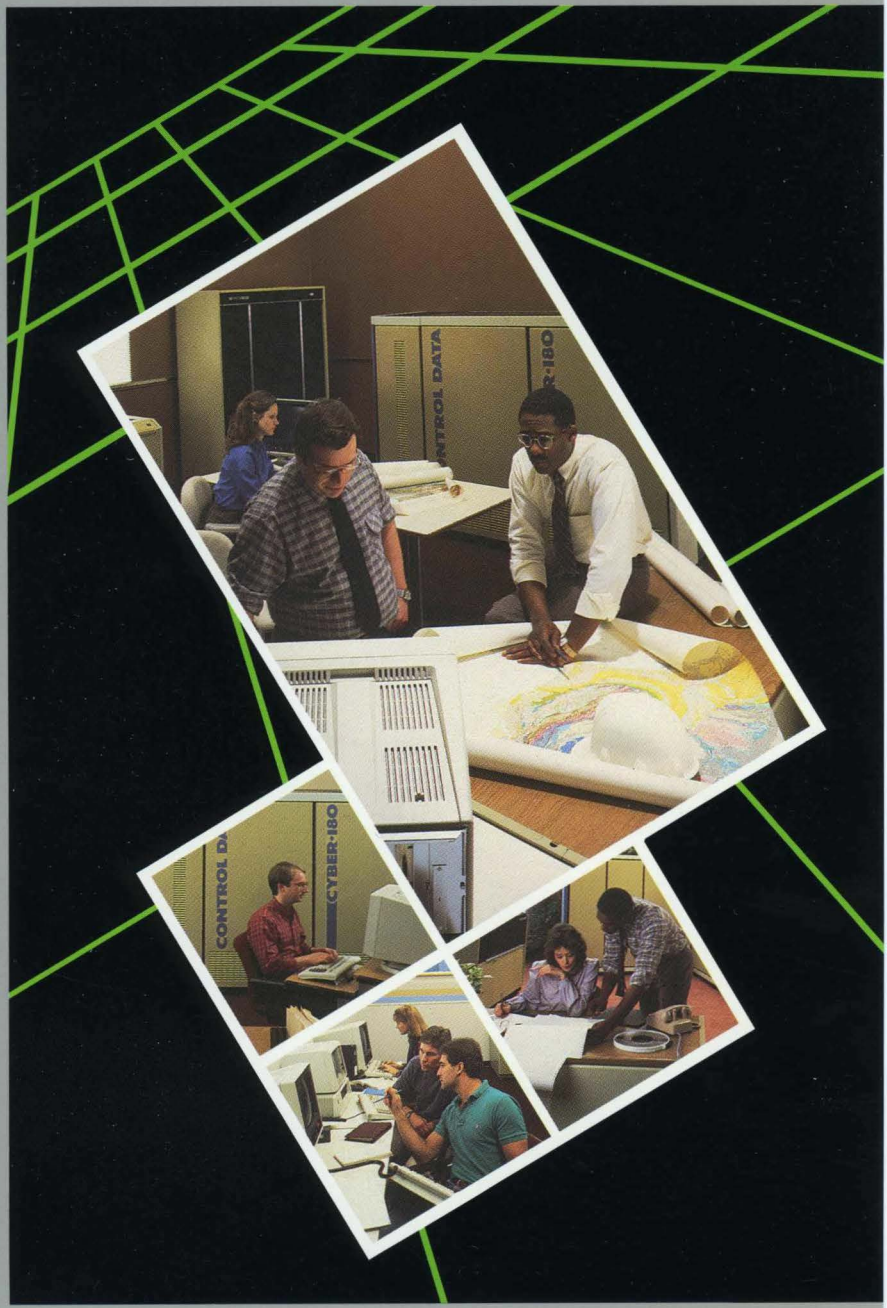
Math Library for NOS/VE

CONTROL DATA



Usage

60486513

Math Library for NOS/VE

Usage

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

Publication Number 60486513

Manual History

Revisions	System Version/ PSR Level	Product Version	Date
A	1.0.2/598	1.0	October 1983
B	1.1.2/630	1.0	April 1985
C	1.1.4/649	1.0	January 1986
D	1.2.1/664	1.0	October 1986
E	1.2.2/678	1.0	April 1987
F	1.2.3/688	1.0	September 1987
G	1.3.1/700	1.0	April 1988

This revision:

Revision G documents the Math Library for NOS/VE Usage Version 1 at release level 1.3.1, PSR level 700. This revision documents algorithm enhancements to the ALOG, ALOG10, COS, EXP, SQRT, SIN, and SINCOS routines. It was published in April 1988.

Contents

Contents

About This Manual

About This Manual

The Math Library usage manual describes the mathematical routines available in the Math Library. These routines can be accessed by Ada, FORTRAN Version 1, FORTRAN Version 2, CYBIL, BASIC, PASCAL, and APL programs. The Math Library is available under the CONTROL DATA Network Operating System/Virtual Environment (NOS/VE) operating system.

Audience

You should be familiar with one of the above programming languages and with the NOS/VE operating system. In addition, you should understand basic numerical techniques.

Organization

The Math Library usage manual is organized into the following chapters:

Chapter 1 - Introduction

Describes the external characteristics of each function. These include the number types used, the calling procedures, and the error handling used by these procedures.

Chapter 2 - Routine Descriptions

Presents detailed information about each mathematical routine.

Chapter 3 - Auxiliary Routines

Presents detailed information on auxiliary routines that are called only by other math routines.

Additional information is available to you in the following appendixes:

A - Glossary

Defines commonly-used terms and phrases.

B - Error Handling

Explains the Math Library error processing procedure.

Conventions

Certain notations are used throughout the manual with consistent meaning. The notations are:

...	In formulas, a horizontal ellipses indicates that the preceding item can be repeated as necessary.
*	In formulas, an asterisk indicates multiplication.
**	In formulas, two successive asterisks indicate exponentiation.
| |	In formulas, vertical bars indicate the absolute value of the quantity.
( )	In intervals, parentheses indicate an open interval (the end points are not included).
[ ]	In intervals, brackets indicate a closed interval (the end points are included).
( ]	In intervals, closure by a left parenthesis and a right bracket includes the right end point, but not the left end point.
[ )	In intervals, closure by a left bracket and a right parenthesis includes the left end point, but not the right end point.
▌	Vertical bars in the margin indicate changes or additions to the text from the previous revision.
●	A dot next to the page number indicates that a significant amount of text (or the entire page) has changed from the previous revision.

All numbers used in this manual are decimal unless otherwise indicated. Other number systems are indicated by a notation after the number (for example, FA34 hexadecimal).

All references to logarithm (log) are base e unless otherwise indicated.

All references to infinite values include positive and negative infinity unless otherwise indicated.

Submitting Comments

The last page of this manual is a comment sheet. Please use it to give us your opinion of the manual's usability, to suggest specific improvements, and to report technical or typographical errors. If the comment sheet has already been used, you can mail your comments to:

Control Data Corporation
Technology and Publications Division
P. O. Box 3492
Sunnyvale, California 94088-3492

Please indicate whether you would like a written response.

Also, if you have access to SOLVER, an online facility for reporting problems, you can use it to submit comments about the manual. Use FN8 as the product identifier for problems that are related to FORTRAN and FV8 as the product identifier for problems related to FORTRAN Version 2.

In Case of Trouble

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help beyond that provided in the documentation or find that the product does not perform as described, call us at one of the following numbers and a support analyst will work with you.

From the USA and Canada: (800) 345-9903

From other countries: (612) 851-4131

The preceding numbers are for help on product usage. Address questions about the physical packaging and/or distribution of printed manuals to Literature and Distribution Services at the following address:

Control Data Corporation
Literature and Distribution Services
308 North Date Street
St. Paul, Minnesota 55103

or you can call (612) 292-2101. If you are a Control Data employee, call CONTROLNET® 243-2100 or (612) 292-2100.

Introduction 1

This section describes the external characteristics of each function. These include the number types used, the calling procedures, and the error handling method used by these procedures.

Introduction 1

This manual describes the math functions found in the math library. The math library contains mathematical functions and exponentiation routines. Table 1-1 gives a summary of the mathematical functions. Table 1-2 shows the input domain and output range for a subset of the mathematical functions. Table 1-3 shows a summary of the exponentiation routines of the math library. Tables 1-4, 1-5, 1-6, and 1-7 show the parameter lists for the vector mathematical functions.

The math library routines can be called directly from Ada, FORTRAN Version 1, FORTRAN Version 2, CYBIL, BASIC, Pascal, and APL programs. Some functions cannot be called directly from CYBIL programs. These functions are:

CABS	DATAN2	DSIGN
CCOS	DCOS	DSIN
CEXP	DCOSH	DSINH
CLOG	DDIM	DSNCOS
CONJG	DEXP	DSQRT
CSQRT	DINT	DTAN
DABS	DLOG	DTANH
DACOS	DLOG10	DTOD
DASIN	DMOD	DTOI
DASNCS	DNINT	DTOX
DATAN	DPROD	DTOZ

Number Types

The math routines perform computations on four number types: integer, single precision floating point (real), double precision floating point (long real), and complex floating point.

Integer

An integer is a one-word, right-justified, two's complement 64-bit representation of all integers from -2**63 to 2**63 - 1. All integers are considered standard forms.

Single Precision

A single precision floating point number consists of a sign bit, S, which is the sign of the fraction, a signed biased exponent (15 bits), and a fraction (48 bits) which is also called a coefficient or a mantissa.

Single precision floating point numbers consist of two types: standard and nonstandard. Standard numbers are numbers that have exponents in the range of 3000 hexadecimal...4FFF hexadecimal, inclusive, and have a nonzero fraction. Standard numbers can be normalized or unnormalized. A normalized standard number has a one in bit position 16, (the most significant bit of the fraction) where bit position zero is the left-most bit.

The range in magnitude, M, covered by standard, normalized single precision numbers is:

$$2^{-4097} \leq M \leq (1-2^{-48}) * 2^{4095}$$

(Approximately 14.4 decimal digits of precision.)

Nonstandard floating point numbers have many representations:

A floating point number with an exponent in the range 5000 hexadecimal...6FFF hexadecimal

A floating point number with an exponent in the range 7000 hexadecimal...7FFF hexadecimal

A floating point number with an exponent in the range 0000 hexadecimal...0FFF hexadecimal

A floating point number with an exponent in the range 1000 hexadecimal...2FFF hexadecimal

An unnormalized floating point number with a zero fraction and a standard exponent

A sign bit followed by 63 zero bits

Double Precision

A double precision floating point number consists of two words, each a single precision floating point number. The coefficient of the second word is considered to be an extension of the fraction of the first word, yielding a 96-bit fraction. The exponent of the second word following an arithmetic operation is identical to that of the first word. The number type of the first single number determines the type of the double number.

The range in magnitude, M, covered by standard, normalized double precision numbers is:

$$2^{**}-4097 \leq M \leq (1-2^{**}-96) * 2^{**}4095$$

(Approximately 28.9 decimal digits of precision.)

Complex

A complex number consists of two words, each a single precision floating point number. The first word represents the real part of the complex number, and the second word represents the imaginary part.

A complex number is considered to be indefinite if either the real or imaginary part is indefinite. Similarly, a complex number is considered to be infinite if either the real or imaginary part is infinite.

General Rules

In the following routines, two rules apply to the use of these number forms in computation:

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

2. Unless otherwise documented, if a nonstandard number other than zero is used in a computation, or if the limits to a standard form of a number type are exceeded, error handling will occur.

Rule 1 does not hold if the answer computed exceeds the range of values for standard numbers, or if a mathematically invalid operation is attempted.

Rule 2 does not hold when various nonstandard numbers are within the domain of the function.

Most of the routines have a default error value of positive indefinite. The following routines have a default error value of zero: CCOS, DEXP, ERFC, EXP IDIM, IDNINT, ITOI, MOD, and NINT.

Routines and Calls

The math functions are predefined routines that can be called directly from an Ada, APL, BASIC, CYBIL, FORTRAN Version 1, FORTRAN Version 2, or Pascal program. (See page 1-1 for a list of functions that cannot be called from a CYBIL program.) These functions return a single value to the calling program. There are two types of calling procedures: call-by-reference and call-by-value.

A math function can be called from a FORTRAN program by reference or by value. Use the EE=R parameter in the FORTRAN or VECTOR_FORTRAN command to access the function through the call-by-reference calling procedure. If you do not use this parameter, the program accesses the math function through the call-by-value calling procedure. Ada and CYBIL programs access the math functions through the call-by-reference calling procedure only. APL, BASIC, and Pascal programs access the math functions through the call-by-value calling procedures only.

In a call-by-reference computation, a parameter list is formed in memory and the first-word-address of this list is stored in register A4 before the routine is invoked. The call-by-reference routine is called through one of two entry points. Argument error processing is set up in this routine. If the argument list is valid, the routine calls or branches to the call-by-value routine, depending on the function. This routine performs the appropriate computation. If the argument list is invalid, the call-by-value routine is not called or branched to.

In a call-by-value computation, the arguments are entered directly into the X registers before the routine is invoked. The first word of the first argument is entered into register X2, and the remaining words of each argument are entered into the registers successively. For example, the calling procedure for the exponential routine ITOD uses registers X2, X3, and X4. Register X2 holds the integer base, and registers X3 and X4 hold the double precision exponent.

The first and second words of a complex argument contain the real and imaginary parts, respectively. The first and second words of a double precision argument contain the high-order and low-order bits, respectively.

If the call-by-value routine is called directly, and if the arguments are out-of-range, the job will abort during the computation. When valid computations occur in both call-by-reference and call-by-value routines, the result is returned in registers XE and XF. One-word results (type integer and single precision) are returned in XF. Two-word results (type double precision and complex) are returned in registers XE and XF, with the second word being in XF.

Table 1-1. Mathematical Functions

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Absolute value	|x|; if x is complex, square root of ((real x)**2 + (imag x)**2)	ABS CABS DABS IABS	real complex double integer	1	real real double integer
Inverse cosine	arccos(x)	ACOS DACOS	real double	1	real double
Imaginary part of a complex argument	Imaginary part of (xr,xi) = xi	AIMAG	complex	1	real
Truncation	int(x)	AINT DINT	real double	1	real double
Natural logarithm	log e (x)	ALOG CLOG DLOG	real complex double	1	real complex double
Common logarithm	log 10 (x)	ALOG10 DLOG10	real double	1	real double
Remaindering	x - int(x/y)*y	AMOD DMOD MOD	real double integer	2	real double integer
Nearest whole number	int(x + 0.5), if x $\geq$ 0 int(x - 0.5), if x $<$ 0	ANINT DNINT	real double	1	real double
Inverse sine or cosine	arcsin(x) arccos(x)	ACOSIN	real	1	real
Inverse sine	arcsin(x)	ASIN DASIN	real double	1	real double
Inverse tangent	arctan(x)	ATAN DATAN	real double	1	real double
Cosine	cos(x), where x is in radians	CCOS COS DCOS	complex real double	1	complex real double
Conjugate	Negation of imaginary part (xr,-xi)	CONJG	complex	1	complex
Cosine	cos(x), where x is in degrees	COSD	real	1	real
Cotangent	cotan(x), where x is in radians	COTAN	real	1	real

(Continued)

Table 1-1. Mathematical Functions (Continued)

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Exponential	e**x	CEXP DEXP EXP	complex double real	1	complex double real
Hyperbolic cosine	cosh(x)	COSH DCOSH	real double	1	real double
Sine and cosine simultaneously	cossin(x)	COSSIN	real	1	real
Sine	sin(x), where x is in radians	CSIN DSIN SIN	complex double real	1	complex double real
Square root	x**(1/2)	CSQRT DSQRT SQRT	complex double real	1	complex double real
Inverse hyperbolic tangent	arctanh(x)	ATANH	real	1	real
Inverse tangent	arctan(y/x)	ATAN2 DATAN2	real double	2	real double
Inverse sine or cosine	arcsin(x) arccos(x)	DASNCS	double	1	double
Positive difference	x - y, if x > y 0, if x $\leq$ y	DDIM DIM IDIM	double real integer	2	double real integer
Product	x*y	DPROD	real	2	double
Transfer of sign	$\lvert x\rvert$, if y $\geq$ 0 $-\lvert x\rvert$, if y $<$ 0	DSIGN ISIGN SIGN	double integer real	2	double integer real
Hyperbolic sine	sinh(x)	DSINH SINH	double real	1	double real
Hyperbolic sine and cosine simultaneously	hyperb(x)	HYPERB	real	1	real
Trigonometric sine or cosine	sin(x) cos(x)	DSNCOS	double	1	double
Tangent	tan(x), where x is in radians	DTAN TAN	double real	1	double real

(Continued)

Table 1-1. Mathematical Functions (Continued)

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Hyperbolic tangent	tanh(x)	DTANH TANH	double real	1	double real
Error function	erf(x)	ERF	real	1	real
Complementary error function	1 - erf(x)	ERFC	real	1	real
Extract bits	extb(x, i1, i2); extracts bits from x starting with position i1 with length of i2	EXTB	x: boolean complex double integer logical real i1: integer i2: integer	3	boolean
Nearest integer	int(x + 0.5), if x $\geq$ 0 int(x - 0.5), if x $<$ 0	IDNINT NINT	double real	1	integer integer
Insert bits	insb(x, i1, i2, y) inserts bits from x starting with position i1 with length of i2 into copy of y	INSB	x,y: boolean complex double integer logical real i1: integer i2: integer	4	boolean
Random number generator	Random number in range (0,1)	RANF	none	0	real
Returns random number seed	Seed is in range (0,1)	RANGET	real	1	real
Sets seed for random number generator	ranset(x)	RANSET	real	1	real
Sine and cosine	sin(x) & cos(x), where x is in degrees	SINCSD	real	1	real
Trigonometric sine or cosine	sin(x) cos(x)	SINCOS	real	1	real
Sine	sin(x), where x is in degrees	SIND	real	1	real

(Continued)

Table 1-1. Mathematical Functions (Continued)

Description	Definition	Function Name	Type of Argument	Number of Arguments	Type of Result
Sum of bits	sum1s(x)	SUM1S	boolean complex double integer real	1	integer
Tangent	tan(x), where x is in degrees	TAND	real	1	real

Table 1-2. Input Domains and Output Ranges

Function	Input Domain	Output Range
ACOS(x)	|x| ≤ 1	0 ≤ ACOS(x) ≤ pi
DACOS(x)	|x| ≤ 1	0 ≤ DACOS(x) ≤ pi
ACOSIN(x)	|x| ≤ 1	0 ≤ ACOSIN(x) ≤ pi
ALOG(x)	x > 0	|ALOG(x)| < 4095*log(2)
CLOG(xr,xi)	(xr, xi) ≠ (0,0) (xr**2 + xi**2)**1/2 in machine range	-pi < CLOG(xi) ≤ pi
DLOG(x)	x > 0	|DLOG (x)| < 4095*log(2)
ALOG10(x)	x > 0	|ALOG10(x)| < 4095*log(2) base 10
DLOG10(x)	x > 0	|DLOG10(x)| < 4095*log(2) base 10
ASIN(x)	|x| ≤ 1	-pi/2 ≤ ASIN(x) ≤ pi/2
DASIN(x)	|x| ≤ 1	-pi/2 ≤ DASIN(x) ≤ pi/2
ATAN(x)	-infinity ≤ x ≤ infinity	-pi/2 ≤ ATAN(x) ≤ pi/2
DATAN(x)	-infinity ≤ x ≤ infinity	-pi/2 ≤ DATAN(x) ≤ pi/2
ATAN2(x,y)	y < 0, x < 0	-pi < ATAN2(x,y) < - pi/2
	y < 0, x ≥ 0	pi/2 ≤ ATAN2(x,y) ≤ pi
	y = 0, x < 0	ATAN2(x,y) = -pi/2
	y = 0, x > 0	ATAN2(x,y) = pi/2
	y > 0, x < 0	-pi/2 < ATAN2(x,y) < 0
	y > 0, x ≥ 0	0 ≤ ATAN2(x,y) < pi/2
DATAN2(x,y)	y < 0, x < 0	-pi < DATAN2(x,y) < - pi/2
	y < 0, x ≥ 0	pi/2 ≤ DATAN2(x,y) ≤ pi
	y = 0, x < 0	DATAN2(x,y) = -pi/2
	y = 0, x > 0	DATAN2(x,y) = pi/2
	y > 0, x < 0	-pi/2 < DATAN2(x,y) < 0
	y > 0, x ≥ 0	0 ≤ DATAN2(x,y) < pi/2
ATANH(x)	|x| < 1	
COS(x)	|x| < 2**47	-1 ≤ COS(x) ≤ 1
CCOS(xr,xi)	|xr| < 2**47 |xi| < 4095*log(2)	-1 ≤ CCOS(x) ≤ 1
DCOS(x)	|x| < 2**47	-1 ≤ DCOS(x) ≤ 1

