



IMSL[®] Fortran Math Special Functions Library

Version 2022.1

PERFORCE

www.perforce.com



Copyright 1970-2022 Rogue Wave Software, Inc., a Perforce company.

Visual Numerics, IMSL, and PV-WAVE are registered trademarks of Rogue Wave Software, Inc., a Perforce company.

IMPORTANT NOTICE: Information contained in this documentation is subject to change without notice. Use of this document is subject to the terms and conditions of a Rogue Wave Software License Agreement, including, without limitation, the Limited Warranty and Limitation of Liability.

ACKNOWLEDGMENTS

Use of the Documentation and implementation of any of its processes or techniques are the sole responsibility of the client, and Perforce Software, Inc., assumes no responsibility and will not be liable for any errors, omissions, damage, or loss that might result from any use or misuse of the Documentation

PERFORCE SOFTWARE, INC. MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THE DOCUMENTATION. THE DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. PERFORCE SOFTWARE, INC. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS WITH REGARD TO THE DOCUMENTATION, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT SHALL PERFORCE SOFTWARE, INC. BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT, PUNITIVE, OR EXEMPLARY DAMAGES IN CONNECTION WITH THE USE OF THE DOCUMENTATION.

The Documentation is subject to change at any time without notice.

IMSL
<https://www.imsl.com/>

Contents

Introduction

The IMSL Fortran Numerical Libraries	1
Getting Started	2
Finding the Right Routine	3
Organization of the Documentation	4
Naming Conventions	6
Using Library Subprograms	8
Programming Conventions	9
Module Usage	10
Programming Tips	11
Optional Subprogram Arguments	12
Error Handling	13
Printing Results	14

Elementary Functions

Routines	15
Usage Notes	16
CARG	17
CBRT	19
EXPRL	21
LOG10	23
ALNREL	25

Trigonometric and Hyperbolic Functions

Routines	28
Usage Notes	29
TAN	30
COT	32
SINDG	35
COSDG	37

ASIN	39
ACOS	41
ATAN	43
ATAN2	45
SINH	47
COSH	49
TANH	51
ASINH	53
ACOSH	55
ATANH	57

Exponential Integrals and Related Functions

Routines.....	60
Usage Notes	61
EI	62
E1	64
ENE	66
ALI	68
SI	71
CI	73
CIN	75
SHI	77
CHI	79
CINH	81

Gamma Function and Related Functions

Routines.....	83
Usage Notes	84
FAC	85
BINOM	87
GAMMA	89
GAMR	92
ALNGAM	94
ALGAMS	97
GAMI	99

GAMIC	102
GAMIT.....	104
PSI.....	106
PSI1.....	109
POCH.....	111
POCH1.....	113
BETA.....	115
ALBETA.....	118
BETAI.....	121

Error Function and Related Functions

Routines.....	123
Usage Notes	124
ERF	125
ERFC.....	127
ERFCE.....	130
CERFE.....	132
ERFI.....	134
ERFCI.....	137
DAWS.....	140
FRESC.....	142
FRESS.....	144

Bessel Functions

Routines.....	146
Usage Notes	147
BSJ0.....	148
BSJ1.....	150
BSY0.....	152
BSY1.....	154
BSI0.....	156
BSI1.....	158
BSK0.....	160
BSK1.....	163
BSIOE.....	165

BSI1E	167
BSK0E	169
BSK1E	171
BSJNS	173
BSINS	176
BSJS	179
BSYS	181
BSIS	183
BSIES	185
BSKS	187
BSKES	189
CBJS	191
CBYS	194
CBIS	197
CBKS	199

Kelvin Functions

Routines	201
Usage Notes	202
BER0	205
BEI0	207
AKER0	209
AKEI0	211
BERP0	213
BEIP0	215
AKERP0	217
AKEIP0	219
BER1	221
BEI1	223
AKER1	225
AKEI1	227

Airy Functions

Routines	229
AI	230

BI	232
AID	234
BID	236
AIE	238
BIE	240
AIDE	242
BIDE	244
CAI	246
CBI	248
CAID	250
CBID	252

Elliptic Integrals

Routines	254
Usage Notes	255
ELK	258
ELE	260
ELRF	262
ELRD	264
ELRJ	266
ELRC	268

Elliptic and Related Functions

Routines	270
Usage Notes	271
CWPL	272
CWPLD	274
CWPQ	276
CWPQD	278
EJSN	280
EJCN	283
EJDN	286

Probability Distribution Functions and Inverses

Routines	289
----------	-----

Usage Notes	291
BINDF	296
BINPR	298
GEODF	301
GEOIN	303
GEOPR	305
HYPDF	307
HYPPR	310
POIDF	313
POIPR	315
UNDDF	318
UNDIN	320
UNDPR	322
AKS1DF	324
AKS2DF	327
ALNDF	330
ALNIN	332
ALNPR	334
ANORDF	336
ANORIN	339
ANORPR	341
BETDF	343
BETIN	346
BETPR	348
BETNDF	350
BETNIN	353
BETNPR	356
BNRDF	359
CHIDF	361
CHIIN	364
CHIPR	366
CSNDF	368
CSNIN	371

CSNPR	373
EXPDF	376
EXPIN	378
EXPPR	380
EXVDF	382
EXVIN	384
EXVPR	386
FDF	388
FIN	391
FPR	393
FNDF	395
FNIN	398
FNPR	401
GAMDF	404
GAMIN	407
GAMPR	409
RALDF	411
RALIN	413
RALPR	415
TDF	417
TIN	420
TPR	422
TNDF	424
TNIN	427
TNPR	429
UNDF	431
UNIN	433
UNPR	435
WBLDF	437
WBLIN	439
WBLPR	441
GCDF	443
GCIN	447

GFNIN	451
Mathieu Functions	
Routines	454
Usage Notes	455
MATEE	456
MATCE	459
MATSE	463
Miscellaneous Functions	
Routines	467
Usage Notes	468
SPENC	470
INITS	472
CSEVL	474
Reference Material	
Routines/Topics	476
User Errors	477
ERSET	480
IERCD and N1RTY	481
Machine-Dependent Constants	482
IMACH	483
AMACH	485
DMACH	487
IFNAN(X)	488
UMACH	490
Reserved Names	492
Deprecated Features and Deleted Routines	493
Appendix A, Alphabetical Summary of Routines	496
A	496
B	497
C	499
D	500

E	501
F.....	502
G.....	502
H.....	503
I.....	503
L.....	504
M.....	504
N.....	504
P.....	504
R.....	505
S.....	505
T.....	505
U.....	506
W.....	506

Appendix B, References	507
-------------------------------------	-----

Index

Introduction

The IMSL Fortran Numerical Libraries

The IMSL Libraries consist of two separate, but coordinated Libraries that allow easy user access. These Libraries are organized as follows:

- MATH LIBRARY general applied mathematics and special functions
The User's Guide for IMSL MATH LIBRARY has two parts:
 - MATH LIBRARY
 - MATH LIBRARY Special Functions
- STAT/LIBRARY statistics

Most of the routines are available in both single and double precision versions. Many routines are also available for complex and complex-double precision arithmetic. The same user interface is found on the many hardware versions that span the range from personal computer to supercomputer. Note that some IMSL routines are not distributed for FORTRAN compiler environments that do not support double precision complex data. The specific names of the IMSL routines that return or accept the type double complex begin with the letter "Z" and, occasionally, "DC."

Getting Started

IMSL MATH LIBRARY Special Functions is a collection of FORTRAN subroutines and functions useful in research and statistical analysis. Each routine is designed and documented to be used in research activities as well as by technical specialists.

To use any of these routines, you must write a program in FORTRAN (or possibly some other language) to call the MATH LIBRARY Special Functions routine. Each routine conforms to established conventions in programming and documentation. We give first priority in development to efficient algorithms, clear documentation, and accurate results. The uniform design of the routines makes it easy to use more than one routine in a given application. Also, you will find that the design consistency enables you to apply your experience with one MATH LIBRARY Special Functions routine to all other IMSL routines that you use.

Finding the Right Routine

The organization of IMSL MATH LIBRARY Special Functions closely parallels that of the National Bureau of Standards' *Handbook of Mathematical Functions*, edited by Abramowitz and Stegun (1964). Corresponding to the NBS Handbook, functions are arranged into separate chapters, such as elementary functions, trigonometric and hyperbolic functions, exponential integrals, gamma function and related functions, and Bessel functions. To locate the right routine for a given problem, you may use either the table of contents located in each chapter introduction, or one of the indexes at the end of this manual.

Organization of the Documentation

This manual contains a concise description of each routine, with at least one demonstrated example of each routine, including sample input and results. You will find all information pertaining to the Special Functions Library in this manual. Moreover, all information pertaining to a particular routine is in one place within a chapter.

Each chapter begins with an introduction followed by a table of contents that lists the routines included in the chapter. Documentation of the routines consists of the following information:

- IMSL Routine's Generic Name
- Purpose: a statement of the purpose of the routine. If the routine is a function rather than a subroutine the purpose statement will reflect this fact.
- Function Return Value: a description of the return value (for functions only).
- Required Arguments: a description of the required arguments in the order of their occurrence. Input arguments usually occur first, followed by input/output arguments, with output arguments described last. Further, the following terms apply to arguments:

Input Argument must be initialized; it is not changed by the routine.

Input/Output Argument must be initialized; the routine returns output through this argument; cannot be a constant or an expression.

Input or Output Select appropriate option to define the argument as either input or output. See individual routines for further instructions.

Output No initialization is necessary; cannot be a constant or an expression. The routine returns output through this argument.

- Optional Arguments: a description of the optional arguments in the order of their occurrence.
- Fortran 90 Interface: a section that describes the generic and specific interfaces to the routine.
- Fortran 77 Style Interface: an optional section, which describes Fortran 77 style interfaces, is supplied for backwards compatibility with previous versions of the Library.
- ScaLAPACK Interface: an optional section, which describes an interface to a ScaLAPACK based version of this routine.
- Description: a description of the algorithm and references to detailed information. In many cases, other IMSL routines with similar or complementary functions are noted.
- Comments: details pertaining to code usage.
- Programming notes: an optional section that contains programming details not covered elsewhere.

- Example: at least one application of this routine showing input and required dimension and type statements.
- Output: results from the example(s). **Note** that unique solutions may differ from platform to platform.
- Additional Examples: an optional section with additional applications of this routine showing input and required dimension and type statements.

Naming Conventions

The names of the routines are mnemonic and unique. Most routines are available in both a single precision and a double precision version, with names of the two versions sharing a common root. The root name is also the generic interface name. The name of the double precision specific version begins with a "D". The single precision specific version begins with an "S_". For example, the following pairs are precision specific names of routines in the two different precisions: **S_GAMDF/D_GAMDF** (the root is "GAMDF," for "Gamma distribution function") and **S_POIDF/D_POIDF** (the root is "POIDF," for "Poisson distribution function"). The precision specific names of the IMSL routines that return or accept the type complex data begin with the letter "C_" or "Z_" for complex or double complex, respectively. Of course the generic name can be used as an entry point for all precisions supported.

When this convention is not followed the generic and specific interfaces are noted in the documentation. For example, in the case of the BLAS and trigonometric intrinsic functions where standard names are already established, the standard names are used as the precision specific names. There may also be other interfaces supplied to the routine to provide for backwards compatibility with previous versions of the Library. These alternate interfaces are noted in the documentation when they are available.

Except when expressly stated otherwise, the names of the variables in the argument lists follow the FORTRAN default type for integer and floating point. In other words, a variable whose name begins with one of the letters "I" through "N" is of type **INTEGER**, and otherwise is of type **REAL** or **DOUBLE PRECISION**, depending on the precision of the routine.

An assumed-size array with more than one dimension that is used as a FORTRAN argument can have an assumed-size declarator for the last dimension only. In the MATH LIBRARY Special Functions routines, the information about the first dimension is passed by a variable with the prefix "LD" and with the array name as the root. For example, the argument **LDA** contains the leading dimension of array *A*. In most cases, information about the dimensions of arrays is obtained from the array through the use of Fortran 90's *size* function. Therefore, arguments carrying this type of information are usually defined as optional arguments.

Where appropriate, the same variable name is used consistently throughout a chapter in the MATH LIBRARY Special Functions. For example, in the routines for random number generation, **NR** denotes the number of random numbers to be generated, and **R** or **IR** denotes the array that stores the numbers.

When writing programs accessing the MATH LIBRARY Special Functions, the user should choose FORTRAN names that do not conflict with names of IMSL subroutines, functions, or named common blocks. The careful user can avoid any conflicts with IMSL names if, in choosing names, the following rules are observed:

- Do not choose a name that appears in the Alphabetical Summary of Routines, at the end of the *User's Manual*, nor one of these names preceded by a **D**, **S_**, **D_**, **C_**, or **Z_**.

- Do not choose a name consisting of more than three characters with a numeral in the second or third position.

For further details, see the section on [Reserved Names](#) in the Reference Material.

Using Library Subprograms

The documentation for the routines uses the generic name and omits the prefix, and hence the entire suite of routines for that subject is documented under the generic name.

Examples that appear in the documentation also use the generic name. To further illustrate this principle, note the `BSJNS` documentation (see [Chapter 6, “Bessel Functions”](#), of this manual). A description is provided for just one data type. There are four documented routines in this subject area: `S_BSJNS`, `D_BSJNS`, `C_BSJNS`, and `Z_BSJNS`.

These routines constitute single-precision, double-precision, complex, and complex double-precision versions of the code.

The appropriate routine is identified by the Fortran 90 compiler. Use of a module is required with the routines. The naming convention for modules joins the suffix “`_int`” to the generic routine name. Thus, the line “use `BSJNS_INT`” is inserted near the top of any routine that calls the subprogram “`BSJNS`”. More inclusive modules are also available. For example, the module named `imsl_libraries` contains the interface modules for all routines in the library.

When dealing with a complex matrix, all references to the *transpose* of a matrix, A^T are replaced by the *adjoint* matrix

$$\overline{A}^T \equiv A^* = A^H$$

where the overstrike denotes complex conjugation. IMSL Fortran Numerical Library linear algebra software uses this convention to conserve the utility of generic documentation for that code subject. All references to *orthogonal* matrices are to be replaced by their complex counterparts, *unitary* matrices. Thus, an $n \times n$ orthogonal matrix Q satisfies the condition $Q^T Q = I_n$. An $n \times n$ unitary matrix V satisfies the analogous condition for complex matrices, $V^* V = I_n$.

Programming Conventions

In general, the IMSL MATH LIBRARY Special Functions codes are written so that computations are not affected by underflow, provided the system (hardware or software) places a zero value in the register. In this case, system error messages indicating underflow should be ignored.

IMSL codes also are written to avoid overflow. A program that produces system error messages indicating overflow should be examined for programming errors such as incorrect input data, mismatch of argument types, or improper dimensioning.

In many cases, the documentation for a routine points out common pitfalls that can lead to failure of the algorithm.

Library routines detect error conditions, classify them as to severity, and treat them accordingly. This error-handling capability provides automatic protection for the user without requiring the user to make any specific provisions for the treatment of error conditions. See the section on [User Errors](#) in the Reference Material for further details.

Module Usage

Users are required to incorporate a “use” statement near the top of their program for the IMSL routine being called when writing new code that uses this library. However, legacy code which calls routines in the previous version of the library without the presence of a “use” statement will continue to work as before. The example programs throughout this manual demonstrate the syntax for including use statements in your program. In addition to the examples programs, common cases of when and how to employ a use statement are described below.

- Users writing new programs calling the generic interface to IMSL routines must include a use statement near the top of any routine that calls the IMSL routines. The naming convention for modules joins the suffix “_int” to the generic routine name. For example, if a new program is written calling the IMSL routines `LFTRG` and `LFSRG`, then the following use statements should be inserted near the top of the program:

```
USE LFTRG_INT
USE LFSRG_INT
```

In addition to providing interface modules for each routine individually, we also provide a module named “`imsl_libraries`”, which contains the generic interfaces for all routines in the library. For programs that call several different IMSL routines using generic interfaces, it can be simpler to insert the line

```
USE IMSL_LIBRARIES
```

rather than list use statements for every IMSL subroutine called.

- Users wishing to update existing programs to call other routines from this library should incorporate a use statement for the new routine being called. (Here, the term “new routine” implies any routine in the library, only “new” to the user’s program.) For example, if a call to the generic interface for the routine `LSARG` is added to an existing program, then

```
USE LSARG_INT
```

should be inserted near the top of your program.

- Users wishing to update existing programs to call the new generic versions of the routines must change their calls to the existing routines to match the new calling sequences and use either the routine specific interface modules or the all encompassing “`imsl_libraries`” module.
- Code which employed the “`use numerical_libraries`” statement from the previous version of the library will continue to work properly with this version of the library.

Programming Tips

It is strongly suggested that users force all program variables to be explicitly typed. This is done by including the line `"IMPLICIT NONE"` as close to the first line as possible. Study some of the examples accompanying an IMSL Fortran Numerical Library routine early on. These examples are available online as part of the product.

Each subject routine called or otherwise referenced requires the `"use"` statement for an interface block designed for that subject routine. The contents of this interface block are the interfaces to the separate routines available for that subject. Packaged descriptive names for option numbers that modify documented optional data or internal parameters might also be provided in the interface block. Although this seems like an additional complication, many typographical errors are avoided at an early stage in development through the use of these interface blocks. The `"use"` statement is required for each routine called in the user's program.

However, if one is only using the Fortran 77 interfaces supplied for backwards compatibility then the `"use"` statements are not required.

Optional Subprogram Arguments

IMSL Fortran Numerical Library routines have *required* arguments and may have *optional* arguments. All arguments are documented for each routine. For example, consider the routine **GCIN** which evaluates the inverse of a general continuous **CDF**. The required arguments are **P**, **X**, and **F**. The optional arguments are **IOPT** and **M**. Both **IOPT** and **M** take on default values so are not required as input by the user unless the user wishes for these arguments to take on some value other than the default. Often there are other output arguments that are listed as optional because although they may contain information that is closely connected with the computation they are not as compelling as the primary problem. In our example code, **GCIN**, if the user wishes to input the optional argument “**IOPT**” then the use of the keyword “**IOPT=**” in the argument list to assign an input value to **IOPT** would be necessary.

For compatibility with previous versions of the IMSL Libraries, the **NUMERICAL_LIBRARIES** interface module includes backwards compatible positional argument interfaces to all routines which existed in the Fortran 77 version of the Library. Note that it is not necessary to use “*use*” statements when calling these routines by themselves. Existing programs which called these routines will continue to work in the same manner as before.

Error Handling

The routines in IMSL MATH LIBRARY Special Functions attempt to detect and report errors and invalid input. Errors are classified and are assigned a code number. By default, errors of moderate or worse severity result in messages being automatically printed by the routine. Moreover, errors of worse severity cause program execution to stop. The severity level as well as the general nature of the error is designated by an “error type” with numbers from 0 to 5. An error type 0 is no error; types 1 through 5 are progressively more severe. In most cases, you need not be concerned with our method of handling errors. For those interested, a complete description of the error-handling system is given in the [Reference Material](#), which also describes how you can change the default actions and access the error code numbers.

Printing Results

None of the routines in IMSL MATH LIBRARY Special Functions print results (but error messages may be printed). The output is returned in FORTRAN variables, and you can print these yourself.

The IMSL routine [UMACH](#) (see the Reference Material section of this manual) retrieves the FORTRAN device unit number for printing. Because this routine obtains device unit numbers, it can be used to redirect the input or output. The section on [Machine-Dependent Constants](#) in the Reference Material contains a description of the routine `UMACH`.

Elementary Functions

Routines

Evaluates the argument of a complex number	CARG	17
Evaluates the cube root of a real or complex number	CBRT	19
Evaluates $(e^x - 1)/x$ for real or complex x	EXPRL	21
Evaluates the complex base 10 logarithm, $\log_{10}z$	LOG10	23
Evaluates $\ln(x + 1)$ for real or complex x	ALNREL	25

Usage Notes

The “relative” function `EXPRL` is useful for accurately computing $e^x - 1$ near $x = 0$. Computing $e^x - 1$ using `EXP(X) - 1` near $x = 0$ is subject to large cancellation errors.

Similarly, `ALNREL` can be used to accurately compute $\ln(x + 1)$ near $x = 0$. Using the routine `ALOG` to compute $\ln(x + 1)$ near $x = 0$ is subject to large cancellation errors in the computation of $1 + x$.

CARG

This function evaluates the argument of a complex number.

Function Return Value

CARG — Function value. (Output)

If $z = x + jy$, then $\arctan(y/x)$ is returned except when both x and y are zero. In this case, zero is returned.

Required Arguments

Z — Complex number for which the argument is to be evaluated. (Input)

FORTRAN 90 Interface

Generic: CARG (Z)

Specific: The specific interface names are S_CARG and D_CARG.

FORTRAN 77 Interface

Single: CARG (Z)

Double: The double precision function name is ZARG.

Description

$\text{Arg}(z)$ is the angle θ in the polar representation $z = |z|e^{j\theta}$, where $i = \sqrt{-1}$.

If $z = x + jy$, then $\theta = \tan^{-1}(y/x)$ except when both x and y are zero. In this case, θ is defined to be zero

Example

In this example, $\text{Arg}(1 + j)$ is computed and printed.

```
USE CARG_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  REAL         VALUE
  COMPLEX      Z
!
  Z           = (1.0, 1.0)
  VALUE      = CARG(Z)
!                               Compute
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CARG(', F6.3, ', ', F6.3, ') = ', F6.3)
  END                               Print the results
```

Output

```
CARG( 1.000, 1.000) = 0.785
```

CBRT

This function evaluates the cube root.

Function Return Value

CBRT — Function value. (Output)

Required Arguments

X — Argument for which the cube root is desired. (Input)

FORTRAN 90 Interface

Generic: **CBRT (X)**

Specific: The specific interface names are **S_CBRT**, **D_CBRT**, **C_CBRT**, and **Z_CBRT**.

FORTRAN 77 Interface

Single: **CBRT (X)**

Double: The double precision name is **DCBRT**.

Complex: The complex precision name is **CCBRT**.

Double Complex: The double complex precision name is **ZCBRT**.

Description

The function **CBRT(X)** evaluates $x^{1/3}$. All arguments are legal. For complex argument, x , the value of $|x|$ must not overflow.

Comments

For complex arguments, the branch cut for the cube root is taken along the negative real axis. The argument of the result, therefore, is greater than $-\pi/3$ and less than or equal to $\pi/3$. The other two roots are obtained by rotating the principal root by $3\pi/3$ and $\pi/3$.

Examples

Example 1

In this example, the cube root of 3.45 is computed and printed.

```

      USE CBRT_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER      NOUT
      REAL         VALUE, X
!
      X           = 3.45
      VALUE = CBRT(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' CBRT(', F6.3, ') = ', F6.3)
      END

```

Output

```
CBRT( 3.450) = 1.511
```

Example 2

In this example, the cube root of $-3 + 0.0076i$ is computed and printed.

```

      USE UMACH_INT
      USE CBRT_INT
      IMPLICIT NONE

!
      INTEGER      NOUT
      COMPLEX      VALUE, Z
!
      Z           = (-3.0, 0.0076)
      VALUE = CBRT(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CBRT(', F7.4, ', ', F7.4, ') = (', &
      F6.3, ', ', F6.3, ')')
      END

```

Output

```
CBRT((-3.0000, 0.0076)) = ( 0.722, 1.248)
```

EXPRL

This function evaluates the exponential function factored from first order, $(\text{EXP}(X) - 1.0)/X$.

Function Return Value

EXPRL — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **EXPRL (X)**

Specific: The specific interface names are **S_EXPRL**, **D_EXPRL**, and **C_EXPRL**.

FORTRAN 77 Interface

Single: **EXPRL (X)**

Double: The double precision function name is **DEXPRL**.

Complex: The complex name is **CEXPRL**.

Description

The function **EXPRL(X)** evaluates $(e^X - 1)/X$. It will overflow if e^X overflows. For complex arguments, z , the argument z must not be so close to a multiple of $2\pi i$ that substantial significance is lost due to cancellation. Also, the result must not overflow and $|\Im z|$ must not be so large that the trigonometric functions are inaccurate.

Examples

Example 1

In this example, **EXPRL(0.184)** is computed and printed.

```
USE EXPRL_INT
```



```

      USE UMACH_INT
      IMPLICIT  NONE
!
      INTEGER  NOUT
      REAL     VALUE, X
!
      X       = 0.184
      VALUE = EXPRL(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' EXPRL(', F6.3, ') = ', F6.3)
      END

```

Output

```
EXPRL( 0.184) = 1.098
```

Example 2

In this example, `EXPRL(0.0076i)` is computed and printed.

```

      USE EXPRL_INT
      USE UMACH_INT
      IMPLICIT  NONE
!
      INTEGER  NOUT
      COMPLEX  VALUE, Z
!
      Z       = (0.0, 0.0076)
      VALUE = EXPRL(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' EXPRL((', F7.4, ', ', F7.4, ') = (', &
      F6.3, ', ', F6.3, ')')
      END

```

Output

```
EXPRL(( 0.0000, 0.0076)) = ( 1.000, 0.004)
```

LOG10

This function extends FORTRAN's generic `log10` function to evaluate the principal value of the complex common logarithm.

Function Return Value

LOG10 — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `LOG10 (Z)`

Specific: The specific interface names are `CLOG10` and `ZLOG10`.

FORTRAN 77 Interface

Complex: `CLOG10 (Z)`

Double complex: The double complex function name is `ZLOG10`.

Description

The function `LOG10(Z)` evaluates $\log_{10}(z)$. The argument must not be zero, and $|z|$ must not overflow.

Example

In this example, the $\log_{10}(0.0076i)$ is computed and printed.

```

      USE LOG10_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!          Compute
      Z = (0.0, 0.0076)

```

```
      VALUE = LOG10(Z)
!                                     Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' LOG10((', F7.4, ', ', F7.4, ')) = (', &
             F6.3, ', ', F6.3, ')')
      END
```

Output

```
LOG10(( 0.0000, 0.0076)) = (-2.119, 0.682)
```

ALNREL

This function evaluates the natural logarithm of one plus the argument, or, in the case of complex argument, the principal value of the complex natural logarithm of one plus the argument.

Function Return Value

ALNREL — Function value. (Output)

Required Arguments

X — Argument for the function. (Input)

FORTRAN 90 Interface

Generic: **ALNREL(X)**

Specific: The specific interface names are **S_ALNREL**, **D_ALNREL**, and **C_ALNREL**.

FORTRAN 77 Interface

Single: **ALNREL(X)**

Double: The double precision name function is **DLNREL**.

Complex: The complex name is **CLNREL**.

Description

For real arguments, the function **ALNREL(X)** evaluates $\ln(1 + x)$ for $x > -1$. The argument x must be greater than -1.0 to avoid evaluating the logarithm of zero or a negative number. In addition, x must not be so close to -1.0 that considerable significance is lost in evaluating $1 + x$.

For complex arguments, the function **CLNREL(Z)** evaluates $\ln(1 + z)$. The argument z must not be so close to -1 that considerable significance is lost in evaluating $1 + z$. If it is, a recoverable error is issued; however, $z = -1$ is a fatal error because $\ln(1 + z)$ is infinite. Finally, $|z|$ must not overflow.

