
  y = array used for result
  i = number of elements to convert

Routines Called

GTODA calls the MTHERR routine.

Type of Arguments

GTODA is a subroutine that is called with three arguments. The first and second arguments must be double-precision arrays. The third argument must be an integer value representing the number of elements to be converted. The first array (x) contains the input values; the second array (y) will contain the results. The input values must be double-precision, G-floating-point values; they can be any such values.

Type of Result

The result returned is an array of double-precision, D-floating-point values; they may be any such values. They are returned in the second array (y) supplied in the call.

Accuracy of Result

The result is exact for each value converted.

Algorithm Used

GTODA(x) is calculated as follows.

Using the number specified in the third argument, GTODA converts each double-precision, G-floating-point value to a double-precision, D-floating-point value and sets the low-order three bits to 0. Each converted value is stored in the second array.

Error Conditions

1. For each resulting exponent that is too small to be represented as a double-precision, D-floating-point number, the following message is issued and the result is set to 0.0.

   GTODA: Result underflow

2. For each resultIng exponent that is too large to be represented as a double-precision, D-floating-point number, the following message is issued and the result is set to +machine infinity.

   GTODA: Result overflow

Description
The GFL.n routine converts its integer argument to a double-precision, G-floating-point value. n is an even octal number from 0 through 14 that designates a register (AC).

Routines Called
None

Calling Sequence
GFL.n is not called like most of the routines in the library (see Section 1.4.1). It is called by:

EXTEND n, GFL.n

Type of Argument
The argument must be an integer value; it can be any such value. It must be stored in the AC specified in the routine name.

Type of Result
The result returned is a double-precision, G-floating-point value less than 2^{35}. It is returned in the AC specified in the routine name.

Accuracy of Result
The result is exact.

Algorithm Used
GFL.n(n) is calculated by moving the value of the argument to the locations used by a double-precision result (see Chapter 1).

Error Conditions
None

Description
The GDB.n routine converts its single-precision, floating-point argument to a double-precision, G-floating-point value. n is an even octal number from 0 through 14 that designates a register (AC).

Routines Called
None

Calling Sequence
GDB.n is not called like most of the routines in the library (see Section 1.4.1). It is called by:

EXTEND n, GDB.n

Type of Argument
The argument must be a single-precision, floating-point value; it can be any such value. It must be stored in the AC specified in the routine name.

Type of Result
The result returned is a double-precision, G-floating-point value; it may be any such value. It is returned in the AC specified in the routine name.

Accuracy of Result
The result is exact.

Algorithm Used
GDB.n(x) is calculated as follows.

The routine uses the GDBLE machine instruction to convert the argument and move it to the locations used for double-precision results.

Error Conditions
None

Description
The DTOG routine converts its double-precision, D-floating-point argument to a double-precision, G-floating-point value.

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result
The result is rounded with an error bound of half a least significant bit.

Algorithm Used
DTOG(x) is calculated by converting the double-precision, D-floating-point value to a double-precision, G-floating-point value and rounding the converted value.

Error Conditions
None

DTOGA

Description

The DTOGA subroutine converts an array of double-precision, D-floating-point values to an array of double-precision, G-floating-point values. It is called as:

DTOGA(x,y,i)
x = input array
y = array used for result
i = number of elements to convert

Routines Called

None

Type of Arguments

DTOGA is a subroutine that is called with three arguments. The first and second arguments must be double-precision arrays. The third argument must be an integer value representing the number of elements to be converted. The first array (x) contains the input values; the second array (y) will contain the result. The input values must be double-precision, D-floating-point values; they can be any such values.

Type of Result

The result returned is an array of double-precision, G-floating-point values; they may be any such values. They are returned in the second array (y) supplied in the call.

Accuracy of Result

Each element of the result is rounded with an error bound of half a least significant bit.

Algorithm Used

DTOGA(x) is calculated as follows.

Using the number specified in the third argument, DTOGA converts each double-precision, D-floating-point value to a double-precision, G-floating-point value and rounds the converted value. Each converted value is stored in the second array.

Error Conditions

None

Description
The CMPL.I routine converts its two integer arguments into a complex, single-precision, floating-point value.

Routines Called
None

Type of Arguments
Both arguments must be integer values; they can be any such values.

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result
The result is rounded with an error bound of half a least significant bit for each part (real and imaginary).

Algorithm Used
CMPL.I(n,m) is calculated as follows.

> The two arguments are converted to single-precision, floating-point values using the FLTR machine instructions. These values are then moved to the locations where the result is stored as a complex value (see Chapter 1). The first argument is used as the real part of the complex number and the second argument as the imaginary part.

Error Conditions
None

Description

The CMPLX routine converts two single-precision arguments into one complex single-precision, floating-point value.

Routines Called

None

Type of Arguments

Both arguments must be single-precision, floating-point values; they can be any such values.

Type of Result

The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

CMPLX(x,y) is calculated by moving the arguments to the locations used for a complex result (see Chapter 1). The first argument is used as the real part of the complex number and the second argument as the imaginary part.

Error Conditions

None

Description
The CMPL.D routine converts its two double-precision, D-floating-point arguments into a complex, single-precision, floating-point value.

Routines Called
None

Type of Arguments
The arguments must be double-precision, D-floating-point values; they can be any such values.

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result
The result is accurate to half a least significant bit for each part because of rounding.

Algorithm Used
CMPL.D(x,y) is calculated by converting the arguments to single-precision and then moving them to the locations used for the real and imaginary parts of the complex result (see Chapter 1). The first argument is used as the real part of the complex number and the second argument as the imaginary part.

Error Conditions
If overflow occurs on the conversions, the result is set to machine infinity for either or both of the parts of the result.

Description
The CMPL.G routine converts its two double-precision, G-floating-point arguments into a complex, single-precision, floating-point value.

Routines Called
None

Type of Arguments
The arguments must be double-precision, G-floating-point values; they can be any such values.

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result
The result is accurate to half a least significant bit for each part because of rounding.

Algorithm Used
CMPL.G(x,y) is calculated by converting the arguments to single-precision and then moving them to the locations used for the real and imaginary parts of the complex result (see Chapter 1). The first argument is used as the real part of the complex number and the second argument as the imaginary part.

Error Conditions

1. If overflow occurs on the conversions, the result is set to machine infinity for either or both of the parts of the result.
2. If underflow occurs on the conversions, the result is set to 0.0 for either or both parts of the result.

Description
The CMPL.C routine creates a complex, single-precision, floating-point value from the real parts of two complex, single-precision, floating-point values.

Routines Called
None

Type of Arguments
The arguments must be complex, single-precision, floating-point values; they can be any such values.

Type of Result
The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
CMPL.C(z,g) is calculated by moving the arguments to the locations used for a complex result (see Chapter 1). The first argument is used as the real part of the complex number and the second argument as the imaginary part.

Error Conditions
None

Chapter 11
Rounding and Truncation Routines

Description

The NINT routine rounds its single-precision, floating-point argument to the nearest integer.

Routines Called

NINT calls the MTHERR routine.

Type of Argument

The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result

The result returned is an integer value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

NINT(x) is calculated as follows.

Let j = INT(|x|+.5)

If $j < 2^{35}$ and
If $x \geq 0.0$
NINT(x) = j
If $x < 0.0$
NINT(x) = -j

If $j = 2^{35}$ and
If $x < 0.0$
NINT(x) = -j

Otherwise, overflow occurs and
If $x > 0.0$
NINT(x) = $2^{35}-1$
If $x < 0.0$
NINT(x) = -2^{35}

Error Conditions

If x is greater than or equal to 2^{35} or less than -2^{35}, the result overflows. When overflow occurs, the following message is issued and the result is set to +machine infinity if x is greater than 0.0 or to -machine infinity if x is less than 0.0.

NINT: Result overflow

IDNINT

Description
The IDNINT routine rounds its double-precision, D-floating-point argument to the nearest integer.