(Continued)

Table 1-2. Input Domains and Output Ranges (Continued)

Function	Input Domain	Output Range
COSD(x)	|x| < 2**47	-1 ≤ COSD(x) ≤ 1
COSH(x) DCOSH(x)	|x| < 4095*log(2) |x| < 4095*log(2)	COSH(x) ≥ 1 DCOSH(x) ≥ 1
COSSIN(x)	|x| < 2**47	-1 ≤ COSSIN(x) ≤ 1
COTAN(x)	0 < |x| < 2**47	
DASNCS(x)	|x| ≤ 1	0 ≤ DASNCS(x) -pi/2 ≤ DASNCS(x) ≤ pi/2
DSNCOS(x)	|x| < 2**47	-1 ≤ DSNCOS(x) ≤ 1
ERF(x)	-infinity ≤ x ≤ infinity	-1 ≤ ERF(x) ≤ 1
ERFC(x)	-infinity ≤ x ≤ 53.037	0 ≤ ERFC(x) ≤ 2
EXP(x) CEXP(xr,xi) DEXP(x)	x < 4095*log(2) & x ≥ 4097*log(2) xr < 4095*log(2) & xi ≥ -4097*log(2) x < 4095*log(2) & x ≥ -4097*log(2)	0 < EXP(x)
HYPERB	|x| < 2**47	-1 ≤ HYPERB(x) ≤ 1
SIN(x) CSIN(xr,xi) DSIN(x)	|x| < 2**47 |xr| < 2**47 |xi| < 4095*log(2) |x| < 2**47	-1 ≤ SIN(x) ≤ 1 -1 ≤ DSIN(x) ≤ 1
SINCOS(x)	|x| < 2**47	-1 ≤ SINCOS(x) ≤ 1
SIND(x)	|x| < 2**47	-1 ≤ SIND(x) ≤ 1

(Continued)

Table 1-2. Input Domains and Output Ranges (Continued)

Function	Input Domain	Output Range
SINH(x) DSINH(x)	|x| < 4095*log(2) |x| < 4095*log(2)	
SQRT(x) CSQRT(xr,xi)	x ≥ 0 (xr**2 + xi**2)**1/2 + |xr| in machine range	SQRT(x) ≥ 0 value in right half of plane CSQRT(xr) ≥ 0)
DSQRT(x)	x ≥ 0	
TAN(x) DTAN(x)	|x| < 2**47 |x| < 2**47	
TAND(x)	|x| < 2**47 x cannot be exact odd multiple of 90	
TANH(x)	-infinity ≤ x ≤ infinity	-1 ≤ TANH(x) ≤ 1

Table 1-3. Exponentiation Routines

Name	Type of Argument	Input Domain	Output Range
DTOD	double double	x ≥ 0; if x = 0 then y > 0	x**y ≥ 0
DTOI	double integer	if x = 0 then y > 0	
DTOX	double real	x ≥ 0; if x = 0 then y > 0	x**y ≥ 0
DTOZ	double complex	if x = 0 then yr > 0, yi = 0	
ITOD	integer double	x ≥ 0; if x = 0 then y > 0	x**y ≥ 0
ITOI	integer integer	if x = 0 then y > 0 |x**y| < 2**63	
ITOX	integer real	if x = 0 then y > 0	x**y ≥ 0
ITOZ	integer complex	if x = 0 then yr > 0, yi = 0	x**y ≥ 0
XTOD	real double	x ≥ 0; if x = 0 then y > 0	x**y ≥ 0
XTOI	real integer	if x = 0 then y > 0	
XTOX	real real	x ≥ 0; if x = 0 then y > 0	x**y ≥ 0
XTOZ	real complex	if x = 0 then yr > 0, yi = 0	
ZTOD	complex double	if (xr,xi) = (0,0) then y > 0	
ZTOI	complex integer	if (xr,xi) = (0,0) then y > 0	
ZTOX	complex real	if (xr,xi) = (0,0) then y > 0	
ZTOZ	complex complex	if (xr,xi) = (0,0) then yr > 0, yi = 0	

Vector Routines

Vector math functions accept vectors as arguments and return vectors as results using vector processing capabilities.

While the vector math functions are available and referenced on any Cyber 180 mainframe model, they function correctly only on models which include vector hardware facilities, currently limited to the Cyber 180/990. If a vector math function is called on a non-vector machine, an unimplemented instruction trap occurs.

The FORTRAN Version 2 compiler guarantees that the length (L) of the vector sent to the Math Library will be within the range $0 \leq L \leq 512$. When the vector length is not within this valid range, an error message is displayed. See the section in this chapter on vector error handling. When the length of the vector argument sent to the CMML vector routine is zero, a no-operation occurs. No-operation means that the contents of the vector are returned without changing any values.

The calling sequence for all vector functions conforms to call-by-reference with one entry point defined for each routine. In all cases, register A4 contains the Process Virtual Address [PVA] to the first entry in the parameter list. Table 1-4 shows the parameter list for real, double precision, and complex single argument vector math functions. The double argument vector math functions are divided into three catagories:

f(scalar, vector)	See table 1-5 for parameter list.
f(vector, scalar)	See table 1-6 for parameter list.
f(vector, vector)	See table 1-7 for parameter list.

where f is a double argument function name, such as, ATAN2.

Table 1-4. Parameter List for Single Argument Vector Math Functions

Word (in Decimal)	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array.
Word 3	Pointer to the result array descriptor.
Word 4	Pointer to the source array descriptor.

Table 1-5. Parameter List for (Scalar, Vector) Functions (double argument vector math functions where argument 1 is scalar and argument 2 is vector)

Word (in Decimal)	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source scalar (argument 1).
Word 3	Pointer to the source array (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	0
Word 6	Pointer to the source array descriptor (argument 2)

Table 1-6. Parameter List for (Vector, Scalar) Functions (double argument vector math functions where argument 1 is vector and argument 2 is scalar)

Word (in Decimal)	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array (argument 1).
Word 3	Pointer to the source scalar (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	Pointer to the source array descriptor (argument 1)
Word 6	0

Table 1-7. Parameter List for (Vector, Vector) Functions (double argument vector math functions where arguments 1 and 2 are vector)

Word (in Decimal)	Description of Contents
Word 1	Pointer to the result array.
Word 2	Pointer to the source array (argument 1).
Word 3	Pointer to the source array (argument 2).
Word 4	Pointer to the result array descriptor.
Word 5	Pointer to the source array descriptor (argument 1)
Word 6	Pointer to the source array descriptor (argument 2)

Routine Error

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

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

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

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

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

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

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

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

The first method is used for single precision algorithms accurate to less than 2E-15, and round-off errors less than 10E-15. Changing the last bit in a single precision number produces a relative change of between 3.5E-15 for a large mantissa, and 7.1E-15 for a small but normalized mantissa. This method is used for the error analysis of the math library routines.

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

Error Handling

Error handling takes place when the argument or result is outside the range of the function. There are two modes of error processing, depending on whether the calling sequence was call-by-reference or call-by-value.

If you are accessing the math library in a language other than FORTRAN, you might want to establish a condition handler to be used in conjunction with the error handling mechanism under the call-by-reference routine. The math library automatically establishes this condition handler for FORTRAN programs. Refer to appendix C for a more complete description of the math library error handling process.

Call-By-Reference Error Handling

When the argument or result is out-of-range in a call-by-reference routine, an error message is displayed and the corresponding default error value is placed in the result registers(s) XE and XF.

Call-By-Value Error Handling

If the call-by-value routine is called directly, that is, if the call-by-reference routine is not called, the job will abort if the following situations occur:

An out-of-range argument is passed to the call-by-value routine.

The result of the computation in a call-by-value routine is out-of-range.

The call-by-value routine does not guarantee any other type of error handling, and the values in registers XE and XF are undefined unless otherwise specified.

Vector Error Handling

The vector math functions use call-by-reference error handling. For example, if an argument within a set of arguments is illegal or produces an out-of-range value, an error message is displayed for that argument. The first argument in error is supplied in the error message. The default error value (usually an indefinite value indicated by +IND) is placed in the result location corresponding to the argument in error within the set. Processing continues and correct results are generated for all arguments which are not in error. However, once an argument is found to be in error, further arguments which are in error are not detected and results are not guaranteed.

Routine Descriptions 2

This chapter describes each of the math routines in detail.

Routine Descriptions 2

The description of each routine is discussed on the following pages. The routines are organized in alphabetical order. Each description includes:

Entry points for the call-by-reference, call-by-value, and vector routines

Input domains and output ranges for the arguments in each routine

Conditions that cause an argument to be invalid, which results in an error

Formulas used to compute the result

Error analysis and the effect of argument error

Entry points to the routines are places in the routines where execution can begin. Some routines can evaluate more than one function, and some routines call other routines to compute a portion of the function.

ABS

ABS is a function that computes the absolute value of an argument. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RABS and ABS, and the call-by-value entry point is MLP$VABS.

The input domain is the collection of all valid real quantities. The output range is included in the set of nonnegative real quantities.

Call-By-Reference Routine

No errors are generated by ABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with its sign bit forced positive. The rightmost 63 bits remain the same.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ACOS

ACOS is a function that computes the inverse cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RACOS and ACOS, the call-by-value entry point is MLP$VACOS, and the vector entry point is MLP$ACOSV.

The input domain is the collection of all valid real quantities in the interval [-1.0,1.0]. The output range is included in the set of nonnegative real quantities less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),  x < -.5
arcos(x)  = pi - arcos(-x),  x < -.5
arcsin(x) = x + x**3*s*((w + z -j)*w + a + m/(e - x**2)),
  where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),  -.5 <= x < .5
arcsin(x) = pi/2 - arcos(x),  .5 <= x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,  .5 <= x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
        w  = (x**2 - c)*z + k
        z  = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =   3.173 170 078 537 13
e =   1.160 394 629 739 02
m =  50.319 055 960 798 3
c =  -2.369 588 855 612 88
i =   8.226 467 970 799 17
j = -35.629 481 597 455 5
k =  37.459 230 925 758 2
a = 349.319 357 025 144
s =    .746 926 199 335 419*10**-3
```

The approximation of arcsin(-.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

a. If ACOS entry, go to step g.

b. If |x| >= .5, go to step h.

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

d. z = (y + r)*y + i
   w = (y - c)*z + k
   p = q + s*q*y*((w + z - j)*w + a + m/(e - y))
   p = u - p
   Y1 = p/2**n

e. If ASIN entry, go to step k.

f. If x is in (-.5,1.0), return.
   XF = 2*u - (Y1)
   Return.

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

h. If x = 1.o or -1.0, go to step l.
   If x is invalid, go to step m.
      n = 0 (Loop counter).
      y = 1.0 - |x|, and normalize y.

i. h = 4*y - 2*y**2
   n = n + 1.0
   If 2*y<= 2 - sqrt(3) = .267949192431, y = h, and go to step i.

j. q = 1.0 - h, and normalize q.
   y = q**2
   u = pi/2
   Go to step d.

k. Y1 = u - (Y1), and normalize Y1.
   Affix sign of x to Y1 = XF.
   Return.

l. XF = pi/2, if x = 1.0.
   XF = -pi/2, if x = -1.0.
   If ASIN entry, return.
   XF = 0, if x = 1.0.
   XF = pi, if x = -1.0.
   Return.

m. Return.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The maximum absolute value of relative error of the ACOS approximation over (-.5,.5) is 1.996*E-15.

The function ACOS was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-1 shows a summary of these statistics.

Table 2-1. Relative Error of ACOS

Interval			
From	To	Maximum	Root Mean Square
-.1250E+00	.1250E+00	.4916E-14	.3233E-14
-.1000E+01	-.7500E+00	.5875E-14	.2068E-14
.7500E+00	.1000E+01	.1987E-13	.7749E-14

Effect of Argument Error

If a small error e´ occurs in the argument x, the error in the result is given approximately by e´/(1.0 - x**2)**.5.

AIMAG

AIMAG is a function that returns the imaginary part of an argument. It accepts a complex argument and returns a real result.

The call-by-reference entry points are MLP$RAIMAG and AIMAG, and the call-by-value entry point is MLP$VAIMAG.

The input domain is the collection of all valid complex quantities. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

No errors are generated by AIMAG. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The imaginary part of the complex argument is returned. The real part of the argument is not used.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

AINT

AINT is a function that returns an integer part of an argument after truncation. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RAINT and AINT, and the call-by-value entry point is MLP$VAINT.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument is added to a special floating point zero that forces truncation. The result is returned.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ALOG

ALOG is a function that computes the natural logarithm function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RALOG and ALOG, the call-by-value entry point is MLP$VALOG, and the vector entry point is MLP$ALOGV.

The input domain is the collection of all valid, positive real quantities. The output range is included in the set of valid real quantities whose absolute value is less than 4095*log(2).

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [1, 2) and n an integer such that x = y*2**n. Log(x) is evaluated by:

```
log(x) = log(y) + n*log(2)
```

To evaluate log(y), the interval [1, 2) is divided into 33 subintervals such that on each the abs(t) < 1/129 where t = (y - c)/(y + c). To achieve this, the subintervals are offset by 1/64. The subintervals are:

```
[1, 65/64)
[65/64, 67/64)
    .
    .
    .
[125/64, 127,64)
[127/64, 2)
```

Log(y) is then computed using the identity:

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

and the center point c is chosen close to the midpoint of the subinterval containing y, except for the first and last subintervals, where the center points are 1 and 2, respectively. By selecting these center points, it ensures that abs(t) < 1/129.

Log ((1 + t)/(1 - t)) is approximated with a 7th degree min-max polynomial of the form:

```
2*t + c3*t**3 + c5*t**5 + c7*t**7
```

The coefficients are:

```
c3 = .6666666666667
c5 = .3999999995486
c7 = .2857343176917
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-2 shows a summary of these statistics.

Table 2-2. Relative Error of ALOG

Test	Interval From	Interval To	Maximum	Root Mean Square
ALOG(x) against ALOG(17x/16) - ALOG(17/16)	.7071E+00	.9375E+00	.1782E-13	.5463E-14
ALOG(x*x) against 2*LOG(x)	.1600E+02	.2400E+03	.7082E-14	.2035E-14
ALOG(x) against Taylor series expansion of ALOG(1 + y)	1-.1526E-04	1+.1526E-04	.1417E-13	.5197E-14

Total Error

The final calculation of log(x) is done by adding the following terms in the order below to achieve maximum precision:

```
log(x) = n*(log(2) - factor) +
         (((c7*t2 + c5)*t2 + c3)*t2)*t +
         t +
         t +
         (log(c)/2) + (factor/2)*n +
         (log(c)/2) + (factor/2)*n
```

The values of c and log(c)/2 for each subinterval are stored in a table. Factor is the nearest floating point value with 8 bits of precision to log(2). Thus, the single precision representation of log(2) - factor is accurate to 56 bits of precision. The sum log(c) + factor*n is split into two equal parts to provide extra precision during the accumulation of the sum of terms.

Effect of Argument Error

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

ALOG10

ALOG10 is a function that computes the common logarithm function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RALOG10 and ALOG10, the call-by-value entry point is MLP$VALOG10, and the vector entry point is MLP$ALOG10V.

The input domain is the collection of all valid, positive real quantities. The output range is included in the set of valid real quantities whose absolute value is less than 4095*log(2) base 10.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [1, 2) and n an integer such that x = y*2**n. Log10(x) is evaluated by:

log10(x) = log10(y) + n*log10(2)

To evaluate log10(y), the interval [1, 2) is divided into 33 subintervals such that on each the abs(t) < 1/129 where t = (y - c)/(y + c). To achieve this, the subintervals are offset by 1/64. The subintervals are:

[1, 65/64)
[65/64, 67/64)
.
.
.
[125/64, 127,64)
[127/64, 2)

Log10(y) is then computed using the identity:

log10(y) = log10(c) + log10((1 + t)/(1 - t))

and the center point c is chosen close to the midpoint of the subinterval containing y, except for the first and last subintervals, where the center points are 1 and 2, respectively. By selecting these center points, it ensures that abs(t) < 1/129.

Log10((1 + t)/(1 - t)) is approximated with a 7th degree min-max polynomial of the form:

```
C1*t + c3*t**3 + c5 + t**5 + c7**t**7
```

The coefficients are:

```
c1 = .8685889638065
c3 = .2895296546022
c5 = .1737177925653
c7 = .1240928374639
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is less than zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function ALOG10 was tested against ALOG10(11x/10) - ALOG10(11/10). Groups of 2,000 arguments were chosen randomly from the interval [.3162E+00,.9000E+00]. Statistics on relative error were observed: maximum relative error was .3011E-13, root mean square relative error was .8125E-14.

Total Error

The final calculation of log10(x) is done by adding the following terms in the order below to achieve maximum precision:

```
log10(x) = n*(log10(2) - factor) +
           (((c7*t2 + c5)*t2 + c3)*t2 + (c1 - 1))*t +
           t +
           (log10(c) + factor*n)
```

The values of c and log10(c) for each subinterval are stored in a table. Factor is the nearest floating point value with 8 bits of precision to log10(2). Thus, the single precision representation of log10(2) - factor is accurate to 56 bits of precision. The leading coefficient of the approximation is split into 1 and (c1 - 1) to provide extra precision to the min-max polynomial approximation.

Effect of Argument Error

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

AMOD

AMOD is a function that returns the remainder of the ratio of two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RAMOD and AMOD, and the call-by-value entry point is MLP$VAMOD.

The input domain is the collection of all valid real pairs (x,y) such that x/y is a valid real quantity, and y is not equal to 0. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

y is equal to zero.

x/y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Given the argument pair (x,y), the formula used for computation is:

x - aint(x/y)*y

The quotient x/y is added to a special floating point zero that forces truncation, to get n = aint(x/y); then the product of n and y is formed in double precision and subtracted from x in double precision. The most significant word of the result is returned. If the result is nonzero, it has the sign of x.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ANINT

ANINT is a function that returns the nearest whole number to an argument. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RANINT and ANINT, and the call-by-value entry point is MLP$VANINT.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it, and the result is added to a special floating point zero that forces truncation. If the argument is < 0, -.5 is added to it, and the result is added to a special floating point zero that forces truncation.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ASIN

ASIN is a function that computes the inverse sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RASIN and ASIN, the call-by-value entry point is MLP$VASIN, and the vector entry point is MLP$ASINV.

The input domain is the collection of all valid real quantities in the interval [-1.0,1.0]. The output range is included in the set of valid real quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),  x < -.5
arcos(x)  = pi - arcos(-x),  x < -.5
arcsin(x) = x + x**3*s*((w + z -j)*w + a + m/(e - x**2)),
    where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),  -.5 <= x < .5
arcsin(x) = pi/2 - arcos(x),  .5 <= x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,  .5 <= x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
        w = (x**2 - c)*z + k
        z = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =    3.173 170 078 537 13
e =    1.160 394 629 739 02
m =   50.319 055 960 798 3
c =   -2.369 588 855 612 88
i =    8.226 467 970 799 17
j =  -35.629 481 597 455 5
k =   37.459 230 925 758 2
a =  349.319 357 025 144
s =     .746 926 199 335 419*10**-3
```

The approximation of arcsin(-.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

a. If ACOS entry, go to step g.

b. If |x| >= .5, go to step h.

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

d. z = (y + r)*y + i
   w = (y - c)*z + k
   p = q + s*q*y*((w + z - j)*w + a + m/(e - y))
   p = u - p
   Y1 = p/2**n

e. If ASIN entry, go to step k.

f. If x is in (-.5,1.0), return.
   XF = 2*u - (Y1)
   Return.