Let $\rho = |z|$, $z = x + iy$ and $r^2 = |1 + z|^2 = (1 + x)^2 + y^2 = 1 + 2x + \rho^2$. Now, if ρ is small, we may evaluate **CLNREL(Z)** accurately by

$$\begin{aligned}\log(1+z) &= \log r + i\text{Arg}(z+1) \\ &= 1/2 \log r^2 + i\text{Arg}(z+1) \\ &= 1/2 \text{ALNREL}(2x + \rho^2) + i\text{CARG}(1+z)\end{aligned}$$

Comments

Informational Error

Type	Code	Description
3	2	Result of ALNREL(x) is accurate to less than one-half precision because x is too near -1.0.

ALNREL evaluates the natural logarithm of $(1 + X)$ accurate in the sense of relative error even when X is very small. This routine (as opposed to the intrinsic ALOG) should be used to maintain relative accuracy whenever X is small and accurately known.

Examples

Example 1

In this example, $\ln(1.189) = \text{ALNREL}(0.189)$ is computed and printed.

```

USE ALNREL_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
REAL VALUE, X
!
X = 0.189
VALUE = ALNREL(X)                            Compute
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ALNREL(', F6.3, ') = ', F6.3)
END

```

Output

```
ALNREL( 0.189) = 0.173
```

Example 2

In this example, $\ln(0.0076i) = \text{ALNREL}(-1 + 0.0076i)$ is computed and printed.

```
USE UMACH_INT
```

```
      USE ALNREL_INT
      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (-1.0, 0.0076)
      VALUE = ALNREL(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ALNREL(', F8.4, ', ', F8.4, ') = (', &
           F8.4, ', ', F8.4, ')')
      END
```

Output

```
ALNREL(( -1.0000,  0.0076)) = ( -4.8796,  1.5708)
```

Trigonometric and Hyperbolic Functions

Routines

2.1 Trigonometric Functions

Evaluates $\tan z$ for complex z	TAN	30
Evaluates $\cot x$ for real x	COT	32
Evaluates $\sin x$ for x a real angle in degrees	SINDG	35
Evaluates $\cos x$ for x a real angle in degrees	COSDG	37
Evaluates $\sin^{-1} z$ for complex z	ASIN	39
Evaluates $\cos^{-1} z$ for complex z	ACOS	41
Evaluates $\tan^{-1} z$ for complex z	ATAN	43
Evaluates $\tan^{-1}(x/y)$ for x and y complex	ATAN2	45

2.2 Hyperbolic Functions

Evaluates $\sinh z$ for complex z	SINH	47
Evaluates $\cosh z$ for complex z	COSH	49
Evaluates $\tanh z$ for complex z	TANH	51

2.3 Inverse Hyperbolic Functions

Evaluates $\sinh^{-1} x$ for real or complex x	ASINH	53
Evaluates $\cosh^{-1} x$ for real or complex x	ACOSH	55
Evaluates $\tanh^{-1} x$ for real or complex x	ATANH	57

Usage Notes

The complex inverse trigonometric hyperbolic functions are single-valued and regular in a slit complex plane. The branch cuts are shown below for $z = x + iy$, i.e., $x = \Re z$ and $y = \Im z$ are the real and imaginary parts of z , respectively.

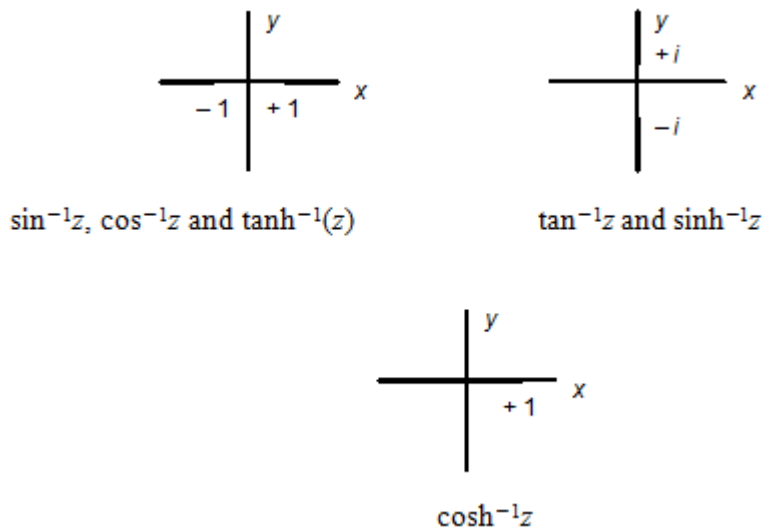


Figure 1, Branch Cuts for Inverse Trigonometric and Hyperbolic Functions

TAN

This function extends FORTRAN's generic tan to evaluate the complex tangent.

Function Return Value

TAN — Complex function value. (Output)

Required Arguments

Z — Complex number representing the angle in radians for which the tangent is desired. (Input)

FORTRAN 90 Interface

Generic: **TAN** (Z)

Specific: The specific interface names are **CTAN** and **ZTAN**.

FORTRAN 77 Interface

Complex: **CTAN** (Z)

Double complex: The double complex function name is **ZTAN**.

Description

Let $z = x + iy$. If $|\cos z|^2$ is very small, that is, if x is very close to $\pi/2$ or $3\pi/2$ and if y is small, then $\tan z$ is nearly singular and a fatal error condition is reported. If $|\cos z|^2$ is somewhat larger but still small, then the result will be less accurate than half precision. When $2x$ is so large that $\sin 2x$ cannot be evaluated to any nonzero precision, the following situation results. If $|y| < 3/2$, then **CTAN** cannot be evaluated accurately to better than one significant figure. If $3/2 \leq |y| < -1/2 \ln \epsilon/2$, then **CTAN** can be evaluated by ignoring the real part of the argument; however, the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Comments

Informational Error

Type	Code	Description
3	2	Result of CTAN(Z) is accurate to less than one-half precision because the real part of z is too near $\pi/2$ or $3\pi/2$ when the imaginary part of z is near zero or because the absolute value of the real part is very large and the absolute value of the imaginary part is small.

Example

In this example, $\tan(1 + i)$ is computed and printed.

```

      USE TAN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (1.0, 1.0)
      VALUE = TAN(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' TAN((', F6.3, ',', F6.3, ') = (', &
             F6.3, ',', F6.3, ')')
      END

```

Output

```
TAN(( 1.000, 1.000)) = ( 0.272, 1.084)
```

COT

This function evaluates the cotangent.

Function Value Return

COT — Function value. (Output)

Required Arguments

X — Angle in radians for which the cotangent is desired. (Input)

FORTRAN 90 Interface

Generic: COT (X)

Specific: The specific interface names are COT, DCOT, CCOT, and ZCOT.

FORTRAN 77 Interface

Single: COT (X)

Double: The double precision function name is DCOT.

Complex: The complex name is CCOT.

Double Complex: The double complex name is ZCOT.

Description

For real x , the magnitude of x must not be so large that most of the computer word contains the integer part of x . Likewise, x must not be too near an integer multiple of π , although x close to the origin causes no accuracy loss. Finally, x must not be so close to the origin that $\text{COT}(x) \approx 1/x$ overflows.

For complex arguments, let $z = x + jy$. If $|\sin z|^2$ is very small, that is, if x is very close to a multiple of π and if $|y|$ is small, then $\cot z$ is nearly singular and a fatal error condition is reported. If $|\sin z|^2$ is somewhat larger but still small, then the result will be less accurate than half precision. When $|2x|$ is so large that $\sin 2x$ cannot be evaluated accurately to even zero precision, the following situation results. If $|y| < 3/2$, then CCOT cannot be evaluated

accurately to be better than one significant figure. If $3/2 \leq |y| < -1/2 \ln \epsilon/2$, where $\epsilon = \mathbf{AMACH}(4)$ is the machine precision, then \mathbf{CCOT} can be evaluated by ignoring the real part of the argument; however, the answer will be less accurate than half precision. Finally, $|z|$ must not be so small that $\cot z \approx 1/z$ overflows.

Comments

1. Informational error for Real arguments

Type	Code	Description
3	2	Result of $\mathbf{COT}(x)$ is accurate to less than one-half precision because $\mathbf{ABS}(x)$ is too large, or x is nearly a multiple of π .

2. Informational error for Complex arguments

Type	Code	Description
3	2	Result of $\mathbf{CCOT}(z)$ is accurate to less than one-half precision because the real part of z is too near a multiple of π when the imaginary part of z is zero, or because the absolute value of the real part is very large and the absolute value of the imaginary part is small.

3. Referencing $\mathbf{COT}(x)$ is **not** the same as computing $1.0/\mathbf{TAN}(x)$ because the error conditions are quite different. For example, when x is near $\pi/2$, $\mathbf{TAN}(x)$ cannot be evaluated accurately and an error message must be issued. However, $\mathbf{COT}(x)$ can be evaluated accurately in the sense of absolute error.

Examples

Example 1

In this example, $\cot(0.3)$ is computed and printed.

```

USE COT_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 0.3
VALUE = COT(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' COT(', F6.3, ') = ', F6.3)
END
```

Output

```
COT( 0.300 ) = 3.233
```

Example 2

In this example, $\cot(1 + i)$ is computed and printed.

```

      USE COT_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (1.0, 1.0)                                Compute
      VALUE = COT(Z)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' COT(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
      END

```

Output

```
COT(( 1.000, 1.000)) = ( 0.218,-0.868)
```

SINDG

This function evaluates the sine for the argument in degrees.

Function Return Value

SINDG — Function value. (Output)

Required Arguments

X — Argument in degrees for which the sine is desired. (Input)

FORTRAN 90 Interface

Generic: SINDG (X)

Specific: The specific interface names are `S_SINDG` and `D_SINDG`.

FORTRAN 77 Interface

Single: SINDG (X)

Double: The double precision function name is `DSINDG`.

Description

To avoid unduly inaccurate results, the magnitude of x must not be so large that the integer part fills more than the computer word. Under no circumstances is the magnitude of x allowed to be larger than the largest representable integer because complete loss of accuracy occurs in this case.

Example

In this example, $\sin 45^\circ$ is computed and printed.

```
      USE SINDG_INT
      USE UMACH_INT

      IMPLICIT NONE
!                                     Declare variables
      INTEGER NOUT
      REAL VALUE, X
```

```
!                                     Compute
  X      = 45.0
  VALUE = SINDG(X)
!                                     Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' SIN(', F6.3, ' deg) = ', F6.3)
  END
```

Output

```
SIN(45.000 deg) = 0.707.
```

COSDG

This function evaluates the cosine for the argument in degrees.

Function Return Value

COSDG — Function value. (Output)

Required Arguments

X — Argument in degrees for which the cosine is desired. (Input)

FORTRAN 90 Interface

Generic: COSDG (X)

Specific: The specific interface names are S_COSDG and D_COSDG.

FORTRAN 77 Interface

Single: COSDG (X)

Double: The double precision function name is DCOSDG.

Description

To avoid unduly inaccurate results, the magnitude of x must not be so large that the integer part fills more than the computer word. Under no circumstances is the magnitude of x allowed to be larger than the largest representable integer because complete loss of accuracy occurs in this case.

Example

In this example, $\cos 100^\circ$ computed and printed.

```

      USE COSDG_INT
      USE UMACH_INT

      IMPLICIT NONE
!                                     Declare variables
      INTEGER NOUT
      REAL VALUE, X

```



```
!                                     Compute
  X      = 100.0
  VALUE = COSDG(X)
!                                     Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' COS(', F6.2, ' deg) = ', F6.3)
  END
```

Output

```
COS(100.00 deg) = -0.174
```

ASIN

This function extends FORTRAN's generic **ASIN** function to evaluate the complex arc sine.

Function Return Value

ASIN — Complex function value in units of radians and the real part in the first or fourth quadrant.
(Output)

Required Arguments

ZINP — Complex argument for which the arc sine is desired. (Input)

FORTRAN 90 Interface

Generic: **ASIN** (**ZINP**)

Specific: The specific interface names are **CASIN** and **ZASIN**.

FORTRAN 77 Interface

Complex: **CASIN** (**ZINP**)

Double complex: The double complex function name is **ZASIN**.

Description

Almost all arguments are legal. Only when $|z| > b/2$ can an overflow occur. Here, $b = \mathbf{AMACH}(2)$ is the largest floating point number. This error is not detected by **ASIN**.

See Pennisi (1963, page 126) for reference.

Example

In this example, $\sin^{-1}(1 - i)$ is computed and printed.

```
USE ASIN_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  COMPLEX      VALUE, Z
!
  Z           = (1.0, -1.0)      Compute
  VALUE = ASIN(Z)
!
  CALL UMACH (2, NOUT)          Print the results
  WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ASIN(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
  END
```

Output

```
ASIN(( 1.000,-1.000)) = ( 0.666,-1.061)
```

ACOS

This function extends FORTRAN's generic **ACOS** function to evaluate the complex arc cosine.

Function Return Value

ACOS — Complex function value in units of radians with the real part in the first or second quadrant.
(Output)

Required Arguments

Z — Complex argument for which the arc cosine is desired. (Input)

FORTRAN 90 Interface

Generic: **ACOS** (Z)

Specific: The specific interface names are **CACOS** and **ZACOS**.

FORTRAN 77 Interface

Complex: **CACOS** (Z)

Double complex: The double complex function name is **ZACOS**.

Description

Almost all arguments are legal. Only when $|z| > b/2$ can an overflow occur. Here, $b = \mathbf{AMACH}(2)$ is the largest floating point number. This error is not detected by **ACOS**.

Example

In this example, $\cos^{-1}(1 - i)$ is computed and printed.

```

      USE ACOS_INT
      USE UMACH_INT

      IMPLICIT NONE
      !                                     Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z

```

```
!                               Compute
  Z      = (1.0, -1.0)
  VALUE = ACOS(Z)
!                               Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ACOS(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
  END
```

Output

```
ACOS(( 1.000,-1.000)) = ( 0.905, 1.061)
```

ATAN

This function extends FORTRAN's generic function **ATAN** to evaluate the complex arc tangent.

Function Return Value

ATAN — Complex function value in units of radians with the real part in the first or fourth quadrant.
(Output)

Required Arguments

Z — Complex argument for which the arc tangent is desired. (Input)

FORTRAN 90 Interface

Generic: **ATAN** (Z)

Specific: The specific interface names are **CATAN** and **ZATAN**.

FORTRAN 77 Interface

Complex: **CATAN** (Z)

Double complex: The double complex function name is **ZATAN**.

Description

The argument z must not be exactly $\pm i$, because $\tan^{-1} z$ is undefined there. In addition, z must not be so close to $\pm i$ that substantial significance is lost.

Comments

Informational error

Type	Code	Description
3	2	Result of ATAN (Z) is accurate to less than one-half precision because $ z^2 $ is too close to -1.0 .

Example

In this example, $\tan^{-1}(0.01 - 0.01i)$ is computed and printed.

```
      USE ATAN_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (0.01, 0.01)
      VALUE = ATAN(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ATAN(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
      END
```

Output

```
ATAN(( 0.010, 0.010)) = ( 0.010, 0.010)
```

ATAN2

This function extends FORTRAN's generic function **ATAN2** to evaluate the complex arc tangent of a ratio.

Function Return Value

ATAN2 — Complex function value in units of radians with the real part between $-\pi$ and π . (Output)

Required Arguments

CSN — Complex numerator of the ratio for which the arc tangent is desired. (Input)

CCS — Complex denominator of the ratio. (Input)

FORTRAN 90 Interface

Generic: **ATAN2** (**CSN**, **CCS**)

Specific: The specific interface names are **CATAN2** and **ZATAN2**.

FORTRAN 77 Interface

Complex: **CATAN2** (**CSN**, **CCS**)

Double complex: The double complex function name is **ZATAN2**.

Description

Let $z_1 = \text{CSN}$ and $z_2 = \text{CCS}$. The ratio $z = z_1/z_2$ must not be $\pm i$ because $\tan^{-1}(\pm i)$ is undefined. Likewise, z_1 and z_2 should not both be zero. Finally, z must not be so close to $\pm i$ that substantial accuracy loss occurs.

Comments

The result is returned in the correct quadrant (modulo 2π).

Example

In this example,

$$\tan^{-1} \frac{(1/2) + (i/2)}{2+i}$$

is computed and printed.

```

      USE ATAN2_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, X, Y
!      Compute
      X = (2.0, 1.0)
      Y = (0.5, 0.5)
      VALUE = ATAN2(Y, X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Y, X, VALUE
99999 FORMAT (' ATAN2(', F6.3, ', ', F6.3, '), (', F6.3, ', ', F6.3, '&
           ')) = (', F6.3, ', ', F6.3, ')')
      END

```

Output

```

ATAN2(( 0.500, 0.500), ( 2.000, 1.000)) = ( 0.294, 0.092)

```

SINH

This function extends FORTRAN's generic function `SINH` to evaluate the complex hyperbolic sine.

Function Return Value

SINH — Complex function value. (Output)

Required Arguments

z — Complex number representing the angle in radians for which the complex hyperbolic sine is desired.
(Input)

FORTRAN 90 Interface

Generic: `SINH (z)`
Specific: The specific interface names are `CSINH` and `ZSINH`.

FORTRAN 77 Interface

Complex: `CSINH (z)`
Double complex: The double complex function name is `ZSINH`.

Description

The argument z must satisfy

$$|\Im z| \leq 1 / \sqrt{\epsilon}$$

where $\epsilon = \text{AMACH}(4)$ is the machine precision and $\Im z$ is the imaginary part of z .

Example

In this example, `sinh(5 - i)` is computed and printed.

```
USE SINH_INT
USE UMACH_INT
```

```
      IMPLICIT  NONE
!
      INTEGER  NOUT
      COMPLEX  VALUE, Z
!
      Z      = (5.0, -1.0)
      VALUE = SINH(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' SINH(', F6.3, ', ', F6.3, ') = (', &
             F7.3, ', ', F7.3, ')')
      END
```

Output

```
SINH(( 5.000,-1.000)) = ( 40.092,-62.446)
```

COSH

The function extends FORTRAN's generic function `COSH` to evaluate the complex hyperbolic cosine.

Function Return Value

COSH — Complex function value. (Output)

Required Arguments

Z — Complex number representing the angle in radians for which the hyperbolic cosine is desired.
(Input)

FORTRAN 90 Interface

Generic: `COSH (Z)`

Specific: The specific interface names are `CCOSH` and `ZCOSH`.

FORTRAN 77 Interface

Complex: `CCOSH (Z)`

Double complex: The double complex function name is `ZCOSH`.

Description

Let $\epsilon = \text{AMACH}(4)$ be the machine precision. If $|\Im z|$ is larger than

$$1 / \sqrt{\epsilon}$$

then the result will be less than half precision, and a recoverable error condition is reported. If $|\Im z|$ is larger than $1/\epsilon$, the result has no precision and a fatal error is reported. Finally, if $|\Re z|$ is too large, the result overflows and a fatal error results. Here, $\Re z$ and $\Im z$ represent the real and imaginary parts of z , respectively.

Example

In this example, `cosh(-2 + 2i)` is computed and printed.

```
      USE COSH_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (-2.0, 2.0)
      VALUE = COSH(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' COSH(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
COSH((-2.000, 2.000)) = (-1.566,-3.298)
```

TANH

This function extends FORTRAN's generic function `TANH` to evaluate the complex hyperbolic tangent.

Function Return Value

TANH — Complex function value. (Output)

Required Arguments

Z — Complex number representing the angle in radians for which the hyperbolic tangent is desired.
(Input)

FORTRAN 90 Interface

Generic: `TANH (Z)`

Specific: The specific interface names are `CTANH` and `ZTANH`.

FORTRAN 77 Interface

Complex: `CTANH (Z)`

Double complex: The double complex function name is `ZTANH`.

Description

Let $z = x + iy$. If $|\cosh z|^2$ is very small, that is, if $y \bmod \pi$ is very close to $\pi/2$ or $3\pi/2$ and if x is small, then $\tanh z$ is nearly singular; a fatal error condition is reported. If $|\cosh z|^2$ is somewhat larger but still small, then the result will be less accurate than half precision. When $2y$ ($z = x + iy$) is so large that $\sin 2y$ cannot be evaluated accurately to even zero precision, the following situation results. If $|x| < 3/2$, then `TANH` cannot be evaluated accurately to better than one significant figure. If $3/2 \leq |y| < -1/2 \ln(\epsilon/2)$, then `TANH` can be evaluated by ignoring the imaginary part of the argument; however, the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $\tanh(1 + i)$ is computed and printed.

```
      USE TANH_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (1.0, 1.0)
      VALUE = TANH(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' TANH((', F6.3, ', ', F6.3, ')) = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
TANH(( 1.000, 1.000)) = ( 1.084, 0.272)
```

ASINH

This function evaluates the arc hyperbolic sine.

Function Return Value

ASINH — Function value. (Output)

Required Arguments

X — Argument for which the arc hyperbolic sine is desired. (Input)

FORTRAN 90 Interface

Generic: **ASINH (X)**

Specific: The specific interface names are **ASINH**, **DASINH**, **CASINH**, and **ZASINH**.

FORTRAN 77 Interface

Single: **ASINH (X)**

Double: The double precision function name is **DASINH**.

Complex: The complex name is **CASINH**.

Double Complex: The double complex name is **ZASINH**.

Description

The function **ASINH (X)** computes the inverse hyperbolic sine of x , $\sinh^{-1}x$.

For complex arguments, almost all arguments are legal. Only when $|z| > b/2$ can an overflow occur, where $b = \text{AMACH} (2)$ is the largest floating point number. This error is not detected by **ASINH**.

Examples

Example 1

In this example, $\sinh^{-1}(2.0)$ is computed and printed.


```

      USE ASINH_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      REAL VALUE, X
!
      X = 2.0                                       Compute
      VALUE = ASINH(X)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ASINH(', F6.3, ') = ', F6.3)
      END

```

Output

```
ASINH( 2.000) = 1.444
```

Example 2

In this example, $\sinh^{-1}(-1 + i)$ is computed and printed.

```

      USE ASINH_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (-1.0, 1.0)                               Compute
      VALUE = ASINH(Z)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ASINH(', F6.3, ', ', F6.3, ') = (', &
      F6.3, ', ', F6.3, ')')
      END

```

Output

```
ASINH((-1.000, 1.000)) = (-1.061, 0.666)
```

ACOSH

This function evaluates the arc hyperbolic cosine.

Function Return Value

ACOSH — Function value. (Output)

Required Arguments

X — Argument for which the arc hyperbolic cosine is desired. (Input)

FORTRAN 90 Interface

Generic: **ACOSH(X)**

Specific: The specific interface names are **ACOSH**, **DACOSH**, **CACOSH**, and **ZACOSH**.

FORTRAN 77 Interface

Single: **ACOSH(X)**

Double: The double precision function name is **DACOSH**.

Complex: The complex name is **CACOSH**.

Double Complex: The double complex name is **ZACOSH**.

Description

The function **ACOSH(X)** computes the inverse hyperbolic cosine of x , $\cosh^{-1}x$.

For complex arguments, almost all arguments are legal. Only when $|z| > b/2$ can an overflow occur, where $b = \mathbf{AMACH}(2)$ is the largest floating point number. This error is not detected by **ACOSH**.

Comments

The result of **ACOSH(X)** is returned on the positive branch. Recall that, like **SQRT(X)**, **ACOSH(X)** has multiple values.

Examples

Example 1

In this example, $\cosh^{-1}(1.4)$ is computed and printed.

```

      USE ACOSH_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL VALUE, X
!
      X = 1.4
      VALUE = ACOSH(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ACOSH(', F6.3, ') = ', F6.3)
      END

```

Output

```
ACOSH( 1.400) = 0.867
```

Example 2

In this example, $\cosh^{-1}(1 - i)$ is computed and printed.

```

      USE ACOSH_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (1.0, -1.0)
      VALUE = ACOSH(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ACOSH(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
      END

```

Output

```
ACOSH(( 1.000,-1.000)) = (-1.061, 0.905)
```

ATANH

This function evaluates the arc hyperbolic tangent.

Function Return Value

ATANH — Function value. (Output)

Required Arguments

X — Argument for which the arc hyperbolic tangent is desired. (Input)

FORTRAN 90 Interface

Generic: **ATANH (X)**

Specific: The specific interface names are **ATANH**, **DATANH**, **CATANH**, and **ZATANH**

FORTRAN 77 Interface

Single: **ATANH (X)**

Double: The double precision function name is **DATANH**.

Complex: The complex name is **CATANH**.

Double Complex: The double complex name is **ZATANH**.

Description

ATANH (X) computes the inverse hyperbolic tangent of x , $\tanh^{-1}x$. The argument x must satisfy

$$|x| < 1 - \sqrt{\epsilon}$$

where $\epsilon = \mathbf{AMACH}(4)$ is the machine precision. Note that $|x|$ must not be so close to one that the result is less accurate than half precision.

Comments

Informational Error

Type	Code	Description
3	2	Result of $\text{ATANH}(X)$ is accurate to less than one-half precision because the absolute value of the argument is too close to 1.0.

Examples

Example 1

In this example, $\tanh^{-1}(-1/4)$ is computed and printed.

```

USE ATANH_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
REAL VALUE, X                                Compute
!
X      = -0.25
VALUE = ATANH(X)
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ATANH(', F6.3, ') = ', F6.3)
END

```

Output

```
ATANH(-0.250) = -0.255
```

Example 2

In this example, $\tanh^{-1}(1/2 + i/2)$ is computed and printed.

```

USE ATANH_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
COMPLEX VALUE, Z
!
Z      = (0.5, 0.5)
VALUE = ATANH(Z)
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ATANH(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
END

```

Output

```
ATANH(( 0.500, 0.500)) = ( 0.402, 0.554)
```

Exponential Integrals and Related Functions

Routines

Evaluates the exponential integral, $Ei(x)$	EI	62
Evaluates the exponential integral, $E_1(x)$	E1	64
Evaluates the scaled exponential integrals, integer order, $E_n(x)$	ENE	66
Evaluates the logarithmic integral, $li(x)$	ALI	68
Evaluates the sine integral, $Si(x)$	SI	71
Evaluates the cosine integral, $CI(x)$	CI	73
Evaluates the cosine integral (alternate definition)	CIN	75
Evaluates the hyperbolic sine integral, $Shi(x)$	SHI	77
Evaluates the hyperbolic cosine integral, $Chi(x)$	CHI	79
Evaluates the hyperbolic cosine integral (alternate definition)	CINH	81

Usage Notes

The notation used in this chapter follows that of Abramowitz and Stegun (1964).

The following is a plot of the exponential integral functions that can be computed by the routines described in this chapter.

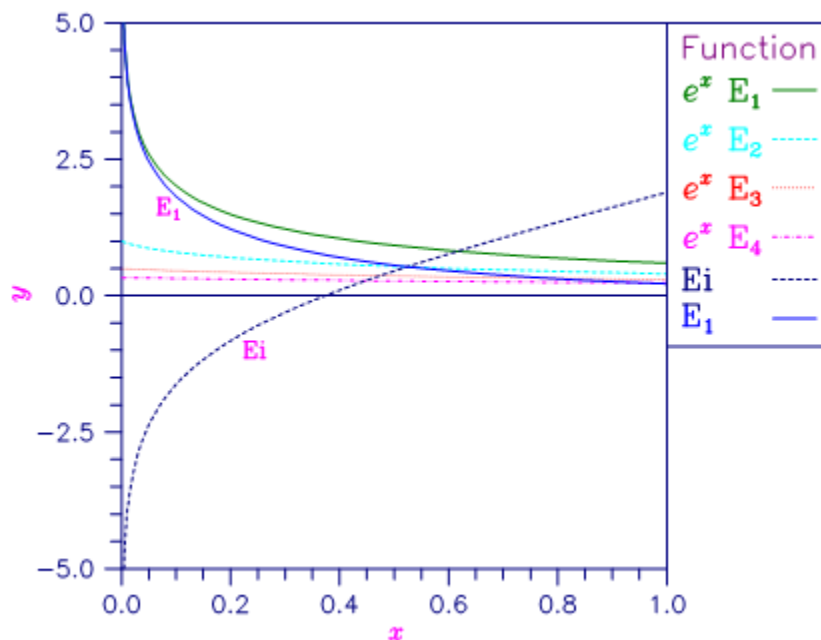


Figure 2, Plot of $e^x E(x)$, $E_1(x)$ and $Ei(x)$

EI

This function evaluates the exponential integral for arguments greater than zero and the Cauchy principal value for arguments less than zero.