Routines Called
IDNINT calls the MTHERR routine.

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is an integer value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
IDNINT(x) is calculated as follows.

Let j = INT(|x|+.5)

If $j < 2^{35}$ and
  If $x \geq 0.0$
    IDNINT(x) = j
  If $x < 0.0$
    IDNINT(x) = -j

If $j = 2^{35}$ and
  If $x < 0.0$
    IDNINT(x) = -j

Otherwise, overflow occurs and
  If $x > 0.0$
    IDNINT(x) = $2^{35}-1$
  If $x < 0.0$
    IDNINT(x) = -2^{35}

Error Conditions
If x is greater than or equal to 2^{35} or less than -2^{35}, the result overflows. When overflow occurs, the following message is issued and the result is set to +machine infinity if x is greater than 0.0 or to -machine infinity if x is less than 0.0.

```
IDNINT: Result overflow
```

Description

The IGNIN. routine rounds its double-precision, G-floating-point argument to the nearest integer.

Routines Called

IGNIN. calls the MTHERR routine.

Type of Argument

The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result

The result returned is an integer value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

IGNIN.(x) is calculated as follows.

Let j = INT(|x|+.5)

If $j < 2^{35}$ and
    If $x \geq 0.0$
        IGNIN.(x) = j
    If $x < 0.0$
        IGNIN.(x) = -j

If $j = 2^{35}$ and
    If $x < 0.0$
        IGNIN.(x) = -j

Otherwise, overflow occurs and
    If $x > 0.0$
        IGNIN.(x) = $2^{35}-1$
    If $x < 0.0$
        IGNIN.(x) = -2^{35}

Error Conditions

If x is greater than or equal to 2^{35} or less than -2^{35}, the result overflows. When overflow occurs, the following message is issued and the result is set to +machine infinity if x is greater than 0.0 or – machine infinity if x is less than 0.0.

```
IGNIN.: Result overflow
```

Description

The ANINT routine rounds its single-precision, floating-point argument to the nearest single-precision, floating-point whole number.

Routines Called

None

Type of Argument

The argument must be a single-precision, floating-point value; it can be any such value.

Type of Return

The result returned is a single-precision, floating-point whole value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

ANINT(x) is calculated as follows.

If $|x| \geq 2^{26}$
ANINT(x) = x because x is an integer

If $|x| < 2^{26}$
If $x > 0.0$
$ANINT(x) = ((|x|+2^{26})\text{rounded})-2^{26}$

If $x < 0.0$
$ANINT(x) = -(((|x|+2^{26})\text{rounded})-2^{26})$

Error Conditions

None

Description
The DNINT routine rounds its double-precision, D-floating-point argument to the nearest double-precision, D-floating-point whole number.

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point whole value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
DNINT is calculated as follows.

If $|x| \geq 2^{61}$
  DNINT(x) = x because x is an integer

If $|x| < 2^{61}$
  If $x > 0.0$
    $DNINT(x) = ((|x|+2^{61})rounded)-2^{61}$

  If $x < 0.0$
    $DNINT(x) = -(((|x|+2^{61})rounded)-2^{61})$

Error Conditions
None

GNINT.

Description
The GNINT. routine rounds its double-precision, G-floating-point argument to the nearest double-precision, G-floating-point whole number.

Routines Called
None

Type of Argument
The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, G-floating-point whole value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
GNINT.(x) is calculated as follows.

If $|x| \geq 2^{58}$
  GNINT.(x) = x because x is an integer

If $|x| < 2^{58}$
  If $x > 0.0$
    $\text{GNINT.}(x) = ((|x|+2^{58})\text{rounded})-2^{58}$

  If $x < 0.0$
    $\text{GNINT.}(x) = -(((|x|+2^{58})\text{rounded})-2^{58})$

Error Conditions
None

Description
The AINT routine truncates its single-precision, floating-point argument to a single-precision, floating-point whole number.

Routines Called
None

Type of Argument
The argument must be a single-precision, floating-point value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point whole value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
AINT(x) is calculated as follows.

If $|x| \geq 2^{26}$
AINT(x) = x because x is an integer

If $|x| < 2^{26}$
If $x > 0.0$
$AINT(x) = ((|x|+2^{26})\text{truncated})-2^{26}$

If $x < 0.0$
$AINT(x) = -(((|x|+2^{26})\text{truncated})-2^{26})$

Error Conditions
None

DINT

Description
The DINT routine truncates its double-precision, D-floating-point argument to a double-precision, D-floating-point whole number.

Routines Called
None

Type of Argument
The argument must be a double-precision, D-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, D-floating-point whole value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
DINT(x) is calculated as follows.

If $|x| \geq 2^{61}$
DINT(x) = x because x is an integer

If $|x| < 1.0$
DINT(x) = 0.0

Otherwise
DINT(x) = sgn(x)•(|x| with fraction bits replaced by zeroes)

Error Conditions
None

Description
The GINT. routine truncates its double-precision, G-floating-point argument to a double-precision, G-floating-point whole number.

Routines Called
None

Type of Argument
The argument must be a double-precision, G-floating-point value; it can be any such value.

Type of Result
The result returned is a double-precision, G-floating-point whole value; it may be any such value.

Accuracy of Result
The result is exact.

Algorithm Used
GINT.(x) is calculated as follows.

If $|x| \geq 2^{58}$
GINT.(x) = x because x is an integer

If $|x| < 1.0$
GINT.(x) = 0.0

Otherwise
GINT.(x) = sgn(x)•(|x| with fraction bits replaced by zeroes)

Error Conditions
None

Chapter 12

Product, Remainder, and Positive Difference Routines

Description

The DPROD routine multiplies two single-precision, floating-point numbers and returns a double-precision, D-floating-point product. That is:

$$DPROD(x,y) = x \cdot y$$

Routines Called

DPROD calls the MTHERR routine.

Type of Arguments

Both arguments must be single-precision, floating-point values; they can be any such values.

Type of Result

The result returned is a double-precision, D-floating-point value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

DPROD(x,y) is calculated as follows.

Let $x = DBLE(x)$
$y = DBLE(y)$

$$DPROD(x,y) = x \cdot y$$

Error Conditions

1. If overflow occurs, the following message is issued and the result is set to ±machine infinity.

   DPROD: Result overflow

2. If underflow occurs, the following message is issued and the result is set to 0.0.

   DPROD: Result underflow

GPROD.

Description

The GPROD. routine multiplies two single-precision, floating-point numbers and returns a double-precision, G-floating-point product. That is:

GPROD.(x,y) = x•y

Routines Called

GPROD. calls the MTHERR routine.

Type of Arguments

Both arguments must be single-precision, floating-point values; they can be any such values.

Type of Result

The result returned is a double-precision, G-floating-point value; it may be any such value.

Accuracy of Result

The result is exact.

Algorithm Used

GPROD.(x,y) is calculated as follows.

Let x = GDB.0(x)
    y = GDB.0(y)

GPROD.(x,y) = x•y

Error Conditions

None

Description

The MOD routine returns the integer remainder of the quotient of its integer arguments. That is:

$$MOD(i,j) = i-[i/j] \cdot j$$

Routines Called

None

Type of Arguments

Both arguments must be integer; the second argument cannot equal zero. If the first argument is negative, the result is negative.

Type of Result

The result returned is an integer value in the range $-|j|$ to $|j|$.

Accuracy of Result

The result is exact.

Algorithm Used

MOD(i,j) is calculated as follows.

$$MOD(i,j) = (|i|-[|i|/j] \cdot j) \cdot sgn(i)$$

$$[|i|/j] = \text{the greatest integer in } |i|/j$$

Error Conditions

None

AMOD

Description

The AMOD routine returns the single-precision, floating-point remainder of the quotient of its single-precision, floating-point arguments. That is:

$$AMOD(x,y) = x-[x/y]\cdot y$$

Routines Called

AMOD calls the MTHERR routine.