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

h. If x = 1.o or -1.0, go to step l.
   If x is invalid, go to step m.
      n = 0 (Loop counter).
      y = 1.0 - |x|, and normalize y.

i. h = 4*y - 2*y**2
   n = n + 1.0
   If 2*y<= 2 - sqrt(3) = .267949192431, y = h, and go to step i.

j. q = 1.0 - h, and normalize q.
   y = q**2
   u = pi/2
   Go to step d.

```
k.  Y1 = u - (Y1), and normalize Y1.
    Affix sign of x to Y1 = XF.
    Return.

1.  XF = pi/2, if x = 1.0.
    XF = -pi/2, if x = -1.0.
    If ASIN entry, return.
    XF = 0, if x = 1.0.
    XF = pi, if x = -1.0.
    Return.

m.  Return.
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than 1.0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The maximum absolute value of relative error of the ASIN approximation over (-.5,.5) is 1.996*E-15.

The function ASIN was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-3 shows a summary of these statistics.

Table 2-3. Relative Error of ASIN

Interval			
From	To	Maximum	Root Mean Square
-.1250E+00	.1250E+00	.7101E-14	.2763E-14
.7500E+00	.1000E+01	.8378E-14	.3462E-14

Effect of Argument Error

If a small error e´ occurs in the argument x, the error in the result is given approximately by e´/(1.0 - x**2)**.5.

ATAN

ATAN is a function that computes the inverse tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RATAN and ATAN, the call-by-value entry point is MLP$VATAN, and the vector entry point is MLP$ATANV.

The input domain is the collection of all valid real quantities. The output range is included in the set of valid real quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

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

```
arctan(u) = -arctan(-u), if u < 0
arctan(u) =  pi/4 + (pi/4 - arctan(1/u)), if u ≥ 1.0
arctan(u) =  arctan(k/16) + arctan((u - k/16)/(1.0 +  u*k/16)),
             if 0 ≤ u < 1.0, and k is the greatest integer not
             exceeding 16*u.
```

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

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

where:

```
a(1) = -.333 333 333 333 128 45
a(2) =  .199 999 995 801 446 4
a(3) = -.142 854 130 508 745 0
a(4) =  .110 228 161 612 614 9
```

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

(Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455.)

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-4 shows a summary of these statistics.

Table 2-4. Relative Error of ATAN

Test	Interval From	Interval To	Maximum	Root Mean Square
ATAN(x) against truncated Taylor series	-.6250E-01	.6250E-01	.7102E-14	.3647E-14
2*ATAN(x) against ATAN(2x/(1 - x*x))	.2679E+00 .4142E+00	.4142E+00 .1000E+01	.1355E-13 .1763E-13	.4023E-14 .5931E-14
ATAN(x) against ATAN(1/16) + ATAN((x - 1/16)/ (1 + x/16))	.6250E-01	.2679E+00	.7117E-14	.2605E-14

Effect of Argument Error

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

ATANH

ATANH is a function that computes the inverse hyperbolic tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RATANH and ATANH, the call-by-value entry point is MLP$VATANH, and the vector entry point is MLP$ATANHV.

The input domain is the collection of all valid real quantities whose absolute value is less than 1.0. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument range can be reduced to the interval [0,1.0] by the identity atanh(-x) = -atanh(x). The expression atanh(x) = .5*ln((1.0 + x)/(1.0 - x)) is formed by using the definition tanh(x) = (e**x - e**-x)/(e**x + e**-x).

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

```
atanh(x) = .5*n*ln(2) + .5*ln(2**-(n)*(1.0 + x)/(1.0 - x))
```

The argument range is reduced to the interval [-.2,.2] by writing the argument of log in the form (1.0 + y)/(1.0 - y), and substituting atanh(y):

$$\text{atanh}(x) = .5*n*\ln(2) + \text{atanh}\left[\frac{2^{-n}*(1.0 + x) - (1.0 - x)}{2^{-n}*(1.0 + x) + (1.0 - x)}\right]$$

The value of n such that 2**-n*(1.0 + x)/(1.0 - x) is in the interval [.75,1.5] is the same as the value of n such that 2**-n*(1.0 + x)/(.75*(1.0 - x)) is in the interval [1.0,2.0]. If .75*(1.0 - x) is written as a*2**m, where a is in interval [1.0,2.0], then 2**(-n - m)*(1.0 + x)/a must be in interval [1.0,2.0]. If (1.0 + x) $\geq$ a, then -n - m = 0 and n = -m. If (1.0 + x) < a, then -n - m = 1.0 and n = 1.0 - m.

The function atanh(z) in the interval [-.2,.2] is approximated by z + z**3*p/q, where p and q are 4th order even polynomials. For atanh(z), the coefficients of p and q were derived from the (7th order odd)/(4th order even) minimax (relative error) rational form in the interval [-.2,.2].

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 1.0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

For abs(x) < .2, n equals zero, and the expected bound of the error is 4.8E-15.

For abs(x) $\geq$ .5, the term n*(ln(2)/2) dominates. This term is computed as n*(ln(2)/2 - .125) - n*.125 - n*.125 because the rounding error in representing ln(2)/2 is large; the above form makes the rounding error relatively small. Since n*.125 is exact and the dominating form, the two additions in (other)n*.125 + n*.125 dominate the error, and the expected relative error is 8.3E-15 in this region.

For .2 $\leq$ abs(x) < .5, n equals one, and the term z = (.5*(1.0 + x) - (1.0 - x))/(.5*(1.0 + x) + (1.0 - x)) may be relatively large. For abs(x) < 0.25, the subtraction 1.0 - x = .5 - x + .5 loses two bits of the original argument. Also, z is negative in this range, and some cancellation occurs in the final combination of terms, costing about one ulp. The expected upper bound in the region .2 < abs(x) < 0.25 is 19.4E-15.

A group of 10,000 arguments was chosen randomly from the interval [-1.0,1.0]. The maximum relative error of these arguments was found to be .3304E-13.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is 1.0/(1.0 - x**2), and that of relative error is x/((1.0 - x**2)*atanh(x)). This increases from 1 at 0 and becomes arbitrarily large near 1.0.

ATAN2

ATAN2 is a function that computes the inverse tangent function of the ratio of two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RATAN2 and ATAN2, and the call-by-value entry point is MLP$VATAN2.

The ATAN2 vector math function is divided into three routines having three separate entry points defined as follows:

```
ATAN2(scalar,vector) = MLP$ATAN2SV
ATAN2(vector,scalar) = MLP$ATAN2VS
ATAN2(vector,vector) = MLP$ATAN2VV
```

The input domain is the collection of all valid real pairs (x,y) such that both quantities are not equal to zero. The output range is included in the set of valid real quantities greater than -pi and less than or equal to pi.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

x/y is equal to plus or minus infinite and y is not equal to zero.

x is not equal to zero and y is equal to plus or minus infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite and y does not equal zero, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

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

```
atan2(y,x) = -atan2(-y,x), y < 0
atan2(y,x) = pi - atan2(y,-x), x < 0, y > 0
```

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

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

by the range reduction:

atan2(y,x) = pi/2 - atan2(x,y), $x \geq 0$ or $y \geq 0$

The routine calls ATAN to evaluate atan2(y,x) as arctan(y/x).

(Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455.)

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

x/y is equal to plus or minus infinite and y is not equal to zero.

x is not equal to zero and y is equal to plus or minus infinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the description of routine ATAN.

Effect of Argument Error

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

CABS

CABS is a function that computes the absolute value of an argument. It accepts a complex argument and returns a real result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCABS and CABS, and the call-by-value entry point is MLP$VCABS.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 is a valid real quantity. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is positive infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

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

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

c. r = v/u
   w = 1.0 + r**2

   where t = w**.5 = (1.0 + r**2)**.5 is computed inline using the same algorithm as used in SQRT.

d. Return u*t to the calling program.

Formulas used are:

```
|x + i*y| = sqrt(x + i*y)
          = max(|x|,|y|)*(1 + r**2)**.5
          = u*t
```

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

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error of these arguments was found to be .1401E-13.

Effect of Argument Error

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

CCOS

CCOS is a function that computes the complex cosine function. It accepts a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCCOS and CCOS, the call-by-value entry point is MLP$VCCOS, and the vector entry point is MLP$CCOSV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y; |x| is less than 2**47 and |y| is less than 4095*log(2). The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The imaginary part is greater than or equal to 4095*log(2).

The imaginary part is less than or equal to -4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

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

The routine evaluates COSSIN inline to simultaneously compute the sine and cosine of the real part of the argument. The routine evaluates HYPERB inline to simultaneously compute the hyperbolic sine and hyperbolic cosine of the imaginary part of the argument. See the descriptions of routines COSSIN and HYPERB for detailed information.

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The imaginary part is greater than or equal to 4095*log(2).

The imaginary part is less than or equal to -4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the descriptions of HYPERB and COSSIN for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.276E-13 + 1.241E-13*exp(abs(y)).

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error of these arguments was found to be .7665E-13.

Effect of Argument Error

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

CEXP

CEXP is a function that computes the complex exponential function. It accepts a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCEXP and CEXP, the call-by-value entry point is MLP$VCEXP, and the vector entry point is MLP$CEXPV.

The input domain is the collection of all valid complex quantities z, where z = x + i * y; x is less than 4095*log(2) and x is greater than -4097*log(2), and |y| is less than 2**47. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The real part is greater than or equal to 4095*log(2) or less than or equal to -4097*log(2).

The absolute value of the imaginary part is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

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

The routine evaluates COSSIN inline to compute cos(y) and sin(y), and calls EXP to compute exp(x).

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The real part is greater than or equal to 4095*log(2) or less than or equal to -4097*log(2).

The absolute value of the imaginary part is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the description of EXP and COSSIN for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.378E-13 + 1.378E-13*exp(abs(x)). If the real part of the argument is large, the error in the routine will be significant.

The function CEXP was tested. A group of 10,000 arguments was chosen randomly from given intervals. Statistics on maximum relative error were observed. Table 2-5 shows a summary of these statistics.

Table 2-5. Relative Error of CEXP

Interval	Maximum
[[-1.0,1.0],[-1.0,1.0]]	.5462E-13
[[1.0,.6700E+03],[1.0,.11E+15]]	.9182E-13

Effect of Argument Error

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

CLOG

CLOG is a function that computes the complex natural logarithm function. It accepts a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCLOG and CLOG, the call-by-value entry point is MLP$VCLOG, and the vector entry point is MLP$CLOGV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 is a valid, positive real quantity. The output range is included in the set of valid complex quantities z, such that the real part of z is a valid real quantity, and the imaginary part is greater than -pi and less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

Both the real part and the imaginary part are zero.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

$$\log(z) = \log(|z|) + i*\arg(z)$$

where $|z|$ is the modulus of z. The routine calls CABS to evaluate the absolute value of z and calls ALOG to compute the logarithm. Then the routine calls ATAN2 to evaluate the function arg(z). When z is nonzero, and in-range, arg(z) is in the interval [-pi,pi].

Vector Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

Both the real part and the imaginary part are zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error of these arguments was found to be .4346E-12.

Effect of Argument Error

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

CONJG

CONJG is a function that returns the conjugate of an argument. It accepts a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCONJG and CONJG, and the call-by-value entry point is MLP$VCONJG.

The input domain is the collection of all valid complex quantities. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

No errors are generated by CONJG. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with its imaginary part negated.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

COS

COS is a function that computes the cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOS and COS, the call-by-value entry point is MLP$VCOS, and the vector entry point is MLP$COSV.

The input domain is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, then COS(x) or SIN(x) is calculated by using the periodic properties of the cosine and sine functions to reduce the task to finding a cosine or sine of an equivalent angle y within [-pi/4, pi/4] as follows:

```
If N + K is even
then
     Z = sin(y)
else
     Z = cos(y)
If MOD(N + K, 4) is 0 or 1 (that is, the second last bit of
N + K is even)
then
     S = 0
else
     S = mask(1)
```

where K is 0, 1, or 2 according to whether the SIN of a positive angle, the COS of any angle, or the SIN of a negative angle is to be calculated. N is the nearest integer to 2/pi*x, and y is the nearest single precision floating point number to x - n*pi/2. The argument x is the absolute value of the angle. The desired SIN or COS is xor(S, Z).

Once the angle has been reduced to the range [-pi/4, pi/4], the following approximations are used to calculate either the cosine or the sine of the angle, providing 48 bits of precision.

If the cosine of the angle is required, the approximation used is

```
cosine(y) = 1 - y*y*P(y*y)
```

where y is the angle and P(w) is the quintic polynomial:

```
P(w) = P0 + P1*w  + P2*w**2 + P3 + w**3 + P4*w**4 + P5*w**5
```

such that P(y*y) is a min-max polynomial approximation to the function (1 - cos(y))/y**2.

The coefficients are:

```
P5 = -2.070062305624629462E-9
P4 =  2.755636997406588778E-7
P3 = -2.480158521206426671E-5
P2 =  1.388888888727866775E-3
P1 = -4.166666666666468116E-2
P0 =  5.000000000000000000E-1
```

If the sine of the angle is required, the approximation used is

```
sine(y) = y - y*y*y*Q(y*y)
```

where y is the angle and Q(w) is the quintic polynomial:

```
Q(w) = Q0 + Q1*w  + Q2*w**2 + Q3*w**3 + Q4*w**4 + Q5*w**5
```

such that Q(y*y) is a min-max polynomial approximation to the function (y - sin(y))/y**3.

The coefficients are:

```
Q5 = -1.591814257033005283E-10
Q4 =  2.505113204973767698E-8
Q3 = -2.755731610365754733E-6
Q2 =  1.984126983676100911E-4
Q1 = -8.333333333330950363E-3
Q0 =  1.666666666666666463E-1
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function COS was tested against 4*COS(x/3)**3 - 3*COS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199E+02,.2356E+02]. Statistics on relative error were observed: maximum relative error was .1404E-13, and root mean square relative error was .3245E-14.

Effect of Argument Error

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

COSD

COSD is a function that computes the cosine function for an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOSD and COSD, the call-by-value entry point is MLP$VCOSD, and the vector entry point is MLP$COSDV.

The input domain is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [-45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x += 360 degrees) =  sin(x)
sin(x += 180 degrees) = -sin(x)
sin(x + 90 degrees)   =  cos(x)
sin(x - 90 degrees)   = -cos(x)
cos(x += 360 degrees) =  cos(x)
cos(x += 180 degrees) = -cos(x)
cos(x + 90 degrees)   = -sin(x)
cos(x - 90 degrees)   =  sin(x)
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The reduction to (-45,+45) is exact; the constant pi/180 has relative error 1.37E-15, and multiplication by this constant has a relative error 5.33E-15, and a total error of 6.7E-15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is, at most, 5.26E-15 plus the maximum error in SINCOS over (-pi/4,+pi/4).

A group of 10,000 arguments was chosen at random from the interval [0,360]. The maximum relative error of these arguments was found to be .7105E-14 for COSD and .1403E-13 for SIND.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (-45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

COSH

COSH is a function that computes the hyperbolic cosine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOSH and COSH, the call-by-value entry point is MLP$VCOSH, and the vector entry point is MLP$COSHV.

The input domain is the collection of all valid real quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid real quantities greater than or equal to 1.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formula used to compute cosh(x) is:

cosh(x) = (exp(x) + exp(-x))/2

The routine calls EXP to compute exp(x) and computes 1.0/exp(x) to obtain exp(-x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-6 shows a summary of these statistics.

Table 2-6. Relative Error of COSH

Test	Interval		Maximum	Root Mean Square
	From	To		
COSH(x) against Taylor series expansion of COSH(x)	0.0000E+00	.5000E+00	.1382E-13	.6875E-14
COSH(x) against c*(COSH(x + 1) + COSH(x - 1))	.3000E+01	.2838E+04	.2296E-13	.8260E-14

Effect of Argument Error

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

COTAN

COTAN is a function that computes the trigonometric circular cotangent of an argument in radians. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RCOTAN and COTAN, the call-by-value entry point is MLP$VCOTAN, and the vector entry point is MLP$COTANV.

The input domain is the collection of all valid real quantities whose absolute value is greater than 0 and less than 2**47. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is 0.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The evaluation is reduced to the interval [-.5,.5] by using the identities:

1. cotan(x) = cotan(x + k * pi/2), if k is even

2. cotan(x) = -1.0/cotan(x + pi/2)

in the form:

3. cotan(x)=1/tan(x)=1/tan((pi/2)*(x*2/pi + k)), if k is even

4. cotan(x)=1/tan(x)=tan((pi/2)*(x*2/pi + 1.0))/-1.0

In effect, the original algorithm for TAN(x) is used to find COTAN(x). The result for COTAN(x) is the reciprocal of TAN(x).

An approximation of tan(pi/2*y) is used. The argument is reduced to the interval [-.5,.5] by subtracting a multiple of pi/2 from x in double precision.

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

Identity 4 is used if the integer subtracted is odd. The result is negated and inverted by dividing -P/Q instead of Q/P.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is 0.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function COTAN was tested against (COTAN(x/2)**2-1)/(2*COTAN(x/2)). Groups of 2,000 arguments were chosen randomly from the interval (.1885E+02, .1963E+02). Statistics on relative error were observed: maximum relative error was .2297E-13, and root mean square relative error was .7847E-14.

Effect of Argument Error

For small errors in the argument x, the amplication of absolute error is sec(x)**2, and that of relative error is x/(sin(x)*cos(x)), which is at least 2x and can be arbitrarily large near a multiple of pi/2.

CSIN

CSIN is a function that computes the complex sine function. It accepts a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RCSIN and CSIN, the call-by-value entry point is MLP$VCSIN, and the vector entry point is MLP$CSINV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y; |x | is less than 2**47, and |y | is less than 4095*log(2). The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The absolute value of the imaginary part is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Let x + i*y be the argument. The formula used for computation is:

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

The routine evaluates COSSIN inline to simultaneously compute sine and cosine, and evaluates HYPERB inline to simultaneously compute hyperbolic sine and hyperbolic cosine. See the descriptions of routines COSSIN and HYPERB for detailed information.

Vector Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

The absolute value of the real part is greater than or equal to 2**47.

The absolute value of the imaginary part is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the description of HYPERB and COSSIN for details. If z = x + i*y is the argument, then the modulus of the error in the routine does not exceed: 1.276E-13 + 1.297E-13*exp(abs(y)).

Effect of Argument Error

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

CSQRT

CSQRT is a function that computes the complex square root function that maps to the right half of the complex plane. It accepts a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RCSQRT and CSQRT, the call-by-value entry point is MLP$VCSQRT, and the vector entry point is MLP$CSQRTV.

The input domain is the collection of all valid complex quantities z, where z = x + i*y, and (x**2 + y**2)**.5 + |x| is a valid real quantity. If the argument is zero, zero is returned. The output range is included in the set of valid complex quantities z such that the real part of z is nonnegative and the imaginary part of z is a valid complex quantity.

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is positive infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

For this computation, values returned by the routine will lie in the right half of the complex plane.

Call-By-Value Routine

Let x + i*y be the argument. The formulas used for computation are:

```
u = (.5*(|x| + |(x,y)|))**.5
v = .5*(y/u)
```

If x is nonnegative, then csqrt(x,y) = u + i*v. If x is negative, then csqrt(x,y) = sign(y)*(v + i*u).

The result of this routine always lies in the first or fourth quadrant of the complex plane. The routine takes complex quantities lying on the axis of the negative reals, to the axis of the positive imaginaries.

Vector Routine

The argument is checked upon entry. It is invalid if:

The real or imaginary part is indefinite.

The real or imaginary part is infinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function CSQRT was tested. A group of 10,000 arguments was chosen randomly from given intervals. Statistics on maximum relative error were observed. Table 2-7 shows a summary of these statistics.

Table 2-7. Relative Error of CSQRT

Interval	Maximum
[[0,0],[100,100]]	.1600E-13
[[0,0],[1.0E+100,1.0E+100]]	.1499E-13

Effect of Argument Error

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

DABS

DABS is a function that computes the absolute value of an argument. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDABS and DABS, and the call-by-value entry point is MLP$VDABS.

The input domain is the collection of all valid double precision quantities. The output range is included in the set of valid, nonnegative double precision quantities.

Call-By-Reference Routine

No errors are generated in DABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The argument is returned with the sign bits of both its upper and lower words forced positive.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DACOS

DACOS is a function that computes the inverse cosine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDACOS and DACOS, and the call-by-value entry point is MLP$VDACOS.