Function Return Value

EI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **EI (X)**

Specific: The specific interface names are **S_EI** and **D_EI**.

FORTRAN 77 Interface

Single: **EI (X)**

Double: The double precision function name is **DEI**.

Description

The exponential integral, $Ei(x)$, is defined to be

$$E_1(x) = \int_x^{\infty} e^{-t}/t dt \quad \text{for } x \neq 0$$

The argument x must be large enough to insure that the asymptotic formula e^x/x does not underflow, and x must not be so large that e^x overflows.

Comments

If principal values are used everywhere, then for all X , $Ei(X) = -E_1(-X)$ and $E_1(X) = -Ei(-X)$.

Example

In this example, $Ei(1.15)$ is computed and printed.

```
      USE EI_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 1.15
      VALUE = EI(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' EI(', F6.3, ') = ', F6.3)
      END
```

Output

```
EI( 1.150) = 2.304
```

E1

This function evaluates the exponential integral for arguments greater than zero and the Cauchy principal value of the integral for arguments less than zero.

Function Return Value

E1 — Function value. (Output)

Required Arguments

X — Argument for which the integral is to be evaluated. (Input)

FORTRAN 90 Interface

Generic: **E1 (X)**

Specific: The specific interface names are **S_E1** and **D_E1**.

FORTRAN 77 Interface

Single: **E1 (X)**

Double: The double precision function name is **DE1**.

Description

The alternate definition of the exponential integral, $E_1(x)$, is

$$E_1(x) = \int_x^{\infty} e^{-t}/t dt \text{ for } x \neq 0$$

The path of integration must exclude the origin and not cross the negative real axis.

The argument x must be large enough that e^{-x} does not overflow, and x must be small enough to insure that e^{-x}/x does not underflow.

Comments

Informational Error

Type	Code	Description
3	2	The function underflows because x is too large.

Example

In this example, $E_1(1.3)$ is computed and printed.

```
USE E1_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 1.3
VALUE = E1(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' E1(', F6.3, ') = ', F6.3)
END
```

Output

```
E1( 1.300) = 0.135
```

ENE

Evaluates the exponential integral of integer order for arguments greater than zero scaled by **EXP(X)**.

Required Arguments

- X** — Argument for which the integral is to be evaluated. (Input)
It must be greater than zero.
- N** — Integer specifying the maximum order for which the exponential integral is to be calculated. (Input)
- F** — Vector of length **N** containing the computed exponential integrals scaled by **EXP(X)**. (Output)

FORTRAN 90 Interface

- Generic: `CALL ENE (X, N, F)`
- Specific: The specific interface names are `S_ENE` and `D_ENE`.

FORTRAN 77 Interface

- Single: `CALL ENE (X, N, F)`
- Double: The double precision function name is `DENE`.

Description

The scaled exponential integral of order n , $E_n(x)$, is defined to be

$$E_n(x) = e^x \int_1^{\infty} e^{-xt} t^{-n} dt \quad \text{for } x > 0$$

The argument x must satisfy $x > 0$. The integer n must also be greater than zero. This code is based on a code due to Gautschi (1974).

Example

In this example, $E_n(10)$ for $n = 1, \dots, n$ is computed and printed.

```
      USE ENE_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=10)
!
      INTEGER K, NOUT
      REAL F(N), X
!           Compute
      X = 10.0
      CALL ENE (X, N, F)
!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) K, X, F(K)
10 CONTINUE
99999 FORMAT (' E sub ', I2, ' (', F6.3, ') = ', F6.3)
      END
```

Output

```
E sub 1 (10.000) = 0.092
E sub 2 (10.000) = 0.084
E sub 3 (10.000) = 0.078
E sub 4 (10.000) = 0.073
E sub 5 (10.000) = 0.068
E sub 6 (10.000) = 0.064
E sub 7 (10.000) = 0.060
E sub 8 (10.000) = 0.057
E sub 9 (10.000) = 0.054
E sub 10 (10.000) = 0.051
```

ALI

This function evaluates the logarithmic integral.

Function Return Value

ALI — Function value. (Output)

Required Arguments

X — Argument for which the logarithmic integral is desired. (Input)
It must be greater than zero and not equal to one.

FORTRAN 90 Interface

Generic: **ALI (X)**
Specific: The specific interface names are **S_ALI** and **D_ALI**.

FORTRAN 77 Interface

Single: **ALI (X)**
Double: The double precision function name is **DALI**.

Description

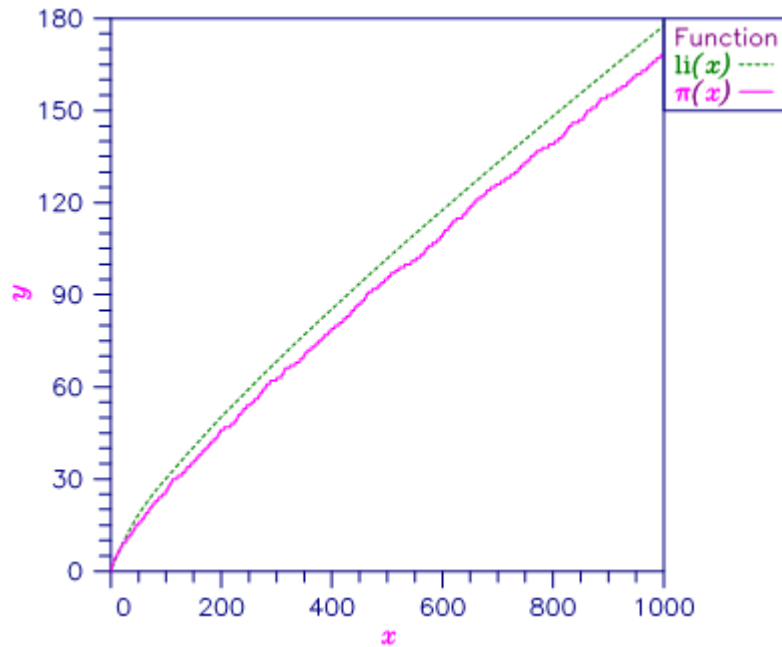
The logarithmic integral, $\text{li}(x)$, is defined to be

$$\text{li}(x) = - \int_0^x \frac{dt}{\ln t} \quad \text{for } x > 0 \text{ and } x \neq 1$$

The argument x must be greater than zero and not equal to one. To avoid an undue loss of accuracy, x must be different from one at least by the square root of the machine precision.

The function $\text{li}(x)$ approximates the function $\pi(x)$, the number of primes less than or equal to x . Assuming the Riemann hypothesis (all non-real zeros of $\zeta(z)$ are on the line $\Re z = 1/2$), then

$$\text{li}(x) - \pi(x) = O(\sqrt{x} \ln x)$$

Figure 3, Plot of $\text{li}(x)$ and $\pi(x)$

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>ALI(X)</code> is accurate to less than one-half precision because <code>x</code> is too close to 1.0.

Example

In this example, $\text{li}(2.3)$ is computed and printed.

```

USE ALI_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 2.3
VALUE = ALI(X)

```

Declare variables

Compute


```
!                                     Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ALI(', F6.3, ') = ', F6.3)
  END
```

Output

```
ALI( 2.300) =  1.439
```

SI

This function evaluates the sine integral.

Function Return Value

SI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `SI (X)`

Specific: The specific interface names are `S_SI` and `D_SI`.

FORTRAN 77 Interface

Single: `SI (X)`

Double: The double precision function name is `DSI`.

Description

The sine integral, $\text{Si}(x)$, is defined to be

$$\text{Si}(x) = \int_0^x \frac{\sin t}{t} dt$$

If

$$|x| > 1/\sqrt{\epsilon}$$

the answer is less accurate than half precision, while for $|x| > 1/\epsilon$, the answer has no precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $\text{Si}(1.25)$ is computed and printed.

```
      USE SI_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 1.25
      VALUE = SI(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' SI(', F6.3, ') = ', F6.3)
      END
```

Output

```
SI( 1.250) = 1.146
```

CI

This function evaluates the cosine integral.

Function Return Value

CI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
It must be greater than zero.

FORTRAN 90 Interface

Generic: **CI (X)**
Specific: The specific interface names are **S_CI** and **D_CI**.

FORTRAN 77 Interface

Single: **CI (X)**
Double: The double precision function name is **DCI**.

Description

The cosine integral, $Ci(x)$, is defined to be

$$Ci(x) = \gamma + \ln(x) + \int_0^x \frac{\cos t - 1}{t} dt$$

Where $\gamma \approx 0.57721566$ is Euler's constant.

The argument x must be larger than zero. If

$$x > 1/\sqrt{\epsilon}$$

then the result will be less accurate than half precision. If $x > 1/\epsilon$, the result will have no precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $\text{Ci}(1.5)$ is computed and printed.

```
      USE CI_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 1.5
      VALUE = CI(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' CI(', F6.3, ') = ', F6.3)
      END
```

Output

```
CI( 1.500) = 0.470
```

CIN

This function evaluates a function closely related to the cosine integral.

Function Return Value

CIN — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **CIN (X)**

Specific: The specific interface names are **S_CIN** and **D_CIN**.

FORTRAN 77 Interface

Single: **CIN (X)**

Double: The double precision function name is **DCIN**.

Description

The alternate definition of the cosine integral, $\text{Cin}(x)$, is

$$\text{Cin}(x) = \int_0^x \frac{1 - \cos t}{t} dt$$

For

$$0 < |x| < \sqrt{s}$$

where $s = \text{AMACH}(1)$ is the smallest representable positive number, the result underflows. For

$$|x| > 1/\sqrt{\epsilon}$$

the answer is less accurate than half precision, while for $|x| > 1/\epsilon$, the answer has no precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too small.

Example

In this example, $\text{Cin}(2\pi)$ is computed and printed.

```

      USE CIN_INT
      USE UMACH_INT
      USE CONST_INT

      IMPLICIT NONE
!
!                                     Declare variables
!
      INTEGER NOUT
      REAL VALUE, X
!
!                                     Compute
      X = CONST('pi')
      X = 2.0* X
      VALUE = CIN(X)
!
!                                     Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' CIN( ', F6.3, ') = ', F6.3)
      END

```

Output

```
CIN( 6.283) = 2.438
```

SHI

This function evaluates the hyperbolic sine integral.

Function Return Value

SHI— Function value. (Output)

SHI equals

$$\int_0^x \sinh(t) / t dt$$

Required Arguments

X— Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: SHI (X)

Specific: The specific interface names are S_SHI and D_SHI.

FORTRAN 77 Interface

Single: SHI (X)

Double: The double precision function name is DSHI.

Description

The hyperbolic sine integral, Shi(x), is defined to be

$$\text{Shi}(x) = \int_0^x \frac{\sinh t}{t} dt$$

The argument x must be large enough that e^{-x}/x does not underflow, and x must be small enough that e^x does not overflow.

Example

In this example, Shi(3.5) is computed and printed.

```
      USE SHI_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 3.5
      VALUE = SHI(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' SHI(', F6.3, ') = ', F6.3)
      END
```

Output

```
SHI( 3.500) = 6.966
```

CHI

This function evaluates the hyperbolic cosine integral.

Function Return Value

CHI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **CHI (X)**

Specific: The specific interface names are **S_CHI** and **D_CHI**.

FORTRAN 77 Interface

Single: **CHI (X)**

Double: The double precision function name is **DCHI**.

Description

The hyperbolic cosine integral, $\text{Chi}(x)$, is defined to be

$$\text{Chi}(x) = \gamma + \ln x + \int_0^x \frac{\cosh t - 1}{t} dt \quad \text{for } x > 0$$

where $\gamma \approx 0.57721566$ is Euler's constant.

The argument x must be large enough that e^{-x}/x does not underflow, and x must be small enough that e^x does not overflow.

Comments

When x is negative, the principal value is used.

Example

In this example, Chi(2.5) is computed and printed.

```
      USE CHI_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 2.5
      VALUE = CHI(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' CHI(', F6.3, ') = ', F6.3)
      END
```

Output

```
CHI(2.500) = 3.524
```

CINH

This function evaluates a function closely related to the hyperbolic cosine integral.

Function Return Value

CINH — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **CINH (X)**

Specific: The specific interface names are **S_CINH** and **D_CINH**.

FORTRAN 77 Interface

Single: **CINH (X)**

Double: The double precision function name is **DCINH**.

Description

The alternate definition of the hyperbolic cosine integral, $\text{Cinh}(x)$, is

$$\text{Cinh}(x) = \int_0^x \frac{\cosh t - 1}{t} dt$$

For

$$0 < |x| < 2\sqrt{s}$$

where $s = \text{AMACH}(1)$ is the smallest representable positive number, the result underflows. The argument x must be large enough that e^{-x}/x does not underflow, and x must be small enough that e^x does not overflow.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too small.

Example

In this example, Cinh(2.5) is computed and printed.

```
USE CINH_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 2.5
VALUE = CINH(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' CINH(', F6.3, ') = ', F6.3)
END
```

Output

```
CINH( 2.500) = 2.031
```

Gamma Function and Related Functions

Routines

4.1	Factorial Function		
	Evaluates the factorial, $n!$	FAC	85
	Evaluates the binomial coefficient.	BINOM	87
4.2	Gamma Function		
	Evaluates the real or complex gamma function, $\Gamma(x)$	GAMMA	89
	Evaluates the reciprocal of the real or complex gamma function, $1/\Gamma(x)$	GAMR	92
	Evaluates the real or complex function, $\ln \Gamma(x) $	ALNGAM	94
	Evaluates the log abs gamma function and its sign.	ALGAMS	97
4.3	Incomplete Gamma Function		
	Evaluates the incomplete gamma function, $\Gamma(a, x)$	GAMI	99
	Evaluates the complementary incomplete gamma function, $\Gamma(a, x)$	GAMIC	102
	Evaluates Tricomi's incomplete gamma function, $\Gamma^*(a, x)$	GAMIT	104
4.4	Psi Function		
	Evaluates the real or complex psi function, $\Psi(x)$	PSI	106
	Evaluates the real psi1 function, $\Psi_1(x)$	PSI1	109
4.5	Pochhammer's Function		
	Evaluates Pochhammer's generalized symbol, $(a)_x$	POCH	111
	Evaluates Pochhammer's symbol starting from the first order.	POCH1	113
4.6	Beta Function		
	Evaluates the real or complex beta function, $\Gamma(a, b)$	BETA	115
	Evaluates the log of the real or complex beta function, $\ln \Gamma(a, b)$	ALBETA	118
	Evaluates the incomplete beta function, $I_x(a, b)$	BETA1	121

Usage Notes

The notation used in this chapter follows that of Abramowitz and Stegun (1964).

The following is a table of the functions defined in this chapter:

FAC	$n! = \Gamma(n + 1)$
BINOM	$n!/m!(n - m)!, 0 \leq m \leq n$
GAMMA	$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt, x \neq 0, -1, -2, \dots$
GAMR	$1/\Gamma(x)$
ALNGAM	$\ln \Gamma(x) , x \neq 0, -1, -2, \dots$
ALGAMS	$\ln \Gamma(x) $ and $\text{sign } \Gamma(x), x \neq 0, -1, -2, \dots$
GAMI	$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt, a > 0, x \geq 0$
GAMIC	$\Gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt, x > 0$
GAMIT	$\gamma^*(a, x) = (x^{-a}/\Gamma(a))\gamma(a, x), x \geq 0$
PSI	$\Psi(x) = \Gamma'(x)/\Gamma(x), x \neq 0, -1, -2, \dots$
PSI1	$\Psi_1(x) = d^2/dx^2 \ln \Gamma(x), x \neq 0, -1, -2, \dots$
POCH	$(a)_x = \Gamma(a + x)/\Gamma(a)$, if $a + x = 0, -1, -2, \dots$ then a must = $0, -1, -2, \dots$
POCH1	$((a)_x - 1)/x$, if $a + x = 0, -1, -2, \dots$ then a must = $0, -1, -2, \dots$
BETA	$\beta(x_1, x_2) = \Gamma(x_1)\Gamma(x_2)/\Gamma(x_1 + x_2), x_1 > 0$ and $x_2 > 0$
CBETA	$\beta(z_1, z_2) = \Gamma(z_1)\Gamma(z_2)/\Gamma(z_1 + z_2), z_1 > 0$ and $z_2 > 0$
ALBETA	$\ln \beta(a, b), a > 0, b > 0$
BETAI	$I_x(a, b) = \beta_x(a, b)/\beta(a, b), 0 \leq x \leq 1, a > 0, b > 0$

FAC

This function evaluates the factorial of the argument.

Function Return Value

FAC — Function value. (Output)

See [Comments](#).

Required Arguments

N — Argument for which the factorial is desired. (Input)

FORTRAN 90 Interface

Generic: **FAC (N)**

Specific: The specific interface names are **S_FAC** and **D_FAC**.

FORTRAN 77 Interface

Single: **FAC (N)**

Double: The double precision function name is **DFAC**.

Description

The factorial is computed using the relation $n! = \Gamma(n + 1)$. The function $\Gamma(x)$ is defined in [GAMMA](#). The argument n must be greater than or equal to zero, and it must not be so large that $n!$ overflows. Approximately, $n!$ overflows when $n^n e^{-n}$ overflows.

Comments

If the generic version of this function is used, the immediate result must be stored in a variable before use in an expression. For example:

```
X = FAC(6)
```

```
Y = SQRT(X)
```


must be used rather than

$$Y = \text{SQRT}(\text{FAC}(6)).$$

If this is too much of a restriction on the programmer, then the specific name can be used without this restriction.

To evaluate the factorial for nonintegral values of the argument, the gamma function should be used. For large values of the argument, the log gamma function should be used.

Example

In this example, 6! is computed and printed.

```
USE FAC_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER N, NOUT
REAL VALUE
!
N = 6
VALUE = FAC(N)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) N, VALUE
99999 FORMAT (' FAC(', I1, ') = ', F6.2)
END
```

Output

```
FAC(6) = 720.00
```

BINOM

This function evaluates the binomial coefficient.

Function Return Value

BINOM — Function value. (Output)

See [Comment 1](#).

Required Arguments

N — First parameter of the binomial coefficient. (Input)

N must be nonnegative.

M — Second parameter of the binomial coefficient. (Input)

M must be nonnegative and less than or equal to **N**.

FORTRAN 90 Interface

Generic: **BINOM** (**N**, **M**)

Specific: The specific interface names are **S_BINOM** and **D_BINOM**.

FORTRAN 77 Interface

Single: **BINOM** (**N**, **M**)

Double: The double precision function name is **DBINOM**.

Description

The binomial function is defined to be

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

with $n \geq m \geq 0$. Also, n must not be so large that the function overflows.

Comments

1. If the generic version of this function is used, the immediate result must be stored in a variable before use in an expression. For example:

$$X = \text{BINOM}(9, 5) \quad Y = \text{SQRT}(X)$$

must be used rather than

$$Y = \text{SQRT}(\text{BINOM}(9, 5)).$$

If this is too much of a restriction on the programmer, then the specific name can be used without this restriction.

2. To evaluate binomial coefficients for nonintegral values of the arguments, the complete beta function or log beta function should be used.

Example

In this example, $\binom{9}{5}$ is computed and printed.

```

      USE BINOM_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER M, N, NOUT
      REAL VALUE
!      Compute
      N = 9
      M = 5
      VALUE = BINOM(N, M)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) N, M, VALUE
99999 FORMAT (' BINOM(', I1, ', ', I1, ') = ', F6.2)
      END

```

Output

```
BINOM(9,5) = 126.00
```

GAMMA

This function evaluates the complete gamma function.

Function Return Value

GAMMA — Function value. (Output)

Required Arguments

X — Argument for which the complete gamma function is desired. (Input)

FORTRAN 90 Interface

Generic: **GAMMA (X)**

Specific: The specific interface names are **S_GAMMA**, **D_GAMMA**, and **C_GAMMA**.

FORTRAN 77 Interface

Single: **GAMMA (X)**

Double: The double precision function name is **DGAMMA**.

Complex: The complex name is **CGAMMA**.

Description

The gamma function, $\Gamma(z)$, is defined to be

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt \quad \text{for } \Re z > 0$$

For $\Re(z) < 0$, the above definition is extended by analytic continuation.

z must not be so close to a negative integer that the result is less accurate than half precision. If $\Re(z)$ is too small, then the result will underflow. Users who need such values should use the log gamma function [ALNGAM](#). When $\Im(z) \approx 0$, $\Re(z)$ should be greater than x_{min} so that the result does not underflow, and $\Re(z)$ should be less than x_{max} so that the result does not overflow. x_{min} and x_{max} are available by

```
CALL R9GAML (XMIN, XMAX)
```

Note that z must not be too far from the real axis because the result will underflow.

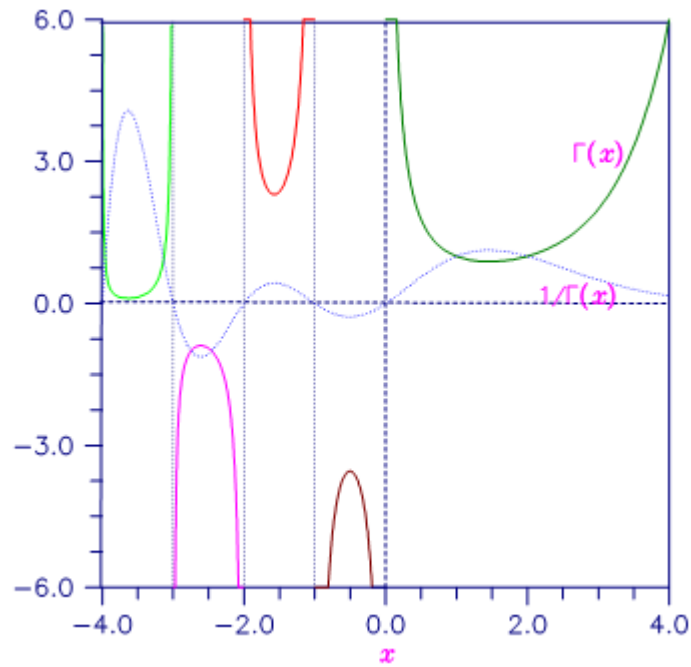


Figure 4, Plot of $\Gamma(x)$ and $1/\Gamma(x)$

Comments

Informational Errors

Type	Code	Description
2	3	The function underflows because x is too small.
3	2	Result is accurate to less than one-half precision because x is too near a negative integer.

Examples

Example 1

In this example, $\Gamma(5.0)$ is computed and printed.

```
USE GAMMA_INT
USE UMACH_INT
```

```

      IMPLICIT  NONE
!
      INTEGER  NOUT
      REAL     VALUE, X
!
      X       = 5.0
      VALUE = GAMMA(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' GAMMA(', F6.3, ') = ', F6.3)
      END

```

Output

```
GAMMA( 5.000) = 24.000
```

Example 2

In this example, $\Gamma(1.4 + 3i)$ is computed and printed.

```

      USE GAMMA_INT
      USE UMACH_INT

      IMPLICIT  NONE
!
      INTEGER  NOUT
      COMPLEX  VALUE, Z
!
      Z       = (1.4, 3.0)
      VALUE = GAMMA(Z)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' GAMMA(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
      END

```

Output

```
GAMMA( 1.400, 3.000) = (-0.001, 0.061)
```

GAMR

This function evaluates the reciprocal gamma function.

Function Return Value

GAMR — Function value. (Output)

Required Arguments

X — Argument for which the reciprocal gamma function is desired. (Input)

FORTRAN 90 Interface

Generic: **GAMR** (**X**)

Specific: The specific interface names are **S_GAMR**, **D_GAMR**, and **C_GAMR**

FORTRAN 77 Interface

Single: **GAMR** (**X**)

Double: The double precision function name is **DGAMR**.

Complex: The complex name is **CGAMR**.

Description

The function **GAMR** computes $1/\Gamma(z)$. See [GAMMA](#) for the definition of $\Gamma(z)$.

For $\Im(z) \approx 0$, z must be larger than x_{min} so that $1/\Gamma(z)$ does not underflow, and x must be smaller than x_{max} so that $1/\Gamma(z)$ does not overflow. Symmetric overflow and underflow limits x_{min} and x_{max} are obtainable from

```
CALL R9GAML (XMIN, XMAX)
```

Note that z must not be too far from the real axis because the result will overflow there.

Comments

This function is well behaved near zero and negative integers.

Examples

Example 1

In this example, $1/\Gamma(1.85)$ is computed and printed.

```

      USE GAMR_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      REAL VALUE, X
!
      X = 1.85
      VALUE = GAMR(X)                               Compute
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' GAMR(', F6.3, ') = ', F6.3)
      END

```

Output

```
GAMR( 1.850) = 1.058
```

Example 2

In this example, $\ln \Gamma(1.4 + 3i)$ is computed and printed.

```

      USE GAMR_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (1.4, 3.0)
      VALUE = GAMR(Z)                               Compute
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' GAMR(', F6.3, ', ', F6.3, ') = (', F7.3, ', ', F7.3, ')')
      END

```

Output

```
GAMR( 1.400, 3.000) = ( -0.303, -16.367)
```


ALNGAM

The function evaluates the logarithm of the absolute value of the gamma function.

Function Return Value

ALNGAM — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `ALNGAM (X)`

Specific: The specific interface names are `S_ALNGAM`, `D_ALNGAM`, and `C_ALNGAM`.

FORTRAN 77 Interface

Single: `ALNGAM (X)`

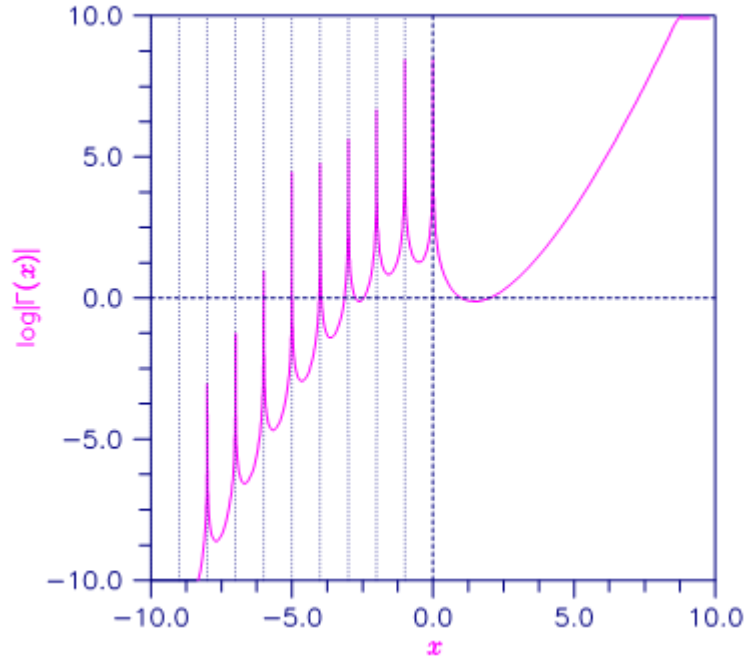
Double: The double precision function name is `DLNGAM`.