Type of Arguments

Both arguments must be single-precision, floating-point values; the second argument cannot equal zero. If the first argument is negative, the result will be negative.

Type of Result

The result returned is a single-precision, floating-point value in the range $-|y|$ to $|y|$.

Accuracy of Result

The result is exact.

Algorithm Used

AMOD(x,y) is calculated as follows.

$$AMOD(x,y) = (|x|-[|x|/y]\cdot y)\cdot sgn(x)$$

$$[|x|/y] = \text{largest integer in } |x|/y$$

Error Conditions

Underflow may occur if y is too small a number. If underflow occurs, the following message is issued and the result is set to 0.0.

```
AMOD: Result underflow
```

Description

The DMOD routine returns the double-precision, D-floating-point remainder of the quotient of its double-precision, D-floating-point arguments. That is:

$$DMOD(x,y) = x-[x/y] \cdot y$$

Routines Called

DMOD calls the MTHERR routine.

Type of Arguments

Both arguments must be double-precision, D-floating-point values; the second argument cannot equal zero. If the first argument is negative, the result will be negative.

Type of Result

The result returned is a double-precision, D-floating-point value in the range $-|y|$ to $|y|$.

Accuracy of Result

The result is exact.

Algorithm Used

DMOD(x,y) is calculated as follows.

$$DMOD(x,y) = (|x|-[|x|/y] \cdot y) \cdot sgn(x)$$
$$[|x|/y] = \text{largest integer in } |x|/y$$

Error Conditions

Underflow may occur if y is too small a number. If underflow occurs, the following message is issued and the result is set to 0.0.

```
DMOD: Result underflow
```

GMOD

Description

The GMOD routine returns the double-precision, G-floating-point remainder of the quotient of its double-precision, G-floating-point arguments. That is:

$$GMOD(x,y) = x-[x/y]\cdot y$$

Routines Called

GMOD calls the MTHERR routine.

Type of Arguments

Both arguments must be double-precision, G-floating-point values; the second argument cannot equal zero. If the first argument is negative, the result will be negative.

Type of Result

The result returned is a double-precision, G-floating-point value in the range $-|y|$ to $|y|$.

Accuracy of Result

The result is exact.

Algorithm Used

GMOD(x,y) is calculated as follows.

$$GMOD(x,y) = (|x|-[|x|/y]\cdot y)\cdot sgn(x)$$
$$[|x|/y] = \text{largest integer in } |x|/y$$

Error Conditions

Underflow may occur if y is too small a number. If underflow occurs, the following message is issued and the result is set to 0.0.

```
GMOD: Result underflow
```

Description
The IDIM routine returns the integer difference between its integer arguments, provided that the difference is positive. If the difference is negative, IDIM returns zero. That is:

IDIM(i,j) = i–j

Routines Called
IDIM calls the MTHERR routine.

Type of Arguments
Both arguments must be integer values; they can be any such values.

Type of Result
The result returned is an integer value greater than or equal to 0.

Accuracy of Result
The result is exact.

Algorithm Used
IDIM is calculated as follows.

If $i \leq j$
IDIM(i,j) = 0

If $i > j$
IDIM(i,j) = i–j

Error Conditions
If overflow occurs during subtraction, the following message is issued and the result is set to machine infinity.

IDIM: Result overflow

Description

The DIM routine returns the single-precision, floating-point difference between its single-precision, floating-point arguments, provided that the difference is positive. If the difference is negative, DIM returns zero. That is:

DIM(x,y) = x−y

Routines Called

DIM calls the MTHERR routine.

Type of Arguments

Both arguments must be single-precision, floating-point values; they can be any such values.

Type of Result

The result returned is a single-precision, floating-point value greater than or equal to 0.0.

Accuracy of Result

The result is rounded with an error bound of half a least significant bit.

Algorithm Used

DIM(x,y) is calculated as follows.

If $x \leq y$
DIM(x,y) = 0.0

If $x > y$
DIM(x,y) = x−y

Error Conditions

1. If overflow occurs during subtraction, the following message is issued and the result is set to machine infinity.

   DIM: Result overflow

2. If underflow occurs during subtraction, the following message is issued and the result is set to 0.0.

   DIM: Result underflow

Description

The DDIM routine returns the double-precision, D-floating-point difference between its double-precision, D-floating-point arguments, provided that the difference is positive. If the difference is negative, DDIM returns zero. That is:

DDIM(x,y) = x−y

Routines Called

DDIM calls the MTHERR routine.

Type of Arguments

Both arguments must be double-precision, D-floating-point values; they can be any such values.

Type of Result

The result returned is a double-precision, D-floating-point value greater than or equal to 0.0.

Accuracy of Result

The result is rounded with an error bound of half a least significant bit.

Algorithm Used

DDIM(x,y) is calculated as follows.

If $x \le y$
DDIM(x,y) = 0.0

If $x > y$
DDIM(x,y) = x−y

Error Conditions

1. If overflow occurs during subtraction, the following message is issued and the result is set to machine infinity.

   DDIM: Result overflow

2. If underflow occurs during subtraction, the following message is issued and the result is set to 0.0.

   DDIM: Result underflow

Description

The GDIM routine returns the double-precision, G-floating-point difference between its double-precision, G-floating-point arguments, provided that the difference is positive. If the difference is negative, GDIM returns zero. That is:

GDIM(x,y) = x–y

Routines Called

GDIM calls the MTHERR routine.

Type of Arguments

Both arguments must be double-precision, G-floating-point values; they can be any such values.

Type of Result

The result returned is a double-precision, G-floating-point value greater than or equal to 0.0.

Accuracy of Result

The result is rounded with an error bound of half a least significant bit.

Algorithm Used

GDIM(x,y) is calculated as follows.

If $x \leq y$
    GDIM(x,y) = 0.0

If $x > y$
    GDIM(x,y) = x–y

Error Conditions

1. If overflow occurs during subtraction, the following message is issued and the result is set to machine infinity.

   GDIM: Result overflow

2. If underflow occurs during subtraction, the following message is issued and the result is set to 0.0.

   GDIM: Result underflow

Chapter 13
Transfer of Sign Routines

Description
The ISIGN routine transfers the sign of its integer second argument to its integer first argument, ignoring the sign of the first argument. That is:

$$ISIGN\ (i,j) = |i| \cdot sgn(j)$$

Routines Called
ISIGN calls the MTHERR routine.

Type of Arguments
Both arguments must be integer values; they can be any such values.

Type of Result
The result returned is an integer value; it has the same magnitude as the first argument.

Accuracy of Result
The result is exact.

Algorithm Used
ISIGN(i,j) is calculated as follows.

$$ISIGN(i,j) = |i| \cdot sgn(j)$$

If $j \geq 0$

$$ISIGN(i,j) = |i|$$

If $j < 0$

$$ISIGN(i,j) = -|i|$$

Error Conditions
If $i = -2^{35}$ and $j > 0$, overflow occurs. If overflow occurs, the following message is issued and the result is set to machine infinity.

```
ISIGN: Result overflow
```

SIGN

Description

The SIGN routine transfers the sign of its single-precision, floating-point second argument to its single-precision, floating-point first argument, ignoring the sign of the first argument. That is:

$$SIGN(x,y) = |x| \cdot sgn(y)$$

Routines Called

None

Type of Arguments

Both arguments must be single-precision, floating-point values; they can be any such values.

Type of Result

The result returned is a single-precision, floating-point value; it has the same magnitude as the first argument.

Accuracy of Result

The result is exact.

Algorithm Used

SIGN(x,y) is calculated as follows.

$$SIGN(x,y) = |x| \cdot sgn(y)$$

If $y \geq 0.0$

$$SIGN(x,y) = |x|$$

If $y < 0.0$

$$SIGN(x,y) = -|x|$$

Error Conditions

None

Description
The DSIGN routine transfers the sign of its double-precision, D-floating-point second argument to its double-precision, D-floating-point first argument, ignoring the sign of the first argument. That is:

$$DSIGN(x,y) = |x| \cdot sgn(y)$$

Routines Called
None

Type of Arguments
Both arguments must be double-precision, D-floating-point values; they can be any such values.