The input domain is the collection of all valid double precision quantities in the interval [-1.0,1.0]. The output range is included in the set of valid, nonnegative double precision quantities less than or equal to pi.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

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

```
arcsin(-x) = -arcsin(x)
arccos(x)  =  pi/2-arcsin(x)
arcsin(x)  =  arccos(sqrt(1.0 - x**2)), if x >= 0
arccos(x)  =  arcsin(sqrt(1.0 - x**2)), if x >= 0
```

The reduced value is called y. If y <= .09375, no further reduction is performed. If not, the losest entry to y in a table of values (z, arcsin(z), sqrt(1.0 - x**2), z = .14, .39, .52, .64) is found, and the formula used is:

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

where w = x(sqrt(1.0 - z**2) - z*sqrt(1.0 - x**2). The value of w is in (-.0792, .0848).

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

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

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

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

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The region of worst error is (.9895,.9966). In this region, the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs.

The function DACOS was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-8 shows a summary of these statistics.

Table 2-8. Relative Error of DACOS

Interval			
From	To	Maximum	Root Mean Square
-.1250D+00	.1250D+00	.2794D-27	.2343D-27
-.1000D+01	-.7500D+00	.3339D-27	.2853D-27
.7500D+00	.1000D+01	.7573D-28	.2257D-28

Effect of Argument Error

If a small error eps occurs in the argument x, the resulting error in DACOS is approximately -eps/(1.0 - x**2)**.5. The amplification of the relative error is approximately x/(f(x)*(1.0 - x**2)**.5), where f(x) is DACOS. The error is attenuated for x > -.44 but can become serious near -1.0. If the argument is generated as 1.0 - y or y - 1.0, then the following identities can be used to get the full significance of y:

```
 asin(x) = acos(sqrt(1.0 - x**2))
 acos(x) = asin(sqrt(1.0 - x**2))
asin(-x) = -asin(x)
acos(-x) = pi + asin(x)
```

DASIN

DASIN is a function that computes the inverse sine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDASIN and DASIN, the call-by-value entry point is MLP$VDASIN, and the vector entry point is MLP$DASINV.

The input domain is the collection of all valid double precision quantities in the interval [-1.0,1.0]. The output range is included in the set of valid double precision quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

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

```
arcsin(-x) = -arcsin(x)
arccos(x)  =  pi/2-arcsin(x)
arcsin(x)  =  arccos(sqrt(1.0 - x**2)), if x >= 0
arccos(x)  =  arcsin(sqrt(1.0 - x**2)), if x >= 0
```

The reduced value is called y. If y <= .09375, no further reduction is performed. If not, the closest entry to y in a table of values (z, arcsin(z), sqrt(1.0 - x**2), z = .14, .39, .52, .64) is found, and the formula used is:

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

where w = x(sqrt(1.0 - z**2) - z*sqrt(1.0 - x**2). The value of w is in (-.0792, .0848).

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

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

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

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

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value exceeds 1.0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The region of worst error is (.09375,.1446). In this region, the final addition is of quantities of almost equal magnitude and opposite sign, and cancellation of about one bit occurs, the worst case being .1451-.0629. For DASIN, the polynomial range was extended to cover the region (.0821,.09375), where the worst error occurs.

The function DASIN was tested against the Taylor series. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-9 shows a summary of these statistics.

Table 2-9. Relative Error of DASIN

Interval			
From	To	Maximum	Root Mean Square
-.1250D+00	.1250D+00	.1017D-27	.2246D-28
.7500D+00	.1000D+01	.4761D-27	.3575D-27

Effect of Argument Error

If a small error eps occurs in the argument x, the resulting errors in DASIN are approximately eps/(1 - x**2)**.5. The amplification of the relative error is approximately x/(f(x)*(1 - x**2)**.5), where f(x) is DASIN. The error is attenuated for abs(x) < .75 but can become serious near -1.0 or +1.0. If the argument is generated as 1 - y or y - 1, then the following identities can be used to get the full significance of y:

```
 asin(x) = acos(sqrt(1.0 - x**2))
 acos(x) = asin(sqrt(1.0 - x**2))
asin(-x) = -asin(x)
acos(-x) = pi + asin(x)
```

DATAN

DATAN is a function that computes the inverse tangent function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDATAN and DATAN, the call-by-value entry point is MLP$VDATAN, and the vector entry point is MLP$DATANV.

The input domain is the collection of all valid double precision quantities. The output range is included in the set of valid double precision quantities in the interval [-pi/2,pi/2].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Register pair (X4,X5) holds the absolute value of the argument.

B4 = (X9) = sign mask for the argument. (B4 holds a mask for the result's sign.)

If $|x| < 1.0$, then:

B3 = (XA) = 0.
B7 = (XB) = 0. (B7 will hold the closest multiple of pi/2 to the absolute value of the result.)
Branch to DATCOM at label DTN to complete processing.

If $|x| \geq 1.0$, then:

B3 = (XA) = 1 in high order bit.
B7 = (XB) = 1.0.
Branch to DATCOM at label DATCOM to complete processing.

At labels DATCOM and DTN:

(X9) = B4 = mask MS = sign of final result.
(XA) = B3 = mask MI.
(XB) = B7 = closest multiple of pi/2 to the absolute value of the result.

At label DATCOM:

    Register pair (X7,X8) = DU.
    Register pair (X4,X5) = DV.

At label DTN:

    Register pair (X7,X8) = DU.

Label ATNU is the start of an 18-word table containing atan(n/8) (0 $\leq$ n $\leq$ 8) in double precision. Label DATCOM corresponds to step a, and label DTN corresponds to step b.

Constants used in the algorithm are:

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

where d3, d5, d7, and d9 are double precision constants, and c11, c13, a, and b are real constants. Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. Boolean operations have B subscripts.

The algorithm used is:

a. DQ = DU/DV computed in double precision.

b. (DQ = DA-DU at DTN) (Note that 0 $\leq$ DQ $\leq$ 1.0.)

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

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

e. DA = (DQ - n/8)/(1.0 + n/8*DA), computed in double precision.

f. Z = 0
   DC = 0
   If (DA)(u) = 0, go to step i.

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

h. w = DA +(d) DC*DA

i. DB = 0
   If (XB) ≠ 0 DB = ATN(9)*2*(XB)

j. BBAR = (B7*pi/2) - (B)B3 (upper and lower)

k. CBAR = BBAR + (D)ATN(n/8). ATN(n/8) is obtained as a double precision quantity from the look-up table.

l. Result = (CBAR + (D) w) - (B) (B3 - (B)B4).

At the end of processing, register pair (XE,XF) contains the DATAN result.

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

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

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-10 shows a summary of these statistics.

Table 2-10. Relative Error of DATAN

Test	Interval		Maximum	Root Mean Square
	From	To		
DATAN(x) against truncated Taylor series	-.6250D-01	.6250D-01	.2556D-28	.1343D-28
2*DATAN(x) against DATAN(2x/(1 - x*x))	.2697D+00 .4142D+00	.4142D+00 .1000D+01	.4821D-28 .5992D-28	.2027D-28 .2449D-28
DATAN(x) against DATAN(1/16) + DATAN((x - 1/16)/ (1 + x/16))	.6250D-01	.2679D+00	.3388D-28	.1557D-28

Total Error

Most of the errors can be traced back to errors in double precision addition.

Effect of Argument Error

If a small error e occurs in the argument x, the error in the result is given by e/(1.0 + x**2).

DATAN2

DATAN2 is a function that computes the inverse tangent function of the ratio of two arguments. It accepts two double precision arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDATAN2 and DATAN2, and the call-by-value entry point is MLP$VDATAN2.

The DATAN2 vector math function is divided into three routines having three separate entry points defined as follows:

```
DTAN2(scalar,vector) = MLP$DATAN2SV
DTAN2(vector,scalar) = MLP$DATAN2VS
DTAN2(vector,vector) = MLP$DATAN2VV
```

The input domain is the collection of all valid double precision pairs (x,y) such that both quantities are not zero. The output range is included in the set of double precision quantities greater than -pi and less than or equal to pi.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x and y are infinite.

x and y are equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Register pair (X4,X5) holds the absolute value of the first argument. Register pair (X7,X8) holds the absolute value of the second argument.

B4 = (X9) = sign mask of the first word of the first argument.
B3 = (XA) = complement of the sign mask of the first word of the second argument.
B7 = (XB) = closest multiple of pi/2 to the result value.

If (X4) > (X7), then:

```
B7 = (XB) = 1.0.
Branch to label DATCOM to complete processing.
```

If (X4) $\leq$ (X7), then:

Exchange (X7) and (X4) and (X8) and (X5).
Complement contents of B3.
B7 = (XB) = 0, if the first word of the second argument is positive.
B7 = (XB) = 2, if the first word of the second argument is negative.
Branch to label DATCOM to complete processing.

At label DATCOM:

(X9) = B4 = mask MS = sign of the final result.
(XA) = B3 = mask MI.
(XB) = B7 = closest multiple of pi/2 to the absolute value of the result.
Register pair (X7,X8) = DU = smaller of DU and DB = min(x,y).
Register pair (X4,X5) = DV = larger of DU and DV = max(x,y).

At label DATCOM10:

Register pair (X7,X8) = DQ = DU/DV, which is < 1.0.

ATNU is the start of an 18-word table containing atan(n/8) ($0 \leq n \leq 8$) in double precision. Label DATCOM corresponds to step a.

Constants used in the algorithm are:

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

where d3, d5, d7, and d9 are double precision constants, and c11, c13, a, and b are real constants. Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. Boolean operations have B subscripts.

The algorithm used is:

a. DQ = DU/DV in double precision.

b. If both DU and DV are zero, error exit occurs.

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

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

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

f. Z = 0
   DC = 0
   If (DA)(u) = 0, go to step i.

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

h. w = DA + (d) DC*DA

i. DB = 0
   If (XB) ≠ 0 DB = ATN(9)*2*(XB)

j. BBAR = (B7*pi/2) - (B)B3 (upper and lower)

k. CBAR = BBAR + (D)ATN(n/8). ATN(n/8) is obtained as a double precision quantity from the look-up table.

l. Result = (CBAR + (D) w) - (B) (B3 - (B)B4).

At the end of processing, register pair (XE,XF) contains DATAN2 result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is infinite.

x and y are infinite.

x and y are equal to 0.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

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

Effect of Argument Error

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

```
(x * e´´ - y * e´)/(x**2 + y**2)
```

DCOS

DCOS is a function that computes the cosine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDCOS and DCOS, the call-by-value entry point is MLP$VDCOS, and the vector entry point is MLP$DCOSV.

The input domain is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x - n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [-pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [-pi/4,pi/4], the sine polynomial approximation is:

```
p(x) =  a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 +
        a(11)x**11 + a(13)x**13** + a(15)x**15 + a(17)x**17 +
        a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) =  b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 +
        b(10)x**10 + b(12)x**12 + b(14)x**14 + b(16)x**16 +
        b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =  .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =  .833 333 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =  .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =  .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =  .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =  .194 362 013 130 224*10**-19
b(0)  =  .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =  .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =  .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =  .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =  .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =  .408 023 947 777 860*10**-18
```

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

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The maximum absolute value of the error of approximation of p(x) to sin(x) over (-pi/4,pi/4) is .2570E-28, and of q(x) to cos(x) is .3786E-28.

The function DCOS was tested against 4*DCOS(x/3)**3 - 3*DCOS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199D+02,.2356D+02]. Statistics on relative error were observed: maximum relative error was .2057D-23; root mean square relative error was .4606D-25.

Effect of Argument Error

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

DCOSH

DCOSH is a function that computes the hyperbolic cosine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDCOSH and DCOSH, the call-by-value entry point is MLP$VDCOSH, and the vector entry point is MLP$DCOSHV.

The input domain is the collection of all valid double precision quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid double precision quantities greater than or equal to 1.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formulas used for computation are:

```
u    = exp(x)*.5
v    = exp(-x)*.5
cosh(x) = u + v
```

The routine calls DEXP to compute exp(x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-11 shows a summary of these statistics.

Table 2-11. Relative Error of DCOSH

Test	Interval		Maximum	Root Mean Square
	From	To		
DCOSH(x) against Taylor series expansion of DCOSH(x)	0.0000D+00	.5000D+00	.2524D-28	.1739D-28
DCOSH(x) against c*(DCOSH(x + 1) + DCOSH(x - 1))	.3000D+01	.2838D+04	.1023D-27	.4548D-28

Effect of Argument Error

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

DDIM

DDIM is a function that computes the positive difference between two arguments. It accepts two double precision arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDDIM and DDIM, and the call-by-value entry point is MLP$VDDIM.

The input domain is the collection of all valid double precision pairs (x,y) such that x - y is a valid double precision quantity. The output range is included in the set of valid, nonnegative double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x - y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit of the difference is extended across another word to form a mask. The boolean product of the mask's complement and the upper and lower word of the difference is formed.

Given arguments (x,y):

result = x - y if $x > y$
result = 0 if $x \leq y$.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DEXP

DEXP is a function that computes the exponential function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDEXP and DEXP, the call-by-value entry point is MLP$VDEXP, and the vector entry point is MLP$DEXPV.

The input domain is the collection of all valid double precision quantities whose value is greater than or equal to -4097*log(2) and less than or equal to 4095*log(2). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than -4097*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The argument reduction performed is:

```
x = argument
y = x - n*log(2)
```

where y = is in [-1/2 log(2), 1/2 log(2)] and n is an integer.

Constants used in the algorithm are:

```
1.0/log(2)
log(2) (in double precision)
d3  =    .166 666 666 666 666 666 666 666 666 709
d5  =    .833 333 333 333 333 333 333 331 234 953*10**-2
d7  =    .198 412 698 412 698 412 700 466 386 658*10**-3
d9  =    .275 573 192 239 858 897 408 325 908 796*10**-5
pc  =   -.474 970 880 178 988*10**-10
pa  =    .566 228 284 957 811*10**-7
pb  = 272.110 632 903 710
c11 =    .250 521 083 854 439*10**-7
```

Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. An operand with a u or 1 subscript denotes the first or second word, respectively, of the double precision pair of words containing the operand.

On input, the argument is in register pair X2-X3, and on output, the result is in register pair XE-XF.

The algorithm used is:

a. x = argument. If x = 0, set DEXP = 1.0. Return.

b. If x ≠ 0,
   n = nearest integer to x/log(2),
   y = x - n*log(2).
   Then y is in [-1/2*log(2),1/2*log(2)].

c. q = (y)(u)*(u)(y)(u)

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

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

f. Compute hm = sqrt(1.0 + s**2).
   hi = 3*q + ((s)(u))**2 in real.
   hj = hi + hi
   hk = 2*(1.0 + hj)
   hl = ((y)(u)*(u)(y)(u) - hj)/hk - hi
   hm = hj +(u) (hk -(u) hl)*(u)(hl/hk)
   (hm now carries cosh - 1.0 in single precision.)

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

h. DC = hm +(d) (DS*DS - 2*hm - hm*hm)/(2(1.0 + hm)) computed in double precision.

i. DX = DS + DC

j. Clean up DS, DC, DX with (X7) = n.
Register pair XA-XB = DS = sinh(y).
Register pair X8-X9 = DC = cosh(y) - 1.0.
Register pair X4-X5 = DX = exp(y).

k. Increase the exponents of exp(y) by n.

l. Return.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than -4097*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-12 shows a summary of these statistics.

Table 2-12. Relative Error of DEXP

Test	Interval		Maximum	Root Mean Square
	From	To		
DEXP(x - 2.8125) against DEXP(x)/ DEXP(2.8125)	-.3466D+01	-.2772D+04	.9240D-28	.2956D-28
DEXP(x - .0625) against DEXP(x)/ DEXP(.0625)	-.2841D+00	.3466D+00	.6449D-28	.1680D-28
DEXP(x - 2.8125) against DEXP(x)/ DEXP(2.8125)	.6931D+01	.2838D+04	.9262D-28	.2907D-28

Effect of Argument Error

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

DIM

DIM is a function that computes the positive difference between two arguments. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RDIM and DIM, and the call-by-value entry point is MLP$VDIM.

The input domain is the collection of all valid real quantities (x,y), such that x - y is a valid real quantity. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x - y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit is extended across another word to form a mask. The boolean product of the mask's complement and the difference is formed.

Given arguments (x,y):

result = x - y if $x > y$
result = 0 if $x \leq y$

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DINT

DINT is a function that returns the integer part of an argument after truncation. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDINT and DINT, and the call-by-value entry point is MLP$VDINT.

The input domain for this routine is the collection of all valid double precision quantities. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument is added to a special floating point zero with an exponent value that forces the argument's fraction bits to be shifted off when it is added to the argument. The result is returned.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DLOG

DLOG is a function that computes the natural logarithm function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDLOG and DLOG, the call-by-value entry point is MLP$VDLOG, and the vector entry point is MLP$DLOGV.

The input domain for this routine is the collection of all valid, positive double precision quantities. The output range is included in the set of double precision quantities whose absolute value is less than 4095*log(2).

Call-By-Reference Routine

The argument is checked upon entry. The argument is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

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

```
log(x) = k*log(2) + log(w)
```

and k*log(2) is computed in double precision. A polynomial approximation u is evaluated in single precision using:

```
u = c(1)*t + c(3)*t**3 + c(5)*t**5 + c(7)*t**7
```

where t = (w - 1.0)/(1.0 + w)

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

See Vector Error Handling in chapter 1 for further information.

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

```
c(1) = 1.999 999 993 734 000
c(3) =  .666 669 486 638 944
c(5) =  .399 657 811 051 126
c(7) =  .301 005 922 238 712
```

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

```
v = u - (1.0 - x*exp(-u)) - (1.0 - x*exp(-u - (1.0 - x*exp(-u))))
```

z is made to be less than 1.0 by writing z = 1.0 - x*exp(-u), and v is computed using:

```
v = u - z(u) - z(1) - (z(u))**2*(.5 + z(u)/3)
```

where z = z(u) + z(1). This formula is obtained by neglecting terms that are not significant for double precision; exp(-u) is evaluated in double precision by the polynomial of degree 17. If entry was made at MLP$VDLOG10, after k*log(2) + log(w) has been evaluated, the result is multiplied by log(e) base 10 in double precision.

Error Analysis

The maximum absolute value of the error of approximation of the algorithm to log(x) is 1.555E-29 over the interval (2**(-.5),2**.5).

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-13 shows a summary of these statistics.

Table 2-13. Relative Error of DLOG

Test	Interval		Maximum	Root Mean Square
	From	To		
DLOG(x*x) against 2*DLOG(x)	.1600D+02	.2400D+03	.4479D-28	.1528D-28
DLOG(x) against DLOG(17x/16) - DLOG(17/16)	.7071D+00	.9375D+00	.9041D-27	.1478D-27

Effect of Argument Error

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

DLOG10

DLOG10 is a function that computes the common logarithm function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDLOG10 and DLOG10, the call-by-value entry point is MLP$VDLOG10, and the vector entry point is MLP$DLOG10V.

The input domain for this routine is the collection of all valid, positive double precision quantities. The output range is included in the set of double precision quantities whose absolute value is less than 4095*log(2) base 10.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is put into the form x = 2**k*w, where k is an integer, and 2**-1/2 <= w <= 2**1/2. Then log(x) is computed from:

```
log(x) = k*log(2) + log(w)
```

and k*log(2) is computed in double precision. A polynomial approximation u is evaluated in single precision using:

```
u = c(1)*t + c(3)*t**3 + c(5)*t**5 + c(7)*t**7
```

where t = (w - 1.0)/(1.0 + w)

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

```
c(1) = 1.999 999 993 734 000
c(3) =  .666 669 486 638 944
c(5) =  .399 657 811 051 126
c(7) =  .301 005 922 238 712
```

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

```
v = u - (1.0 - x*exp(-u)) - (1.0 - x*exp(-u - (1.0 - x*exp(-u))))
```

z is made to be less than 1.0 by writing z = 1.0 - x*exp(-u), and v is computed using:

```
v = u - z(u) - z(1) - (z(u))**2*(.5 + z(u)/3)
```

where z = z(u) + z(1). This formula is obtained by neglecting terms that are not significant for double precision; exp(-u) is evaluated in double precision by the polynomial of degree 17. If entry was made at MLP$VDLOG10, after k*log(2) + log(w) has been evaluated, the result is multiplied by log(e) base 10 in double precision.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is equal to zero.

It is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function DLOG10 was tested against DLOG10(11x/10) - DLOG10(11/10). Groups of 2000 arguments were chosen randomly from the interval [.3162D+00,.9000D+00]. Statistics on relative error were observed: maximum relative error was .5417D-27; root mean square relative error was .8117D-28.