Complex: The complex name is `CLNGAM`.

Description

The function `ALNGAM` computes $\ln |\Gamma(x)|$. See [GAMMA](#) for the definition of $\Gamma(x)$.

The gamma function is not defined for integers less than or equal to zero. Also, $|x|$ must not be so large that the result overflows. Neither should x be so close to a negative integer that the accuracy is worse than half precision.

Figure 5, Plot of $\log|\Gamma(x)|$

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>ALNGAM(x)</code> is accurate to less than one-half precision because <code>x</code> is too near a negative integer.

Examples

Example 1

In this example, $\ln |\Gamma(1.85)|$ is computed and printed.

```

      USE ALNGAM_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      Declare variables

```

```

REAL      VALUE, X
!
X      = 1.85           Compute
VALUE = ALNGAM(X)
!
CALL UMACH (2, NOUT)   Print the results
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ALNGAM(', F6.3, ') = ', F6.3)
END

```

Output

```
ALNGAM( 1.850) = -0.056
```

Example 2

In this example, $\ln \Gamma(1.4 + 3i)$ is computed and printed.

```

USE ALNGAM_INT
USE UMACH_INT

IMPLICIT NONE           Declare variables
!
INTEGER    NOUT
COMPLEX    VALUE, Z
!
Z      = (1.4, 3.0)     Compute
VALUE = ALNGAM(Z)
!
CALL UMACH (2, NOUT)   Print the results
WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' ALNGAM(', F6.3, ', ', F6.3, ') = (' ,&
             F6.3, ', ', F6.3, ')')
END

```

Output

```
ALNGAM( 1.400, 3.000) = (-2.795, 1.589)
```

ALGAMS

Returns the logarithm of the absolute value of the gamma function and the sign of gamma.

Required Arguments

X — Argument for which the logarithm of the absolute value of the gamma function is desired. (Input)

ALGM — Result of the calculation. (Output)

S — Sign of $\text{gamma}(x)$. (Output)

If $\text{gamma}(x)$ is greater than or equal to zero, $S = 1.0$. If $\text{gamma}(x)$ is less than zero, $S = -1.0$.

FORTRAN 90 Interface

Generic: `CALL ALGAMS (X, ALGM, S)`

Specific: The specific interface names are `S_ALGAMS` and `D_ALGAMS`.

FORTRAN 77 Interface

Single: `CALL ALGAMS (X, ALGM, S)`

Double: The double precision function name is `DLGAMS`.

Description

The function `ALGAMS` computes $\ln |\Gamma(x)|$ and the sign of $\Gamma(x)$. See [GAMMA](#) for the definition of $\Gamma(x)$.

The result overflows if $|x|$ is too large. The accuracy is worse than half precision if x is too close to a negative integer.

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>ALGAMS</code> is accurate to less than one-half precision because x is too near a negative integer.

Example

In this example, $\ln |\Gamma(1.85)|$ and the sign of $\Gamma(1.85)$ are computed and printed.

```
      USE ALGAMS_INT
      USE UMACH_INT

      IMPLICIT   NONE
!           Declare variables
      INTEGER   NOUT
      REAL      VALUE, S, X
!           Compute
      X = 1.85
      CALL ALGAMS(X, VALUE, S)
!           Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99998) X, VALUE
99998 FORMAT (' Log Abs(Gamma(', F6.3, ')) = ', F6.3)
      WRITE (NOUT,99999) X, S
99999 FORMAT (' Sign(Gamma(', F6.3, ')) = ', F6.2)
      END
```

Output

```
Log Abs(Gamma( 1.850)) = -0.056
Sign(Gamma( 1.850)) = 1.00
```

GAMI

This function evaluates the incomplete gamma function.

Function Return Value

GAMI — Function value. (Output)

Required Arguments

A — The integrand exponent parameter. (Input)
It must be positive.

X — The upper limit of the integral definition of **GAMI**. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: **GAMI (A, X)**

Specific: The specific interface names are **S_GAMI** and **D_GAMI**.

FORTRAN 77 Interface

Single: **GAMI (A, X)**

Double: The double precision function name is **DGAMI**.

Description

The incomplete gamma function is defined to be

$$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt \quad \text{for } a > 0 \text{ and } x \geq 0$$

The function $\gamma(a, x)$ is defined only for a greater than zero. Although $\gamma(a, x)$ is well defined for $x > -\infty$, this algorithm does not calculate $\gamma(a, x)$ for negative x . For large a and sufficiently large x , $\gamma(a, x)$ may overflow. $\gamma(a, x)$ is bounded by $\Gamma(a)$, and users may find this bound a useful guide in determining legal values of a .

Because logarithmic variables are used, a slight deterioration of two or three digits of accuracy will occur when GAMI is very large or very small.

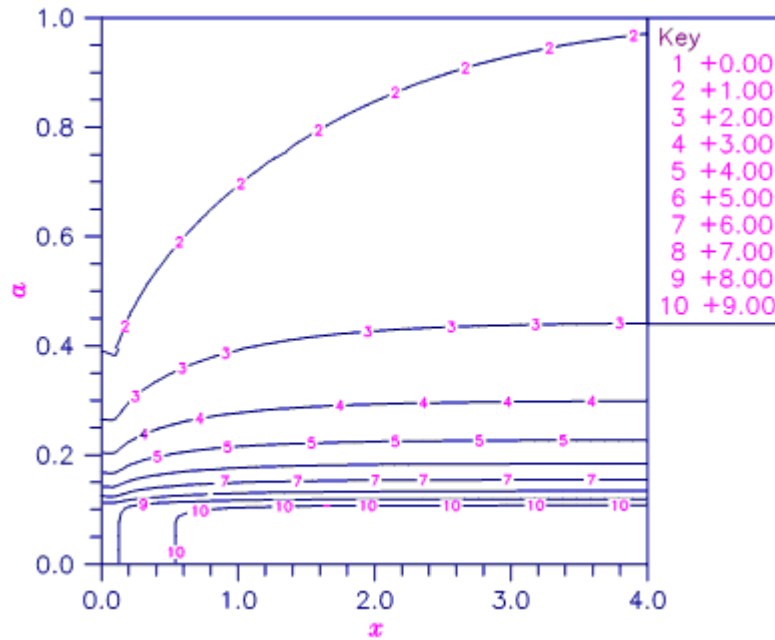


Figure 6, Contour Plot of $\gamma(\alpha, x)$

Example

In this example, $\gamma(2.5, 0.9)$ is computed and printed.

```

USE GAMI_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
REAL A, VALUE, X
!
A = 2.5
X = 0.9
VALUE = GAMI(A, X)                           Compute
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' GAMI(', F6.3, ', ', F6.3, ') = ', F6.4)
END

```

Output

```
GAMI( 2.500, 0.900) = 0.1647
```

GAMIC

Evaluates the complementary incomplete gamma function.

Function Return Value

GAMIC — Function value. (Output)

Required Arguments

A — The integrand exponent parameter as per the remarks. (Input)

X — The upper limit of the integral definition of **GAMIC**. (Input)

If **A** is positive, then **X** must be positive. Otherwise, **X** must be nonnegative.

FORTRAN 90 Interface

Generic: **GAMIC** (**A**, **X**)

Specific: The specific interface names are **S_GAMIC** and **D_GAMIC**.

FORTRAN 77 Interface

Single: **GAMIC** (**A**, **X**)

Double: The double precision function name is **DGAMIC**.

Description

The incomplete gamma function is defined to be

$$\Gamma(a, x) = \int_x^{\infty} t^{a-1} e^{-t} dt$$

The only general restrictions on a are that it must be positive if x is zero; otherwise, it must not be too close to a negative integer such that the accuracy of the result is less than half precision. Furthermore, $\Gamma(a, x)$ must not be so small that it underflows, or so large that it overflows. Although $\Gamma(a, x)$ is well defined for $x > -\infty$ and $a > 0$, this algorithm does not calculate $\Gamma(a, x)$ for negative x .

The function **GAMIC** is based on a code by Gautschi (1979).

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>GAMIC(A, X)</code> is accurate to less than one-half precision because <code>A</code> is too near a negative integer.

Example

In this example, $\Gamma(2.5, 0.9)$ is computed and printed.

```

USE GAMIC_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL A, VALUE, X
!
A = 2.5
X = 0.9
VALUE = GAMIC(A, X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' GAMIC(', F6.3, ', ', F6.3, ') = ', F6.4)
END

```

Output

```
GAMIC( 2.500, 0.900) = 1.1646
```

GAMIT

This function evaluates the Tricomi form of the incomplete gamma function.

Function Return Value

GAMIT — Function value. (Output)

Required Arguments

A — The integrand exponent parameter as per the comments. (Input)

X — The upper limit of the integral definition of **GAMIT**. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: **GAMIT (A, X)**

Specific: The specific interface names are **S_GAMIT** and **D_GAMIT**.

FORTRAN 77 Interface

Single: **GAMIT (A, X)**

Double: The double precision function name is **DGAMIT**.

Description

The Tricomi's incomplete gamma function is defined to be

$$\gamma^*(a, x) = \frac{x^{-a} \gamma(a, x)}{\Gamma(a)} = \frac{x^{-a}}{\Gamma(a)} \int_x^\infty t^{a-1} e^{-t} dt$$

where $\gamma(a, x)$ is the incomplete gamma function. See [GAMI](#) for the definition of $\gamma(a, x)$.

The only general restriction on a is that it must not be too close to a negative integer such that the accuracy of the result is less than half precision. Furthermore, $|\gamma^*(a, x)|$ must not underflow or overflow. Although $\gamma^*(a, x)$ is well defined for $x > -\infty$, this algorithm does not calculate $\gamma^*(a, x)$ for negative x .

A slight deterioration of two or three digits of accuracy will occur when **GAMIT** is very large or very small in absolute value because logarithmic variables are used. Also, if the parameter a is very close to a negative integer (but not quite a negative integer), there is a loss of accuracy which is reported if the result is less than half machine precision.

The function **GAMIT** is based on a code by Gautschi (1979).

Comments

Informational Error

Type	Code	Description
3	2	Result of GAMIT (A, X) is accurate to less than one-half precision because A is too close to a negative integer.

Example

In this example, $\gamma^*(3.2, 2.1)$ is computed and printed.

```

USE GAMIT_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL A, VALUE, X
!
A = 3.2
X = 2.1
VALUE = GAMIT(A, X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' GAMIT(', F6.3, ', ', F6.3, ') = ', F6.4)
END

```

Output

```
GAMIT( 3.200, 2.100) = 0.0284
```

PSI

This function evaluates the derivative of the log gamma function.

Function Return Value

PSI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `PSI (X)`

Specific: The specific interface names are `S_PSI`, `D_PSI`, and `C_PSI`.

FORTRAN 77 Interface

Single: `PSI (X)`

Double: The double precision function name is `DPSI`.

Complex: The complex name is `CPST`.

Description

The `psi` function, also called the digamma function, is defined to be

$$\psi(x) = \frac{d}{dx} \ln \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

See [GAMMA](#) for the definition of $\Gamma(x)$.

The argument x must not be exactly zero or a negative integer, or $\psi(x)$ is undefined. Also, x must not be too close to a negative integer such that the accuracy of the result is less than half precision.

Comments

Informational Error

Type	Code	Description
3	2	Result of $\text{PSI}(x)$ is accurate to less than one-half precision because x is too near a negative integer.

Examples

Example 1

In this example, $\psi(1.915)$ is computed and printed.

```

      USE PSI_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      REAL VALUE, X
!
      X = 1.915
      VALUE = PSI(X)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' PSI(', F6.3, ') = ', F6.3)
      END

```

Output

```
PSI( 1.915) = 0.366
```

Example 2

In this example, $\psi(1.9 + 4.3i)$ is computed and printed.

```

      USE PSI_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (1.9, 4.3)
      VALUE = PSI(Z)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' PSI(', F6.3, ', ', F6.3, ') = (', F6.3, ', ', F6.3, ')')
      END

```

Output

```
PSI( 1.900, 4.300) = ( 1.507, 1.255)
```

PSI1

This function evaluates the second derivative of the log gamma function.

Function Return Value

PSI1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `PSI1 (X)`

Specific: The specific interface names are `S_PSI1` and `D_PSI1`.

Description

The `psi1` function, also called the trigamma function, is defined to be

$$\psi_1(x) = \frac{d^2}{dx^2} \ln \Gamma(x)$$

See [GAMMA](#) for the definition of $\Gamma(x)$.

The argument x must not be exactly zero or a negative integer, or $\psi_1(x)$ is undefined. Also, x must not be too close to a negative integer such that the accuracy of the result is less than half precision.

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>PSI1(x)</code> is accurate to less than one-half precision because x is too near a negative integer.

Example

In this example, $\psi_1(1.915)$ is computed and printed.

```
      USE PSI1_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 1.915
      VALUE = PSI1(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' PSI1(', F6.3, ') = ', F6.3)
      END
```

Output

```
PSI1( 1.915) = 0.681
```

POCH

This function evaluates a generalization of Pochhammer's symbol.

Function Return Value

POCH — Function value. (Output)

The generalized Pochhammer symbol is $\Gamma(a + x)/\Gamma(a)$.

Required Arguments

A — The first argument. (Input)

X — The second, differential argument. (Input)

FORTRAN 90 Interface

Generic: **POCH** (**A**, **X**)

Specific: The specific interface names are **S_POCH** and **D_POCH**.

FORTRAN 77 Interface

Single: **POCH** (**A**, **X**)

Double: The double precision function name is **DPOCH**.

Description

Pochhammer's symbol is $(a)_n = (a)(a - 1)\dots(a - n + 1)$ for n a nonnegative integer. Pochhammer's generalized symbol is defined to be

$$(a)_x = \frac{\Gamma(a + x)}{\Gamma(a)}$$

See [GAMMA](#) for the definition of $\Gamma(x)$.

Note that a straightforward evaluation of Pochhammer's generalized symbol with either gamma or log gamma functions can be especially unreliable when a is large or x is small.

Substantial loss can occur if $a + x$ or a are close to a negative integer unless $|x|$ is sufficiently small. To insure that the result does not overflow or underflow, one can keep the arguments a and $a + x$ well within the range dictated by the gamma function routine `GAMMA` or one can keep $|x|$ small whenever a is large. `POCH` also works for a variety of arguments outside these rough limits, but any more general limits that are also useful are difficult to specify.

Comments

Informational Errors

Type	Code	Description
3	2	Result of <code>POCH(A, X)</code> is accurate to less than one-half precision because the absolute value of the x is too large. Therefore, $A + X$ cannot be evaluated accurately.
3	2	Result of <code>POCH(A, X)</code> is accurate to less than one-half precision because either A or $A + X$ is too close to a negative integer.

For X a nonnegative integer, `POCH(A, X)` is just Pochhammer's symbol.

Example

In this example, $(1.6)_{0.8}$ is computed and printed.

```

USE POCH_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL A, VALUE, X
!
A = 1.6
X = 0.8
VALUE = POCH(A, X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' POCH( ', F6.3, ', ', F6.3, ') = ', F6.4)
END

```

Output

```
POCH( 1.600, 0.800) = 1.3902
```

POCH1

This function evaluates a generalization of Pochhammer's symbol starting from the first order.

Function Return Value

POCH1 — Function value. (Output)

$$\text{POCH1}(A, X) = (\text{POCH}(A, X) - 1)/X.$$

Required Arguments

A — The first argument. (Input)

X — The second, differential argument. (Input)

FORTRAN 90 Interface

Generic: POCH1 (A, X)

Specific: The specific interface names are S_POCH1 and D_POCH1.

FORTRAN 77 Interface

Single: POCH1 (A, X)

Double: The double precision function name is DPOCH1.

Description

Pochhammer's symbol from the first order is defined to be

$$\text{POCH1}(a,x) = \frac{(a)_x - 1}{x} = \left(\frac{\Gamma(a+x)}{\Gamma(a)} - 1 \right) / x$$

where $(a)_x$ is Pochhammer's generalized symbol. See [POCH](#) for the definition of $(a)_x$. It is useful in special situations that require especially accurate values when x is small. This specification is particularly suited for stability when computing expressions such as

$$\left[\frac{\Gamma(a+x)}{\Gamma(a)} - \frac{\Gamma(b+x)}{\Gamma(b)} \right] / x = \text{POCH1}(a,x) - \text{POCH1}(b,x)$$

Note that $\text{POCH1}(a, 0) = \psi(a)$. See [PSI](#) for the definition of $\psi(a)$.

When $|x|$ is so small that substantial cancellation will occur if the straightforward formula is used, we use an expansion due to fields and discussed by Luke (1969).

The ratio $(a)_x = \Gamma(a+x)/\Gamma(a)$ is written by Luke as $(a + (x-1)/2)^x$ times a polynomial in $(a + (x-1)/2)^{-2}$. To maintain significance in POCH1 , we write for positive a ,

$$(a + (x-1)/2)^x = \exp(x \ln(a + (x-1)/2)) = e^q = 1 + q\text{EXPRL}(q)$$

where $\text{EXPRL}(x) = (e^x - 1)/x$. Likewise, the polynomial is written $P = 1 + xP_1(a, x)$. Thus,

$$\text{POCH1}(a, x) = ((a)_x - 1)/x = \text{EXPRL}(q)(q/x + qP_1(a, x)) + P_1(a, x)$$

Substantial significance loss can occur if $a+x$ or a are close to a negative integer even when $|x|$ is very small. To insure that the result does not overflow or underflow, one can keep the arguments a and $a+x$ well within the range dictated by the gamma function routine [GAMMA](#) or one can keep $|x|$ small whenever a is large. POCH also works for a variety of arguments outside these rough limits, but any more general limits that are also useful are difficult to specify.

Example

In this example, $\text{POCH1}(1.6, 0.8)$ is computed and printed.

```

      USE POCH1_INT
      USE UMACH_INT

      IMPLICIT      NONE
!
      INTEGER      NOUT
      REAL         A, VALUE, X
!
      A           = 1.6
      X           = 0.8
      VALUE = POCH1(A, X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' POCH1(', F6.3, ', ', F6.3, ') = ', F6.4)
      END

```

Output

```
POCH1( 1.600, 0.800) = 0.4878
```

BETA

This function evaluates the complete beta function.

Function Return Value

BETA — Function value. (Output)

Required Arguments

- A** — First beta parameter. (Input)
For real arguments, **A** must be positive.
- B** — Second beta parameter. (Input)
For real arguments, **B** must be positive.

FORTRAN 90 Interface

- Generic: **BETA (A, B)**
Specific: The specific interface names are **S_BETA**, **D_BETA**, and **C_BETA**.

FORTRAN 77 Interface

- Single: **BETA (A, B)**
Double: The double precision function name is **DBETA**.
Complex: The complex name is **CBETA**.

Description

The beta function is defined to be

$$\beta(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} = \int_0^1 t^{a-1}(1-t)^{b-1} dt$$

See [GAMMA](#) for the definition of $\Gamma(x)$.

For real arguments the function **BETA** requires that both arguments be positive. In addition, the arguments must not be so large that the result underflows.

For complex arguments, the arguments a and $a + b$ must not be close to negative integers. The arguments should not be so large (near the real axis) that the result underflows. Also, $a + b$ should not be so far from the real axis that the result overflows.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because A and/or B is too large.

Examples

Example 1

In this example, $\beta(2.2, 3.7)$ is computed and printed.

```

      USE BETA_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      REAL A, VALUE, X
!
      A = 2.2
      X = 3.7
      VALUE = BETA(A, X)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' BETA(', F6.3, ', ', F6.3, ') = ', F6.4)
      END

```

Output

```
BETA( 2.200, 3.700) = 0.0454
```

Example 2

In this example, $\beta(1.7 + 2.2i, 3.7 + 0.4i)$ is computed and printed.

```

      USE BETA_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX A, B, VALUE
!
      A = (1.7, 2.2)

```

```
B      = (3.7, 0.4)
VALUE = BETA(A, B)
!
                                Print the results
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) A, B, VALUE
99999 FORMAT (' BETA(', F6.3, ', ', F6.3, '), (', F6.3, ', ', F6.3, '&
            ') = (', F6.3, ', ', F6.3, ')')
END
```

Output

```
BETA(( 1.700, 2.200), ( 3.700, 0.400)) = (-0.033,-0.017)
```


ALBETA

This function evaluates the natural logarithm of the complete beta function for positive arguments.

Function Return Value

ALBETA — Function value. (Output)

ALBETA returns $\ln \beta(A, B) = \ln(\Gamma(A) \Gamma(B) / \Gamma(A + B))$.

Required Arguments

A — The first argument of the **BETA** function. (Input)
For real arguments, **A** must be greater than zero.

B — The second argument of the **BETA** function. (Input)
For real arguments, **B** must be greater than zero.

FORTRAN 90 Interface

Generic: **ALBETA** (A, B)

Specific: The specific interface names are **S_ALBETA**, **D_ALBETA**, and **C_ALBETA**.

FORTRAN 77 Interface

Single: **ALBETA** (A, B)

Double: The double precision function name is **DLBETA**.

Complex: The complex name is **CLBETA**.

Description

ALBETA computes $\ln \beta(a, b) = \ln \beta(b, a)$. See **BETA** for the definition of $\beta(a, b)$.

For real arguments, the function **ALBETA** is defined for $a > 0$ and $b > 0$. It returns accurate results even when a or b is very small. It can overflow for very large arguments; this error condition is not detected except by the computer hardware.

For complex arguments, the arguments a , b and $a + b$ must not be close to negative integers (even though some combinations ought to be allowed). The arguments should not be so large that the logarithm of the gamma function overflows (presumably an improbable condition).

Comments

Note that $\ln \beta(A, B) = \ln \beta(B, A)$.

Examples

Example 1

In this example, $\ln \beta(2.2, 3.7)$ is computed and printed.

```

USE ALBETA_INT
USE UMACH_INT

!      IMPLICIT      NONE                Declare variables
      INTEGER      NOUT
      REAL         A, VALUE, X
!
      A           = 2.2
      X           = 3.7
      VALUE = ALBETA(A, X)
!
      CALL UMACH (2, NOUT)                Print the results
      WRITE (NOUT,99999) A, X, VALUE
99999 FORMAT (' ALBETA(', F6.3, ', ', F6.3, ') = ', F8.4)
      END

```

Output

```
ALBETA( 2.200, 3.700) = -3.0928
```

Example 2

In this example, $\ln \beta(1.7 + 2.2i, 3.7 + 0.4i)$ is computed and printed.

```

USE ALBETA_INT
USE UMACH_INT

!      IMPLICIT      NONE                Declare variables
      INTEGER      NOUT
      COMPLEX      A, B, VALUE
!
      A           = (1.7, 2.2)
      B           = (3.7, 0.4)
      VALUE = ALBETA(A, B)
!
      CALL UMACH (2, NOUT)                Print the results
      WRITE (NOUT,99999) A, B, VALUE

```

```
99999 FORMAT (' ALBETA(', F6.3, ', ', F6.3, ')', (' ', F6.3, ', ', F6.3, '&
              ') = (', F6.3, ', ', F6.3, ')')
END
```

Output

```
ALBETA(( 1.700, 2.200), ( 3.700, 0.400)) = (-3.280,-2.659)
```

BETAI

This function evaluates the incomplete beta function ratio.

Function Return Value

BETAI — Probability that a random variable from a beta distribution having parameters **PIN** and **QIN** will be less than or equal to **X**. (Output)

Required Arguments

X — Upper limit of integration. (Input)

X must be in the interval (0.0, 1.0) inclusive.

PIN — First beta distribution parameter. (Input)

PIN must be positive.

QIN — Second beta distribution parameter. (Input)

QIN must be positive.

FORTRAN 90 Interface

Generic: **BETAI (X, PIN, QIN)**

Specific: The specific interface names are **S_BETAI** and **D_BETAI**.

FORTRAN 77 Interface

Single: **BETAI (X, PIN, QIN)**

Double: The double precision function name is **DBETAI**.

Description

The incomplete beta function is defined to be

$$I_x(p,q) = \frac{\beta_x(p,q)}{\beta(p,q)} = \frac{1}{\beta(p,q)} \int_0^x t^{p-1} (1-t)^{q-1} dt$$

for $0 \leq x \leq 1$, $p > 0$, $q > 0$

See [BETA](#) for the definition of $\beta(p, q)$.

The parameters p and q must both be greater than zero. The argument x must lie in the range 0 to 1. The incomplete beta function can underflow for sufficiently small x and large p ; however, this underflow is not reported as an error. Instead, the value zero is returned as the function value.

The function **BETAI** is based on the work of Bosten and Battiste (1974).

Example

In this example, $I_{0.61}(2.2, 3.7)$ is computed and printed.

```

      USE BETAI_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL PIN, QIN, VALUE, X
!
      X = 0.61
      PIN = 2.2
      QIN = 3.7
      VALUE = BETAI(X, PIN, QIN)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, PIN, QIN, VALUE
99999 FORMAT (' BETAI(', F6.3, ', ', F6.3, ', ', F6.3, ') = ', F6.4)
      END

```

Output

```
BETAI( 0.610, 2.200, 3.700) = 0.8822
```

Error Function and Related Functions

Routines

5.1 Error Functions

Evaluates the error function, $\text{erf } x$	ERF	125
Evaluates the complementary error function, $\text{erfc } x$	ERFC	127
Evaluates the scaled complementary error function, $\exp(x^2) \text{erfc}(x)$. . .	ERFCE	130
Evaluates a scaled function related to erfc , $\exp(-z^2) \text{erfc}(-iz)$	CERFE	132
Evaluates the inverse error function, $\text{erf}^{-1} x$	ERFI	134
Evaluates the inverse complementary error function, $\text{erfc}^{-1} x$	ERFCI	137
Evaluates Dawson's function	DAWS	140

5.2 Fresnel Integrals

Evaluates the cosine Fresnel integral, $C(x)$	FRESC	142
Evaluates the sine Fresnel integral, $S(x)$	FRESS	144

Usage Notes

The error function is

$$e(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

The complementary error function is $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$. Dawson's function is

$$e^{-x^2} \int_0^x e^{t^2} dt$$

The Fresnel integrals are

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2} t^2\right) dt$$

and

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2} t^2\right) dt$$

They are related to the error function by

$$C(z) + iS(z) = \frac{1+i}{2} \operatorname{erf}\left(\frac{\sqrt{\pi}}{2} (1-i)z\right)$$

ERF

This function evaluates the error function.

Function Return Value

ERF — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **ERF (X)**

Specific: The specific interface names are **S_ERF** and **D_ERF**.