Type of Result
The result returned is a double-precision, D-floating-point value; it has the same magnitude as the first argument.

Accuracy of Result
The result is exact.

Algorithm Used
DSIGN(x,y) is calculated as follows.

$$DSIGN(x,y) = |x| \cdot sgn(y)$$

If $y \geq 0.0$
$$DSIGN(x,y) = |x|$$

If $y < 0.0$
$$DSIGN(x,y) = -|x|$$

Error Conditions
None

Description

The GSIGN routine transfers the sign of its double-precision, G-floating-point second argument to its double-precision, G-floating-point first argument, ignoring the sign of the first argument. That is:

$$GSIGN(x,y) = |x| \cdot sgn(y)$$

Routines Called

None

Type of Arguments

Both arguments must be double-precision, G-floating-point values; they can be any such values.

Type of Result

The result returned is a double-precision, G-floating-point value; it has the same magnitude as the first argument.

Accuracy of Result

The result is exact.

Algorithm Used

GSIGN(x,y) is calculated as follows.

$$GSIGN(x,y) = |x| \cdot sgn(y)$$

If $y \geq 0.0$
$$GSIGN(x,y) = |x|$$

If $y < 0.0$
$$GSIGN(x,y) = -|x|$$

Error Conditions

None

Chapter 14

Maximum/Minimum Routines

Description
The MAX0 routine finds the integer maximum of a series of integer arguments.

Routines Called
None

Type of Arguments
All the arguments must be integer values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is an integer value; it is the largest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
MAX0(i,...j) is calculated as follows.

The MAX0 routine compares each argument in succession with the current largest argument, which is held in a register. Each time an argument exceeds the current largest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then returned as the result.

Error Conditions
None

Description

The MAX1 routine finds the integer maximum of a series of single-precision, floating-point arguments.

Routines Called

None

Type of Arguments

All the arguments must be single-precision, floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result

The result returned is the largest value in the series converted to integer format.

Accuracy of Result

The result is exact except for possible overflow during the conversion to integer.

Algorithm Used

MAX1(x,...y) is calculated as follows.

The MAX1 routine compares each argument in succession with the current largest argument, which is held in a register. Each time an argument exceeds the current largest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then converted to integer format and returned as the result.

Error Conditions

Overflow can occur during conversion to integer. If overflow occurs, the result is set to $\pm$ machine infinity.

Description
The AMAX0 routine finds the single-precision, floating-point maximum of a series of integer arguments.

Routines Called
None

Type of Arguments
All the arguments must be integer; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is the largest value in the series converted to single-precision, floating-point format.

Accuracy of Result
The result is exact unless a rounding error occurs during conversion, in which case the error could be half a least significant bit.

Algorithm Used
AMAX0(i,...j) is calculated as follows.

The AMAX0 routine compares each argument in succession with the current largest argument, which is held in a register. Each time an argument exceeds the current largest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then converted to single-precision, floating-point format and returned as the result.

Error Conditions
None

Description
The AMAX1 routine finds the single-precision, floating-point maximum of a series of single-precision, floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be single-precision, floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is a single-precision, floating-point value; it is the largest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
AMAX1(x,...y) is calculated as follows.

The AMAX1 routine compares each argument in succession with the current largest argument, which is held in a register. Each time an argument exceeds the current largest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then returned as the result.

Error Conditions
None

Description
The DMAX1 routine finds the double-precision, D-floating-point maximum of a series of double-precision, D-floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be double-precision, D-floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is a double-precision, D-floating-point value; it is the largest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
DMAX1(x,...y) is calculated as follows.

The DMAX1 routine compares each argument in succession with the current largest argument, which is held in two registers. Each time an argument exceeds the current largest argument, the registers are updated. This loop continues until the final argument is processed. The contents of the registers are then returned as the result.

Error Conditions
None

Description
The GMAX1 routine finds the double-precision, G-floating-point maximum of a series of double-precision, G-floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be double-precision, G-floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is a double-precision, G-floating-point value; it is the largest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
GMAX1(x,...y) is calculated as follows.

The GMAX1 routine compares each argument in succession with the current largest argument, which is held in two registers. Each time an argument exceeds the current largest argument, the registers are updated. This loop continues until the final argument is processed. The contents of the registers are then returned as the result.

Error Conditions
None

Description
The MIN0 routine finds the integer minimum of a series of integer arguments.

Routines Called
None

Type of Arguments
All the arguments must be integer values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is an integer value; it is the smallest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
MIN0(i,...j) is calculated as follows.

The MIN0 routine compares each argument in succession to the current smallest argument, which is held in a register. Each time an argument is less than the current smallest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then returned as the result.

Error Conditions
None

Description
The MIN1 routine finds the integer minimum of a series of single-precision, floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be single-precision, floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is the smallest value in the series converted to integer format.

Accuracy of Result
The result is exact except for possible overflow during the conversion to integer.

Algorithm Used
MIN1(x,...y) is calculated as follows.

The MIN1 routine compares each argument in succession with the current smallest argument, which is held in a register. Each time an argument is smaller than the current smallest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then converted to integer and returned as the result.

Error Conditions
Overflow can occur during conversion to integer. If overflow occurs, the result is set to ± machine infinity.

Description
The AMINO routine finds the single-precision, floating-point minimum of a series of integer arguments.

Routines Called
None

Type of Arguments
All the arguments must be integer; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is the smallest value in the series converted to single-precision, floating-point format.

Accuracy of Result
The result is exact unless a rounding error occurs during conversion, in which case the error could be half a least significant bit.

Algorithm Used
AMINO(i,...j) is calculated as follows.

The AMINO routine compares each argument in succession with the current smallest argument, which is held in a register. Each time an argument is smaller than the current smallest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then converted to single-precision, floating-point format and returned as the result.

Error Conditions
None

AMIN1

Description
The AMIN1 routine finds the single-precision, floating-point minimum of a series of single-precision, floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be single-precision, floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is a single-precision, floating-point value; it is the smallest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
AMIN1(x,...y) is calculated as follows.

The AMIN1 routine compares each argument in succession with the current smallest argument, which is held in a register. Each time an argument is smaller than the current smallest argument, the register is updated. This loop continues until the final argument is processed. The contents of the register are then returned as the result.

Error Conditions
None

Description
The DMIN1 routine finds the double-precision, D-floating-point minimum of a series of double-precision, D-floating-point arguments.

Routines Called
None

Type of Arguments
All the arguments must be double-precision, D-floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result
The result returned is a double-precision, D-floating-point value; it is the smallest value in the series.

Accuracy of Result
The result is exact.

Algorithm Used
DMIN1(x,...y) is calculated as follows.

The DMIN1 routine compares each argument in succession with the current smallest argument, which is held in two registers. Each time an argument is less than the current smallest argument, the registers are updated. This loop continues until the final argument is processed. The contents of the registers are then returned as the result.

Error Conditions
None

GMIN1

Description

The GMIN1 routine finds the double-precision, G-floating-point minimum of a series of double-precision, G-floating-point arguments.

Routines Called

None

Type of Arguments

All the arguments must be double-precision, G-floating-point values; they can be any such values. There can be as many arguments as desired.

Type of Result

The result returned is a double-precision, G-floating-point value; it is the smallest value in the series.

Accuracy of Result

The result is exact.

Algorithm Used

GMIN1(x,...y) is calculated as follows.

The GMIN1 routine compares each argument in succession with the current smallest argument, which is held in two registers. Each time an argument is less than the current smallest argument, the registers are updated. This loop continues until the final argument is processed. The contents of the registers are then returned as the result.