Effect of Argument Error

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

DMOD

DMOD is a function that returns the remainder of the ratio of two arguments. It accepts two double precision arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDMOD and DMOD, and the call-by-value entry point is MLP$VDMOD.

The input domain for this routine is the collection of all valid double precision pairs (x,y), where y is nonzero and x/y is a valid quantity. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

y is equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The function computed by DMOD(x,y) is:

```
x - (x/y)*y
```

where parentheses denote truncation. The result of x/y is found and then added to a special floating point zero that forces truncation.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DNINT

DNINT is a function that returns the nearest whole number to an argument. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDNINT and DNINT, and the call-by-value entry point is MLP$VDNINT.

The input domain for this routine is the collection of all valid double precision quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it and the result is added to a special floating point zero that forces truncation. If the argument is < 0, -.5 is added to it and the result is treated as above.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DPROD

DPROD is a function that computes the product of two arguments. It accepts two real arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDPROD and DPROD, and the call-by-value entry point is MLP$VDPROD.

The input domain for this routine is the collection of all valid real pairs (x,y) such that x*y is a valid double precision quantity. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Given argument pair (x,y), the result of x*y is found.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DSIGN

DSIGN is a function that transfers the sign of the second argument to the sign of the first. It accepts two double precision arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDSIGN and DSIGN, and the call-by-value entry point is MLP$VDSIGN.

The input domain for this routine is the collection of all valid double precision pairs (x,y). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

No errors are generated by DSIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the second argument is isolated in a mask with all other bits zero. The sign bits of the upper and lower words of the first argument are cleared by a boolean AND mask and replaced by the sign of the second argument by a boolean inclusive OR with the complement of the mask.

Given arguments (x,y):

result = $|x|$ if y is nonnegative
result = $-|x|$ if y is negative

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

DSIN

DSIN is a function that computes the sine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDSIN and DSIN, the call-by-value entry point is MLP$VDSIN, and the vector entry point is MLP$DSINV.

The input domain for this routine is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result is returned to the calling program.

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x - n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [-pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [-pi/4,pi/4], the sine polynomial approximation is:

```
p(x) =  a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 +
        a(11)x**11 + a(13)x**13** + a(15)x**15 + a(17)x**17 +
        a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) =  b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 +
        b(10)x**10 + b(12)x**12 + b(14)x**14 + b(16)x**16 +
        b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =  .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =  .833 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =  .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =  .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =  .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =  .194 362 013 130 224*10**-19
b(0)  =  .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =  .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =  .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =  .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =  .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =  .408 023 947 777 860*10**-18
```

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

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The maximum absolute value of the error of approximation of p(x) to sin(x) over (-pi/4,pi/4) is .2570E-28, and of q(x) to cos(x) is .3786E-28.

The function DSIN was tested against the 3*DSIN(x/3) - 4*DSIN(x/3)**3. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-14 shows a summary of these statistics.

Table 2-14. Relative Error of DSIN

Interval			
From	To	Maximum	Root Mean Square
0.0000D+00	.1571D+01	.5153D-28	.1254D-28
.1885D+02	.2042D+02	.2764D-23	.6188D-25

Effect of Argument Error

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

DSINH

DSINH is a function that computes the hyperbolic sine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDSINH and DSINH, the call-by-value entry point is MLP$VDSINH, and the vector entry point is MLP$SINHV.

The input domain for this routine is the collection of all valid double precision quantities whose absolute value is less than 4095*log(2). The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result is returned to the calling program.

Call-By-Value Routine

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

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

where n is an integer, and y is in the interval [-1/2*log(2),1/2*log(2)].

The formula used for computation is:

```
sinh(y + n*log(2)) = (cosh(y) + sinh(y))*2**(n-1.0) - (cosh(y) -
                     sinh(y))*2**(-n-1.0)
```

where

cosh(y) = DC, and sinh(y) = DS as computed in routine DEULER.

On input, the argument is in register pair (X2,X3), and on output, the result is in register pair (XE,XF).

See the description of routine DEULER for detailed information.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-15 shows a summary of these statistics.

Table 2-15. Relative Error of DSINH

Test	Interval		Maximum	Root Mean Square
	From	To		
DSINH(x) against Taylor series expansion of DSINH(x)	0.0000D+00	.5000D+00	.1184D-27	.3084D-28
DDINH(x) against c*(DSINH(x + 1) + DSINH(x - 1))	.3000D+01	.2838D+04	.1178D-27	.4582D-28

Effect of Argument Error

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

DSQRT

DSQRT is a function that computes the square root. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDSQRT and DSQRT, the call-by-value entry point is MLP$VDSQRT, and the vector entry point is MLP$DSQRTV.

The input domain for this routine is the collection of all valid, nonnegative double precision quantities. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

An initial approximation to sqrt(y) is obtained by evaluating inline, the sqrt of y(u) in single precision.

One Heron's iteration is performed in double precision using y and the initial approximation of sqrt(y), giving the double precision result.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The algorithm error is at most 2.05E-31, and is always positive.

The function DSQRT was tested against DSQRT(x*x) - x. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-16 shows a summary of these statistics.

Table 2-16. Relative Error of DSQRT

Interval			
From	To	Maximum	Root Mean Square
.1000D+01	.1414D+01	.0000D+00	.0000D+00
.7071D+00	.1000D+01	.1785D-28	.9981D-29

Effect of Argument Error

For a small error in the argument y, the amplification of absolute error is 1/2*sqrt(y)) and that of relative error is .5.

DTAN

DTAN is a routine that computes the tangent function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTAN and DTAN, the call-by-value entry point is MLP$VDTAN, and the vector entry point is MLP$DTANV.

The input domain for this routine is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument reduction is performed in two steps:

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

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

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

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

```
t = tan(m/8 + e(m))              (table look-up)
r = L(U(x)**2)/2U(x) + L(x)
a = L(U(x)**2) + 2L(x)U(x)
b = U(U(x)**2)
```

Since:

```
tan(x) = tan(sqrt(x**2))
       = tan(sqrt(U(U(x)**2 + L(U(x)**2) + 2L(x)U(x)))
       = tan(sqrt(b + a))
       = tan(sqrt(b) + a/2b)
       = tan(sqrt(b) + r)
```

Then s = sqrt(b) = U(x) - L(U(x)**2)/2U(x)

The value of the original argument y is:

```
tan(y) = tan(x + n*pi/2 + m/8 + e(m))
```

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

```
tan(y) = tan(x + m/8 + e(m)), if n is even
tan(y) = 1/tan(x + m/8 + e(m)), if n is odd
```

Applying the tangent addition formula gives:

$$\tan(x + m/8 + e(m)) = \tan(s + r + (m/8 + e(m)))$$

$$= \frac{\tan(s) + \tan(r) + t - \tan(s)*\tan(r)*t}{1.0 - \tan(s)*\tan(r) - \tan(r)*t - t*\tan(s)}$$

$$= \frac{\tan(s) + r + t - \tan(s)*r*t}{1 - \tan(s)*r - r*t - t*\tan(s)}$$

Tan(s) is computed by using the general polynomial form:

```
x + x**3/3 + x**5*2/315 ...
```

After Chebyshev is applied to the coefficients, the form is:

```
tan(s) = s + s*(c(1)s**2 + c(2)s**4 + c(3)s**6 + c(4)s**8 +
         (a/(b - s**2))s**10)
```

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

The quotient is inverted if n is odd.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The algorithm error has a negligible effect on the total error. The worst relative error of the algorithm is 1.032E-29. There is a negligible error introduced by the pi/2 range reduction except for points close to nonzero multiples of pi/2. Near pi/2 the pi/2 reduction relative error is bounded by 2**(n-155) where n is the number of bits of precision to which the argument represents pi/2. At larger multiples of pi/2, similar problems occur.

The function DTAN was tested against 2*DTAN(x/2)/(1 - DTAN(x/2)**2). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-17 shows a summary of these statistics.

Table 2-17. Relative Error of DTAN

Interval			
From	To	Maximum	Root Mean Square
.0000D+00	.7854D+00	.1946D-27	.4491D-28
.1885D+02	.1963D+02	.1729D-27	.4480D-28
.2749D+01	.3534D+01	.2008D-27	.5363D-28

Effect of Argument Error

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

DTANH

DTANH is a function that computes the hyperbolic tangent function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTANH and DTANH, the call-by-value entry point is MLP$VDTANH, and the vector entry point is MLP$DTANHV.

The input domain for this routine is the collection of all valid double precision quantities. The output range is included in the set of valid quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

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

1. For argument in [-47*log(2),47*log(2)] but not in [-1/2*log(2),1/2*log(2)]:

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

   where n is an integer, and y is in [-1/2*log(2),1/2*log(2)]

```
tanh(x) = u/v where

    u = 1.0 - 2**-n - 2**-n*(DC - DS)
    v = 1.0 - 2**-n + 2**-n*(DC - DS)
```

2. For argument in [-1/2*log(2),1/2*log(2)]:

```
    x = argument
    y = reduced argument
    y = x
tanh(x) = DS(2*+DC)
```

3. For argument outside [-47*log(2),47*log(2)]:

```
    x = argument
    y = reduced argument
tanh(x) = 1.0 - 2((1.0 + DC - DS)*2**-n - ((1.0 + DC -
DS)*2**-n)**2)
```

In steps 1, 2, and 3, DC = cosh(y) - 1.0 and DS = sinh(y), where DC + DS are computed in DEULER.

On input, the argument is in register pair (X2-X3), and on output, the result is in register pair (XE-XF).

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function DTANH was tested against (DTANH(x - 1/8) + DTANH(1/8))/(1 + DTANH(x - 1/8)*DTANH(1/8)). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-18 shows a summary of these statistics.

Table 2-18. Relative Error of DTANH

Interval			
From	To	Maximum	Root Mean Square
.1250D+00	.5493D+00	.9403D-28	.2612D-28
.6743D+00	.3431D+02	.3282D-27	.2348D-28

Algorithm Error

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

Effect of Argument Error

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

DTOD

DTOD is an exponentiation routine that accepts compiler-generated calls. DTOD performs exponentiation for program statements that raise double precision quantities to double precision exponents. It accepts two double precision arguments and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTOD and DTOD, and the call-by-value entry point is MLP$VDTOD.

The DTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOD(scalar,vector) = MLP$DTODV
DTOD(vector,scalar) = MLP$DVTOD
DTOD(vector,vector) = MLP$DVTODV
```

The input domain for this routine is the collection of all valid double precision pairs (x,y), where x is positive and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, positive double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

x**y = exp(y*log(x)), where x > 0.

Upon entry, the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The routine DTOD was tested. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-19 shows a summary of these statistics.

Table 2-19. Relative Error DTOD

Test	Interval		Maximum	Root Mean Square
	From	To		
x**y against x**2**(y/2)	x interval .1000D-01	.1000D+02	.5172D-25	.9207D-26
	y interval -.6167D+03	.6167D+03		
x**2**1.5 against x**2*x	.1000D+01 .5000D+00	.8053+411 .1000D+01	.1133D-24 .1143D-27	.4805D-25 .3978D-28
x**1.0 against x	.5000D+00	.1000D+01	.7133D-28	.3195D-28

Effect of Argument Error

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

```
r*(log(b)*e(p) + p*e(b)/b)
```

DTOI

DTOI is an exponentiation routine that accepts compiler-generated calls. DTOI performs exponentiation for program statements that raise double precision quantities to integer exponents. It accepts a double precision argument and an integer argument, and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTOI and DTOI, and the call-by-value entry point is MLP$VDTOI.

The DTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOI(scalar,vector) = MLP$DTOIV
DTOI(vector,scalar) = MLP$DVTOI
DTOI(vector,vector) = MLP$DVTOIV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a double precision quantity and y is an integer quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

```
y = i(0)*2**0 + i(1)*2**1 + ... i(n)*2**n
```

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

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1].
```

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

```
x**y    = 1, if y = 0 and x ≠ 0.
    = (x**2)**(y/2), if y > 0 and y is even.
    = x*(x**2)**((y - 1)/2), if y > 0 and y is odd.
```

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

```
(1/x)(u) = n(1/a(u)) + (((n(1 - ((1/a(u))*a(u)) +
           (1 - (1)(1/a(u)*a(u)) - (1/a(u)*(1)a(u)) -
           (1/a(u)*a(1)/a(u))) + (1/a(u) + (1)(((n(1 -
           (1/a(u))*a*u) + (1 - (1)(1/x(u))*x(u))) -
           (1/a(u))*(1)a(u) - (1/a(u))*a(1))/a(u))(1/x)(1) =
           n(...) + (1) (...)
```

In the routine, double precision quantities a = a(u)*a(1) and b = b(u)*b(1) are multiplied according to:

```
a*b = (a*b)(u)*(a*b)(1)
```

where:

```
(a*b)(u) = (((a(u)*b(1)) + (a(1)*b(u))) + (a(u)*(1)b(u))) +
(a(u)*b(u))
```

and

```
(a*b)(1) = (((a(u)*b(1)) + (a(1)*b(u))) + (a(u)*(1)b(u))) +
(1)(a(u)*b(u))
```

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Not applicable.

Effect of Argument Error

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

DTOX

DTOX is an exponentiation routine that accepts compiler-generated calls. DTOX performs exponentiation for program statements that raise double precision quantities to real exponents. It accepts a double precision argument and a real argument and returns a double precision result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTOX and DTOX, and the call-by-value entry point is MLP$VDTOX.

The DTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOX(scalar,vector) = MLP$DTOXV
DTOX(vector,scalar) = MLP$DVTOX
DTOX(vector,vector) = MLP$DVTOXV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a nonnegative double precision quantity and y is a real quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, positive double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the description of routine DTOD.

Effect of Argument Error

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

```
r*(e(p)*log(b) + p*e(b)/b)
```

DTOZ

DTOZ is an exponentiation routine that accepts compiler-generated calls. DTOZ performs exponentiation for program statements that raise double precision quantities to complex exponents. It accepts a double precision argument and a complex argument and returns a complex result. This function cannot be called from a CYBIL program.

The call-by-reference entry points are MLP$RDTOZ and DTOZ, and the call-by-value entry point is MLP$VDTOZ.

The DTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
DTOZ(scalar,vector)=MLP$DTOZV
DTOZ(vector,scalar)=MLP$DVTOZ
DTOZ(vector,vector)=MLP$DVTOZV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a double precision quantity and y is a complex quantity. If x is equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y is not equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

```
base**exponent = x + i*y
```

Upon entry, the double precision base, x, is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y is not equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error of these arguments was found to be 1.7431E-11.

Effect of Argument Error

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

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

ERF

ERF is a function that computes the error function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RERF and ERF, the call-by-value entry point is MLP$VERF, and the vector entry point is MLP$ERFV.

The input domain for this routine is the collection of all valid real quantities. The output range is included in the set of real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The routine calculates the smaller of erf(abs(x)) and erfc(abs(x)). The final value, which is the sum of a signed function and a constant, is computed by using the identities:

```
erf(-x) = -erf(x)
erf(x)  = 1.0 - erfc(x)
```

The forms used are given in table 2-20.

Table 2-20. Forms Used in ERF. (y = ABS(x))

Range	ERF	ERFC
[-INF,-5.625]	-1.0	+2.0
(-5.625,-.477)	-1.0+p2(y)	+2.0-p2(y)
[-.477,0)	-p1(y)	+1.0+p1(y)
[0,+.477]	+p1(y)	+1.0-p1(y)
[.477,5.625)	+1.0-p2(y)	p2(y)
[5.625,8.0)	+1.0	p2(y)
[8.0,53.0]	+1.0	p3(y)
(53.0,+INF)	+1.0	underflow
+INF	+1.0	0.0

The constants .477 and 53.0 are inverse erf(.5) and inverse erfc(2**-975), which are approximately .47693627620447 and 53.0374219959898.

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

(The coefficients for p2 and p3 are from Hart, Cheney, Lawson, et al., Computer Approximations, New York, 1968, John Wiley and Sons).

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function ERF was tested against 1 - e**(-x**2)*p(x)/q(x)´. A group of 10,000 arguments was chosen randomly from the interval (0.0,8.0). The maximum relative error of these arguments was found to be .2050E-13.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(-x**2) and that of relative error is (2/sqrt(pi))*x*exp(-x**2)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC when x < .53. For x > .53, the relative error for ERFC is amplified by approximately 2x.

ERFC

ERFC is a function that computes the complementary error function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RERFC and ERFC, the call-by-value entry point is MLP$VERFC, and the vector entry point is MLP$ERFCV.

The input domain for this routine is the collection of all valid real quantities less than 53.037, but not equal to infinity. The output range is included in the set of valid, nonnegative real quantities less than or equal to 2.0.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is greater than 53.037, but not equal to infinity.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

See the description of routine ERF.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is greater than 53.037, but not equal to infinity.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function ERFC was tested against e**(-x**2)*p(x)/q(x)´. A group of 10,000 arguments was chosen randomly from the interval (0.0,8.0). The maximum relative error of these arguments was found to be .9531E-11.

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is (2/sqrt(pi))*exp(-x**2) and that of relative error is (2/sqrt(pi))*x*exp(-x**2)/f(x) where f is erf or erfc. The relative error is attenuated for ERF everywhere and for ERFC when x < .53. For x > .53, the relative error for ERFC is amplified by approximately 2x.

EXP

EXP is a function that computes the exponential function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$REXP and EXP, the call-by-value entry point is MLP$VEXP, and the vector entry point is MLP$EXPV.

The input domain for this routine is the collection of all valid real quantities whose value is greater than or equal to -4097*log(2) and less than or equal to 4095*log(2). The output range is included in the set of valid positive real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than -4097*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

If x is valid, EXP(x) is calculated by reducing it to the simpler task of approximating e**g*2**(NL/32). This reduction is derived as follows:

```
exp(x) = e**(g + (32*NH + NL)*(ln(2)/32))
       = e**(g + NH*ln(2) + (NL/32)*ln(2))
       = e**g*2**NH*2**(NL/32)
       = (e**g*2**(NL/32))*2**NH
```

where

- n is the nearest integer to 32*x/ln(2).
- g is a real number such that x = g + n*(ln(2)/32). Thus, abs(g) is less than or equal to ln(2)/64.
- NH is floor(n/32).
- NL is greater than or equal to 0, less than or equal to 31, and is the integer such that n = 32*NH + NL.

The reduction:

```
e**g*2**(NL/32)
```

is approximated to 48 bits of precision using the following min-max approximation:

```
Z = Q(NL, g) + Qbias(NL)
```

where for each of the 32 values of NL, Qbias(NL) is a number that is represented exactly in binary floating point and which is slightly less than 2**(-1/64)*2**(NL/32), which is the minimum value of e**g*2**(NL/32).

Q(NL, g) denotes the 32 quintic polynomials in g which approximate e**g*2**(NL/32) - Qbias(NL) with the lowest maximum relative error for abs(g) < = ln(2)/64. Z is evaluated with almost no error since the low bits of Q(NL, g), which may be inaccurate due to truncation errors, are insignificant with respect to Qbias(NL). Thus, Z*2**NH, which is evaluated simply by adding NH to the exponent of Z, is an accurate approximation to EXP(x).

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is greater than 4095*log(2).

It is less than -4097*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-21 shows a summary of these statistics.

Table 2-21. Relative Error of EXP

Test	Interval From	Interval To	Maximum	Root Mean Square
EXP(x - 2.8125) against EXP(x)/EXP(2.8125)	-.3466E+01	-.2805E+04	.7335E-14	.3766E-14
EXP(x - .0625) against EXP(x)/EXP(.0625)	-.2841E+00	.3466E+00	.7557E-14	.3945E-14
EXP(x - 2.8125) against EXP(x)/EXP(2.8125)	.6931E+01	.2838E+04	.7384E-14	.3850E-14

Effect of Argument Error

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

EXTB

EXTB is a function that extracts bits from the first argument, x, as specified by the second and third arguments, i1 and i2. It accepts any type except character for argument x and accepts integer for arguments i1 and i2. The result is boolean. If x is of type double precision or complex, only the first word is used. The result returned contains 0 fill in the second word; however, the first 4 bytes of the first word is duplicated in the second word for double precision.

Argument x must be byte aligned and be at least 64 bits in length. The argument used is the leftmost 64 bits of x. Argument i1 indicates the first bit to be extracted numbering from bit 0 on the left. Argument i2 indicates the number of bits to be extracted. The extracted bits occupy the rightmost bits of the result, with 0 bits as fill on the left.