FORTRAN 77 Interface

Single: **ERF (X)**

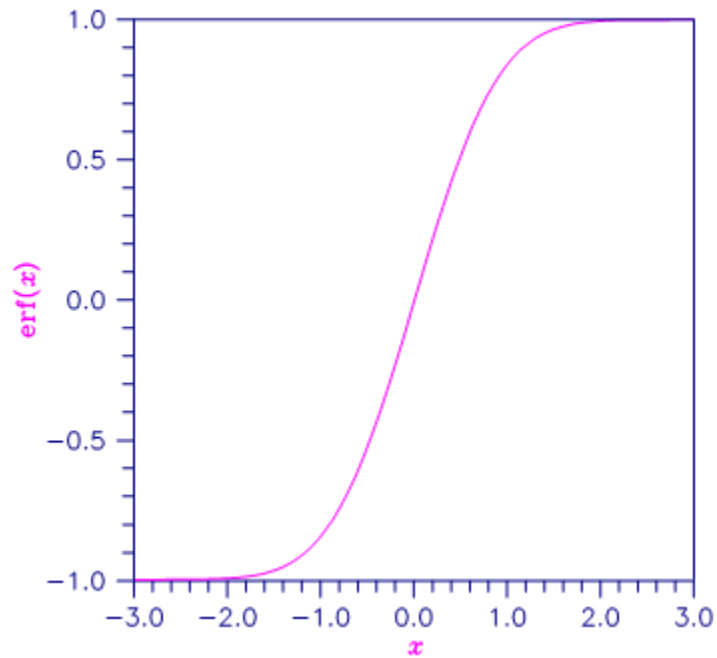
Double: The double precision function name is **DERF**.

Description

The error function, $\text{erf}(x)$, is defined to be

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

All values of x are legal.

Figure 7, Plot of $\text{erf}(x)$

Example

In this example, $\text{erf}(1.0)$ is computed and printed.

```
USE ERF_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 1.0
VALUE = ERF(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ERF(', F6.3, ') = ', F6.3)
END
```

Output

```
ERF( 1.000) = 0.843
```

ERFC

This function evaluates the complementary error function.

Function Return Value

ERFC — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **ERFC (X)**

Specific: The specific interface names are **S_ERFC** and **D_ERFC**.

FORTRAN 77 Interface

Single: **ERFC (X)**

Double: The double precision function name is **DERFC**.

Description

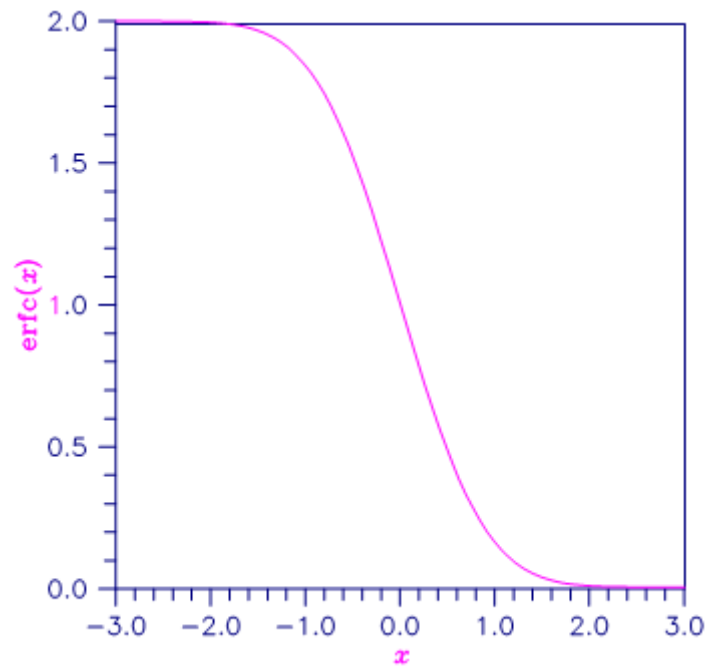
The complementary error function, $\text{erfc}(x)$, is defined to be

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

The argument x must not be so large that the result underflows. Approximately, x should be less than

$$\left[-\ln(\sqrt{\pi} s) \right]^{1/2}$$

where $s = \text{AMACH}(1)$ (see the [Reference Material](#) section of this manual) is the smallest representable positive floating-point number.

Figure 8, Plot of $\text{erfc}(x)$

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too large.

Example

In this example, $\text{erfc}(1.0)$ is computed and printed.

```

USE ERFC_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 1.0
VALUE = ERFC(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE

```

```
99999 FORMAT ( ' ERFC( ', F6.3, ' ) = ', F6.3 )  
END
```

Output

```
ERFC( 1.000 ) = 0.157
```

ERFCE

This function evaluates the exponentially scaled complementary error function.

Function Return Value

ERFCE — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **ERFCE** (**X**)

Specific: The specific interface names are **S_ERFCE** and **D_ERFCE**.

FORTRAN 77 Interface

Single: **ERFCE** (**X**)

Double: The double precision function name is **DERFCE**.

Description

The function **ERFCE**(**X**) computes

$$e^{x^2} \operatorname{erfc}(x)$$

where $\operatorname{erfc}(x)$ is the complementary error function. See [ERFC](#) for its definition.

To prevent the answer from underflowing, x must be greater than

$$x_{\min} \simeq -\sqrt{\ln(b/2)}$$

where $b = \mathbf{AMACH}(2)$ is the largest representable floating-point number.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too large.

Example

In this example, `ERFCE(1.0) = e1.0 erfc(1.0)` is computed and printed.

```
      USE ERFCE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 1.0
      VALUE = ERFCE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ERFCE(', F6.3, ') = ', F6.3)
      END
```

Output

```
ERFCE( 1.000) = 0.428
```

CERFE

This function evaluates a scaled function related to `ERFC`.

Function Return Value

CERFE — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `CERFE (Z)`

Specific: The specific interface names are `C_CERFE` and `Z_CERFE`.

FORTRAN 77 Interface

Complex: `CERFE (Z)`

Double complex: The double complex function name is `ZERFE`.

Description

Function `CERFE` is defined to be

$$e^{-z^2} \operatorname{erfc}(-iz) = -ie^{-z^2} \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{t^2} dt$$

Let $b = \text{AMACH}(2)$ be the largest floating-point number. The argument z must satisfy

$$|z| \leq \sqrt{b}$$

or else the value returned is zero. If the argument z does not satisfy $(\Im z)^2 - (\Re z)^2 \leq \log b$, then b is returned. All other arguments are legal (Gautschi 1969, 1970).

Example

In this example, `CERFE(2.5 + 2.5i)` is computed and printed.

```
      USE CERFE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (2.5, 2.5)
      VALUE = CERFE(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CERFE(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
CERFE( 2.500, 2.500) = ( 0.117, 0.108)
```

ERFI

This function evaluates the inverse error function.

Function Return Value

ERFI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **ERFI (X)**

Specific: The specific interface names are **S_ERFI** and **D_ERFI**.

FORTRAN 77 Interface

Single: **ERFI (X)**

Double: The double precision function name is **DERFI**.

Description

Function **ERFI(X)** computes the inverse of the error function $\operatorname{erf} x$, defined in [ERF](#).

The function **ERFI(X)** is defined for $|x| < 1$. If $x_{\max} < |x| < 1$, then the answer will be less accurate than half precision. Very approximately,

$$x_{\max} \approx 1 - \sqrt{\varepsilon / (4\pi)}$$

where $\varepsilon = \mathbf{AMACH}(4)$ is the machine precision.

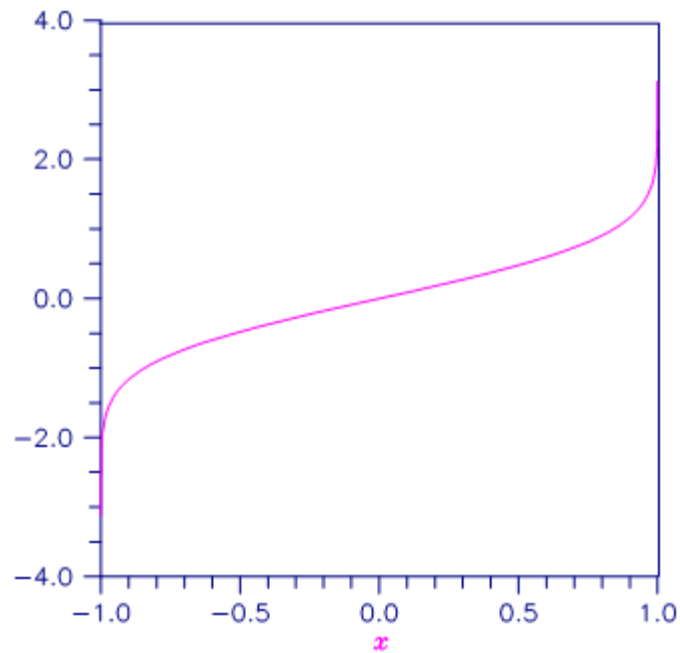


Figure 9, Plot of $\text{erf}^{-1}(x)$

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>ERFI(X)</code> is accurate to less than one-half precision because the absolute value of the argument is too large .

Example

In this example, $\text{erf}^{-1}(\text{erf}(1.0))$ is computed and printed.

```

USE ERFI_INT
USE ERF_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = ERF(1.0)
VALUE = ERFI(X)

```

Declare variables
Compute

```
!                                Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ERFI(', F6.3, ') = ', F6.3)
  END
```

Output

```
ERFI( 0.843) =  1.000
```

ERFCI

This function evaluates the inverse complementary error function.

Function Return Value

ERFCI — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **ERFCI (X)**

Specific: The specific interface names are **S_ERFCI** and **D_ERFCI**.

FORTRAN 77 Interface

Single: **ERFCI (X)**

Double: The double precision function name is **DERFCI**.

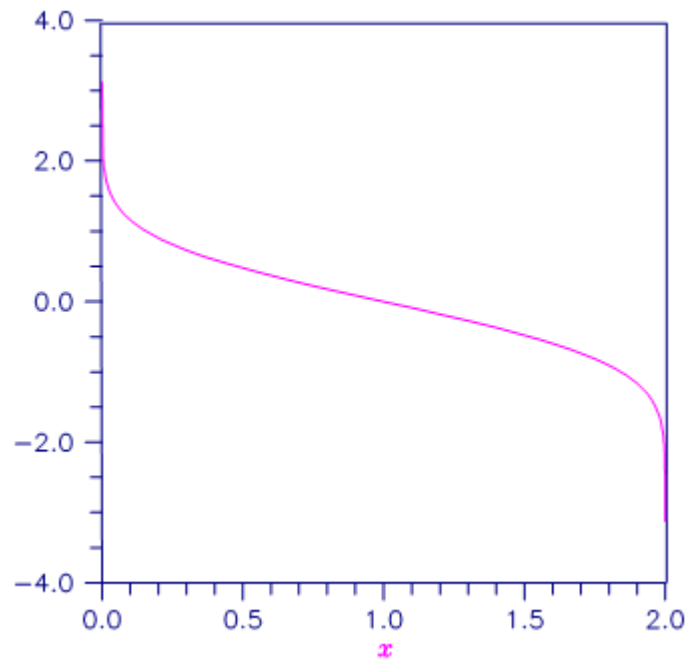
Description

The function **ERFCI(X)** computes the inverse of the complementary error function $\operatorname{erfc} x$, defined in [ERFC](#).

The function **ERFCI(X)** is defined for $0 < x < 2$. If $x_{\max} < x < 2$, then the answer will be less accurate than half precision. Very approximately,

$$x_{\max} \approx 2 - \sqrt{\varepsilon / (4\pi)}$$

Where $\varepsilon = \mathbf{AMACH}(4)$ is the machine precision.

Figure 10, Plot of $\text{erf}^{-1}(x)$

Comments

Informational Error

Type	Code	Description
3	2	Result of <code>ERFCI(X)</code> is accurate to less than one-half precision because the argument is too close to 2.0.

Example

In this example, $\text{erfc}^{-1}(\text{erfc}(1.0))$ is computed and printed.

```

USE ERFCI_INT
USE ERFC_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = ERFC(1.0)
VALUE = ERFCI(X)

```

Declare variables
Compute

```
!                                Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ERFCI(', F6.3, ') = ', F6.3)
  END
```

Output

```
ERFCI( 0.157) =  1.000
```

DAWS

This function evaluates Dawson's function.

Function Return Value

DAWS — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **DAWS** (**X**)

Specific: The specific interface names are **S_DAWS** and **D_DAWS**.

FORTRAN 77 Interface

Single: **DAWS** (**X**)

Double: The double precision function name is **DDAWS**.

Description

Dawson's function is defined to be

$$e^{-x^2} \int_0^x e^{t^2} dt$$

It is closely related to the error function for imaginary arguments.

So that Dawson's function does not underflow, $|x|$ must be less than $1/(2s)$. Here, $s = \mathbf{AMACH}(1)$ is the smallest representable positive floating-point number.

Comments

Informational Error

Type	Code	Description
3	2	The function underflows because the absolute value of x is too large.

The Dawson function is closely related to the error function for imaginary arguments.

Example

In this example, `DAWS(1.0)` is computed and printed.

```
      USE DAWS_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      REAL VALUE, X
!          Compute
      X = 1.0
      VALUE = DAWS(X)
!          Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999  FORMAT (' DAWS(', F6.3, ') = ', F6.3)
      END
```

Output

```
DAWS( 1.000 ) = 0.538
```

FRESC

This function evaluates the cosine Fresnel integral.

Function Return Value

FRESC — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **FRESC (X)**

Specific: The specific interface names are **S_FRESC** and **D_FRESC**.

FORTRAN 77 Interface

Single: **FRESC (X)**

Double: The double precision function name is **DFRESC**.

Description

The cosine Fresnel integral is defined to be

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right) dt$$

All values of x are legal.

Example

In this example, $C(1.75)$ is computed and printed.

```
USE FRESC_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  REAL         VALUE, X
!                               Compute
  X           = 1.75
  VALUE = FRESC(X)
!                               Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' FRESC(', F6.3, ') = ', F6.3)
  END
```

Output

```
FRESC( 1.750) = 0.322
```

FRESS

This function evaluates the sine Fresnel integral.

Function Return Value

FRESS — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **FRESS (X)**

Specific: The specific interface names are **S_FRESS** and **D_FRESS**.

FORTRAN 77 Interface

Single: **FRESS (X)**

Double: The double precision function name is **DFRESS**.

Description

The sine Fresnel integral is defined to be

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right) dt$$

All values of x are legal.

Example

In this example, $S(1.75)$ is computed and printed.

```
USE FRESS_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  REAL         VALUE, X
!                               Compute
  X           = 1.75
  VALUE = FRESS(X)
!                               Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' FRESS(' , F6.3, ') = ' , F6.3)
  END
```

Output

```
FRESS( 1.750) =  0.499
```

Bessel Functions

Routines

6.1	Bessel Functions of Order 0 and 1		
	Evaluates $J_0(x)$	BSJ0	148
	Evaluates $J_1(x)$	BSJ1	150
	Evaluates $Y_0(x)$	BSY0	152
	Evaluates $Y_1(x)$	BSY1	154
	Evaluates $I_0(x)$	BSI0	156
	Evaluates $I_1(x)$	BSI1	158
	Evaluates $K_0(x)$	BSK0	160
	Evaluates $K_1(x)$	BSK1	163
	Evaluates $e^{- x }I_0(x)$	BSIOE	165
	Evaluates $e^{- x }I_1(x)$	BSI1E	167
	Evaluates $e^xK_0(x)$	BSK0E	169
	Evaluates $e^xK_1(x)$	BSK1E	171
6.2	Series of Bessel Functions, Integer Order		
	Evaluates $J_k(x), k = 0, \dots, n-1$	BSJNS	173
	Evaluates $I_k(x), k = 0, \dots, n-1$	BSINS	176
6.3	Series of Bessel Functions, Real Order and Argument		
	Evaluates $J_{\nu+k}(x), k = 0, \dots, n-1$	BSJS	179
	Evaluates $Y_{\nu+k}(x), k = 0, \dots, n-1$	BSYS	181
	Evaluates $I_{\nu+k}(x), k = 0, \dots, n-1$	BSIS	183
	Evaluates $e^{-x}I_{\nu+k}(x), k = 0, \dots, n-1$	BSIES	185
	Evaluates $K_{\nu+k}(x), k = 0, \dots, n-1$	BSKS	187
	Evaluates $e^xK_{\nu+k}(x), k = 0, \dots, n-1$	BSKES	189
6.4	Series of Bessel Functions, Real Order and Complex Argument		
	Evaluates $J_{\nu+k}(z), k = 0, \dots, n-1$	CBJS	191
	Evaluates $Y_{\nu+k}(z), k = 0, \dots, n-1$	CBYS	194
	Evaluates $I_{\nu+k}(z), k = 0, \dots, n-1$	CBIS	197
	Evaluates $K_{\nu+k}(z), k = 0, \dots, n-1$	CBKS	199

Usage Notes

The following table lists the Bessel function routines by argument and order type:

Function	Real Argument				Complex Argument	
	Order				Order	
	0	1	Integer	Real	Integer	Real
$J_\nu(x)$	BSJ0	BSJ1	BSJNS	BSJS	BSJNS	CBJS
$Y_\nu(x)$	BSY0	BSY1		BSYS		CBYS
$I_\nu(x)$	BSI0	BSI1	BSINS	BSIS	BSINS	CBIS
$e^{- x }I_\nu(x)$	BSI0E	BSI1E		BSIES		
$K_\nu(x)$	BSK0	BSK1		BSKS		CBKS
$e^{- x }K_\nu(x)$	BSK0E	BSK1E		BSKES		

BSJ0

This function evaluates the Bessel function of the first kind of order zero.

Function Return Value

BSJ0 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: BSJ0 (**x**)

Specific: The specific interface names are S_BSJ0 and D_BSJ0.

FORTRAN 77 Interface

Single: BSJ0 (**x**)

Double: The double precision function name is DBSJ0.

Description

The Bessel function $J_0(x)$ is defined to be

$$J_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta) d\theta$$

To prevent the answer from being less accurate than half precision, $|x|$ should be smaller than

$$1 / \sqrt{\epsilon}$$

For the result to have any precision at all, $|x|$ must be less than $1/\epsilon$. Here, ϵ is the machine precision, $\epsilon = \text{AMACH}(4)$.

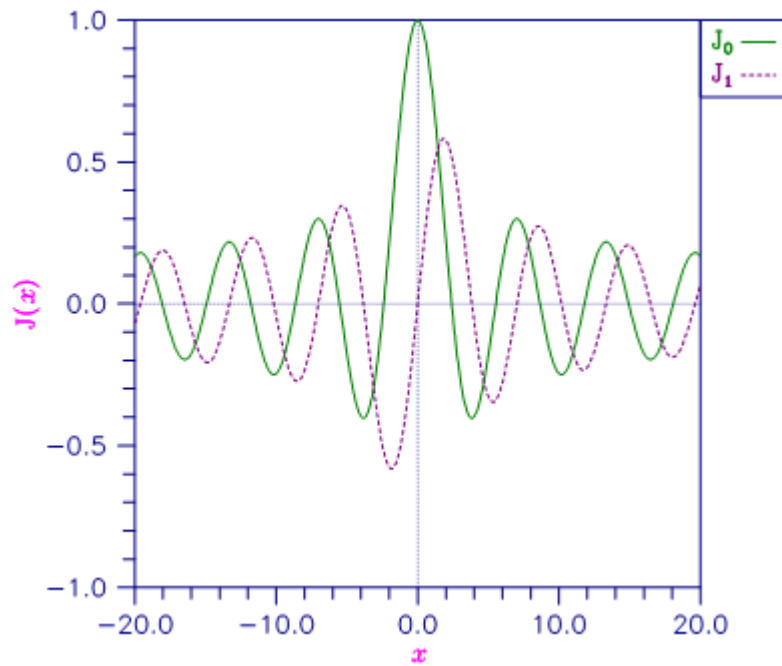


Figure 11, Plot of $J_0(x)$ and $J_1(x)$

Example

In this example, $J_0(3.0)$ is computed and printed.

```

      USE BSJO_INT
      USE UMACH_INT

      IMPLICIT NONE
      !
      INTEGER NOUT
      REAL VALUE, X
      !
      X = 3.0
      VALUE = BSJO(X)
      !
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
      99999 FORMAT (' BSJO(', F6.3, ') = ', F6.3)
      END
  
```

Output

```
BSJO( 3.000) = -0.260
```


BSJ1

This function evaluates the Bessel function of the first kind of order one.

Function Return Value

BSJ1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: `BSJ1 (X)`

Specific: The specific interface names are `S_BSJ1` and `D_BSJ1`.

FORTRAN 77 Interface

Single: `BSJ1 (X)`

Double: The double precision function name is `DBSJ1`.

Description

The Bessel function $J_1(x)$ is defined to be

$$J_1(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin \theta - \theta) d\theta$$

The argument x must be zero or larger in absolute value than $2s$ to prevent $J_1(x)$ from underflowing. Also, $|x|$ should be smaller than

$$1 / \sqrt{\epsilon}$$

to prevent the answer from being less accurate than half precision. $|x|$ must be less than $1/\epsilon$ for the result to have any precision at all. Here, ϵ is the machine precision, $\epsilon = \mathbf{AMACH}(4)$, and $s = \mathbf{AMACH}(1)$ is the smallest representable positive floating-point number.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because the absolute value of x is too small.

Example

In this example, $J_1(2.5)$ is computed and printed.

```

      USE BSJ1_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      REAL VALUE, X
!          Compute
      X = 2.5
      VALUE = BSJ1(X)
!          Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSJ1(', F6.3, ') = ', F6.3)
      END

```

Output

```
BSJ1( 2.500) = 0.497
```

BSY0

This function evaluates the Bessel function of the second kind of order zero.

Function Return Value

BSY0 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: BSY0 (x)

Specific: The specific interface names are S_BSY0 and D_BSY0.

FORTRAN 77 Interface

Single: BSY0 (x)

Double: The double precision function name is DBSY0.

Description

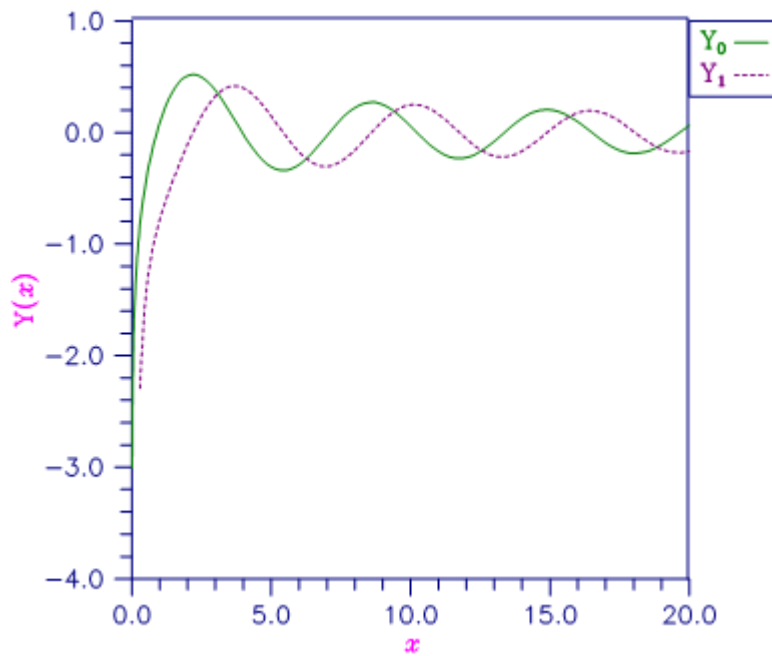
The Bessel function $Y_0(x)$ is defined to be

$$Y_0(x) = \frac{1}{\pi} \int_0^\pi \sin(x \sin \theta) d\theta - \frac{2}{\pi} \int_0^\infty e^{-x \sinh t} dt$$

To prevent the answer from being less accurate than half precision, x should be smaller than

$$1 / \sqrt{\epsilon}$$

For the result to have any precision at all, $|x|$ must be less than $1/\epsilon$. Here, ϵ is the machine precision, $\epsilon = \text{AMACH}(4)$.

Figure 12, Plot of $Y_0(x)$ and $Y_1(x)$

Example

In this example, $Y_0(3.0)$ is computed and printed.

```

      USE BSY0_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X

!      Compute
      X = 3.0
      VALUE = BSY0(X)

!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSY0(', F6.3, ') = ', F6.3)
      END

```

Output

```
BSY0( 3.000) = 0.377
```

BSY1

This function evaluates the Bessel function of the second kind of order one.

Function Return Value

BSY1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSY1 (X)**

Specific: The specific interface names are **S_BSY1** and **D_BSY1**.

FORTRAN 77 Interface

Single: **BSY1 (X)**

Double: The double precision function name is **DBSY1**.

Description

The Bessel function $Y_1(x)$ is defined to be

$$Y_1(x) = -\frac{1}{\pi} \int_0^\pi \sin(\theta - x \sin \theta) d\theta - \frac{1}{\pi} \int_0^\infty \{e^t - e^{-t}\} e^{-x \sinh t} dt$$

$Y_1(x)$ is defined for $x > 0$. To prevent the answer from being less accurate than half precision, x should be smaller than

$$1 / \sqrt{\epsilon}$$

For the result to have any precision at all, $|x|$ must be less than $1/\epsilon$. Here, ϵ is the machine precision, $\epsilon = \text{AMACH}(4)$.

Example

In this example, $Y_1(3.0)$ is computed and printed.

```
      USE BSY1_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 3.0
      VALUE = BSY1(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSY1(', F6.3, ') = ', F6.3)
      END
```

Output

```
BSY1( 3.000) = 0.325
```

BSI0

This function evaluates the modified Bessel function of the first kind of order zero.

Function Return Value

BSI0 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: BSI0 (**x**)

Specific: The specific interface names are S_BSI0 and D_BSI0.

FORTRAN 77 Interface

Single: BSI0 (**x**)

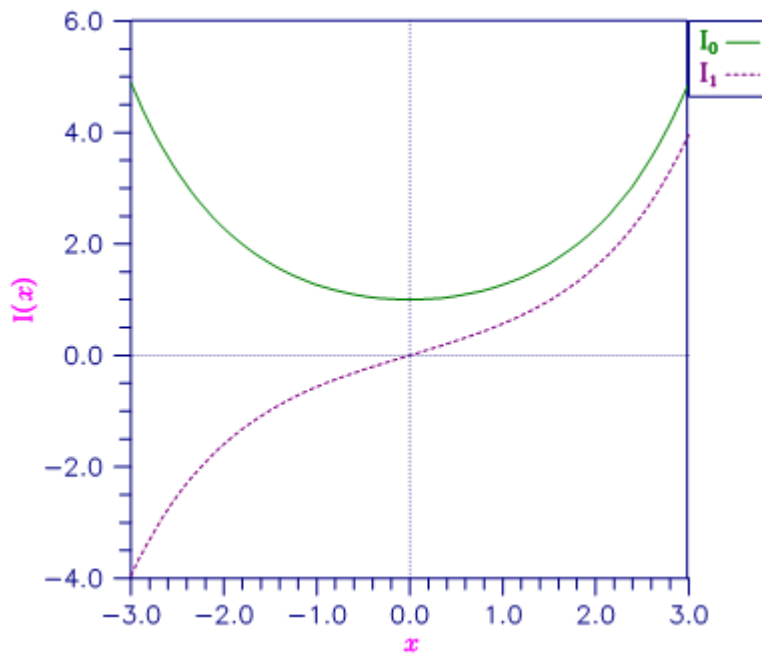
Double: The double precision function name is DBSI0.

Description

The Bessel function $I_0(x)$ is defined to be

$$I_0(x) = \frac{1}{\pi} \int_0^{\pi} \cosh(x \cos \theta) d\theta$$

The absolute value of the argument x must not be so large that $e^{|x|}$ overflows.