Error Conditions

None

Chapter 15
Miscellaneous Complex Routines

Description
The REAL.C routine returns the real part of a complex number. That is:

REAL.C(z) = REAL.C(x+i•y) = x

Routines Called
None

Type of Argument
The argument must be a complex value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value.

Accuracy of Result
The result is exact.

Algorithm Used
REAL.C(z) is calculated by copying the real part of the argument to the return location.

Error Conditions
None

AIMAG

Description
The AIMAG routine returns the imaginary part of a complex number. That is:

AIMAG(z) = AIMAG(x+i•y) = y

Routines Called
None

Type of Argument
The argument must be a complex value; it can be any such value.

Type of Result
The result returned is a single-precision, floating-point value; it is the imaginary part of the number.

Accuracy of Result
The result is exact.

Algorithm Used
AIMAG(z) is calculated by copying the imaginary part of the argument to the return location.

Error Conditions
None

Description

The CONJ routine finds the conjugate of a complex number. That is:

CONJ(z) = conj(x+i•y) = x–i•y

Routines Called

None

Type of Argument

The argument must be a complex value; it can be any such value.

Type of Result

The result returned is a complex value; it is the conjugate of the argument value.

Accuracy of Result

The result is exact.

Algorithm Used

CONJ(z) is calculated as follows.

Let z = x+i•y
conj(x+i•y) = x+(–i•y)
CONJ(z) = x–i•y

Error Conditions

None

Description

The CFM subroutine finds the complex, single-precision, floating-point product of two complex, single-precision, floating-point values. That is:

CFM(z,g) = z•g

Routines Called

CFM calls the MTHERR routine.

Type of Arguments

CFM is a subroutine with two arguments; both must be complex, single-precision, floating-point values. They can be any such values.

Type of Result

The result returned is a complex, single-precision, floating-point value.

Accuracy of Result

test interval:	-10000. through 10000. for z (real) -10000. through 10000. for z (imaginary) -10000. through 10000. for g (real) -10000. through 10000. for g (imaginary)
MRE:	1.20×10^{-5} (16.4 bits) real 1.47×10^{-6} (19.4 bits) imaginary
RMS:	2.64×10^{-7} (21.9 bits) real 5.81×10^{-8} (24.0 bits) imaginary

LSB error distribution:

-4+	-3	-2	-1	0	+1	+2	+3	+4+	
2%	1%	1%	14%	64%	15%	1%	1%	2%	real
1%	1%	1%	15%	64%	14%	1%	1%	2%	imaginary

Algorithm Used

CFM(z,g) is calculated as follows.

Let z = a+i•b
Let g = c+i•d

If CFM(z,g) = (a+i•b)•(c+i•d)
CFM(z,g) = (a•c–b•d)+i•(b•c+a•d)

Error Conditions

1. If either part of the result overflows, the following message is issued and that part of the result is set to machine infinity.

   CMATH: Complex overflow

2. If either part of the result underflows, the following message is issued and that part of the result is set to 0.0.

   CMATH: Complex underflow

Description

The CFDV subroutine finds the complex, single-precision, floating-point quotient of two complex, single-precision, floating-point values. That is:

CFDV(z,g) = z/g

Routines Called

CFDV calls the MTHERR routine.

Type of Arguments

CFDV is a subroutine with two arguments; both must be complex, single-precision, floating-point values. They can be any such values.

Type of Result

The result returned is a complex, single-precision, floating-point value; it may be any such value.

Accuracy of Result

test interval:	-10000. through 10000. for z (real) -10000. through 10000. for z (imaginary) -10000. through 10000. for g (real) -10000. through 10000. for g (imaginary)
MRE:	2.87×10^{-7} (21.7 bits) real 7.60×10^{-7} (20.3 bits) imaginary
RMS:	1.33×10^{-8} (26.2 bits) real 2.30×10^{-8} (25.4 bits) imaginary

LSB error distribution:

	-4^{+}	-3	-2	-1	0	+1	+2	+3	$+4^{+}$	
	1%	1%	3%	22%	49%	21%	2%	0%	1%	real
	1%	1%	3%	21%	50%	20%	3%	1%	1%	imaginary

Algorithm Used

CFDV(z,g) is calculated as follows.

Let z = a+i•b
Let g = c+i•d

If CFDV(z,g) = (a+i•b)/(c+i•d)
CFDV(z,g) = ((a•c+b•d)+i•(b•c−a•d))/(c^2+d^2)

Error Conditions

1. If either part of the result underflows, the following message is issued and that part of the result is set to 0.0.

   CMATH: Complex underflow

2. If either part of the result overflows, that part of the result is set to machine infinity.

Appendix A
ELEFUNT Test Results

This appendix contains the results of the ELEFUNT tests of W. J. Cody, Argonne National Laboratory. For each test, the test interval, maximum relative error (MRE), and root mean square (RMS) relative error are given. Note that it is not meaningful to compare these test results with the test results given for each routine under the heading "Accuracy of Result."

ACOS(x) vs Taylor Series
test interval: -1.0000 through -0.7500
MRE: 0.1231×10^{-7} (26.3 bits)
RMS: 0.2868×10^{-8} (28.4 bits)

ACOS(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.1488×10^{-7} (26.0 bits)
RMS: 0.1330×10^{-8} (29.5 bits)

ACOS(x) vs Taylor Series
test interval: -0.1250 through 0.1250
MRE: 0.1030×10^{-7} (26.5 bits)
RMS: 0.2647×10^{-8} (28.5 bits)

ALOG(x•x) vs $2 \cdot \log_e x$
test interval: 0.1600×10^{2} through 0.2400×10^{3}
MRE: 0.1466×10^{-7} (26.0 bits)
RMS: 0.2292×10^{-8} (28.7 bits)

ALOG(x) vs Taylor Series expansion of ALOG(1+y)
test interval: $1-0.1953 \times 10^{-2}$ through $1+0.1953 \times 10^{-2}$
MRE: 0.2466×10^{-7} (25.3 bits)
RMS: 0.6614×10^{-8} (27.2 bits)

ALOG(x) vs ALOG(17x/16)-ALOG(17/16)
test interval: 0.7071 through 0.9375
MRE: 0.2264×10^{-7} (25.4 bits)
RMS: 0.6426×10^{-8} (27.2 bits)

ALOG10(x) vs ALOG10(11x/10)-ALOG10(11/10)
test interval: 0.3162 through 0.9000
MRE: 0.3863×10^{-7} (24.6 bits)
RMS: 0.1122×10^{-7} (26.4 bits)

ASIN(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.1478×10^{-7} (26.0 bits)
RMS: 0.3245×10^{-8} (28.2 bits)

ASIN(x) vs Taylor Series
test interval: -0.1250 through 0.1250
MRE: 0.1190×10^{-7} (26.3 bits)
RMS: 0.6733×10^{-9} (30.5 bits)

ATAN(x) vs truncated Taylor Series
test interval: -0.6250×10^{-1} through 0.6250×10^{-1}
MRE: 0.8032×10^{-8} (26.9 bits)
RMS: 0.1796×10^{-9} (32.4 bits)

ATAN(x) vs ATAN(1/16)+ATAN((x-1/16)/(1+x/16))
test interval: $0.6250.10^{-1}$ through 0.2679
MRE: 0.1488×10^{-7} (26.0 bits)
RMS: 0.6219×10^{-8} (27.3 bits)

2•ATAN(x) vs ATAN(2x/(1-x•x))
test interval: 0.2679 through 0.4142
MRE: 0.1423×10^{-7} (26.1 bits)
RMS: 0.6597×10^{-8} (27.2 bits)

2•ATAN(x) vs ATAN(2x/(1-x•x))
test interval: 0.4142 through 1.0000
MRE: 0.1484×10^{-7} (26.0 bits)
RMS: 0.3894×10^{-8} (27.9 bits)