The call-by-reference entry points are MLP$REXTB and EXTB, and the call-by-value entry point is MLP$VEXTB.

The input domain for this routine is such that i1 is greater than or equal to 0 and less than 64; i2 is greater than or equal to 0; and i1 + i2 is less than or equal to 64. If i2 = 0, the result is 0 (all 0 bits). The data type of argument x is not significant to the processing of this function. The output range is included in the set of valid boolean quantities.

Call-By-Reference Routine

The arguments i1 and i2 are checked upon entry. They are invalid if:

i1 is less than zero.

i2 is less than zero.

i1 is greater than or equal to 64.

i1 + i2 is greater than 64.

If the arguments are invalid, a diagnostic message is displayed. If the arguments are valid, the call-by-value routine is branched to, and the result of the function is returned to the calling program.

Call-By-Value Routine

The extracted bits from the first argument, x, as specified by the second and third arguments i1 and i2 is returned. The leftmost 64 bits of x is used.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

IABS

IABS is a function that computes the absolute value of an argument. It accepts an integer argument and returns an integer result.

The call-by-reference entry points are MLP$RIABS and IABS, and the call-by-value entry point is MLP$VIABS.

The input domain for this routine is the collection of all valid integer quantities. The output range is included in the set of valid, nonnegative integer quantities.

Call-By-Reference Routine

No errors are generated by IABS. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the argument is extended throughout a word to form a mask. The argument is subtracted from the exclusive OR of the mask and the argument to form the result.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

IDIM

IDIM is a function that computes the positive difference between two arguments. It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RIDIM and IDIM, and the call-by-value entry point is MLP$VIDIM.

The input domain for this routine is the collection of all valid integer pairs (x,y) such that x - y is less than 2**63. The output range is included in the set of valid, nonnegative integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

x - y is greater than or equal to 2**63.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

Upon entry, the difference between the two arguments is formed, and the sign bit is extended across another word to form a mask. The boolean product of the mask's complement and the difference is formed.

Given arguments (x,y):

result = x - y if $x > y$
result = 0 if $x \leq y$.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

IDNINT

IDNINT is a function that returns the nearest integer to an argument. It accepts a double precision argument and returns an integer result.

The call-by-reference entry points are MLP$RIDNINT and IDNINT, and the call-by-value entry point is MLP$VIDNINT.

The input domain for this routine is the collection of all valid double precision quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is ≥ 0, .5 is added to it, and the result is added to a special floating point zero that forces truncation. If the argument is < 0, -.5 is added to it, and the result is added to a special floating point zero that forces truncation.

If the value of the argument is not in the range [-2**63 - 2**15,2**63 - 2**15], then the high order bits of the resulting integer are lost (the result is truncated in its leftmost position).

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

INSB

INSB is a function that inserts bits from the first argument, x, into a copy of the fourth argument, y, as specified by the second and third arguments, i1 and i2. It accepts any type except character for arguments x and y, and accepts integer for arguments i1 and i2. The result is boolean. If x or y is of type double precision or complex, only the first word is used. The result returned contains 0 fill in the second word; however, the first 4 bytes of the first word is duplicated in the second word for double precision.

Arguments x and y must be byte aligned and be at least 64 bits in length. The argument used is the leftmost 64 bits of each x and y. Argument i1 indicates first bit position in y for insertion. Argument i2 indicates the rightmost number of bits taken from x to be inserted into y.

The call-by-reference entry points are MLP$RINSB and INSB, and the call-by-value entry point is MLP$VINSB.

The input domain for this routine is such that i1 is greater than or equal to 0 and less than 64; i2 is greater than or equal to 0; and i1 + i2 is less than or equal to 64. If i2 = 0, the result is the value of y. The data type of arguments x and y is not significant to the processing of this function. The output range is included in the set of valid boolean quantities.

Call-By-Reference Routine

The arguments i1 and i2 are checked upon entry. They are invalid if:

i1 is less than zero.

i2 is less than zero.

i1 is greater than or equal to 64.

i1 + i2 is greater than 64.

If the arguments are invalid, a diagnostic message is displayed. If the arguments are valid, the call-by-value routine is branched to, and the result of the function is returned to the calling program.

Call-By-Value Routine

The inserted bits from the first argument, x, into a copy of the fourth argument, y, as specified by the second and third arguments, i1 and i2 is returned. The leftmost 64 bits of x and y are used.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ISIGN

ISIGN is a function that transfers the sign of one argument to another argument. It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RISIGN and ISIGN, and the call-by-value entry point is MLP$VISIGN.

The input domain for this routine is the collection of all valid integer quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

No errors are generated by ISIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The exclusive OR of the first argument with the second argument is shifted to extend its sign bit across a word to produce a mask. The mask is then subtracted from the exclusive OR of the mask and argument to form the result.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ITOD

ITOD is an exponentiation routine that accepts compiler-generated calls. ITOD performs exponentiation for statements that raise integer quantities to double precision exponents. It accepts an integer argument and a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RITOD and ITOD, and the call-by-value entry point is MLP$VITOD.

The input domain for this routine is the collection of all valid pairs (x,y), where x is a nonnegative integer quantity and y is a double precision quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

y is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

x**y = exp(y*log(x)), where x > 0.

Upon entry, the integer argument is converted to double precision, and the routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Error Analysis

See the description of routine DTOD.

Effect of Argument Error

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

ITOI

ITOI is an exponentiation routine that accepts compiler-generated calls. ITOI performs exponentiation for program statements that raise integer quantities to integer exponents. It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RITOI and ITOI, and the call-by-value entry point is MLP$VITOI.

The input domain for this routine is the collection of all valid integer pairs (x,y) such that the absolute value of x**y is less than 2**63. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

x is zero and y is zero or negative.

If the argument pair is invalid, zero is returned, and a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

```
 0**0 = error
 0**J = error if J < 0
 1**J = 1
-1**J = +1 or -1  (J even or odd)
 I**0 = 1
 I**J = 0 if J < 0
```

If the exponentiation does not fit any special case, the algorithm listed below is used for the computation.

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

```
y = i(0)*2**0 + i(1)*2**1 + ... + i(n)*2**n
n = (log(2)y) = greatest integer not exceeding log(2)y
```

Then:

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1]
```

The numbers x = x**0, x**2**0, x**2, x**4, ..., x**(2)**n are generated during the computation by successive squarings, and the coefficients i(0), ..., i(n) are obtained as sign bits of successive right shifts of y within the computer. A running product is formed during the computation so that smaller powers of x can be discarded. The computation then becomes an iteration of the algorithm:

```
x**y = 1, if y = 1, and x ≠ 0
     = (x*x)**(y/2), if y > 0 and y is even
     = (x*x)**((y-1)/2)*x, if y > 0 and y is odd
```

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

ITOX

ITOX is an exponentiation routine that accepts compiler-generated calls. ITOX performs exponentiation for program statements that raise integer quantities to real exponents. It accepts an integer argument and a real argument and returns a real result.

The call-by-reference entry points are MLP$RITOX and ITOX, and the call-by-value entry point is MLP$VITOX.

The input domain for this routine is the collection of all valid pairs (x,y), where x is a nonnegative integer quantity, y is a real quantity, and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. The argument pair is invalid if:

y is indefinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

$x^{**}y = \exp(y*\log(x))$, where $x \geq 1$

Upon entry, x is converted to real, and the routine calls to XTOX to compute the result. Zero is returned if the base is zero and the exponent is positive.

Error Analysis

See the description of routine XTOX.

Effect of Argument Error

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

ITOZ

ITOZ is an exponentiation routine that accepts compiler-generated calls. ITOZ performs exponentiation for statements that raise integer quantities to complex exponents. It accepts an integer argument and a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RITOZ and ITOZ, and the call-by-value entry point is MLP$VITOZ.

The ITOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
ITOZ(scalar,vector) = MLP$ITOZV
ITOZ(vector,scalar) = MLP$IVTOZ
ITOZ(vector,vector) = MLP$IVTOZV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a nonnegative nonzero integer quantity and y is a complex quantity. If x is equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is infinite.

x is equal to zero, and the real part of y is zero or negative, or the imaginary part of y is not equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

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

```
n**(x + i*y) = exp(x*log(n))*cos(y*log(n)) +
i*exp(x*log(n))*sin(y*log(n))
```

Upon entry, n is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

y is indefinite.

y is infinite.

x is equal to zero, and the real part of y is zero or negative, or the imaginary part of y is not equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error these arguments was found to be 1.7431E-11.

Effect of Argument Error

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

MOD

MOD is a function that computes the remainder of the ratio of two arguments It accepts two integer arguments and returns an integer result.

The call-by-reference entry points are MLP$RMOD and MOD, and the call-by-value entry point is MLP$VMOD.

The input domain for this routine is the collection of all valid integer pairs (x,y), where x is an integer quantity and y is a nonzero integer quantity. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

Upon entry, the argument pair (x,y) is checked. It is invalid if:

y is equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is branched to, and the result is returned.

Call-By-Value Routine

Upon entry, the arguments x and y are converted to real, the quotient x/y is formed, and the result is multiplied by y and then subtracted from x.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

NINT

NINT is a function that finds the nearest integer to an argument. It accepts a real argument and returns an integer result.

The call-by-reference entry points are MLP$RNINT and NINT, and the call-by-value entry point is MLP$VNINT.

The input domain for this routine is the collection of all valid real quantities. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is branched to, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If the argument is $\geq$ 0, .5 is added to it, or if the argument is < 0, -.5 is added to it. This sum is converted from floating point to integer and returned.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

RANF

RANF is a function that generates the next random number in a series of random numbers. It accepts a dummy argument and returns a real result.

The call-by-reference entry points are MLP$RRANF and RANF, and the call-by-value entry point is MLP$VRANF.

There is no input domain to this routine. The output range is included in the set of positive real quantities less than 1.0.

Call-By-Reference Routine

No errors are generated in RANF. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

RANF uses the multiplicative congruential method modulo 2**48. The formula is:

```
x(n + 1.0) = a*x(n) (mod 2**48)
```

The library holds a random seed (mlv$initial¢seed) and a multiplier (mlv$random¢multiplier). The random seed can be changed to any valid seed value prior to calling RANF by use of the routine RANSET. Upon entry at RANF, the random seed is multiplied in double precision by mlv$random¢multiplier to generate a 96-bit product, which is the new seed partially normalized by one bit. This result is then denormalized. The lower 48 bits are formed with an exponent that yields a result between 0 and 1.0 to become the new random seed (mlv$random¢seed). The current seed for the task is updated with the newly formed unnormalized seed. The seed is used to generate subsequent random numbers. The default initial value of mlv$initial¢seed is 40002BC68CFE166D hexadecimal. The new random seed is normalized and returned as the random number.

The multiplier (mlv$random¢multiplier) is constant and has a value of 40302875A2E7B175 hexadecimal. This multiplier passes the Coveyou-MacPherson test, the auto-correlation test with lag $\leq$ 100, the pair triplet test, and other statistical tests for randomness.

(Algorithm and Constants, Copyright 1970 by Krzysztof Frankowski, Computer Information and Control Science, University of Minnesota, 55455.)

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

RANGET

RANGET is a callable program procedure that returns the current random number seed of a task. It accepts a real argument.

The call-by-reference entry points are MLP$RANGET and RANGET. There is no call-by-value routine for RANGET.

The result is returned through parameter n and is a positive real quantity in the interval (0,1.0).

Call-By-Reference Routine

RANGET returns the current seed, between 0 and 1, of the random number generator. The value returned might not be normalized. This seed can be used to restart the random sequence at exactly the same point. The current seed is mlv$random¢seed.

Call-By-Value Routine

There are no call-by-value entry points for RANGET.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

RANSET

RANSET is a callable program procedure that sets the seed of the random number generator. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RANSET and RANSET. There is no call-by-value routine.

The input domain for this routine is the collection of all possible full word bit patterns. There is no output.

Call-By-Reference Routine

RANSET uses the value passed to it to form a valid seed for the random number generator. If the argument is zero, the seed is set to its initial value (mlv$initial¢seed) at load time. Otherwise, the value passed has its exponent set to 4000 hexadecimal, and the coefficient is made odd. This value is then saved and becomes the new seed (mlv$random¢seed) for the task.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

SIGN

SIGN is a function that transfers the sign from one argument to another argument. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RSIGN and SIGN, and the call-by-value entry point is MLP$VSIGN.

The input domain for this routine is the collection of all valid real quantities. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

No errors are generated by SIGN. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The sign bit of the second argument is inserted into the sign bit of the first argument.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

SIN

SIN is a function that computes the sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSIN and SIN, the call-by-value entry point is MLP$VSIN, and the vector entry point is MLP$SINV.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

See the description of routine COS.

This page intentionally left blank.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function SIN was tested against 3*SIN(x/3) - 4*SIN(x/3)**3. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-22 shows a summary of these statistics.

Table 2-22. Relative Error of SIN

Interval			
From	To	Maximum	Root Mean Square
0.0000E+00	.1571E+01	.8305E-14	.2874E-14
.1885E+02	.2042E+02	.1355E-13	.3168E-14

Effect of Argument Error

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

SIND

SIND is a function that computes the sine function of an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSIND and SIND, the call-by-value entry point is MLP$VSIND, and the vector entry point is MLP$SINDV.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [-45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x += 360 degrees) =  sin(x)
sin(x += 180 degrees) = -sin(x)
sin(x + 90 degrees)   =  cos(x)
sin(x - 90 degrees)   = -cos(x)
cos(x += 360 degrees) =  cos(x)
cos(x += 180 degrees) = -cos(x)
cos(x + 90 degrees)   = -sin(x)
cos(x - 90 degrees)   =  sin(x)
```

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The reduction to (-45,+45) is exact; the constant pi/180 has relative error 1.37E-15, and multiplication by this constant has a relative error 5.33E-15, and a total error of 6.7E-15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is at most 5.26E-15 plus the maximum error in SINCOS over (-pi/4,+pi/4). The maximum relative error observed for a group of 10,000 arguments chosen randomly in the interval [0,360] was, .1403E-13 for SIND, and .7105E-14 for COSD.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (-45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

SINH

SINH is a function that computes the hyperbolic sine function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSINH and SINH, the call-by-value entry point is MLP$VSINH, and the vector entry point is MLP$SINHV.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 4095*log(2). The output range is included in the set of all valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The formulas used to compute sinh(x) are:

```
      x  = n*log(2) + a, where |a| ≤ 1/2*log(2)
           and n is an integer
sinh(x) = (cosh(a) + sinh(a))*2**(n-1), when n > 25
sinh(x) = sinh(a), when n = 0, otherwise,
sinh(x) = (c - s)*2**(n-1) + (c + s)*2**(-n-1)
```

where:

```
s = sinh(a) = a + s(3)*a**3*(s(5) + TOP/(BOT - a**2))
c = cosh(a) = 1.0 + a**2*(.5 + a**2*(c(4) + a**2*(c(6) +
              c(10)*a**2*(c(8) + a**2))))
```

Constants used in the algorithm are:

```
s(3)  =   .166 666 666 666 935 58
s(5)  =  -.005 972 995 665 652 368
TOP   =  1.031 539 921 161
BOT   = 72.103 746 707 22
c(4)  =   .041 666 666 666 488 081
c(6)  =   .001 388 888 895 231 804 5
c(8)  = 89.754 738 973 150 22
c(10) =  2.763 250 805 803*10**-7
```

The algorithm used is:

a. u = |x|

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

c. s = w + w**3(s(3) + w**2(s(5) + TOP/(BOT - w**2)))
   d = w**2(1/2 + w**2(c(4) + w**2(c(6) + w**2(c(8) + w**2)*c(10))))
   a = (1.0 + d - s)*2**(-n-1)
   b = d + s

d. c = (1/4 + (1/4 + b))*2**(n-1) + (2**(n-3) + (2**(n-3) - a))
   XF = c with the sign of x

e. Return

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 4095*log(2).

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-23 shows a summary of these statistics.

Table 2-23. Relative Error of SINH

Test	Interval		Maximum	Root Mean Square
	From	To		
SINH(x) against Taylor series expansion of SINH(x)	0.0000E+00	.5000E+00	.3374E-13	.9969E-14
SINH(x) against c*(SINH(x + 1) + SINH(x - 1))	.3000E+01	.2838E+04	.2894E-13	.9979E-14

Effect of Argument Error

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

SQRT

SQRT is a function that computes the square root function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RSQRT and SQRT, the call-by-value entry point is MLP$VSQRT, and the vector entry point is MLP$SQRTV.

The input domain for this routine is the collection of all valid, nonnegative real quantities. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

If x is valid, let y be a real number in [0.5, 2) and n an integer such that x = y*2**(2*n). Then SQRT(x) is evaluated by:

```
SQRT(x) = SQRT(y)*2**n
```

Then SQRT(y) is approximated to 48 bits of precision by applying one iteration of Heron's rule to an initial approximation which is accurate to at least 24 bits of precision. The initial approximation is computed by dividing the interval [0.5, 2) into the following 64 subintervals:

```
[32/64, 33/64)
     .
     .
     .
[63/64, 64/64)
[32/32, 33/32)
     .
     .
     .
[63/32, 64/32)
```

The coefficients of these 64 min-max approximations are stored in three tables p0, p1, and p2 such that:

```
z1 = p0[i] + p1[i]*y + p2[i]*y**2
```

is the quadratic min-max approximation to the square root of y over the subinterval whose index is i. The required initial approximation is obtained by calculating the index i of the subinterval that contains y and then evaluating the above quadratic polynomial so that z1 approximates SQRT(y) to at least 24 bits of precision.

Using Heron's rule, the computation:

```
twoz2 = z1 + y/z1
```

approximates SQRT(y) to 48 bits precision followed by the computation:

```
SQRT(x) = twoz2*2**(n - 1)
```

which approximates SQRT(x) to 48 bits of precision.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

It is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The function SQRT was tested in the form SQRT(x*x) - x. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-24 shows a summary of these statistics.

Table 2-24. Relative Error of SQRT

Interval			
From	To	Maximum	Root Mean Square
.1000E+01	.1414E+01	.7099E-14	.5677E-14
.7071E+00	.1000E+01	.5023E-14	.4106E-14

Effect of Argument Error

For a small error in the argument y, the amplification of absolute error is 1/(2*sqrt(y)) and that of relative error is .5.

SUM1S

SUM1S is a function that returns the number of bits in a word. It accepts any type of argument except character and logical and returns an integer result. If the argument is of type double precision or complex, only the first word is used.

The call-by-reference entry points are MLP$RSUM1S and SUM1S, and the call-by-value entry point is MLP$VSUM1S.

The input domain for this routine is the collection of all valid boolean, real, complex, integer, or double precision quantities. Character and logical are not allowed. The output range is included in the set of valid integer quantities.

Call-By-Reference Routine

No errors are generated by SUM1S. The call-by-reference routine branches to the call-by-value routine.

Call-By-Value Routine

The number of bits in a word is returned. The argument can be any type except character and logical.

Error Analysis

Not applicable.

Effect of Argument Error

Not applicable.

TAN

TAN is a function that computes the trigonometric circular tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTAN and TAN, the call-by-value entry point is MLP$VTAN, and the vector entry point is MLP$TANV.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The evaluation is reduced to the interval [-.5,.5] by using the identities:

1. tan(x) = tan(x + k*pi/2), if k is even

2. tan(x) = -1.0/tan(x + pi/2)

in the form:

3. tan(x) = tan((pi/2)*(x*2/pi + k)), if k is even

4. tan(x) = -1.0/tan((pi/2)*(x*2/pi + 1.0))

An approximation of tan(pi/2*y) is used. The argument is reduced to the interval [-.5,.5] by subtracting a multiple of pi/2 from x in double precision.

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

Identity 4 is used if the integer subtracted is odd. The result is negated and inverted by dividing -Q/P instead of P/Q.

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The range reduction, the final add in each part of the rational form, the final multiply in P, and the divide dominate the error. Each of these operations contributes directly to the final error, and each is accurate to about 1/2 ulp.

The function TAN was tested against 2*TAN(x/2)/(1 - TAN(x/2)**2). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-25 shows a summary of these statistics.

Table 2-25. Relative Error of TAN

Interval			
From	To	Maximum	Root Mean Square
0.0000E+00	.7854E+00	.2177E-13	.5613E-14
.1885E+02	.1963E+02	.1993E-13	.5617E-14
.2749E+01	.3534E+01	.2190E-13	.7286E-14

Effect of Argument Error

For small errors in the argument x, the amplification of absolute error is sec(x)**2, and that of relative error is x/(sin(x)*cos(x)), which is at least 2x and can be arbitrarily large near a multiple of pi/2.