Figure 13, Plot of $I_0(x)$ and $I_1(x)$

Example

In this example, $I_0(4.5)$ is computed and printed.

```

      USE BSI0_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X

!      Compute
      X = 4.5
      VALUE = BSI0(X)

!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSI0(', F6.3, ') = ', F6.3)
      END

```

Output

```
BSI0( 4.500) = 17.481
```

BSI1

This function evaluates the modified Bessel function of the first kind of order one.

Function Return Value

BSI1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSI1 (X)**

Specific: The specific interface names are **S_BSI1** and **D_BSI1**.

FORTRAN 77 Interface

Single: **BSI1 (X)**

Double: The double precision function name is **DBSI1**.

Description

The Bessel function $I_1(x)$ is defined to be

$$I_1(x) = \frac{1}{\pi} \int_0^{\pi} e^{x \cos \theta} \cos \theta d\theta$$

The argument should not be so close to zero that $I_1(x) \approx x/2$ underflows, nor so large in absolute value that $e^{|x|}$ and, therefore, $I_1(x)$ overflows.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because the absolute value of x is too small.

Example

In this example, $I_1(4.5)$ is computed and printed.

```
      USE BSI1_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 4.5
      VALUE = BSI1(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSI1(', F6.3, ') = ', F6.3)
      END
```

Output

```
BSI1( 4.500) = 15.389
```

BSK0

This function evaluates the modified Bessel function of the second kind of order zero.

Function Return Value

BSK0 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

Fortran 90 Interface

Generic: BSK0 (x)

Specific: The specific interface names are S_BSK0 and D_BSK0.

FORTRAN 77 Interface

Single: BSK0 (x)

Double: The double precision function name is DBSK0.

Description

The Bessel function $K_0(x)$ is defined to be

$$K_0(x) = \int_0^{\infty} \cos(x \sinh t) dt$$

The argument must be larger than zero, but not so large that the result, approximately equal to

$$\sqrt{\pi / (2x)} e^{-x}$$

underflows.

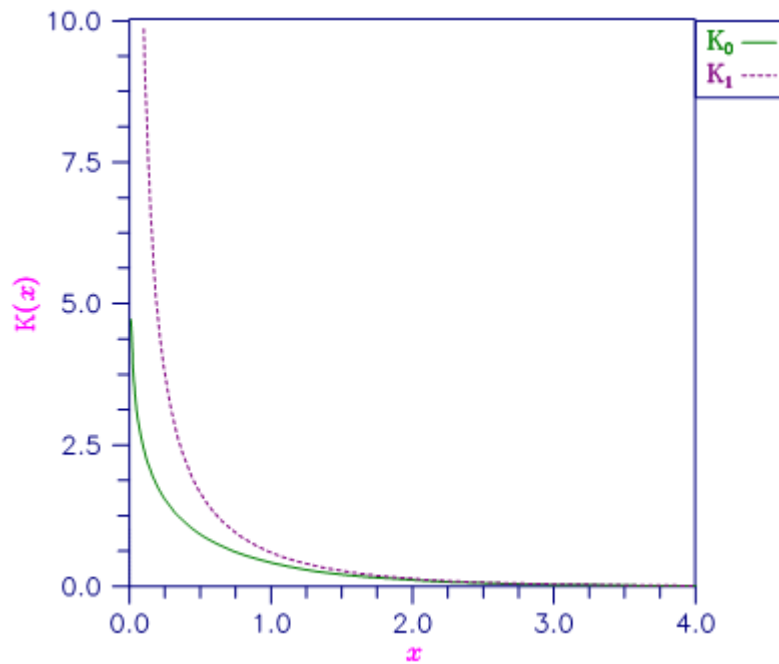


Figure 14, Plot of $K_0(x)$ and $K_1(x)$

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too large.

Example

In this example, $K_0(0.5)$ is computed and printed.

```

USE BSK0_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 0.5
VALUE = BSK0(X)
!
Print the results

```

```
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSK0(', F6.3, ') = ', F6.3)
END
```

Output

```
BSK0( 0.500 ) = 0.924
```

BSK1

This function evaluates the modified Bessel function of the second kind of order one.

Function Return Value

BSK1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSK1 (X)**

Specific: The specific interface names are **S_BSK1** and **D_BSK1**.

FORTRAN 77 Interface

Single: **BSK1 (X)**

Double: The double precision function name is **DBSK1**.

Description

The Bessel function $K_1(x)$ is defined to be

$$K_1(x) = \int_0^{\infty} \sin(x \sinh t) \sinh t \, dt$$

The argument x must be large enough ($> \max(1/b, s)$) that $K_1(x)$ does not overflow, and x must be small enough that the approximate answer,

$$\sqrt{\pi / (2x)} e^{-x}$$

does not underflow. Here, s is the smallest representable positive floating-point number, $s = \mathbf{AMACH}(1)$, and $b = \mathbf{AMACH}(2)$ is the largest representable floating-point number.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is too large.

Example

In this example, $K_1(0.5)$ is computed and printed.

```
USE BSK1_INT
USE UMACH_INT

IMPLICIT NONE
!
INTEGER NOUT
REAL VALUE, X
!
X = 0.5
VALUE = BSK1(X)
!
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSK1(', F6.3, ') = ', F6.3)
END
```

Output

```
BSK1( 0.500) = 1.656
```

BSIOE

This function evaluates the exponentially scaled modified Bessel function of the first kind of order zero.

Function Return Value

BSIOE — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSIOE (X)**

Specific: The specific interface names are **S_BSIOE** and **D_BSIOE**.

FORTRAN 77 Interface

Single: **BSIOE (X)**

Double: The double precision function name is **DBSIOE**.

Description

Function **BSIOE** computes $e^{-|x|} I_0(x)$. For the definition of the Bessel function $I_0(x)$, see [BSI0](#).

Example

In this example, **BSIOE(4.5)** is computed and printed.

```
      USE BSIOE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 4.5
      VALUE = BSIOE(X)
```



```
!                                     Print the results
   CALL UMACH (2, NOUT)
   WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSIOE(', F6.3, ') = ', F6.3)
   END
```

Output

```
BSIOE( 4.500) = 0.194
```

BSI1E

This function evaluates the exponentially scaled modified Bessel function of the first kind of order one.

Function Return Value

BSI1E — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSI1E** (**X**)

Specific: The specific interface names are **S_BSI1E** and **D_BSI1E**.

FORTRAN 77 Interface

Single: **BSI1E** (**X**)

Double: The double precision function name is **DBSI1E**.

Description

Function **BSI1E** computes $e^{-|x|} I_1(x)$. For the definition of the Bessel function $I_1(x)$, see [BSI1](#). The function **BSI1E** underflows if $|x|/2$ underflows.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because the absolute value of x is too small.

Example

In this example, BSI1E(4.5) is computed and printed.

```
      USE BSI1E_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 4.5
      VALUE = BSI1E(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSI1E(', F6.3, ') = ', F6.3)
      END
```

Output

```
BSI1E( 4.500) = 0.171
```

BSK0E

This function evaluates the exponentially scaled modified Bessel function of the second kind of order zero.

Function Return Value

BSK0E — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BSK0E (X)**

Specific: The specific interface names are **S_BSK0E** and **D_BSK0E**.

FORTRAN 77 Interface

Single: **BSK0E (X)**

Double: The double precision function name is **DBSK0E**.

Description

Function **BSK0E** computes $e^{-x}K_0(x)$. For the definition of the Bessel function $K_0(x)$, see [BSK0](#). The argument must be greater than zero for the result to be defined.

Example

In this example, **BSK0E(0.5)** is computed and printed.

```
      USE BSK0E_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      REAL VALUE, X
!          Compute
```

```
X      = 0.5
VALUE = BSK0E(X)
!
                                Print the results
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSK0E(', F6.3, ') = ', F6.3)
END
```

Output

```
BSK0E( 0.500) = 1.524
```



```
X      = 0.5
VALUE = BSK1E(X)
!
                                Print the results
CALL UMACH (2, NOUT)
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BSK1E(', F6.3, ') = ', F6.3)
END
```

Output

```
BSK1E( 0.500) = 2.731
```

BSJNS

Evaluates a sequence of Bessel functions of the first kind with integer order and real or complex arguments.

Required Arguments

- X** — Argument for which the sequence of Bessel functions is to be evaluated. (Input)
The absolute value of real arguments must be less than 10^4 .
The absolute value of complex arguments must be less than 10^4 .
- N** — Number of elements in the sequence. (Input)
It must be a positive integer.
- BS** — Vector of length **N** containing the values of the function through the series. (Output)
BS(I) contains the value of the Bessel function of order **I** - 1 at *x* for **I** = 1 to **N**.

FORTRAN 90 Interface

- Generic: `CALL BSJNS (X, N, BS)`
- Specific: The specific interface names are `S_BSJNS`, `D_BSJNS`, `C_BSJNS`, and `Z_BSJNS`.

FORTRAN 77 Interface

- Single: `CALL BSJNS (X, N, BS)`
- Double: The double precision name is `DBSJNS`.
- Complex: The complex name is `CBJNS`.
- Double Complex: The double complex name is `DCBJNS`.

Description

The complex Bessel function $J_n(z)$ is defined to be

$$J_n(z) = \frac{1}{\pi} \int_0^\pi \cos(z \sin \theta - n\theta) d\theta$$

This code is based on the work of Sookne (1973a) and Olver and Sookne (1972). It uses backward recursion with strict error control.

Examples

Example 1

In this example, $J_n(10.0)$, $n = 0, \dots, 9$ is computed and printed.

```

      USE BSJNS_INT
      USE UMACH_INT

      IMPLICIT      NONE
!
!               Declare variables
      INTEGER      N
      PARAMETER    (N=10)
!
      INTEGER      K, NOUT
      REAL         BS(N), X
!
!               Compute
      X = 10.0
      CALL BSJNS (X, N, BS)
!
!               Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) K-1, X, BS(K)
10    CONTINUE
99999 FORMAT (' J sub ', I2, ' (', F6.3, ') = ', F6.3)
      END

```

Output

```

J sub 0 (10.000) = -0.246
J sub 1 (10.000) =  0.043
J sub 2 (10.000) =  0.255
J sub 3 (10.000) =  0.058
J sub 4 (10.000) = -0.220
J sub 5 (10.000) = -0.234
J sub 6 (10.000) = -0.014
J sub 7 (10.000) =  0.217
J sub 8 (10.000) =  0.318
J sub 9 (10.000) =  0.292

```

Example 2

In this example, $J_n(10 + 10i)$, $n = 0, \dots, 10$ is computed and printed.

```

      USE BSJNS_INT
      USE UMACH_INT

      IMPLICIT      NONE
!
!               Declare variables
      INTEGER      N
      PARAMETER    (N=11)
!
      INTEGER      K, NOUT
      COMPLEX      CBS(N), Z
!
!               Compute
      Z = (10.0, 10.0)
      CALL BSJNS (Z, N, CBS)
!
!               Print the results
      CALL UMACH (2, NOUT)

```

```
DO 10 K=1, N
  WRITE (NOUT,99999) K-1, Z, CBS(K)
10 CONTINUE
99999 FORMAT (' J sub ', I2, ' ((' , F6.3, ', ', F6.3, &
            ')) = ( ', F9.3, ', ', F9.3, ' )' )
END
```

Output

```
J sub 0 ((10.000,10.000)) = (-2314.975, 411.563)
J sub 1 ((10.000,10.000)) = (-460.681,-2246.627)
J sub 2 ((10.000,10.000)) = ( 2044.245, -590.157)
J sub 3 ((10.000,10.000)) = ( 751.498, 1719.746)
J sub 4 ((10.000,10.000)) = (-1302.871, 880.632)
J sub 5 ((10.000,10.000)) = (-920.394, -846.345)
J sub 6 ((10.000,10.000)) = ( 419.501, -843.607)
J sub 7 ((10.000,10.000)) = ( 665.930, 88.480)
J sub 8 ((10.000,10.000)) = ( 108.586, 439.392)
J sub 9 ((10.000,10.000)) = (-227.548, 176.165)
J sub 10 ((10.000,10.000)) = (-154.831, -76.050)
```

BSINS

Evaluates a sequence of modified Bessel functions of the first kind with integer order and real or complex arguments.

Required Arguments

- X** — Argument for which the sequence of Bessel functions is to be evaluated. (Input)
 For real argument $\exp(|x|)$ must not overflow. For complex arguments x must be less than 10^4 in absolute value.
- N** — Number of elements in the sequence. (Input)
- BSI** — Vector of length **N** containing the values of the function through the series. (Output)
BSI(I) contains the value of the Bessel function of order $I - 1$ at x for $I = 1$ to **N**.

FORTRAN 90 Interface

- Generic: `CALL BSINS (X, N, BSI)`
 Specific: The specific interface names are `S_BSINS`, `D_BSINS`, `C_BSINS`, and `Z_BSINS`.

FORTRAN 77 Interface

- Single: `CALL BSINS (X, N, BSI)`
 Double: The double precision name is `DBSINS`.
 Complex: The complex name is `CBINS`.
 Double Complex: The double complex name is `DCBINS`.

Description

The complex Bessel function $I_n(z)$ is defined to be

$$I_n(z) = \frac{1}{\pi} \int_0^\pi e^{z \cos \theta} \cos(n\theta) d\theta$$

This code is based on the work of Sookne (1973a) and Olver and Sookne (1972). It uses backward recursion with strict error control.

Examples

Example 1

In this example, $I_n(10.0)$, $n = 0, \dots, 10$ is computed and printed.

```

      USE BSINS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=11)

!           INTEGER K, NOUT
      REAL BSI(N), X

!           Compute
      X = 10.0
      CALL BSINS (X, N, BSI)

!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
          WRITE (NOUT,99999) K-1, X, BSI(K)
10 CONTINUE
99999 FORMAT (' I sub ', I2, ' (', F6.3, ') = ', F9.3)
      END

```

Output

```

I sub 0 (10.000) = 2815.716
I sub 1 (10.000) = 2670.988
I sub 2 (10.000) = 2281.519
I sub 3 (10.000) = 1758.381
I sub 4 (10.000) = 1226.490
I sub 5 (10.000) = 777.188
I sub 6 (10.000) = 449.302
I sub 7 (10.000) = 238.026
I sub 8 (10.000) = 116.066
I sub 9 (10.000) = 52.319
I sub 10 (10.000) = 21.892

```

Example 2

In this example, $I_n(10 + 10i)$, $n = 0, \dots, 10$ is computed and printed.

```

      USE BSINS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=11)

!           INTEGER K, NOUT
      COMPLEX CBS(N), Z

!           Compute
      Z = (10.0, 10.0)
      CALL BSINS (Z, N, CBS)

!           Print the results

```

```
CALL UMACH (2, NOUT)
DO 10 K=1, N
  WRITE (NOUT,99999) K-1, Z, CBS(K)
10 CONTINUE
99999 FORMAT (' I sub ', I2, ' ((' , F6.3, ', ', F6.3, &
            ')) = (', F9.3, ', ', F9.3, ')')
END
```

Output

```
I sub 0 ((10.000,10.000)) = (-2314.975, -411.563)
I sub 1 ((10.000,10.000)) = (-2246.627, -460.681)
I sub 2 ((10.000,10.000)) = (-2044.245, -590.157)
I sub 3 ((10.000,10.000)) = (-1719.746, -751.498)
I sub 4 ((10.000,10.000)) = (-1302.871, -880.632)
I sub 5 ((10.000,10.000)) = (-846.345, -920.394)
I sub 6 ((10.000,10.000)) = (-419.501, -843.607)
I sub 7 ((10.000,10.000)) = (-88.480, -665.930)
I sub 8 ((10.000,10.000)) = ( 108.586, -439.392)
I sub 9 ((10.000,10.000)) = ( 176.165, -227.548)
I sub 10 ((10.000,10.000)) = ( 154.831, -76.050)
```

BSJS

Evaluates a sequence of Bessel functions of the first kind with real order and real positive arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
It must be at least zero and less than one.
- X** — Real argument for which the sequence of Bessel functions is to be evaluated. (Input)
It must be nonnegative.
- N** — Number of elements in the sequence. (Input)
- BS** — Vector of length **N** containing the values of the function through the series. (Output)
BS(I) contains the value of the Bessel function of order **XNU + I - 1** at **x** for **I = 1** to **N**.

FORTRAN 90 Interface

- Generic: **CALL BSJS (XNU, X, N, BS)**
Specific: The specific interface names are **S_BSJS** and **D_BSJS**.

FORTRAN 77 Interface

- Single: **CALL BSJS (XNU, X, N, BS)**
Double: The double precision name is **DBSJS**.

Description

The Bessel function $J_\nu(x)$ is defined to be

$$J_\nu(x) = \frac{(x/2)^\nu}{\sqrt{\pi}\Gamma(\nu + 1/2)} \int_0^\pi \cos(x \cos \theta) \sin^{2\nu} \theta \, d\theta$$

This code is based on the work of Gautschi (1964) and Skovgaard (1975). It uses backward recursion.

Comments

Workspace may be explicitly provided, if desired, by use of **B2JS/DB2JS**. The reference is

```
CALL B2JS (XNU, X, N, BS, WK)
```

The additional argument is

WK — work array of length $2 * N$.

Example

In this example, $J_\nu(2.4048256)$, $\nu = 0, \dots, 10$ is computed and printed.

```

USE BSJS_INT
USE UMACH_INT

IMPLICIT NONE
!
!           Declare variables
INTEGER N
PARAMETER (N=11)
!
INTEGER K, NOUT
REAL BS(N), X, XNU
!
!           Compute
XNU = 0.0
X = 2.4048256
CALL BSJS (XNU, X, N, BS)
!
!           Print the results
CALL UMACH (2, NOUT)
DO 10 K=1, N
    WRITE (NOUT,99999) XNU+K-1, X, BS(K)
10 CONTINUE
99999 FORMAT (' J sub ', F6.3, ' (', F6.3, ') = ', F10.3)
END

```

Output

```

J sub 0.000 ( 2.405) = 0.000
J sub 1.000 ( 2.405) = 0.519
J sub 2.000 ( 2.405) = 0.432
J sub 3.000 ( 2.405) = 0.199
J sub 4.000 ( 2.405) = 0.065
J sub 5.000 ( 2.405) = 0.016
J sub 6.000 ( 2.405) = 0.003
J sub 7.000 ( 2.405) = 0.001
J sub 8.000 ( 2.405) = 0.000
J sub 9.000 ( 2.405) = 0.000
J sub 10.000 ( 2.405) = 0.000

```

BSYS

Evaluates a sequence of Bessel functions of the second kind with real nonnegative order and real positive arguments.

Required Arguments

XNU — Real argument which is the lowest order desired. (Input)
It must be at least zero and less than one.

X — Real positive argument for which the sequence of Bessel functions is to be evaluated. (Input)

N — Number of elements in the sequence. (Input)

BSY — Vector of length **N** containing the values of the function through the series. (Output)
BSY(I) contains the value of the Bessel function of order $I - 1 + \mathbf{XNU}$ at x for $I = 1$ to **N**.

FORTRAN 90 Interface

Generic: `CALL BSYS (XNU, X, N, BSY)`

Specific: The specific interface names are `S_BSYS` and `D_BSYS`.

FORTRAN 77 Interface

Single: `CALL BSYS (XNU, X, N, BSY)`

Double: The double precision name is `DBSYS`.

Description

The Bessel function $Y_\nu(x)$ is defined to be

$$Y_\nu(x) = \frac{1}{\pi} \int_0^\pi \sin(x \sin \theta - \nu \theta) d\theta - \frac{1}{\pi} \int_0^\infty [e^{\nu t} + e^{-\nu t} \cos(\nu \pi)] e^{-x \sinh t} dt$$

The variable ν must satisfy $0 \leq \nu < 1$. If this condition is not met, then **BSY** is set to $-b$. In addition, x must be in $[x_m, x_M]$ where $x_m = 6(16^{-32})$ and $x_M = 16^9$. If $x < x_M$, then $-b$ ($b = \mathbf{AMACH}(2)$, the largest representable number) is returned; and if $x > x_M$, then zero is returned.

The algorithm is based on work of Cody and others, (see Cody et al. 1976; Cody 1969; *NATS FUNPACK* 1976). It uses a special series expansion for small arguments. For moderate arguments, an analytic continuation in the argument based on Taylor series with special rational minimax approximations providing starting values is employed. An asymptotic expansion is used for large arguments.

Example

In this example, $Y_{0.015625+\nu-1}(0.0078125)$, $\nu = 1, 2, 3$ is computed and printed.

```

      USE BSYS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=3)

!           Compute
      INTEGER K, NOUT
      REAL BSY(N), X, XNU

      XNU = 0.015625
      X = 0.0078125
      CALL BSYS (XNU, X, N, BSY)

!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) XNU+K-1, X, BSY(K)
10 CONTINUE
99999 FORMAT (' Y sub ', F6.3, ' ( ', F6.3, ' ) = ', F10.3)
      END

```

Output

```

Y sub  0.016 ( 0.008 ) =    -3.189
Y sub  1.016 ( 0.008 ) =   -88.096
Y sub  2.016 ( 0.008 ) = -22901.732

```

BSIS

Evaluates a sequence of modified Bessel functions of the first kind with real order and real positive arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
It must be greater than or equal to zero and less than one.
- X** — Real argument for which the sequence of Bessel functions is to be evaluated. (Input)
- N** — Number of elements in the sequence. (Input)
- BSI** — Vector of length **N** containing the values of the function through the series. (Output)
BSI(I) contains the value of the Bessel function of order $I - 1 + \text{XNU}$ at x for $I = 1$ to **N**.

FORTRAN 90 Interface

- Generic: `CALL BSIS (XNU, X, N, BSI)`
Specific: The specific interface names are `S_BSIS` and `D_BSIS`.

FORTRAN 77 Interface

- Single: `CALL BSIS (XNU, X, N, BSI)`
Double: The double precision name is `DBSIS`.

Description

The Bessel function $I_\nu(x)$ is defined to be

$$I_\nu(x) = \frac{1}{\pi} \int_0^\pi e^{x \cos \theta} \cos(\nu \theta) d\theta - \frac{\sin(\nu \pi)}{\pi} \int_0^\infty e^{-x \cosh t - \nu t} dt$$

The input x must be nonnegative and less than or equal to $\log(b)$ ($b = \text{AMACH}(2)$, the largest representable number). The argument $\nu = \text{XNU}$ must satisfy $0 \leq \nu \leq 1$.

Function **BSIS** is based on a code due to Cody (1983), which uses backward recursion.

Example

In this example, $I_{\nu-1}(10.0)$, $\nu = 1, \dots, 10$ is computed and printed.

```

      USE BSIS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=10)

!           Compute
      INTEGER K, NOUT
      REAL BSI(N), X, XNU

      XNU = 0.0
      X = 10.0
      CALL BSIS (XNU, X, N, BSI)

!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) XNU+K-1, X, BSI(K)
10 CONTINUE
99999 FORMAT (' I sub ', F6.3, ' (', F6.3, ') = ', F10.3)
      END

```

Output

```

I sub 0.000 (10.000) = 2815.717
I sub 1.000 (10.000) = 2670.988
I sub 2.000 (10.000) = 2281.519
I sub 3.000 (10.000) = 1758.381
I sub 4.000 (10.000) = 1226.491
I sub 5.000 (10.000) = 777.188
I sub 6.000 (10.000) = 449.302
I sub 7.000 (10.000) = 238.026
I sub 8.000 (10.000) = 116.066
I sub 9.000 (10.000) = 52.319

```

BSIES

Evaluates a sequence of exponentially scaled modified Bessel functions of the first kind with nonnegative real order and real positive arguments.

Required Arguments

XNU — Real argument which is the lowest order desired. (Input)
It must be at least zero and less than one.

X — Real positive argument for which the sequence of Bessel functions is to be evaluated. (Input)
It must be nonnegative.

N — Number of elements in the sequence. (Input)

BSI — Vector of length **N** containing the values of the function through the series. (Output)
BSI(I) contains the value of the Bessel function of order $I - 1 + \text{XNU}$ at x for $I = 1$ to **N** multiplied by $\exp(-X)$.

FORTRAN 90 Interface

Generic: `CALL BSIES (XNU, X, N, BSI)`

Specific: The specific interface names are `S_BSIES` and `D_BSIES`.

FORTRAN 77 Interface

Single: `CALL BSIES (XNU, X, N, BSI)`

Double: The double precision name is `DBSIES`.

Description

Function **BSIES** evaluates $e^{-x} I_{\nu+k+1}(x)$, for $k = 1, \dots, n$. For the definition of $I_{\nu}(x)$, see **BSIS**. The algorithm is based on a code due to Cody (1983), which uses backward recursion.

Example

In this example, $I_{\nu-1}(10.0)$, $\nu = 1, \dots, 10$ is computed and printed.

```

      USE BSIES_INT
      USE UMACH_INT

      IMPLICIT NONE
!
!           Declare variables
      INTEGER N
      PARAMETER (N=10)
!
      INTEGER K, NOUT
      REAL BSI(N), X, XNU
!
!           Compute
      XNU = 0.0
      X = 10.0
      CALL BSIES (XNU, X, N, BSI)
!
!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) X, XNU+K-1, X, BSI(K)
10 CONTINUE
99999 FORMAT ( ' exp(-', F6.3, ') * I sub ', F6.3, &
             ' (', F6.3, ') = ', F6.3)
      END

```

Output

```

exp(-10.000) * I sub 0.000 (10.000) = 0.128
exp(-10.000) * I sub 1.000 (10.000) = 0.121
exp(-10.000) * I sub 2.000 (10.000) = 0.104
exp(-10.000) * I sub 3.000 (10.000) = 0.080
exp(-10.000) * I sub 4.000 (10.000) = 0.056
exp(-10.000) * I sub 5.000 (10.000) = 0.035
exp(-10.000) * I sub 6.000 (10.000) = 0.020
exp(-10.000) * I sub 7.000 (10.000) = 0.011
exp(-10.000) * I sub 8.000 (10.000) = 0.005
exp(-10.000) * I sub 9.000 (10.000) = 0.002

```

BSKS

Evaluates a sequence of modified Bessel functions of the second kind of fractional order.

Required Arguments

- XNU** — Fractional order of the function. (Input)
 XNU must be less than one in absolute value.
- X** — Argument for which the sequence of Bessel functions is to be evaluated. (Input)
- NIN** — Number of elements in the sequence. (Input)
- BK** — Vector of length NIN containing the values of the function through the series. (Output)

FORTRAN 90 Interface

- Generic: `CALL BSKS (XNU, X, NIN, BK)`
 Specific: The specific interface names are `S_BSKS` and `D_BSKS`.

FORTRAN 77 Interface

- Single: `CALL BSKS (XNU, X, NIN, BK)`
 Double: The double precision name is `DBSKS`.

Description

The Bessel function $K_\nu(x)$ is defined to be

$$K_\nu(x) = \frac{\pi}{2} e^{v\pi i/2} \left[iJ_\nu\left(xe^{\frac{\pi}{2}i}\right) - Y_\nu\left(xe^{\frac{\pi}{2}i}\right) \right] \quad \text{for } -\pi < \arg x \leq \frac{\pi}{2}$$

Currently, ν is restricted to be less than one in absolute value. A total of $|n|$ values is stored in the array `BK`. For positive n , `BK(1) = $K_\nu(x)$` , `BK(2) = $K_{\nu+1}(x)$` , ..., `BK(n) = $K_{\nu+n-1}(x)$` . For negative n , `BK(1) = $K_\nu(x)$` , `BK(2) = $K_{\nu-1}(x)$` , ..., `BK($|n|$) = $K_{\nu+n+1}$`

BSKS is based on the work of Cody (1983).