COS(x) vs 4•COS(x/3)^3-3•COS(x/3)
test interval: 0.2199×10^{2} through 0.2356×10^{2}
MRE: 0.2070×10^{-7} (25.5 bits)
RMS: 0.6463×10^{-8} (27.2 bits)

COSH(x) vs C•(COSH(x+1)+COSH(x-1))
test interval: 3.0000 through 0.8803×10^{2}
MRE: 0.2219×10^{-7} (25.4 bits)
RMS: 0.7007×10^{-8} (27.1 bits)

COSH(x) vs Taylor Series expansion of COSH(x)
test interval: 0.0000 through 0.5000
MRE: 0.1490×10^{-7} (26.0 bits)
RMS: 0.5491×10^{-8} (27.4 bits)

COT(x) vs (COT(x/2)^2-1)/(2•COT(x/2))
test interval: 0.1885×10^{2} through 0.1963×10^{2}
MRE: 0.2975×10^{-7} (25.0 bits)
RMS: 0.8629×10^{-8} (26.8 bits)

DACOS(x) vs Taylor Series
test interval: –1.0000 through –0.7500
MRE: 0.3582×10^{-18} (61.3 bits)
RMS: 0.1211×10^{-18} (62.8 bits)

DACOS(x) vs Taylor Series
test interval: –0.1250 through –0.1250
MRE: 0.3000×10^{-18} (61.5 bits)
RMS: 0.1224×10^{-18} (62.8 bits)

DACOS(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.4337×10^{-18} (61.0 bits)
RMS: 0.1682×10^{-18} (62.4 bits)

DASIN(x) vs Taylor Series
test interval: –0.1250 through 0.1250
MRE: 0.4334×10^{-18} (61.0 bits)
RMS: 0.1715×10^{-18} (62.3 bits)

DASIN(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.4326×10^{-18} (61.0 bits)
RMS: 0.1168×10^{-18} (62.9 bits)

DATAN(x) vs truncated Taylor Series
test interval: -0.6250×10^{-1} through -0.6250×10^{-1}
MRE: 0.4326×10^{-18} (61.0 bits)
RMS: 0.1370×10^{-18} (62.7 bits)

DATAN(x) vs DATAN(1/16)+DATAN((x–1/16)/(1+x/16))
test interval: 0.6250×10^{-1} through 0.2679
MRE: 0.4333×10^{-18} (61.0 bits)
RMS: 0.1755×10^{-18} (62.3 bits)

2•DATAN(x) vs DATAN(2x/(1–x•x))
test interval: 0.2679 through 0.4142
MRE: 0.6610×10^{-18} (60.4 bits)
RMS: 0.1987×10^{-18} (62.1 bits)

2•DATAN(x) vs DATAN(2x/(1–x•x))
test interval: 0.4142 through 1.0000
MRE: 0.4319×10^{-18} (61.0 bits)
RMS: 0.1167×10^{-18} (62.9 bits)

DCOS(x) vs 4•DCOS(x/3)^3–3•DCOS(x/3)
test interval: 0.2199×10^{2} through 0.2356×10^{2}
MRE: 0.6523×10^{-18} (60.4 bits)
RMS: 0.1960×10^{-18} (62.2 bits)

DCOSH(x) vs Taylor Series expansion of DCOSH(x)
test interval: 0.0000 through 0.5000
MRE: 0.4337×10^{-18} (61.0 bits)
RMS: 0.1550×10^{-18} (62.5 bits)

DCOSH(x) vs C•(DCOSH(x+1)+DCOSH(x-1))
test interval: 3.0000 through 0.8803×10^{2}
MRE: 0.8440×10^{-18} (60.0 bits)
RMS: 0.2805×10^{-18} (61.6 bits)

DCOT(x) vs (DCOT(x/2)²-1)/(2•DCOT(x/2))
test interval: 0.1885×10^{2} through 0.1963×10^{2}
MRE: 0.9064×10^{-18} (59.9 bits)
RMS: 0.2632×10^{-18} (61.7 bits)

DEXP(x-0.0625) vs DEXP(x)/DEXP(0.0625)
test interval: -0.2841 through 0.3466
MRE: 0.4336×10^{-18} (61.0 bits)
RMS: 0.1689×10^{-18} (62.4 bits)

DEXP(x-2.8125) vs DEXP(x)/DEXP(2.8125)
test interval: -3.4660 through -0.4505×10^{2}
MRE: 0.6394×10^{-18} (60.4 bits)
RMS: 0.1670×10^{-18} (62.4 bits)

DEXP(x-2.8125) vs DEXP(x)/DEXP(2.8125)
test interval: -6.9310 through 0.8792×10^{2}
MRE: 0.6350×10^{-18} (60.4 bits)
RMS: 0.1808×10^{-18} (62.3 bits)

DEXP3. ($x^{1.0}$ vs x)
test interval: 0.5000 through 1.0000
The result is exact.

DEXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 0.5000 through 1.0000
MRE: 0.4336×10^{-18} (61.0 bits)
RMS: 0.1585×10^{-18} (62.4 bits)

DEXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 1.0000 through 0.5541×10^{13}
MRE: 0.4330×10^{-18} (61.0 bits)
RMS: 0.1678×10^{-18} (62.4 bits)

DEXP3. (x^{y} vs $XSQ^{y/2}$)
test interval: 0.1000×10^{-1} through 0.1000×10^{2} for x
-0.1942×10^{2} through 0.1942×10^{2} for y
MRE: 0.5499×10^{-18} (60.7 bits)
RMS: 0.1196×10^{-18} (62.9 bits)

DLOG(x) vs Taylor Series expansion of DLOG(1+y)
test interval: $1-9537 \times 10^{-6}$ through $1+9537 \times 10^{-6}$
MRE: 0.5605×10^{-18} (60.6 bits)
RMS: 0.1922×10^{-18} (62.2 bits)

DLOG(x) vs DLOG(17x/16)-DLOG(17/16)
test interval: 0.7071 through 0.9375
MRE: 0.9228×10^{-18} (59.9 bits)
RMS: 0.3347×10^{-18} (61.4 bits)

DLOG(x•x) vs 2•DLOG(x)
test interval: 0.1600×10^{2} through 0.2400×10^{3}
MRE: 0.4306×10^{-18} (61.0 bits)
RMS: 0.7895×10^{-19} (63.5 bits)

DLOG10(x) vs DLOG10(11x/10)–DLOG10(11/10)
test interval: 0.3162 through 0.9000
MRE: 0.1476×10^{-17} (59.2 bits)
RMS: 0.3747×10^{-18} (61.2 bits)

DSIN(x) vs 3•DSIN(x/3)–4•DSIN(x/3)^{3}
test interval: 0.0000 through 1.5710
MRE: 0.5378×10^{-18} (60.7 bits)
RMS: 0.1802×10^{-18} (62.3 bits)

DSIN(x) vs 3•DSIN(x/3)–4•DSIN(x/3)^{3}
test interval: 0.1885×10^{2} through 0.2042×10^{2}
MRE: 0.6115×10^{-18} (60.5 bits)
RMS: 0.1960×10^{-18} (62.2 bits)

DSINH(x) vs Taylor Series expansion of DSINH(x)
test interval: 0.0000 through 0.5000
MRE: 0.4336×10^{-18} (61.0 bits)
RMS: 0.8776×10^{-19} (63.3 bits)

DSINH(x) vs C•(DSINH(x+1)+DSINH(x–1))
test interval: 3.0000 through 0.8803×10^{2}
MRE: 0.8643×10^{-18} (60.0 bits)
RMS: 0.2736×10^{-18} (61.7 bits)

DSQRT(x•x)–x
test interval: 0.7071 through 1.0000
MRE: 0.3064×10^{-18} (61.5 bits)
RMS: 0.7383×10^{-19} (63.6 bits)

DSQRT(x•x)–x
test interval: 1.0000 through 1.4140
The result is exact.