TAND

TAND is a function that computes the trigonometric tangent for an argument in degrees. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTAND and TAND, the call-by-value entry point is MLP$VTAND, and the vector entry point is MLP$TANDV.

The input domain for this routine is the collection of all valid real arguments whose absolute value is less than 2**47, excluding odd multiples of 90. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The result is put in the interval [-45,45] by finding the nearest integer n to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The routine calls TAN to compute the tangent, and if the multiple n of 90 is odd, the result is negated and inverted by using the identities:

$$\tan(x \pm 180 \text{ degrees}) = \tan(x)$$
$$\tan(x \pm 90 \text{ degrees}) = -1/\tan(x)$$

Vector Routine

The argument is checked upon entry. It is invalid if:

It is indefinite.

It is infinite.

Its absolute value is greater than or equal to 2**47.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The reduction to (-45,+45) is exact; the constant pi/180 has a relative error of 1.37E-15, and multiplication by this constant has a relative error of 5.33E-15, so the total error is 6.7E-15. The maximum relative error observed for 10,000 arguments chosen randomly in the interval [0,360], was .2130E-13.

Effect of Argument Error

Errors in the argument x are amplified at most by x/(sin(x)*cos(x)). This function has a maximum of pi/2 within (-45,+45) but has poles at all multiples of 90 degrees except zero.

TANH

TANH is a function that computes the hyperbolic tangent function. It accepts a real argument and returns a real result.

The call-by-reference entry points are MLP$RTANH and TANH, the call-by-value entry point is MLP$VTANH, and the vector entry point is MLP$TANHV.

The input domain for this routine is the collection of all valid real quantities. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Reference Routine

The argument is checked upon entry. It is invalid if it is indefinite.

If the argument is invalid, a diagnostic message is displayed. If the argument is valid, the call-by-value routine is called, and the result of the computation is returned to the calling program.

Call-By-Value Routine

The argument range is reduced to:

```
tanh(x) = 1.0 - 2*(q - p)/((q - p) + 2**n*(q + p))
```

by the identities:

```
tanh(-x) = -tanh(x) for x < 0
tanh(x)  = p(x)/q(x) approximately, in the interval [0,.55]
tanh(x)  = 1.0 - 2/(exp(2*x) + 1.0)
exp(2*x) = (1.0 + tanh(x))/(1.0 - tanh(x))
exp(2*x) = 2**n*exp(2*(x - n*ln(2)/2))
```

where n is chosen to be nint(x*2/ln(2)) and p and q are evaluated on x - n*ln(2)/2. This choice of n minimizes abs(x - n*ln(2)/2).

When abs(x) $\leq$ .55 = atanh(.5), the approximation p(x)/q(x) is used. When abs(x) $>$ .55, the above range reduction is used. For abs(x) > 17.1, tanh(x) = sign(1.0,x).

The approximation p/q is a minimax (relative error) rational form (5th order odd)/(6th order even). The range reduction is simplified by scaling the coefficients so that (x*2/ln(2) - n) can be used instead of (x - n*ln(2)/2). The coefficients are further scaled by an amount sufficient to reduce truncation error in the leading coefficients without otherwise affecting accuracy.

Vector Routine

The argument is checked upon entry. It is invalid if it is indefinite.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The algorithm error due to finite approximation and coefficient truncation is 1.7E-15. For abs(x) < .55, the form p(x)/q(x) is used. The final operations z = x*2/ln(2) and tanh(z*(p0+small))/(q0+small) dominate the error. For abs(x) > 1.25 the final subtraction, 1.0 - small dominates.

For .55 $\leq$ abs(x) $\leq$ 1.25, the final operation is 1-R, where R becomes smaller as x approaches 1.25. Thus, the worst relative error is near .55, namely, (contribution from R) + (error in final sum), where R = 2*(q - p)/((q - p) + 4*(q + p)).

The function TANH was tested against (TANH(x - 1/8) + TANH(1/8))/(1 + TANH(x - 1/8)*TANH(1/8)). Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-26 shows a summary of these statistics.

Table 2-26. Relative Error of TANH

Interval			
From	To	Maximum	Root Mean Square
.1250E+00	.5493E+00	.4091E-13	.1085E-13
.6743E+00	.1768E+02	.2842E-13	.3730E-14

Effect of Argument Error

For small errors in the argument x, the amplification of the absolute error is 1/cosh**(x) and of relative error is x/(sinh(x)*cosh(x)). Both have maximum values of 1.0 at zero and approach zero as x gets large.

XTOD

XTOD is an exponentiation routine that accepts compiler-generated calls. XTOD performs exponentiation for program statements that raise real quantities to double precision exponents. It accepts a real argument and a double precision argument and returns a double precision result.

The call-by-reference entry points are MLP$RXTOD and XTOD, and the call-by-value entry point is MLP$VXTOD.

The XTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOD(scalar,vector) = MLP$XTODV
XTOD(vector,scalar) = MLP$XVTOD
XTOD(vector,vector) = MLP$XVTODV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a nonnegative real quantity and y is a double precision quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid double precision quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result if valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the argument x is converted to double precision, and all operations are carried out in double precision. The routine calls DLOG to compute log(x), and DEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

See the description of routine DTOD.

Effect of Argument Error

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

```
r*(e(p)*log(b) + p*e(b)/b).
```

XTOI

XTOI is an exponentiation routine that accepts compiler-generated calls. XTOI performs exponentiation for program statements that raise real quantities to integer exponents. It accepts a real argument and an integer argument, and returns a real result.

The call-by-reference entry points are MLP$RXTOI and XTOI, and the call-by-value entry point is MLP$VXTOI.

The XTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOI(scalar,vector) = MLP$XTOIV
XTOI(vector,scalar) = MLP$XVTOI
XTOI(vector,vector) = MLP$XVTOIV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a real quantity and y is an integer quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The arguments are checked to see whether the exponentiation conforms to a special case. If it does, the proper value is immediately returned. If the special case is an error condition, an error message is displayed. The special cases are:

```
x indefinite = error
x infinite   = error
0**0         = error
x**i         = 1.0 if i = 0 and x > 0
x**i         = 1.0/X**-i if i < 0
x = 0        = error if i < 0
```

If the exponentiation is not a special case, one of two methods is used to perform the exponentiation. Method 1 is a quick algorithm and is usually used. Method 2 is used when the number of bits in i plus the number of bits in x is greater than 8.

Method 1

Starting with the most significant bit, the binary representation of i is scanned. The result, which was initialized to x, is squared for each bit. If the next bit is one, the result is also multiplied by x.

Method 2

Ten bits of i are scanned as described in method 1. This procedure is repeated until i is used up. The result is returned if the exponent is not too large.

Error Analysis

Not applicable.

Effect of Argument Error

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

XTOX

XTOX is an exponentiation routine that accepts compiler-generated calls. XTOX performs exponentiation for program statements that raise real quantities to real exponents. It accepts two real arguments and returns a real result.

The call-by-reference entry points are MLP$RXTOX and XTOX, and the call-by-value entry point is MLP$VXTOX.

The XTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOX(scalar,vector) = MLP$XTOXV
XTOX(vector,scalar) = MLP$XVTOX
XTOX(vector,vector) = MLP$XVTOXV
```

The input domain for this routine is the collection of all valid real pairs (x,y), where x is a nonnegative quantity and x**y is a valid quantity. If x is equal to zero, then y must be greater than zero. The output range is included in the set of valid, nonnegative real quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0
```

Upon entry, the routine calls ALOG to compute log(x), and EXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

The routine XTOX was tested. Groups of 2,000 arguments were chosen randomly from given intervals. Statistics on relative error were observed. Table 2-27 shows a summary of these statistics.

Table 2-27. Relative Error XTOX

Test	Interval From	Interval To	Maximum	Root Mean Square
x**y against x**2**(y/2)	x interval .1000E-01	.1000E+02	.3547E-12	.6352E-13
	y interval -.6167E+03	.6167E+03		
x**2**1.5 against x**2*x	.1000E+01	.8053+411	.1360E-13	.5687E-14
	.5000E+00	.1000E+01	.1360E-13	.5715E-14
x**1.0 against x	.5000E+00	.1000E+01	.6802E-14	.3442E-14

Effect of Argument Error

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

```
r*(log(b)*e**p + p*(e(b))/b).
```

XTOZ

XTOZ is an exponentiation routine that accepts compiler-generated calls. XTOZ performs exponentiation for program statements that raise real quantities to complex exponents. It accepts a real argument and a complex argument and returns a complex result.

The call-by-reference entry points are MLP$RXTOZ and XTOZ, and the call-by-value entry point is MLP$VXTOZ.

The XTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
XTOZ(scalar,vector) = MLP$XTOZV
XTOZ(vector,scalar) = MLP$XVTOZ
XTOZ(vector,vector) = MLP$XVTOZV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a real quantity, y is a complex quantity, and x**y is a valid quantity. If x is zero, the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, or the imaginary part of y does not equal zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the real argument x is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

x is negative.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error these arguments was found to be 1.7431E-11.

Effect of Argument Error

If a small error e(x) occurs in the base x, and a small error e(z) (e^(x) + i*e^(y)) occurs in the exponent z, the error in the result w is given approximately by:

w*(log(x)* e(z) + z*e(x)/x).

ZTOD

ZTOD is an exponentiation routine that accepts compiler-generated calls. ZTOD performs exponentiation for program statements that raise complex quantities to double precision exponents. It accepts a complex argument and a double precision argument and returns a complex result.

The call-by-reference entry points are MLP$RZTOD and ZTOD, and the call-by-value entry point is MLP$VZTOD.

The ZTOD vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOD(scalar,vector) = MLP$ZTODV
ZTOD(vector,scalar) = MLP$ZVTOD
ZTOD(vector,vector) = MLP$ZVTODV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a complex quantity, y is a double precision quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the double precision argument y is converted to complex, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error these arguments was found to be 1.7431E-11.

Effect of Argument Error

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

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

ZTOI

ZTOI is an exponentiation routine that accepts compiler-generated calls. ZTOI performs exponentiation for program statements that raise complex quantities to integer exponents. It accepts a complex argument and an integer argument, and returns a complex result.

The call-by-reference entry points are MLP$RZTOI and ZTOI, and the call-by-value entry point is MLP$VZTOI.

The ZTOI vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOI(scalar,vector) = MLP$ZTOIV
ZTOI(vector,scalar) = MLP$ZVTOI
ZTOI(vector,vector) = MLP$ZVTOIV
```

The input domain for this routine is the collection of all valid pairs (x,y), where x is a complex quantity, y is a integer quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Let x represent the base and y represent the exponent. If y has binary representation 000... .000i(n)i(n-1) ...i(1)i(0), where each i(j)($0 \leq j \leq n$) is 0 or 1, then:

```
y = i(0)*2**0 + i(1)*2**1 + ... + i(n)*2**n
n = (log(2)y) = greatest integer not exceeding log(2)y
```

Then:

```
x**y = prod[x**2**j : 0 ≤ j ≤ n and i(j) = 1]
```

The numbers x**0, x = x**2**0, x**2, x**4, ..., x**(2)**n are generated during the computation by successive squarings, and the coefficients i(0), ..., i(n) are obtained as sign bits of successive circular right shifts of y within the computer. A running product is formed during the computation so that smaller powers of x can be discarded. The computation then becomes an iteration of the algorithm:

```
x**y    = 1, if y = 0 and x is ≠ 0
     = (x*x)**(y/2), if y > 0 and y is even
     = (x*x)**((y-1)/2)*x, if y > 0 and y is odd
```

Upon entry, if the exponent y is negative, y is replaced by -y and a sign flag is set. x**y is computed according to this algorithm, and if the sign flag was set, the result is reciprocated before being returned to the calling program.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

x is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

Not applicable.

Effect of Argument Error

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

ZTOX

ZTOX is an exponentiation routine that accepts compiler-generated calls. ZTOX performs exponentiation for program statements that raise real quantities to complex exponents. It accepts a complex argument and a real argument, and returns a complex result.

The call-by-reference entry points are MLP$RZTOX and ZTOX, and the call-by-value entry point is MLP$VZTOX.

The ZTOX vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOX(scalar,vector) = MLP$ZTOXV
ZTOX(vector,scalar) = MLP$ZVTOX
ZTOX(vector,vector) = MLP$ZVTOXV
```

The input domain for this routine is the collection of all valid argument pairs (x,y), where x is a complex quantity, y is a real quantity, and x**y is a valid quantity. If the real and imaginary parts of x are equal to zero, then y must be greater than zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

Upon entry, the real argument is converted to a complex argument, and the routine calls ZTOZ to compute the result.

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error these arguments was found to be 1.7431E-11.

Effect of Argument Error

If a small error e(z1) occurs in the base z1 and a small error e(z2) occurs in the exponent z2, the error in the result w is given approximately by:

w*(e(z2)*log(z1) + z2*e(z1)/z1.

ZTOZ

ZTOZ is an exponentiation routine that accepts compiler-generated calls. ZTOZ performs exponentiation for program statements that raise complex quantities to complex exponents. It accepts two complex arguments and returns a complex result.

The call-by-reference entry points are MLP$RZTOZ and ZTOZ, and the call-by-value entry point is MLP$VZTOZ.

The ZTOZ vector math function is divided into three routines having three separate entry points defined as follows:

```
ZTOZ(scalar,vector) = MLP$ZTOZV
ZTOZ(vector,scalar) = MLP$ZVTOZ
ZTOZ(vector,vector) = MLP$ZVTOZV
```

The input domain is the collection of all valid complex pairs (x,y). If the real and imaginary parts of x are equal to zero, then the real part of y must be greater than zero, and the imaginary part must be equal to zero. The output range is included in the set of valid complex quantities.

Call-By-Reference Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero, and the real part of y is less than or equal to zero, and the imaginary part of y does not equal zero.

If the argument pair is invalid, a diagnostic message is displayed. If the argument pair is valid, the call-by-value routine is called, and the result of the computation is returned to the call-by-reference routine. The result is checked. If the result is infinite, it is invalid, and a diagnostic message is displayed. If the result is valid, it is returned to the calling program.

Call-By-Value Routine

The formula used for computation is:

```
x**y = exp(y*log(x)), where x > 0.
```

Upon entry, argument checking is performed. If the arguments are valid, the routine calls CLOG to compute log(x), and CEXP to compute exp(y*log(x)).

Vector Routine

The argument pair (x,y) is checked upon entry. It is invalid if:

x is indefinite.

y is indefinite.

x is infinite.

y is infinite.

x is equal to zero and y is less than or equal to zero.

See Vector Error Handling in chapter 1 for further information.

Error Analysis

A group of 10,000 arguments was chosen randomly from the interval [[-1.0,1.0],[-1.0,1.0]] and [[-1.0,1.0],[-1.0,1.0]]. The maximum relative error of these arguments was found to be 1.7431E-11.

Effect of Argument Error

If a small error e(z1) occurs in the base z1 and a small error e(z2) occurs in the exponent z2, the error in the result w is given approximately by:

```
w*(e(z2)*log(z1) + z2*e(z1)/z1.
```

Auxiliary Routines 3

This chapter describes the auxiliary routines of the math library. These routines are called only by other math routines.

ACOSIN

ACOSIN is an auxiliary routine that computes the inverse sine or inverse cosine function. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for ACOSIN. The call-by-value entry points are MLP$VACOS and MLP$VASIN.

The input domain is the collection of all valid real quantities in the interval [-1.0,1.0]. The output range is included in the set of valid, nonnegative real quantities less than or equal to pi.

Call-By-Value Routine

Formulas used in the computation are:

```
arcsin(x) = -arcsin(-x),    x < -.5
arcos(x)  = pi - arcos(-x),    x < -.5
arcsin(x) = x + x**3*s*((w + z - j)*w + a + m/(e - x**2)),
  where -.5 < x < .5
arcos(x)  = pi/2 - arcsin(x),    -.5 ≤ x < .5
arcsin(x) = pi/2 - arcos(x),    .5 ≤ x < 1.0
arcos(x)  = arcos(1-ITER((1 - x),n))/2**n,    .5 ≤ x < 1.0
arcsin(1) = pi/2
arcos(1)  = 0
```

where:

```
        w  = (x**2 - c)*z + k
        z  = (x**2 + r)x**2 + i
ITER(y,n) = n iterations of y = 4*y - 2*y**2
```

The constants used are:

```
r =    3.173 170 078 537 13
e =    1.160 394 629 739 02
m =   50.319 055 960 798 3
c =   -2.369 588 855 612 88
i =    8.226 467 970 799 17
j =  -35.629 481 597 455 5
k =   37.459 230 925 758 2
a =  349.319 357 025 144
s =     .746 926 199 335 419*10**-3
```

The approximation of arcsin (-.5,.5) is an economized approximation obtained by varying r,e,m,...,s.

The algorithm used is:

a. If ACOS entry, go to step g.

b. If $|x| \geq .5$, go to step h.

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

d. z = (y + r)*y + i
   w = (y - c)*z + k
   p = q + s*q*y*((w + z - j)*w + a + m/(e - y))
   p = u - p
   Y1 = p/2**n

e. If ASIN entry, go to step k.

f. If x is in (-.5,1.0), return.
   XF = 2*u - (Y1)
   Return.

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

h. If x = 1.0 or -1.0, go to step l.
   If x is invalid, go to step m.
      n = 0 (Loop counter).
      y = 1.0 - |x|, and normalize y.

i. h = 4*y - 2*y**2
   n = n + 1.0
   If $2*y \leq 2 - \text{sqrt}(3) = .267949192431$, y = h, and go to step i.

j. q = 1.0 - h, and normalize q.
   y = q**2
   u = pi/2
   Go to step d.

k. Y1 = u - (Y1), and normalize Y1.
   Affix sign of x to Y1 = XF.
   Return.

l. XF = pi/2, if x = 1.0.
   XF = -pi/2, if x = -1.0.
   If ASIN entry, return.
   XF = 0, if x = 1.0.
   XF = pi, if x = -1.0.
   Return.

m. Return.

Error Analysis

See the description of routines ACOS and ASIN.

Effect of Argument Error

If a small error e´ occurs in the argument x, the error in the result is given approximately by e´/(1.0 - x**2)**.5.

COSSIN

COSSIN is an auxiliary routine that accepts calls from other math routines that require simultaneous computation of the sine and cosine of the same argument. COSSIN accepts a real argument and returns two real results.

The entry points and input and output ranges for this routine are described in routines CSIN and CCOS.

Call-By-Value Routine

The argument is reduced to the interval [-pi/4,pi/4]. Polynomials p(x) and q(x) of degrees 11 and 12 are used to compute sin(x) and cos(x) over that interval. Upon entry, the argument x is multiplied by 2/pi. Then, the nearest integer n to 2/pi*x is computed. The upper and lower halves of the result are added. The value of y is in the interval (-pi/4,pi/4). y = x - n*pi/2 is computed in double precision as the reduced argument for input to p(y) and q(y). Then sin(x) and cos(x) are computed from these as indicated by the value k = mod(n,4), where k = 0, 1, 2, 3. The formula used to compute sine(x) is:

```
sin(x) = sin(y + n*p/2) = sin(y + k*pi/2)
       = sin(y)*cos*(k*pi/2) + cos(y)*sin(k*pi/2)
```

A similar formula is used for the computation of cosine(x). Depending upon k, either the sine(k = 0,1) or cosine(k = 2,3) of y is evaluated and complemented if necessary.

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

```
p(x) = s(0)x + s(1) x**3 + s(2)x**5 + s(3)x**7 + s(4)x**9 +
       s(5)x**11

q(x) = c(0) + c(1)x**2 + c(2)x**4 + c(3)x**6 + c(4)x**8 +
       c(5)x**10 + c(6)x**12
```

where the coefficients are:

```
s(0) =  .999 999 999 999 972
s(1) = -.166 666 666 665 404
s(2) =  .833 333 331 696 029*10**-2
s(3) = -.198 426 073 537 90*10**-3
s(4) =  .275 548 564 509 884*10**-5
s(5) = -.247 320 720 952 463*10**-7
c(0) =  .999 999 999 999 996
c(1) = -.499 999 999 999 991
c(2) =  .041 666 666 666 470 5
c(3) = -.138 888 888 888 159*10**-2
c(4) =  .248 015 784 673 257*10**-4
c(5) = -.275 552 187 277 097*10**-6
c(6) =  .206 291 063 476 645*10**-8
```

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

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

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

where $n \geq 1$.