Comments

1. If **NIN** is positive, **BK(1)** contains the value of the function of order **XNU**, **BK(2)** contains the value of the function of order **XNU + 1**, ... and **BK(NIN)** contains the value of the function of order **XNU + NIN - 1**.
2. If **NIN** is negative, **BK(1)** contains the value of the function of order **XNU**, **BK(2)** contains the value of the function of order **XNU - 1**, ... and **BK(ABS(NIN))** contains the value of the function of order **XNU + NIN + 1**.

Example

In this example, $K_{\nu-1}(10.0)$, $\nu = 1, \dots, 10$ is computed and printed.

```

      USE BSKS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER NIN
      PARAMETER (NIN=10)

!
      INTEGER K, NOUT
      REAL BS(NIN), X, XNU
!           Compute
      XNU = 0.0
      X = 10.0
      CALL BSKS (XNU, X, NIN, BS)

!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, NIN
         WRITE (NOUT,99999) XNU+K-1, X, BS(K)
10 CONTINUE
99999 FORMAT (' K sub ', F6.3, ' (', F6.3, ') = ', E10.3)
      END

```

Output

```

K sub 0.000 (10.000) = 0.178E-04
K sub 1.000 (10.000) = 0.186E-04
K sub 2.000 (10.000) = 0.215E-04
K sub 3.000 (10.000) = 0.273E-04
K sub 4.000 (10.000) = 0.379E-04
K sub 5.000 (10.000) = 0.575E-04
K sub 6.000 (10.000) = 0.954E-04
K sub 7.000 (10.000) = 0.172E-03
K sub 8.000 (10.000) = 0.336E-03
K sub 9.000 (10.000) = 0.710E-03

```

BSKES

Evaluates a sequence of exponentially scaled modified Bessel functions of the second kind of fractional order.

Required Arguments

- XNU** — Fractional order of the function. (Input)
XNU must be less than 1.0 in absolute value.
- X** — Argument for which the sequence of Bessel functions is to be evaluated. (Input)
- NIN** — Number of elements in the sequence. (Input)
- BKE** — Vector of length **NIN** containing the values of the function through the series. (Output)

FORTRAN 90 Interface

- Generic: `CALL BSKES (XNU, X, NIN, BKE)`
 Specific: The specific interface names are `S_BSKES` and `D_BSKES`.

FORTRAN 77 Interface

- Single: `CALL BSKES (XNU, X, NIN, BKE)`
 Double: The double precision name is `DBSKES`.

Description

Function **BSKES** evaluates $e^x K_{\nu+k-1}(x)$, for $k = 1, \dots, n$. For the definition of $K_\nu(x)$, see [BSKS](#).

Currently, ν is restricted to be less than 1 in absolute value. A total of $|n|$ values is stored in the array **BKE**. For n positive, **BKE**(1) contains $e^x K_\nu(x)$, **BKE**(2) contains $e^x K_{\nu+1}(x)$, ..., and **BKE**(**N**) contains $e^x K_{\nu+n-1}(x)$. For n negative, **BKE**(1) contains $e^x K_\nu(x)$, **BKE**(2) contains $e^x K_{\nu-1}(x)$, ..., and **BKE**($|n|$) contains $e^x K_{\nu+n+1}(x)$. This routine is particularly useful for calculating sequences for large x provided $n \leq x$. (Overflow becomes a problem if $n \ll x$.) n must not be zero, and x must not be greater than zero. Moreover, $|\nu|$ must be less than 1. Also, when $|n|$ is large compared with x , $|\nu+n|$ must not be so large that $e^x K_{\nu+n}(x) \approx e^x \Gamma(|\nu+n|) / [2(x/2)^{|\nu+n|}]$ overflows.

BSKES is based on the work of Cody (1983).

Comments

1. If **NIN** is positive, **BKE(1)** contains **EXP(X)** times the value of the function of order **XNU**, **BKE(2)** contains **EXP(X)** times the value of the function of order **XNU + 1**, ..., and **BKE(NIN)** contains **EXP(X)** times the value of the function of order **XNU + NIN - 1**.
2. If **NIN** is negative, **BKE(1)** contains **EXP(X)** times the value of the function of order **XNU**, **BKE(2)** contains **EXP(X)** times the value of the function of order **XNU - 1**, ..., and **BKE(ABS(NIN))** contains **EXP(X)** times the value of the function of order **XNU + NIN + 1**.

Example

In this example, $K_{\nu-1/2}(2.0)$, $\nu = 1, \dots, 6$ is computed and printed.

```

      USE BSKES_INT
      USE UMACH_INT

      IMPLICIT NONE
!                                     Declare variables
      INTEGER NIN
      PARAMETER (NIN=6)

!
      INTEGER K, NOUT
      REAL BKE(NIN), X, XNU
!                                     Compute
      XNU = 0.5
      X = 2.0
      CALL BSKES (XNU, X, NIN, BKE)

!                                     Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, NIN
         WRITE (NOUT,99999) X, XNU+K-1, X, BKE(K)
10 CONTINUE
99999 FORMAT (' exp( ', F6.3, ') * K sub ', F6.3, &
            ' ( ', F6.3, ') = ', F8.3)
      END

```

Output

```

exp( 2.000) * K sub 0.500 ( 2.000) = 0.886
exp( 2.000) * K sub 1.500 ( 2.000) = 1.329
exp( 2.000) * K sub 2.500 ( 2.000) = 2.880
exp( 2.000) * K sub 3.500 ( 2.000) = 8.530
exp( 2.000) * K sub 4.500 ( 2.000) = 32.735
exp( 2.000) * K sub 5.500 ( 2.000) = 155.837

```

CBJS

Evaluates a sequence of Bessel functions of the first kind with real order and complex arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
XNU must be greater than $-1/2$.
- Z** — Complex argument for which the sequence of Bessel functions is to be evaluated. (Input)
- N** — Number of elements in the sequence. (Input)
- CBS** — Vector of length **N** containing the values of the function through the series. (Output)
CBS(I) contains the value of the Bessel function of order XNU + I - 1 at Z for I = 1 to N.

FORTRAN 90 Interface

- Generic: `CALL CBJs (XNU, Z, N, CBS)`
- Specific: The specific interface names are `S_CBJs` and `D_CBJs`.

FORTRAN 77 Interface

- Single: `CALL CBJs (XNU, Z, N, CBS)`
- Double: The double precision name is `DCBJs`.

Description

The Bessel function $J_\nu(z)$ is defined to be

$$J_\nu(z) = \frac{1}{\pi} \int_0^\pi \cos(z \sin \theta - \nu \theta) d\theta - \frac{\sin(\nu\pi)}{\pi} \int_0^\infty e^{-z \sinh t - \nu t} dt$$

for $|\arg z| < \frac{\pi}{2}$

This code is based on the code `BESSCC` of Barnett (1981) and Thompson and Barnett (1987).

This code computes $J_\nu(z)$ from the modified Bessel function $I_\nu(z)$, `CBIS`, using the following relation:

$$J_\nu(z) = \begin{cases} i^\nu I_\nu(-iz) & \text{for } -\frac{\pi}{2} < \arg z \leq \pi \\ i^{-3\nu} I_\nu(-iz) & \text{for } \pi < \arg z \leq \frac{3\pi}{2} \end{cases}$$

CBJS implements the Yousif and Melka (Y&M) algorithm (Yousif and Melka (1997)) for approximating $J_\nu(z)$ with a Taylor series expansion when $x \sim 0$ or $y \sim 0$, where complex argument $z = x + iy$ and “ $x \sim 0$ ” == “ $|x| < \mathbf{amach}(4)$ ”. To be consistent with the existing CBJS argument definitions, the original Y&M algorithm, which was limited to integral order and to ($x \sim 0$ and $y \geq 0$) or ($y \sim 0$ and $x \geq 0$), has been generalized to also work for integral and real order $\nu > -1$, and for ($x \sim 0$ and $y < 0$) and ($y \sim 0$ and $x < 0$).

To deal with the Bessel function discontinuity that occurs at the negative x axis, the following procedure is used for calculating the Y&M approximation of $J_\nu(z)$ with argument $z = x + iy$ when (($x \sim 0$ and $y < 0$) or ($y \sim 0$ and $x < 0$)):

1. Calculate the Y&M approximation of $J_\nu(-z)$.
2. If ($y > 0$), use forward rotation, otherwise use backward rotation, to calculate the Bessel function $J_\nu(z)$, where the “forward” and “backward” rotation transformations are defined as:

forward: $J_\nu(z) = e^{\nu\pi i} j_\nu(-z) = i^{2\nu} J_\nu(-z)$

backward: $J_\nu(z) = e^{-\nu\pi i} j_\nu(-z) = i^{-2\nu} J_\nu(-z)$

These definitions are based on [Abramowitz and Stegun \(1972\)](#), eq. 9.1.35: $J_\nu(ze^{m\pi i}) = e^{m\nu\pi i} J_\nu(z)$, where $m = 1$ represents forward transformation and $m = -1$ represents backward transformation. These specified rotations insure that the continuous rotation transformation $J_\nu(-z) \rightarrow J_\nu(z)$ does not cross the negative x axis, so no discontinuity is encountered.

Comments

Informational Errors

Type	Code	Description
3	1	One of the continued fractions failed.
4	2	Only the first several entries in CBS are valid.

Example

In this example, $J_{0.3+k-1}(1.2 + 0.5i)$, $k = 1, \dots, 4$ is computed and printed.

```
USE CBJS_INT
USE UMACH_INT
```

```

      IMPLICIT    NONE
!
      INTEGER    N
      PARAMETER (N=4)
!
      INTEGER    K, NOUT
      REAL       XNU
      COMPLEX    CBS(N), Z
!
      XNU = 0.3
      Z    = (1.2, 0.5)
      CALL CBJ5 (XNU, Z, N, CBS)
!
      CALL UMACH (2, NOUT)
      DO 10    K=1, N
         WRITE (NOUT,99999) XNU+K-1, Z, CBS(K)
10    CONTINUE
99999 FORMAT (' J sub ', F6.3, ' ((', F6.3, ', ', F6.3, &
            ')) = (', F9.3, ', ', F9.3, ')')
      END

```

Output

```

J sub  0.300 (( 1.200, 0.500)) = (  0.774,  -0.107)
J sub  1.300 (( 1.200, 0.500)) = (  0.400,   0.159)
J sub  2.300 (( 1.200, 0.500)) = (  0.087,   0.092)
J sub  3.300 (( 1.200, 0.500)) = (  0.008,   0.024)

```

CBYS

Evaluates a sequence of Bessel functions of the second kind with real order and complex arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
 XNU must be greater than $-1/2$.
- Z** — Complex argument for which the sequence of Bessel functions is to be evaluated. (Input)
- N** — Number of elements in the sequence. (Input)
- CBS** — Vector of length **N** containing the values of the function through the series. (Output)
 CBS(I) contains the value of the Bessel function of order $XNU + I - 1$ at **Z** for **I** = 1 to **N**.

FORTRAN 90 Interface

- Generic: `CALL CBYS (XNU, Z, N, CBS)`
 Specific: The specific interface names are `S_CBYS` and `D_CBYS`.

FORTRAN 77 Interface

- Single: `CALL CBYS (XNU, Z, N, CBS)`
 Double: The double precision name is `DCBYS`.

Description

The Bessel function $Y_\nu(z)$ is defined to be

$$Y_\nu(z) = \frac{1}{\pi} \int_0^\pi \sin(z \sin \theta - \nu \theta) d\theta - \frac{1}{\pi} \int_0^\infty [e^{\nu t} + e^{-\nu t} \cos(\nu \pi)] e^{-z \sinh t} dt$$

for $|\arg z| < \frac{\pi}{2}$

This code is based on the code `BESSEC` of Barnett (1981) and Thompson and Barnett (1987).

This code computes $Y_\nu(z)$ from the modified Bessel functions $I_\nu(z)$ and $K_\nu(z)$, `CBIS` and `CBKS`, using the following relation:

$$Y_\nu(z e^{\pi i/2}) = e^{(\nu+1)\pi i/2} I_\nu(z) - \frac{2}{\pi} e^{-\nu\pi i/2} K_\nu(z) \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

CBYS implements the Yousif and Melka (Y&M) algorithm (Yousif and Melka(2003)) for approximating $Y_\nu(z)$ with a Taylor series expansion when $x \sim 0$ or $y \sim 0$, where complex argument $z = x + iy$ and " $x \sim 0$ " == " $|x| < \text{amach}(4)$ ". To be consistent with the existing CBYS argument definitions, the original Y&M algorithm, which was limited to integral order and to ($x \sim 0$ and $y \geq 0$) or ($y \sim 0$ and $x \geq 0$), has been generalized to also work for integral and real order $\nu > -1$, and for ($x \sim 0$ and $y < 0$) and ($y \sim 0$ and $x < 0$).

To deal with the Bessel function discontinuity occurring at the negative x axis, the following procedure is used for calculating the Y&M approximation of $Y_\nu(z)$ with argument $z = x + iy$ when ($x \sim 0$ and $y < 0$) or ($y \sim 0$ and $x < 0$):

1. Calculate the Y&M approximation of $Y_\nu(-z)$.
2. If ($y > 0$), use forward rotation, otherwise use backward rotation, to calculate the Bessel function $Y_\nu(z)$, where the "forward" and "backward" rotation transformations are defined as:

forward: $Y_\nu(z) = i^{-2\nu} Y_\nu(-z) + 2i \cos(\nu\pi) J_\nu(-z)$

backward: $Y_\nu(z) = i^{2\nu} Y_\nu(-z) - 2i \cos(\nu\pi) J_\nu(-z)$

These definitions are based on [Abramowitz and Stegun \(1972\)](#), eq. 9.1.36:

$Y_\nu(z e^{m\pi i}) = e^{-m\nu\pi i} Y_\nu(z) + 2i \sin(m\nu\pi) \cot(\nu\pi) J_\nu(z)$, where $m = 1$ represents forward transformation and $m = -1$ represents backward transformation. These specified rotations insure that the continuous rotation transformation $Y_\nu(-z) \rightarrow Y_\nu(z)$ does not cross the negative x axis, so no discontinuity is encountered.

Comments

1. Workspace may be explicitly provided, if desired, by use of C2YS/DC2Y. The reference is:

CALL C2YS (XNU, Z, N, CBS, FK)

The additional argument is:

FK — complex work vector of length N.

2. Informational errors

Type	Code	Description
3	1	One of the continued fractions failed.
4	2	Only the first several entries in CBS are valid.

Example

In this example, $Y_{0.3+k-1}(1.2 + 0.5i)$, $k = 1, \dots, 4$ is computed and printed.

```

      USE CBYS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=4)

!           Compute
      INTEGER K, NOUT
      REAL XNU
      COMPLEX CBS(N), Z
!
      XNU = 0.3
      Z = (1.2, 0.5)
      CALL CBYS (XNU, Z, N, CBS)
!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) XNU+K-1, Z, CBS(K)
10 CONTINUE
99999 FORMAT (' Y sub ', F6.3, ' ((', F6.3, ', ', F6.3, &
           ')) = (', F9.3, ', ', F9.3, ')')
      END

```

Output

```

Y sub 0.300 (( 1.200, 0.500)) = ( -0.013, 0.380)
Y sub 1.300 (( 1.200, 0.500)) = ( -0.716, 0.338)
Y sub 2.300 (( 1.200, 0.500)) = ( -1.048, 0.795)
Y sub 3.300 (( 1.200, 0.500)) = ( -1.625, 3.684)

```

CBIS

Evaluates a sequence of modified Bessel functions of the first kind with real order and complex arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
XNU must be greater than $-1/2$.
- Z** — Complex argument for which the sequence of Bessel functions is to be evaluated. (Input)
- N** — Number of elements in the sequence. (Input)
- CBS** — Vector of length **N** containing the values of the function through the series. (Output)
CBS(I) contains the value of the Bessel function of order $XNU + I - 1$ at **Z** for $I = 1$ to **N**.

FORTRAN 90 Interface

- Generic: `CALL CBIS (XNU, Z, N, CBS)`
- Specific: The specific interface names are `S_CBIS` and `D_CBIS`.

FORTRAN 77 Interface

- Single: `CALL CBIS (XNU, Z, N, CBS)`
- Double: The double precision name is `DCBIS`.

Description

The modified Bessel function $I_\nu(z)$ is defined to be

$$I_\nu(z) = e^{-\nu\pi i/2} J_\nu\left(ze^{\pi i/2}\right) \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

where the Bessel function $J_\nu(z)$ is defined in [BSJS](#).

This code is based on the code `BESSCC` of Barnett (1981) and Thompson and Barnett (1987).

For large arguments, z , Temme's (1975) algorithm is used to find $I_\nu(z)$. The $I_\nu(z)$ values are recurred upward (if this is stable). This involves evaluating a continued fraction. If this evaluation fails to converge, the answer may not be accurate. For moderate and small arguments, Miller's method is used.

Comments

Informational Errors

Type	Code	Description
3	1	One of the continued fractions failed.
4	2	Only the first several entries in CBS are valid.

Example

In this example, $I_{0.3+\nu-1}(1.2 + 0.5i)$, $\nu = 1, \dots, 4$ is computed and printed.

```

      USE CBIS_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER N
      PARAMETER (N=4)
!
      INTEGER K, NOUT
      REAL XNU
      COMPLEX CBS(N), Z
!           Compute
      XNU = 0.3
      Z = (1.2, 0.5)
      CALL CBIS (XNU, Z, N, CBS)
!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) XNU+K-1, Z, CBS(K)
10 CONTINUE
99999 FORMAT (' I sub ', F6.3, ' ((', F6.3, ', ', F6.3, '&
           ')) = (', F9.3, ', ', F9.3, ')')
      END

```

Output

```

I sub 0.300 (( 1.200, 0.500)) = ( 1.163, 0.396)
I sub 1.300 (( 1.200, 0.500)) = ( 0.447, 0.332)
I sub 2.300 (( 1.200, 0.500)) = ( 0.082, 0.127)
I sub 3.300 (( 1.200, 0.500)) = ( 0.006, 0.029)

```

CBKS

Evaluates a sequence of modified Bessel functions of the second kind with real order and complex arguments.

Required Arguments

- XNU** — Real argument which is the lowest order desired. (Input)
 XNU must be greater than $-1/2$.
- Z** — Complex argument for which the sequence of Bessel functions is to be evaluated. (Input)
- N** — Number of elements in the sequence. (Input)
- CBS** — Vector of length **N** containing the values of the function through the series. (Output)
 CBS(I) contains the value of the Bessel function of order $XNU + I - 1$ at **Z** for $I = 1$ to **N**.

FORTRAN 90 Interface

- Generic: `CALL CBKS (XNU, Z, N, CBS)`
 Specific: The specific interface names are `S_CBKS` and `D_CBKS`.

FORTRAN 77 Interface

- Single: `CALL CBKS (XNU, Z, N, CBS)`
 Double: The double precision name is `DCBKS`.

Description

The Bessel function $K_\nu(z)$ is defined to be

$$K_\nu(z) = \frac{\pi}{2} e^{y\pi i/2} \left[iJ_\nu\left(ze^{\pi i/2}\right) - Y_\nu\left(ze^{\pi i/2}\right) \right] \quad \text{for } -\pi < \arg z \leq \frac{\pi}{2}$$

where the Bessel function $J_\nu(z)$ is defined in [CBJS](#) and $Y_\nu(z)$ is defined in [CBYS](#).

This code is based on the code `BESSCC` of Barnett (1981) and Thompson and Barnett (1987).

For moderate or large arguments, z , Temme's (1975) algorithm is used to find $K_\nu(z)$. This involves evaluating a continued fraction. If this evaluation fails to converge, the answer may not be accurate. For small z , a Neumann series is used to compute $K_\nu(z)$. Upward recurrence of the $K_\nu(z)$ is always stable.

Comments

1. Workspace may be explicitly provided, if desired, by use of `C2KS/DC2KS`. The reference is

```
CALL C2KS (XNU, Z, N, CBS, FK)
```

The additional argument is

FK — Complex work vector of length **N**.

2. Informational errors

Type	Code	Description
3	1	One of the continued fractions failed.
4	2	Only the first several entries in CBS are valid.

Example

In this example, $K_{0.3+v-1}(1.2 + 0.5i)$, $v = 1, \dots, 4$ is computed and printed.

```

USE UMACH_INT
USE CBKS_INT

IMPLICIT NONE
!
INTEGER N
PARAMETER (N=4)
!
INTEGER K, NOUT
REAL XNU
COMPLEX CBS(N), Z
!
XNU = 0.3
Z = (1.2, 0.5)
CALL CBKS (XNU, Z, N, CBS)
!
CALL UMACH (2, NOUT)
DO 10 K=1, N
WRITE (NOUT,99999) XNU+K-1, Z, CBS(K)
10 CONTINUE
99999 FORMAT (' K sub ', F6.3, ' ((', F6.3, ', ', F6.3, &
' )) = (', F9.3, ', ', F9.3, ')')
END
```

Output

```

K sub 0.300 (( 1.200, 0.500)) = ( 0.246, -0.200)
K sub 1.300 (( 1.200, 0.500)) = ( 0.336, -0.362)
K sub 2.300 (( 1.200, 0.500)) = ( 0.587, -1.126)
K sub 3.300 (( 1.200, 0.500)) = ( 0.719, -4.839)
```

Kelvin Functions

Routines

Evaluates $\text{ber}_0(x)$	BERO	205
Evaluates $\text{bei}_0(x)$	BEIO	207
Evaluates $\text{ker}_0(x)$	AKER0	209
Evaluates $\text{kei}_0(x)$	AKEIO	211
Evaluates $\text{ber}'_0(x)$	BERPO	213
Evaluates $\text{bei}'_0(x)$	BEIPO	215
Evaluates $\text{ker}'_0(x)$	AKERPO	217
Evaluates $\text{kei}'_0(x)$	AKEIPO	219
Evaluates $\text{ber}_1(x)$	BER1	221
Evaluates $\text{bei}_1(x)$	BEI1	223
Evaluates $\text{ker}_1(x)$	AKER1	225
Evaluates $\text{kei}_1(x)$	AKEI1	227

Usage Notes

The notation used in this chapter follows that of Abramowitz and Stegun (1964). The Kelvin functions are related to the Bessel functions by the following relations.

$$\operatorname{ber}_\nu x + i\operatorname{bei}_\nu x = J_\nu\left(xe^{3\pi i/4}\right)$$

$$\operatorname{ker}_\nu x + i\operatorname{kei}_\nu x = e^{-\nu\pi i/2} K_\nu\left(xe^{\pi i/4}\right)$$

The derivatives of the Kelvin functions are related to the values of the Kelvin functions by the following:

$$\sqrt{2}\operatorname{ber}'_0 x = \operatorname{ber}_1 x + \operatorname{bei}_1 x$$

$$\sqrt{2}\operatorname{bei}'_0 x = -\operatorname{ber}_1 x + \operatorname{bei}_1 x$$

$$\sqrt{2}\operatorname{ker}'_0 x = \operatorname{ker}_1 x + \operatorname{kei}_1 x$$

$$\sqrt{2}\operatorname{kei}'_0 x = -\operatorname{ker}_1 x + \operatorname{kei}_1 x$$

Plots of $\text{ber}_n(x)$, $\text{bei}_n(x)$, $\text{ker}_n(x)$ and $\text{kei}_n(x)$ for $n = 0, 1$ follow:

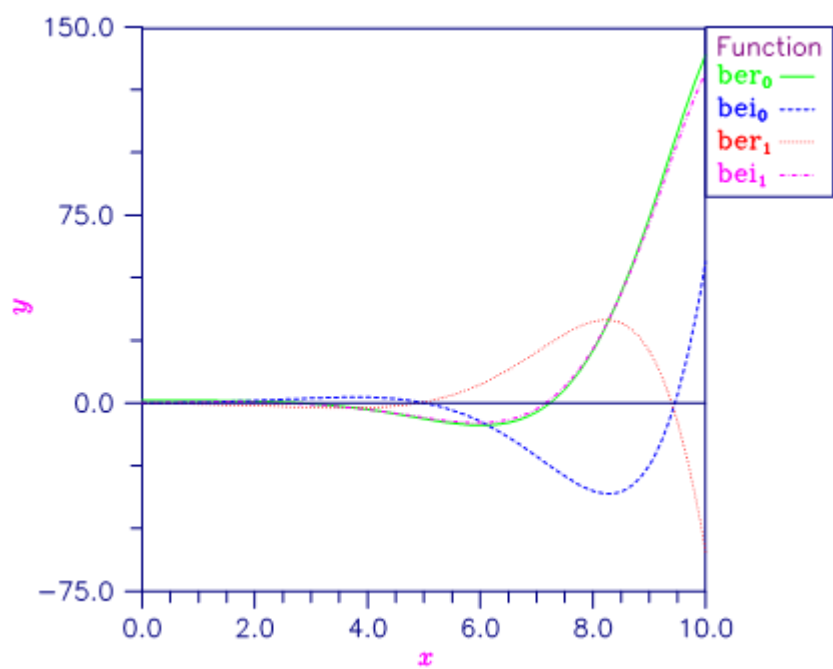


Figure 15, Plot of $\text{ber}_n(x)$ and $\text{bei}_n(x)$

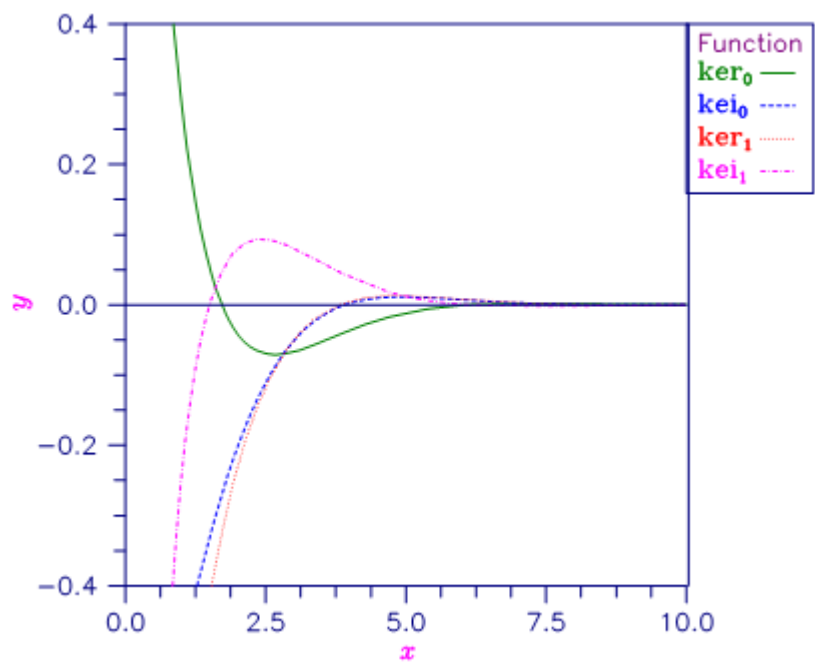


Figure 16, Plot of $ker_n(x)$ and $kei_n(x)$

BERO

This function evaluates the Kelvin function of the first kind, ber , of order zero.