DTAN(x) vs 2•TAN(x/2)/(1–DTAN(x/2)^{2})
test interval: 0.1885×10^{2} through 0.1963×10^{2}
MRE: 0.1262×10^{-17} (59.5 bits)
RMS: 0.3402×10^{-18} (61.4 bits)

DTAN(x) vs 2•DTAN(x/2)/(1–DTAN(x/2)^{2})
test interval: 2.7490 through 3.5340
MRE: 0.1216×10^{-17} (59.5 bits)
RMS: 0.2492×10^{-18} (61.8 bits)

DTAN(x) vs 2•DTAN(x/2)/(1–DTAN(x/2)^{2})
test interval: 0.0000 through 0.7854
MRE: 0.1094×10^{-17} (59.7 bits)
RMS: 0.3331×10^{-18} (61.4 bits)

DTANH(x) vs (DTANH(x-1/8)+DTANH(1/8))/(1+DTANH(x-1/8)DTANH(1/8))
test interval: 0.1250 through 0.5493
MRE: 0.8436×10^{-18} (60.0 bits)
RMS: 0.2150×10^{-18} (62.0 bits)

DTANH(x) vs (DTANH(x-1/8)+DTANH(1/8))/(1+DTANH(x-1/8)DTANH(1/8))
test interval: 0.6743 through 0.2253×10^{2}
MRE: 0.4952×10^{-18} (60.8 bits)
RMS: 0.1966×10^{-18} (62.1 bits)

EXP(x-0.0625) vs EXP(x)/EXP(0.0625)
test interval: -0.2841 through 0.3466
MRE: 0.1489×10^{-7} (26.0 bits)
RMS: 0.5801×10^{-8} (27.4 bits)

EXP(x-2.8125) vs EXP(x)/EXP(2.8125)
test interval: -3.4660 through -0.6931×10^{2}
MRE: 0.1489×10^{-7} (26.0 bits)
RMS: 0.5879×10^{-8} (27.3 bits)

EXP(x-2.8125) vs EXP(x)/EXP(2.8125)
test interval: 6.9310 through 0.8792×10^{2}
MRE: 0.2108×10^{-7} (25.5 bits)
RMS: 0.5768×10^{-8} (27.4 bits)

EXP3. ($x^{1.0}$ vs x)
test interval: 0.5000 through 1.0000
The result is exact.

EXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 0.5000 through 1.0000
MRE: 0.1487×10^{-7} (26.0 bits)
RMS: 0.5433×10^{-8} (27.5 bits)

EXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 1.0000 through 0.5541×10^{13}
MRE: 0.1461×10^{-7} (26.0 bits)
RMS: 0.5347×10^{-8} (27.5 bits)

EXP3. (x^{y} vs $XSQ^{y/2}$)
test interval: $0.1.000 \times 10^{-1}$ through 0.1000×10^{2} for x
-0.1942×10^{2} through 0.1942×10^{2} for y
MRE: 0.2065×10^{-7} (25.5 bits)
RMS: 0.3572×10^{-8} (28.0 bits)

GACOS(x) vs Taylor Series
test interval: -1.0000 through -0.7500
MRE: 0.2869×10^{-17} (58.3 bits)
RMS: 0.1515×10^{-17} (59.2 bits)

GACOS(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.3443×10^{-17} (58.0 bits)
RMS: 0.4924×10^{-18} (60.8 bits)

GACOS(x) vs Taylor Series
test interval: –0.1250 through 0.1250
MRE: 0.2399×10^{-17} (58.5 bits)
RMS: 0.1297×10^{-17} (59.4 bits)

GASIN(x) vs Taylor Series
test interval: 0.7500 through 1.0000
MRE: 0.3457×10^{-17} (58.0 bits)
RMS: 0.1452×10^{-17} (59.3 bits)

GASIN(x) vs Taylor Series
test interval: –0.1250 through 0.1250
MRE: 0.3462×10^{-17} (58.0 bits)
RMS: 0.4997×10^{-18} (60.8 bits)

GATAN(x) vs truncated Taylor Series
test interval: -0.6250×10^{-1} through 0.6250×10^{-1}
MRE: 0.3389×10^{-17} (58.0 bits)
RMS: 0.3674×10^{-18} (61.2 bits)

GATAN(x) vs GATAN(1/16)+GATAN((x–1/16)/(1+x/16))
test interval: 0.6250×10^{-1} through 0.2679
MRE: 0.3899×10^{-17} (57.8 bits)
RMS: 0.1436×10^{-17} (59.3 bits)

2•GATAN(x) vs GATAN(2x/(1–x•x))
test interval: 0.2679 through 0.4142
MRE: 0.3308×10^{-17} (58.1 bits)
RMS: 0.1601×10^{-17} (59.1 bits)

2•GATAN(x) vs GATAN(2x/(1–x•x))
test interval: 0.4142 through 1.0000
MRE: 0.4360×10^{-17} (57.7 bits)
RMS: 0.9839×10^{-18} (59.8 bits)

GCOS(x) vs $4 \cdot GCOS(x/3)^3 - 3 \cdot GCOS(x/3)$
test interval: 0.2199×10^{2} through 0.2356×10^{2}
MRE: 0.4779×10^{-17} (57.5 bits)
RMS: 0.1515×10^{-17} (59.2 bits)

GCOSH(x) vs C•(GCOSH(x+1)+GCOSH(x–1))
test interval: 3.0000 through 0.7091×10^{3}
MRE: 0.4770×10^{-17} (57.5 bits)
RMS: 0.1712×10^{-17} (59.0 bits)

GCOSH(x) vs Taylor Series expansion of GCOSH(x)
test interval: 0.0000 through 0.5000
MRE: 0.3469×10^{-17} (58.0 bits)
RMS: 0.1234×10^{-17} (59.5 bits)

GCOT(x) vs $(GCOT(x/2)^2 - 1)/(2 \cdot GCOT(x/2))$
test interval: 0.1885×10^{2} through 0.1963×10^{2}
MRE: 0.7609×10^{-17} (56.9 bits)
RMS: 0.2096×10^{-17} (58.7 bits)

GEXP(x−2.8125) vs GEXP(x)/GEXP(2.8125)
test interval: 6.9310 through 0.7090×10^{3}
MRE: 0.4706×10^{-17} (57.6 bits)
RMS: 0.1391×10^{-17} (59.3 bits)

GEXP(x−2.8125) vs GEXP(x)/GEXP(2.8125)
test interval: −3.4660 through -0.6682×10^{3}
MRE: 0.4690×10^{-17} (57.6 bits)
RMS: 0.1395×10^{-17} (59.3 bits)

GEXP(x−0.0625) vs GEXP(x)/GEXP(0.0625)
test interval: −0.2841 through 0.3466
MRE: 0.3469×10^{-17} (58.0 bits)
RMS: 0.1384×10^{-17} (59.3 bits)

GEXP3. ($x^{1.0}$ vs x)
test interval: 0.5000 through 1.0000
The result is exact.

GEXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 0.5000 through 1.0000
MRE: 0.3464×10^{-17} (58.0 bits)
RMS: 0.1334×10^{-17} (59.4 bits)

GEXP3. ($XSQ^{1.5}$ vs XSQ•x)
test interval: 1.0000 through 0.4479×10^{103}
MRE: 0.3464×10^{-17} (58.0 bits)
RMS: 0.1347×10^{-17} (59.4 bits)

GEXP3. (x^{y} vs $XSQ^{y/2}$)
test interval: 1.0000 through 0.1000×10^{2} for x
-0.1543×10^{3} through 0.1543×10^{3} for y
MRE: 0.3371×10^{-16} (54.7 bits)
RMS: 0.4759×10^{-17} (57.5 bits)

GLOG(x) vs Taylor Series expansion of GLOG(1+y)
test interval: $1-0.1907 \times 10^{-5}$ through $1+0.1907 \times 10^{-5}$
MRE: 0.5771×10^{-17} (57.3 bits)
RMS: 0.1557×10^{-17} (59.2 bits)

GLOG(x) vs GLOG(17x/16)−GLOG(17/16)
test interval: 0.7071 through 0.9375
MRE: 0.3501×10^{-17} (58.0 bits)
RMS: 0.1488×10^{-17} (59.2 bits)

GLOG(x•x) vs 2•GLOG(x)
test interval: 0.1600×10^{2} through 0.2400×10^{3}
MRE: 0.3393×10^{-17} (58.0 bits)
RMS: 0.4781×10^{-18} (60.9 bits)

GLOG10(x) vs GLOG10(11x/10)−GLOG10(11/10)
test interval: 0.3162 through 0.9000
MRE: 0.9112×10^{-17} (56.6 bits)
RMS: 0.2560×10^{-17} (58.4 bits)

GSIN(x) vs $3 \cdot GSIN(x/3) - 4 \cdot GSIN(x/3)^3$
test interval: 0.0000 through 1.5710
MRE: 0.3794×10^{-17} (57.9 bits)
RMS: 0.1394×10^{-17} (59.3 bits)

GSIN(x) vs $3 \cdot GSIN(x/3) - 4 \cdot GSIN(x/3)^3$
test interval: 0.1885×10^2 through 0.2042×10^2
MRE: 0.5320×10^{-17} (57.4 bits)
RMS: 0.1719×10^{-17} (59.0 bits)

GSINH(x) vs $C \cdot (GSINH(x+1) + GSINH(x-1))$
test interval: 3.0000 through 0.7091×10^3
MRE: 0.5035×10^{-17} (57.5 bits)
RMS: 0.1730×10^{-17} (59.0 bits)

GSINH(x) vs Taylor Series expansion of GSINH(x)
test interval: 0.0000 through 0.5000
MRE: 0.3459×10^{-17} (58.0 bits)
RMS: 0.2973×10^{-18} (61.5 bits)

$GSQRT(x \cdot x) - x$
test interval: 0.7071 through 1.0000
MRE: 0.2450×10^{-17} (58.5 bits)
RMS: 0.6269×10^{-18} (60.5 bits)

$GSQRT(x \cdot x) - x$
test interval: 1.0000 through 1.4140
The result is exact.

GTAN(x) vs $2 \cdot GTAN(x/2)/(1-GTAN(x/2)^2)$
test interval: 2.7490 through 3.5340
MRE: 0.6827×10^{-17} (57.0 bits)
RMS: 0.2028×10^{-17} (58.8 bits)

GTAN(x) vs $2 \cdot GTAN(x/2)/(1-GTAN(x/2)^2)$
test interval: 0.1885×10^2 through 0.1963×10^2
MRE: 0.9834×10^{-17} (56.5 bits)
RMS: 0.2760×10^{-17} (58.3 bits)

GTAN(x) vs $2 \cdot GTAN(x/2)/(1-GTAN(x/2)^2)$
test interval: 0.0000 through 0.7854
MRE: 0.9663×10^{-17} (56.5 bits)
RMS: 0.2678×10^{-17} (58.4 bits)

GTANH(x) vs (GTANH(x-1/8)+GTANH(1/8))/(1+GTANH(x-1/8)GTANH(1/8))
test interval: 0.1250 through 0.5493
MRE: 0.4684×10^{-17} (57.6 bits)
RMS: 0.1608×10^{-17} (59.1 bits)

GTANH(x) vs (GTANH(x-1/8)+GTANH(1/8))/(1+GTANH(x-1/8)GTANH(1/8))
test interval: 0.6743 through 2149×10^2
MRE: 0.3750×10^{-17} (57.9 bits)
RMS: 0.1621×10^{-17} (59.1 bits)

SIN(x) vs 3 SIN(x/3)-4•SIN(x/3)^3

test interval: 0.0000 through 1.5710

MRE: 0.1934×10^{-7} (25.6 bits)

RMS: 0.5980×10^{-8} (27.3 bits)

SIN(x) vs 3•SIN(x/3)-4•SIN(x/3)^3

test interval: 0.1885×10^{2} through 0.2042×10^{2}

MRE: 0.2736×10^{-7} (25.1 bits)

RMS: 0.6923×10^{-8} (27.1 bits)

SINH(x) vs C•(SINH(x+1)+SINH(x-1))

test interval: 3.0000 through 0.8803×10^{2}

MRE: 0.3020×10^{-7} (25.0 bits)

RMS: 0.7083×10^{-8} (27.1 bits)

SINH(x) vs Taylor Series expansion of SINH(x)

test interval: 0.0000 through 0.5000

MRE: 0.1479×10^{-7} (26.0 bits)

RMS: 0.1143×10^{-8} (29.7 bits)

SQRT(x•x)-x

test interval: 0.7071 through 1.0000

The result is exact.

SQRT(x•x)-x

test interval: 1.0000 through 1.4140

The result is exact.

TAN(x) vs 2•TAN(x/2)/(1-TAN(x/2)^2)

test interval: 0.1885×10^{2} through 0.1963×10^{2}

MRE: 0.3059×10^{-7} (25.0 bits)

RMS: 0.1039×10^{-7} (26.5 bits)

TAN(x) vs 2•TAN(x/2)/(1-TAN(x/2)^2)

test interval: 2.7490 through 3.5340

MRE: 0.2940×10^{-7} (25.0 bits)

RMS: 0.7439×10^{-8} (27.0 bits)

TAN(x) vs 2•TAN(x/2)/(1-TAN(x/2)^2)

test interval: 0.0000 through 0.7854

MRE: 0.2994×10^{-7} (25.0 bits)

RMS: 0.1074×10^{-7} (26.5 bits)

TANH(x) vs (TANH(x-1/8)+TANH(1/8))/(1+TANH(x-1/8)TANH(1/8))

test interval: 0.1250 through 0.5493

MRE: 0.2020×10^{-7} (25.6 bits)

RMS: 0.6944×10^{-8} (27.1 bits)

TANH(x) vs (TANH(x-1/8)+TANH(1/8))/(1+TANH(x-1/8)TANH(1/8))

test interval: 0.6743 through 0.1040×10^{2}

MRE: 0.2156×10^{-7} (25.5 bits)

RMS: 0.6360×10^{-8} (27.2 bits)

Appendix B
Using the Common Math Library with MACRO Programs

The Math Library was designed to be used mainly by compiler-level languages. The object-time systems of such languages have facilities to handle error conditions that may occur when a routine from the Math Library is executed. MACRO programmers must include such facilities in their programs.

There are two facilities necessary for use of the Math Library: a trap handler and an error handler. The trap handler is needed, since under certain circumstances the Math Library executes floating-point instructions which may overflow or underflow. In these cases, the library routines expect that the result will be set to the largest possible number for floating overflow, or set to zero for underflow. The central processor does not set the results — the overflows and underflows must be detected by the APR trapping system and interpreted by the trap handler. If the overflow/underflow settings are not done properly, the math routine in question will very likely return mathematically incorrect results.

The error handler is a general error printout routine. It is called by the Math Library when the arguments passed to a Math Library routine are out of range or otherwise incorrect.

Provided with the Math Library are modules for handling APR traps and properly setting the results (MTHTRP) and for providing error handling and reporting (MTHDUM). A MACRO program must initialize these modules before using any other components of the Math Library, as follows:

```
PUSHJ    P,%TRPIN##    ;INITIALIZE TRAP HANDLER
PUSHJ    P,%ERINI##    ;INITIALIZE ERROR HANDLER
```

Index

D

H

I

J

K

L

M

T

TOPS-10/TOPS-20
Common Math Library
Reference Manual
AA-M400A-TK

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____

Name ____ Date ____

Organization ____ Telephone ____

Street ____

City ____ State ____ Zip Code or Country ____

Do Not Tear – Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1–2/L12
MARLBOROUGH, MA 01752

Do Not Tear – Fold Here and Tape

Cut Along Dotted Line