For $n \geq 1.0$, T(n)(x) is the unique polynomial 2(n - 1.0)*x**n + ... of degree n whose maximum absolute value over the interval [-1.0,1.0] is minimal. This maximum absolute value is one.

The formulas used for the range reduction are:

```
    sin(x) = (-1)**n*sin(y)
    cos(x) = (-1)**n*cos(y)
if x = y + n*pi, n an integer

    sin(x) = cos(x - pi/2)
    cos(x) = -sin(x - pi/2)
if pi/4 ≤ x ≤ pi/2
```

Error Analysis

The maximum absolute error in the approximation of sin(x) by p(x) in the interval (-pi/4,pi/4) is .1893E-14. The maximum absolute error in the approximation of cos(x) by q(x) is .3687E-14.

Effect of Argument Error

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

DASNCS

DASNCS is an auxiliary routine that computes the inverse sine or inverse cosine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

There are no call-by-reference entry points for DASNCS. The call-by-value entry points are MLP$VDACOS and MLP$VDASIN.

The input domain is the collection of all valid double precision quantities in the interval [-1.0,1.0]. The output range at entry point MLP$VDACOS is included in the set of valid, nonnegative double precision quantities less than or equal to pi. The output range at entry point MLP$VDASIN is included in the set of valid double precision quantities in the interval [-pi/2,pi/2].

Call-By-Value Routine

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

```
arcsin(-x) = -arcsin(x)
arccos(x)  =  pi/2-arcsin(x)
arcsin(x)  =  arccos(sqrt(1.0 - x**2)), if x ≥ 0
arccos(x)  =  arcsin(sqrt(1.0 - x**2)), if x ≥ 0
```

The reduced value is called y. If $y \leq .09375$, no further reduction is performed. If not, the closest entry to y in a table of values (z,arcsin(z),sqrt(1.0 - z**2),
z = .14, .39, .52, .64) is found, and the formula used is:

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

where w = x*sqrt(1.0 - z**2) - z*sqrt(1.0 - x**2). The value of w is in (-.0792,.0848).

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

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

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

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

Error Analysis

See the description of routines DACOS and DASIN.

Effect of Argument Error

See the description of routines DACOS and DASIN.

DEULER

DEULER is an auxiliary routine that accepts calls from other math routines. It performs computations that are common among these routines.

The input and output ranges are described in routines DEXP and DTANH.

Call-By-Value Routine

Constants used in the algorithm are:

```
1.0/log(2)
log(2)  (in double precision)
d3  =    .166 666 666 666 666 666 666 666 666 709
d5  =    .833 333 333 333 333 333 333 333 331 234 953*10**-2
d7  =    .198 412 698 412 698 412 700 466 386 658*10**-3
d9  =    .275 573 192 239 858 897 408 325 908 796*10**-5
pc  =   -.474 970 880 178 988*10**-10
pa  =    .566 228 284 957 811*10**-7
pb  = 272.110 632 903 710
c11 =    .250 521 083 854 439*10**-7
```

Arithmetic operations with d subscripts are done in double precision, and operations with u subscripts are done in single precision. For example, d3 +(d) q indicates that the addition is in double precision. An operand with a u or l subscript denotes the first or second word, respectively, of the double precision pair of words containing the operand.

The algorithm used is:

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

b. q = (y)(u)*(u)(y)(u)

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

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

e. Compute hm = sqrt(1.0 + s**2).
   hi = 3*q + ((s)(u))**2 computed in single precision.
   hj = hi + hi
   hk = 2*(1.0 + hj)
   hl = ((y)(u)*(u)(y)(u) - hj)/hk - hi
   hm = hj + (u)(hk - (u)hl)*(u)(hl/hk)
        (hm now carries cosh-1.0 in single precision.)

f. DS = s + (d)(((y)(1) + (r)(y)(1)*(u)hm) + (r)((s)(1) +
(r)((y)(u)* (1)(p)(u) + (r)(y)(u)*(r)(p)(1))))
(DS now contains sinh(y) in double precision.)

g. DC = hm + (d)(DS*DS - 2*hm - hm*hm)/(2(1.0 + hm)) computed in double precision.

h. DX = DS + DC

i. Clean up DS, DC, DX with (X7) = n.
Register pair XA-XB = DS = sinh(y).
Register pair X8-X9 = DC = cosh(y) - 1.0.
Register pair X4-X5 = DX = exp(y).

j. Return.

Error Analysis

See the descriptions in routines DEXP and DTANH.

Effect of Argument Error

See the descriptions in routines DEXP and DTANH.

DSNCOS

DSNCOS is an auxiliary routine that computes the trigonometric sine or trigonometric cosine function. It accepts a double precision argument and returns a double precision result. This function cannot be called from a CYBIL program.

There are no call-by-reference entry points for DSNCOS. The call-by-value entry points are MLP$VDCOS and MLP$VDSIN.

The input domain for this routine is the collection of all valid double precision quantities whose absolute value is less than 2**47. The output range is included in the set of valid double precision quantities in the interval [-1.0,1.0].

Call-By-Value Routine

Upon entry, the argument x is made positive and is multiplied by 2/pi in double precision, and the nearest integer n to x*2/pi is computed. At this stage, x*2/pi is checked to see that it does not exceed 2**47. If it does, a diagnostic message is returned. Otherwise, y = x - n*pi/2 is computed in double precision as the reduced argument, and y is in the interval [-pi/4,pi/4]. The value of mod(n,4), the entry point called, and the original sign of x determine whether a sine polynomial approximation p(x) or a cosine polynomial approximation q(x) is to be used. A flag is set to indicate the sign of the final result.

For x in the interval [-pi/4,pi/4], the sine polynomial approximation is:

```
p(x) = a(1)x + a(3)x**3 + a(5)x**5 + a(7)x**7 + a(9)x**9 +
       a(11)x**11 + a(13)x**13** + a(15)x**15 + a(17)x**17 +
       a(19)x**19 + a(21)x**21
```

and the cosine polynomial approximation is:

```
q(x) = b(0) + b(2)x**2 + b(4)x**4 + b(6)x**6 + b(8)x**8 +
       b(10)x**10 + b(12)x**12 + b(14)x**14 + b(16)x**16 +
       b(18)x**18 + b(20)x**20
```

The coefficients are:

```
a(1)  =  .999 999 999 999 999 999 999 999 999 99
a(3)  = -.166 666 666 666 666 666 666 666 666 52
a(5)  =  .833 333 333 333 333 333 333 332 709 57*10**-2
a(7)  = -.198 412 698 412 698 412 698 291 344 78*10**-3
a(9)  =  .275 573 192 239 858 906 394 406 844 01*10**-5
a(11) = -.250 521 083 854 417 101 138 076 473 5*10**-7
a(13) =  .160 590 438 368 179 417 271 194 064 61*10**-9
a(15) = -.764 716 373 079 886 084 755 348 748 91*10**-12
a(17) =  .281 145 706 930 018*10**-14
a(19) = -.822 042 461 317 923*10**-17
a(21) =  .194 362 013 130 224*10**-19
b(0)  =  .999 999 999 999 999 999 999 999 999 99
b(2)  = -.499 999 999 999 999 999 999 999 999 19
b(4)  =  .416 666 666 666 666 666 666 666 139 02
b(6)  = -.138 888 888 888 888 888 888 755 436 28*10**-2
b(8)  =  .248 015 873 015 873 015 699 922 737 30*10**-4
b(10) = -.275 573 192 239 858 775 558 669 957 11*10**-6
b(12) =  .208 767 569 878 619 214 898 747 461 35*10**-8
b(14) = -.114 707 455 958 584 315 495 950 765 75*10**-10
b(16) =  .477 947 696 822 393 115 933 106 267 21*10**-13
b(18) = -.156 187 668 345 316*10**-15
b(20) =  .408 023 947 777 860*10**-18
```

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

Error Analysis

See the description of routines DCOS and DSIN.

Effect of Argument Error

See the description of routines DCOS and DSIN.

HYPERB

HYPERB is an auxiliary routine that accepts calls from other math routines that require the simultaneous hyperbolic sine and hyperbolic cosine of the same argument. HYPERB accepts a real argument and returns two real results.

The entry points and input and output ranges for this routine are described in routines CSIN and CCOS.

Call-By-Value Routine

Upon entry, the routine computes e**x = exp(x), where x is the angle passed to HYPERB. The hyperbolic cosine is computed by:

```
cosh(x) = 0.5*(exp(x) + exp(-x))
```

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

```
sinh(x) = 0.5*(exp(x) - exp(-x))
```

For $|x| < 0.22$, the MacLaurin series for sinh is truncated after the term x**9/9! and the resulting polynomial is taken as the approximation:

```
sinh(x) = x + x**3/3! + x**5/5! + x**7/7! + x**9/9!
```

Error Analysis

See the description of routine COSH and SINH.

Effect of Argument Error

See the description of routine COSH and SINH.

SINCOS

SINCOS is an auxiliary routine that computes the trigonometric sine and cosine functions. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for SINCOS. The call-by-value entry points are MLP$VCOS and MLP$VSIN.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Value Routine

If x is valid, then COS(x) or SIN(x) is calculated by using the periodic properties of the cosine and sine functions to reduce the task to finding a cosine or sine of an equivalent angle y within [-pi/4, pi/4] as follows:

```
If N + K is even
then
     Z = sin(y)
else
     Z = cos(y)
If MOD(N + K, 4) is 0 or 1 (that is, the second last bit of
N + K is even)
then
     S = 0
else
     S = mask(1)
```

where K is 0, 1, or 2 according to whether the SIN of a positive angle, the COS of any angle, or the SIN of a negative angle is to be calculated. N is the nearest integer to 2/pi*x, and y is the nearest single precision floating point number to x - n*pi/2. The argument x is the absolute value of the angle. The desired SIN or COS is xor(S, Z).

Once the angle has been reduced to the range [-pi/4, pi/4], the following approximations are used to calculate either the cosine or the sine of the angle, providing 48 bits of precision.

If the cosine or the angle is required, the approximation used is

```
cosine(y) = 1 - y*y*P(y*y)
```

where y is the angle and P(w) is the quintic polynomial:

```
P(w) = P0 + P1*w  + P2*w**2 +P3 + w**3 + P4*w**4 + P5*w**5
```

such that P(y*y) is a min-max polynomial approximation to the function (1 - cos(y))/y**2.

The coefficients are:

```
P5 = -2.07006230562462 9462E-9
P4 =  2.75563699740658 8778E-7
P3 = -2.48015852120642 6671E-5
P2 =  1.38888888872786 6775E-3
P1 = -4.16666666666646 8116E-2
P0 =  5.00000000000000 0000E-1
```

If the sine of the angle is required, the approximation used is

```
sine(y) = y - y*y*y*Q(y*y)
```

where y is the angle and Q(w) is the quintic polynomial:

```
Q(w) = Q0 + Q1*w  + Q2*w**2 +Q3*w**3 + Q4*w**4 + Q5*w**5
```

such that Q(y*y) is a min-max polynomial approximation to the function (y - sin(y))/y**3.

The coefficients are:

```
Q5 = -1.59181425703300 5283E-10
Q4 =  2.50511320497376 7698E-8
Q3 = -2.75573161036575 4733E-6
Q2 =  1.98412698367610 0911E-4
Q1 = -8.33333333330950 363E-3
Q0 =  1.66666666666666 463E-1
```

Error Analysis

The function SINCOS was tested against 4*COS(x/3)**3 - 3*COS(x/3). Groups of 2,000 arguments were chosen randomly from the interval [.2199E+02,.2356E+02]. Statistics on relative error were observed: maximum relative error was .1404E-13, and root mean square relative error was .3245E-14.

Effect of Argument Error

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

SINCSD

SINCSD is a function that computes the sine and cosine functions for arguments in degrees. It accepts a real argument and returns a real result.

There are no call-by-reference entry points for SINCSD. The call-by-value entry points are MLP$VCOSD and MLP$VSIND.

The input domain for this routine is the collection of all valid real quantities whose absolute value is less than 2**47. The output range is included in the set of valid real quantities in the interval [-1.0,1.0].

Call-By-Value Routine

The result is put in the interval [-45,45] by finding the nearest integer, n, to x/90, and subtracting n*90 from the argument. The reduced argument is then multiplied by pi/180. The appropriate sign is copied to the value of the appropriate function, sine or cosine, as determined by these identities:

```
sin(x ± 360 degrees) =  sin(x)
sin(x ± 180 degrees) = -sin(x)
sin(x ± 90 degrees)  =  cos(x)
sin(x - 90 degrees)  = -cos(x)
cos(x ± 360 degrees) =  cos(x)
cos(x ± 180 degrees) = -cos(x)
cos(x ± 90 degrees)  = -sin(x)
cos(x - 90 degrees)  =  sin(x)
```

Error Analysis

The reduction to (-45,+45) is exact; the constant pi/180 has relative error 1.37E-15, and multiplication by this constant has a relative error 5.33E-15, and a total error of 6.7E-15. Since errors in the argument of SIN and COS contribute only pi/4 of their value to the result, the error due to the reduction and conversion is, at most, 5.26E-15 plus the maximum error in SINCOS over (-pi/4,+pi/4).

A group of 10,000 arguments was chosen at random from the interval [0,360]. The maximum relative error of these arguments was found to be .7105E-14 for COSD and .1403E-13 for SIND.

Effect of Argument Error

Errors in the argument x are amplified by x/tan(x) for SIND and x*tan(x) for COSD. These functions have a maximum value of pi/4 in the interval (-45,+45) but have poles at even (SIND) or odd (COSD) multiples of 90 degrees, and are large between multiples of 90 degrees if x is large.

Appendixes

Glossary A

A

Argument

A variable or constant that is passed to a routine and used by that routine to compute a function. The actual value of the variable is passed when a routine is called by value; the address of the variable is passed when the routine is called by name.

Argument Set

An ordered list of one or more arguments.

Auxiliary Routine

A math routine which is not directly called from program code, but assists in the computation of a math library function.

C

Call-by-Reference

A method of referencing a subprogram in which the addresses of the arguments are passed.

Call-by-Value

A method of referencing a subprogram in which the values of the arguments are passed.

D

Dummy Argument

A variable or constant that is passed to a routine, but is not used by the routine to compute a function.

E

Entry Point

A statement within a math routine at which execution can begin. There may be more than one entry point into a math routine.

Error

The computed value of a function minus the true value.

Exponentiation Routine

A math routine which accepts compiler-generated calls from a source program to perform exponentiation. These calls are generated when a program statement involves exponentiation of certain number types. Exponentiation routines are not called directly using their function names.

External Routine

A predefined subprogram that accepts calls from program code to compute certain mathematical functions.

F

Function Name

A symbolic name that appears in a program and causes a math routine to be executed.

I

Indefinite Value

A value that results from a mathematical operation that cannot be resolved, such as a division where the dividend and divisor are both zero.

Infinite Value

A value that results from a computation whose result exceeds the capacity of the computer.

Input Range

A collection of argument sets for which a given math routine will return a valid result.

N

Number Types

A classification of the numbers processed by the math routines. The math routines perform computations on four number types: integer, single precision floating point, double precision floating point, and complex floating point.

O

Output Range

The collection of results obtained by using the arguments in the input domain of each math routine for computation of the function or routine.

R

Relative Error

The error of a function divided by the true value. The maximum relative error approximates the worst-case behavior of the function in the given interval.

Root Mean Square Relative Error

The square root of the sum of the squares of the relative errors of all the arguments, divided by the number of arguments.

Routine

A computer subprogram, that computes commonly occurring math functions, and performs other tasks such as input and output.

S

Scalar

A constant, variable, array element, or substring of any type.

V

Vector

One-dimensional array of up to 512 elements.

Related Manuals B

Table B-1 lists all manuals that are referenced in this manual or that contain background information. A complete list of NOS/VE manuals is given in the NOS/VE System Usage manual. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
explain
```

Ordering Printed Manuals

You can order Control Data manuals through Control Data sales offices or through:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

Accessing Online Manuals

To access an online manual, log in to NOS/VE and specify the online manual title (listed in table B-1) on the EXPLAIN command. For example, to read the FORTRAN online manual, enter:

```
explain manual=fortran
```

Table B-1. Related Manuals

Manual Title	Publication Number	Online Title
Ada for NOS/VE Usage	60498113	
APL for NOS/VE Usage	60485813	
BASIC for NOS/VE Usage	60486313	BASIC
CYBIL Language Definition Usage	60464113	
FORTRAN Version 1 Language Definition Usage	60485913	
FORTRAN Version 1 Quick Reference	L60485918	FORTRAN
FORTRAN Version 2 Language Definition Usage	60487113	
FORTRAN Version 2 Quick Reference	L60487118	VFORTRAN
NOS/VE Diagnostic Messages	60484613	MESSAGES
NOS/VE System Usage	60464014	
Pascal for NOS/VE Usage	60485613	PASCAL

Error Handling C

Under call-by-reference, the Math Library will generate the special software condition MATH_LIBRARY_ERROR. The system product under which you are executing ordinarily handles the processing of this condition. If no condition handler for MATH_LIBRARY_ERROR has been established, the operating system handles the processing of this condition.

You can also write your own condition handler. The following information defines the interface between the Math Library and the operating system, whether or not a condition handler has been established. For detailed information on the procedures used in establishing a user-defined condition handler, see the NOS/VE Program Interface manual.

When an error occurs in a math library function under a call-by-reference routine, the following events occur:

1. An appropriate abnormal status is set into global variable MLV$STATUS (of type OST$STATUS).

2. The appropriate default error value is placed in the result register(s) XE and/or XF. Register A4 contains the pointer to the parameter list passed to the call-by-reference routine. Register XD contains the number of parameters for the call-by-reference routine. For example, register XD will contain a 1 for the call-by-reference routine MLP$RSIN, and a 2 for MLP$RZTOZ. The User Condition Register is cleared of all arithmetic errors.

3. Ungated routine MLP$ERROR_PROCESSOR is called with all registers saved in the save area so that they can be accessed by a condition handler.

4. MLP$ERROR_PROCESSOR calls the PMP$CAUSE_CONDITION procedure with user condition MATH_LIBRARY_ERROR and a pointer to the previous save area (the registers saved by the call-by-reference routine) as the condition descriptor.

5. Upon return from the PMP$CAUSE_CONDITION procedure, MLP$ERROR_PROCESSOR is exited if the returned status is normal. If the return status is not normal, the PMP$ABORT procedure is called with one of two statuses. If there is no established condition handler for MATH_LIBRARY_ERROR, status MLV$STATUS is used. If there is an established condition handler for MATH_LIBRARY_ERROR, the status returned from the PMP$CAUSE_CONDITION procedure is used.

6. The call-by-reference routine immediately returns if it is returned to from MLP$ERROR_PROCESSOR.

Index

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

V

X

Z

Math Library for NOS/VE Usage 60486513 G

We would like your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who Are You?

___ Manager
___ Systems Analyst or Programmer
___ Applications Programmer
___ Operator
___ Other _____

How Do You Use This Manual?

___ As an Overview
___ To learn the Product/System
___ For Comprehensive Reference
___ For Quick Look-up

Do You Also Have?

___ FORTRAN for NOS/VE Usage
___ CYBIL for NOS/VE Usage

What programming languages do you use? _____

Which are helpful to you? ___ Parameter Summary (inside cover) ___ Related Manuals Page
___ Character Set ___ Other _____

How Do You Like This Manual? Check those that apply.

Yes	Somewhat	No	
___	___	___	Is the manual easy to read (print size, page layout, and so on)?
___	___	___	Is it easy to understand?
___	___	___	Is the order of topics logical?
___	___	___	Are there enough examples?
___	___	___	Are the examples helpful? (___ Too simple ___ Too complex)
___	___	___	Is the technical information accurate?
___	___	___	Can you easily find what you want?
___	___	___	Do the illustrations help you?
___	___	___	Does the manual tell you what you need to know about the topic?

Comments? If applicable, note page number and paragraph.

Would you like a reply? ___ Yes ___ No

Continue on other side

From:

Name _____ Company _____

Address _____ Date _____

_____ Phone No. _____

_____

Please send program listing and output if applicable to your comment.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Technology and Publications Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD FOLD

Comments (continued from other side)

CONTROL DATA