Function Return Value

BERO — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
 $ABS(X)$ must be less than 119.

FORTRAN 90 Interface

Generic: BERO (X)
 Specific: The specific interface names are S_BERO and D_BERO.

FORTRAN 77 Interface

Single: BERO (X)
 Double: The double precision name is DBERO.

Description

The Kelvin function $ber_0(x)$ is defined to be $\Re J_0(xe^{3\pi i/4})$. The Bessel function $J_0(x)$ is defined in [BSJ0](#). Function **BERO** is based on the work of Burgoyne (1963).

Example

In this example, $ber_0(0.4)$ is computed and printed.

```

      USE BERO_INT
      USE UMACH_INT

      IMPLICIT NONE
      !                                     Declare variables
      INTEGER NOUT
      REAL VALUE, X
  
```



```
!                                     Compute
      X      = 0.4
      VALUE = BERO(X)
!                                     Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BERO(', F6.3, ') = ', F6.3)
      END
```

Output

```
BERO( 0.400) = 1.000
```

BEIO

This function evaluates the Kelvin function of the first kind, bei , of order zero.

Function Return Value

BEIO — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
 $ABS(X)$ must be less than 119.

FORTRAN 90 Interface

Generic: **BEIO** (**x**)
Specific: The specific interface names are **S_BEIO** and **D_BEIO**.

FORTRAN 77 Interface

Single: **BEIO** (**x**)
Double: The double precision name is **DBEIO**.

Description

The Kelvin function $bei_0(x)$ is defined to be $\Re J_0(xe^{3\pi i/4})$. The Bessel function $J_0(x)$ is defined in [BSJ0](#). Function **BEIO** is based on the work of Burgoyne (1963).

In **BEIO**, x must be less than 119.

Example

In this example, $bei_0(0.4)$ is computed and printed.

```
USE BEIO_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  REAL         VALUE, X
!                               Compute
  X           = 0.4
  VALUE      = BEIO(X)
!                               Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BEIO(', F6.3, ') = ', F6.3)
  END
```

Output

```
BEIO( 0.400) = 0.040
```

AKERO

This function evaluates the Kelvin function of the second kind, \ker , of order zero.

Function Return Value

AKERO — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: AKERO (**x**)
Specific: The specific interface names are S_AKERO and D_AKERO.

FORTRAN 77 Interface

Single: AKERO (**x**)
Double: The double precision name is DKERO.

Description

The modified Kelvin function $\ker_0(x)$ is defined to be $\Re K_0(xe^{\pi i/4})$. The Bessel function $K_0(x)$ is defined in [BSK0](#). Function **AKERO** is based on the work of Burgoyne (1963). If $x < 0$, then NaN (not a number) is returned. If $x \geq 119$, then zero is returned.

Example

In this example, $\ker_0(0.4)$ is computed and printed.

```

USE AKERO_INT
USE UMACH_INT

IMPLICIT NONE
!                               Declare variables

```

```
      INTEGER      NOUT
      REAL         VALUE, X
!
      X           = 0.4
      VALUE = AKERO(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKERO(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKERO( 0.400) = 1.063
```

AKEI0

This function evaluates the Kelvin function of the second kind, kei , of order zero.

Function Return Value

AKEI0 — Function value. (Output)

Required Arguments

x — Argument for which the function value is desired. (Input)
It must be nonnegative and less than 119.

FORTRAN 90 Interface

Generic: AKEI0 (x)
Specific: The specific interface names are S_AKEI0 and D_AKEI0.

FORTRAN 77 Interface

Single: AKEI0 (x)
Double: The double precision name is DKEI0.

Description

The modified Kelvin function $kei_0(x)$ is defined to be $\mathfrak{K}K_0(xe^{\pi i/4})$. The Bessel function $K_0(x)$ is defined in [BSK0](#). Function **AKEI0** is based on the work of Burgoyne (1963).

In **AKEI0**, x must satisfy $0 \leq x < 119$. If $x < 0$, then NaN (not a number) is returned. If $x \geq 119$, then zero is returned.

Example

In this example, $kei_0(0.4)$ is computed and printed.

```
USE AKEI0_INT
USE UMACH_INT
```

```
      IMPLICIT  NONE
!
      INTEGER  NOUT
      REAL     VALUE, X
!
      X       = 0.4
      VALUE  = AKEI0(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKEI0(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKEI0( 0.400) = -0.704
```

BERPO

This function evaluates the derivative of the Kelvin function of the first kind, ber , of order zero.

Function Return Value

BERPO — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: BERPO (x)

Specific: The specific interface names are S_BERPO and D_BERPO.

FORTRAN 77 Interface

Single: BERPO (x)

Double: The double precision name is DBERPO.

Description

The function $\text{ber}'_0(x)$ is defined to be

$$\frac{d}{dx}\text{ber}_0(x)$$

where $\text{ber}_0(x)$ is a Kelvin function, see [BER0](#). Function **BERPO** is based on the work of Burgoyne (1963).

If $|x| > 119$, then NaN (not a number) is returned.

Example

In this example, $\text{ber}'_0(0.6)$ is computed and printed.

```
USE BERPO_INT
USE UMACH_INT
```



```
!      IMPLICIT  NONE
!      INTEGER  NOUT
!      REAL     VALUE, X
!
!      X       = 0.6
!      VALUE = BERPO(X)
!
!      CALL UMACH (2, NOUT)
!      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BERPO(' , F6.3, ') = ' , F6.3)
!      END
```

Output

```
BERPO( 0.600) = -0.013
```

BEIPO

This function evaluates the derivative of the Kelvin function of the first kind, bei , of order zero.

Function Return Value

BEIPO — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: BEIPO (**x**)

Specific: The specific interface names are S_BEIPO and D_BEIPO.

FORTRAN 77 Interface

Single: BEIPO (**x**)

Double: The double precision name is DBEIPO.

Description

The function $bei'_0(x)$ is defined to be

$$\frac{d}{dx} bei_0(x)$$

where $bei_0(x)$ is a Kelvin function, see [BEI0](#). Function **BEIPO** is based on the work of Burgoyne (1963).

If $|x| > 119$, then NaN (not a number) is returned.

Example

In this example, $bei'_0(0.6)$ is computed and printed.

```
USE BEIPO_INT
USE UMACH_INT
```

```
!      IMPLICIT  NONE                                Declare variables
!      INTEGER   NOUT
!      REAL      VALUE, X
!
!      X        = 0.6                                Compute
!      VALUE = BEIPO(X)
!
!      CALL UMACH (2, NOUT)                          Print the results
!      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BEIPO(', F6.3, ') = ', F6.3)
!      END
```

Output

```
BEIPO( 0.600) = 0.300
```

AKERPO

This function evaluates the derivative of the Kelvin function of the second kind, \ker , of order zero.

Function Return Value

AKERPO — Function value. (Output)

Required Arguments

x — Argument for which the function value is desired. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: AKERPO (**x**)
Specific: The specific interface names are S_AKERPO and D_AKERPO.

FORTRAN 77 Interface

Single: AKERPO (**x**)
Double: The double precision name is DKERPO.

Description

The function $\ker'_0(x)$ is defined to be

$$\frac{d}{dx}\ker_0(x)$$

where $\ker_0(x)$ is a Kelvin function, see [AKER0](#). Function **AKERPO** is based on the work of Burgoyne (1963). If $x < 0$, then NaN (not a number) is returned. If $x > 119$, then zero is returned.

Example

In this example, $\ker'_0(0.6)$ is computed and printed.

```
USE AKERPO_INT
```

```
      USE UMACH_INT
      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X, AKERPO
!      Compute
      X = 0.6
      VALUE = AKERPO(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKERPO(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKERPO( 0.600) = -1.457
```

AKEIPO

This function evaluates the derivative of the Kelvin function of the second kind, kei , of order zero.

Function Return Value

AKEIPO — Function value. (Output)

Required Arguments

x — Argument for which the function value is desired. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: AKEIPO (x)
Specific: The specific interface names are S_AKEIPO and D_AKEIPO.

FORTRAN 77 Interface

Single: AKEIPO (x)
Double: The double precision name is DKEIPO.

Description

The function $kei'_0(x)$ is defined to be

$$\frac{d}{dx}kei_0(x)$$

where $kei_0(x)$ is a Kelvin function, see [AKEI0](#). Function **AKEIPO** is based on the work of Burgoyne (1963).

If $x < 0$, then NaN (not a number) is returned. If $x > 119$, then zero is returned.

Example

In this example, $kei'_0(0.6)$ is computed and printed.

```
      USE AKEIPO_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X, AKEIPO
!      Compute
      X = 0.6
      VALUE = AKEIPO(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKEIPO(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKEIPO( 0.600) = 0.348
```

BER1

This function evaluates the Kelvin function of the first kind, ber_1 , of order one.

Function Return Value

BER1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BER1 (X)**

Specific: The specific interface names are **S_BER1** and **D_BER1**.

FORTRAN 77 Interface

Single: **BER1 (X)**

Double: The double precision name is **DBER1**.

Description

The Kelvin function $ber_1(x)$ is defined to be $\Re J_1(xe^{3\pi i/4})$. The Bessel function $J_1(x)$ is defined in [BSJ1](#). Function **BER1** is based on the work of Burgoyne (1963).

If $|x| > 119$, then NaN (not a number) is returned.

Example

In this example, $ber_1(0.4)$ is computed and printed.

```

      USE BER1_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
```



```
!      REAL      VALUE, X
!      X        = 0.4           Compute
!      VALUE = BER1(X)
!
!      CALL UMACH (2, NOUT)      Print the results
!      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BER1(', F6.3, ') = ', F6.3)
!      END
```

Output

```
BER1( 0.400) = -0.144
```

BEI1

This function evaluates the Kelvin function of the first kind, bei_1 , of order one.

Function Return Value

BEI1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BEI1 (X)**

Specific: The specific interface names are **S_BEI1** and **D_BEI1**.

FORTRAN 77 Interface

Single: **BEI1 (X)**

Double: The double precision name is **DBEI1**.

Description

The Kelvin function $bei_1(x)$ is defined to be $\Im J_1(xe^{3\pi i/4})$. The Bessel function $J_1(x)$ is defined in [BSJ1](#). Function **BEI1** is based on the work of Burgoyne (1963).

If $|x| > 119$, then NaN (not a number) is returned.

Example

In this example, $bei_1(0.4)$ is computed and printed.

```

      USE BEI1_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
```

```
      REAL          VALUE, X
!
      X            = 0.4           Compute
      VALUE = BEI1(X)
!
      CALL UMACH (2, NOUT)        Print the results
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BEI1(', F6.3, ') = ', F6.3)
      END
```

Output

```
BEI1( 0.400) = 0.139
```

AKER1

This function evaluates the Kelvin function of the second kind, \ker , of order one.

Function Return Value

AKER1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: **AKER1 (X)**
Specific: The specific interface names are **S_AKER1** and **D_AKER1**.

FORTRAN 77 Interface

Single: **AKER1 (X)**
Double: The double precision name is **DKER1**.

Description

The modified Kelvin function $\ker_1(x)$ is defined to be $e^{-\pi i/2} \Re K_1(xe^{\pi i/4})$. The Bessel function $K_1(x)$ is defined in [BSK1](#). Function **AKER1** is based on the work of Burgoyne (1963).

If $x < 0$, then NaN (not a number) is returned. If $x \geq 119$, then zero is returned.

Example

In this example, $\ker_1(0.4)$ is computed and printed.

```
USE AKER1_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
      INTEGER      NOUT
      REAL         VALUE, X
!                               Compute
      X           = 0.4
      VALUE = AKER1(X)
!                               Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKER1(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKER1( 0.400) = -1.882
```

AKEI1

This function evaluates the Kelvin function of the second kind, kei_1 , of order one.

Function Return Value

AKEI1 — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: **AKEI1 (X)**
Specific: The specific interface names are **S_AKEI1** and **D_AKEI1**.

FORTRAN 77 Interface

Single: **AKEI1 (X)**
Double: The double precision name is **DKEI1**.

Description

The modified Kelvin function $kei_1(x)$ is defined to be $e^{-\pi i/2} \Im K_1(xe^{\pi i/4})$. The Bessel function $K_1(x)$ is defined in [BSK1](#). Function **AKEI1** is based on the work of Burgoyne (1963).

If $x < 0$, then NaN (not a number) is returned. If $x \geq 119$, then zero is returned.

Example

In this example, $kei_1(0.4)$ is computed and printed.

```
USE UMACH_INT
USE AKEI1_INT

IMPLICIT NONE
```

```
!                               Declare variables
      INTEGER      NOUT
      REAL         VALUE, X
!                               Compute
      X           = 0.4
      VALUE = AKEI1(X)
!                               Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AKEI1(', F6.3, ') = ', F6.3)
      END
```

Output

```
AKEI1( 0.400) = -1.444
```

Airy Functions

Routines

8.1 Real Airy Functions

Evaluates $Ai(x)$	AI	230
Evaluates $Bi(x)$	BI	232
Evaluates $Ai'(x)$	AID	234
Evaluates $Bi'(x)$	BID	236
Evaluates exponentially scaled $Ai(x)$	AIE	238
Evaluates exponentially scaled $Bi(x)$	BIE	240
Evaluates exponentially scaled $Ai'(x)$	AIDE	242
Evaluates exponentially scaled $Bi'(x)$	BIDE	244

8.2 Complex Airy Functions

Evaluates $Ai(z)$	CAI	246
Evaluates $Bi(z)$	CBI	248
Evaluates $Ai'(z)$	CAID	250
Evaluates $Bi'(z)$	CBID	252

AI

This function evaluates the Airy function.

Function Return Value

AI — Function value. (Output)

Required Arguments

X — Argument for which the Airy function is desired. (Input)

FORTRAN 90 Interface

Generic: **AI** (X)

Specific: The specific interface names are **S_AI** and **D_AI**.

FORTRAN 77 Interface

Single: **AI** (X)

Double: The double precision name is **DAI**.

Description

The Airy function $Ai(x)$ is defined to be

$$Ai(x) = \frac{1}{\pi} \int_0^{\infty} \cos\left(xt + \frac{1}{3}t^3\right) dt = \sqrt{\frac{x}{3\pi^2}} K_{1/3}\left(\frac{2}{3}x^{3/2}\right)$$

The Bessel function $K(x)$ is defined in [BSKS](#).

If $x < -1.31e^{-2/3}$, then the answer will have no precision. If $x < -1.31e^{-1/3}$, the answer will be less accurate than half precision. Here, $\epsilon = \mathbf{AMACH}(4)$ is the machine precision. Finally, x should be less than x_{\max} so the answer does not underflow. Very approximately, $x_{\max} = \{-1.5 \ln s\}$, where $s = \mathbf{AMACH}(1)$, the smallest representable positive number. If underflows are a problem for large x , then the exponentially scaled routine [AIE](#) should be used.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is greater than XMAX, where $XMAX = (-3/2 \ln(AMACH(1)))^{2/3}$.

Example

In this example, Ai(-4.9) is computed and printed.

```

      USE AI_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      REAL VALUE, X
!          Compute
      X = -4.9
      VALUE = AI(X)
!          Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AI(', F6.3, ') = ', F6.3)
      END

```

Output

```
AI(-4.900) = 0.375
```

BI

This function evaluates the Airy function of the second kind.

Function Return Value

BI — Function value. (Output)

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BI (X)**

Specific: The specific interface names are **S_BI** and **D_BI**.

FORTRAN 77 Interface

Single: **BI (X)**

Double: The double precision name is **DBI**.

Description

The Airy function of the second kind $\text{Bi}(x)$ is defined to be

$$\text{Bi}(x) = \frac{1}{\pi} \int_0^{\infty} \exp\left(xt - \frac{1}{3}t^3\right) dt + \frac{1}{\pi} \int_0^{\infty} \sin\left(xt + \frac{1}{3}t^3\right) dt$$

It can also be expressed in terms of modified Bessel functions of the first kind, $I_{\nu}(x)$, and Bessel functions of the first kind, $J_{\nu}(x)$ (see [BSIS](#) and [BSJS](#)):

$$\text{Bi}(x) = \sqrt{\frac{x}{3}} \left[I_{-1/3}\left(\frac{2}{3}x^{3/2}\right) + I_{1/3}\left(\frac{2}{3}x^{3/2}\right) \right] \quad \text{for } x > 0$$

and

$$\text{Bi}(x) = \sqrt{-\frac{x}{3}} \left[J_{-1/3} \left(\frac{2}{3} |x|^{3/2} \right) - J_{1/3} \left(\frac{2}{3} |x|^{3/2} \right) \right] \quad \text{for } x < 0$$

Let $\epsilon = \text{AMACH}(4)$, the machine precision. If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, the answer will be less accurate than half precision. In addition, x should not be so large that $\exp\left[\left(\frac{2}{3}\right)x^{3/2}\right]$ overflows. If overflows are a problem, consider using the exponentially scaled form of the Airy function of the second kind, [BIE](#), instead.

Example

In this example, $\text{Bi}(-4.9)$ is computed and printed.

```

USE BI_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
REAL VALUE, X
!
X = -4.9
VALUE = BI(X)                                Compute
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BI(', F6.3, ') = ', F6.3)
END

```

Output

```
BI(-4.900) = -0.058
```

AID

This function evaluates the derivative of the Airy function.

Function Return Value

AID — Function value. (Output)

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **AID (X)**

Specific: The specific interface names are **S_AID** and **D_AID**.

FORTRAN 77 Interface

Single: **AID (X)**

Double: The double precision name is **DAID**.

Description

The function $Ai'(x)$ is defined to be the derivative of the Airy function, $Ai(x)$ (see [AI](#)).

If $x < -1.31e^{-2/3}$, then the answer will have no precision. If $x < -1.31e^{-1/3}$, the answer will be less accurate than half precision. Here, $\epsilon = \mathbf{AMACH}(4)$ is the machine precision. Finally, x should be less than x_{\max} so that the answer does not underflow. Very approximately, $x_{\max} = \{-1.5 \ln s\}$, where $s = \mathbf{AMACH}(1)$, the smallest representable positive number. If underflows are a problem for large x , then the exponentially scaled routine [AIDE](#) should be used.

Comments

Informational Error

Type	Code	Description
2	1	The function underflows because x is greater than X_{MAX} , where $X_{MAX} = -3/2 \ln(AMACH(1))$.

Example

In this example, $Ai'(-4.9)$ is computed and printed.

```
      USE AID_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL VALUE, X
!
      X = -4.9
      VALUE = AID(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AID(', F6.3, ') = ', F6.3)
      END
```

Output

```
AID(-4.900) = 0.147
```

BID

This function evaluates the derivative of the Airy function of the second kind.

Function Return Value

BID — Function value. (Output)

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BID (X)**

Specific: The specific interface names are **S_BID** and **D_BID**.

FORTRAN 77 Interface

Single: **BID (X)**

Double: The double precision name is **DBID**.

Description

The function $Bi'(x)$ is defined to be the derivative of the Airy function of the second kind, $Bi(x)$ (see [BI](#)).

If $x < -1.31e^{-2/3}$, then the answer will have no precision. If $x < -1.31e^{-1/3}$, the answer will be less accurate than half precision. In addition, x should not be so large that $\exp\left[\left(\frac{2}{3}\right)x^{3/2}\right]$ overflows. If overflows are a problem, consider using [BIDE](#) instead. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $Bi'(-4.9)$ is computed and printed.

```
USE BID_INT
USE UMACH_INT

IMPLICIT NONE
```

```
!                               Declare variables
  INTEGER      NOUT
  REAL         VALUE, X
!                               Compute
  X            = -4.9
  VALUE = BID(X)
!                               Print the results
  CALL UMACH (2, NOUT)
  WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BID(', F6.3, ') = ', F6.3)
  END
```

Output

```
BID(-4.900) = 0.827
```


AIE

This function evaluates the exponentially scaled Airy function.

Function Return Value

AIE — Function value. (Output)

The Airy function for negative arguments and the exponentially scaled Airy function, $e^{\zeta} \text{Ai}(x)$, for positive arguments where

$$\zeta = \frac{2}{3} X^{3/2}$$

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **AIE (X)**

Specific: The specific interface names are **S_AIE** and **D_AIE**.

FORTRAN 77 Interface

Single: **AIE (X)**

Double: The double precision name is **DAIE**.

Description

The exponentially scaled Airy function is defined to be

$$\text{AIE}(x) = \begin{cases} \text{Ai}(x) & \text{if } x \leq 0 \\ e^{[2/3]x^{3/2}} \text{Ai}(x) & \text{if } x > 0 \end{cases}$$

If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, then the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $\text{AIE}(0.49)$ is computed and printed.

```
      USE AIE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 0.49
      VALUE = AIE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AIE(', F6.3, ') = ', F6.3)
      END
```

Output

```
AIE( 0.490) = 0.294
```

BIE

This function evaluates the exponentially scaled Airy function of the second kind.

Function Return Value

BIE — Function value. (Output)

The Airy function of the second kind for negative arguments and the exponentially scaled Airy function of the second kind, $e^{\zeta}\text{Bi}(x)$, for positive arguments where

$$\zeta = -\frac{2}{3}X^{3/2}$$

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BIE** (X)

Specific: The specific interface names are **S_BIE** and **D_BIE**.

FORTRAN 77 Interface

Single: **BIE** (X)

Double: The double precision name is **DBIE**.

Description

The exponentially scaled Airy function of the second kind is defined to be

$$\text{BIE}(x) = \begin{cases} \text{Bi}(x) & \text{if } x \leq 0 \\ e^{-[2/3]x^{3/2}} \text{Bi}(x) & \text{if } x > 0 \end{cases}$$

If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, then the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, $\text{BIE}(0.49)$ is computed and printed.

```
      USE BIE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 0.49
      VALUE = BIE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BIE(', F6.3, ') = ', F6.3)
      END
```

Output

```
BIE( 0.490) = 0.675
```

AIDE

This function evaluates the exponentially scaled derivative of the Airy function.

Function Return Value

AIDE — Function value. (Output)

The derivative of the Airy function for negative arguments and the exponentially scaled derivative of the Airy function, $e^{\zeta} \text{Ai}'(\mathbf{x})$, for positive arguments where

$$\zeta = -\frac{2}{3}X^{3/2}$$

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **AIDE (X)**

Specific: The specific interface names are **S_AIDE** and **D_AIDE**.

FORTRAN 77 Interface

Single: **AIDE (X)**

Double: The double precision name is **DAIDE**.

Description

The exponentially scaled derivative of the Airy function is defined to be

$$\text{AIDE}(x) = \begin{cases} \text{Ai}'(x) & \text{if } x \leq 0 \\ e^{[2/3]x^{3/2}} \text{Ai}'(x) & \text{if } x > 0 \end{cases}$$

If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, then the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, `AIDE(0.49)` is computed and printed.

```
      USE AIDE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 0.49
      VALUE = AIDE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' AIDE(', F6.3, ') = ', F6.3)
      END
```

Output

```
AIDE( 0.490) = -0.284
```

BIDE

This function evaluates the exponentially scaled derivative of the Airy function of the second kind.

Function Return Value

BIDE — Function value. (Output)

The derivative of the Airy function of the second kind for negative arguments and the exponentially scaled derivative of the Airy function of the second kind, $e^{\zeta}\text{Bi}'(x)$, for positive arguments where

$$\zeta = -\frac{2}{3}x^{3/2}$$

Required Arguments

X — Argument for which the Airy function value is desired. (Input)

FORTRAN 90 Interface

Generic: **BIDE (X)**

Specific: The specific interface names are **S_BIDE** and **D_BIDE**.

FORTRAN 77 Interface

Single: **BIDE (X)**

Double: The double precision name is **DBIDE**.

Description

The exponentially scaled derivative of the Airy function of the second kind is defined to be

$$\text{BIDE}(x) = \begin{cases} \text{Bi}'(x) & \text{if } x \leq 0 \\ e^{-[2/3]x^{3/2}} \text{Bi}'(x) & \text{if } x > 0 \end{cases}$$

If $x < -1.31\epsilon^{-2/3}$, then the answer will have no precision. If $x < -1.31\epsilon^{-1/3}$, then the answer will be less accurate than half precision. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Example

In this example, `BIDE(0.49)` is computed and printed.

```
      USE BIDE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 0.49
      VALUE = BIDE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' BIDE(', F6.3, ') = ', F6.3)
      END
```

Output

```
BIDE( 0.490) = 0.430
```

CAI

This function evaluates the Airy function of the first kind for complex arguments.

Function Return Value

CAI — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the Airy function is desired. (Input)

Optional Arguments

SCALING — Logical argument specifying whether or not the scaling function will be applied to the $Ai(z)$ function value. (Input)
Default: `SCALING = .false.`

FORTRAN 90 Interface

Generic: `CAI (Z)`
Specific: The specific interface names are `C_CAI` and `Z_CAI`.

Description

The Airy function $Ai(z)$ is a solution of the differential equation

$$\frac{d^2 w}{dz^2} = zw$$

The mathematical development and algorithm, 838, used here are found in the work by Fabijonas *et al.* Function `CAI` returns the complex values of $Ai(z)$.

An optional argument, `SCALING`, defines a scaling function $s(z)$ that multiplies the results. This scaling function is

Scaling	Action
<code>.false.</code>	$s(z) = 1$
<code>.true.</code>	$s(z) = e^{[2/3]z^{3/2}}$

Comments

Informational Errors

Type	Code	Description
2	1	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially small; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .
2	2	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially large; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .

Example

In this example, `Ai(0.49, 0.49)` is computed and printed.

```

USE CAI_INT
USE UMACH_INT
IMPLICIT NONE

!
!           Declare variables
INTEGER      NOUT
COMPLEX      Y, Z, W

!
!           Compute
W = CMPLX(0.49,0.49)
Y = CAI(W)

!
!           Print the results
CALL UMACH (2, NOUT)
WRITE (NOUT,99998) W, Y

!
99998  FORMAT(12x,"CAI(",F6.3 " , " ,F6.3 ") = ( ",F6.3, " , ",F6.3," )" )
End

```

Output

```
CAI( 0.490, 0.490) = ( 0.219, -0.113 )
```

CBI

This function evaluates the Airy function of the second kind for complex arguments.

Function Return Value

CBI — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the Airy function value is desired. (Input)

Optional Arguments

SCALING — Logical argument specifying whether or not the scaling function will be applied to the $Ai(z)$ function value used to compute $Bi(z)$. (Input)
Default: `SCALING = .false.`

FORTRAN 90 Interface

Generic: CBI (Z)

Specific: The specific interface names are `C_CBI` and `Z_CBI`.

Description

The Airy function of the second kind $Bi(z)$ is expressed using the connection formula

$$Bi(z) = e^{-\pi i/6} Ai(ze^{-2\pi i/3}) + e^{\pi i/6} Ai(ze^{2\pi i/3})$$

using function `CAI` for $Ai(z)$.

An optional argument, `SCALING`, defines a scaling function $s(z)$ that multiplies the results. This scaling function is

Scaling	Action
<code>.false.</code>	$s(z) = 1$
<code>.true.</code>	$s(z) = e^{[2/3]z^{3/2}}$

The values for $\text{Bi}(z)$ are returned with the scaling for $\text{Ai}(z)$.

Comments

Informational Errors

Type	Code	Description
2	1	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially small; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .
2	2	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially large; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .

Example

In this example, $\text{Bi}(0.49, 0.49)$ is computed and printed.

```

USE CBI_INT
USE UMACH_INT
IMPLICIT NONE

!
!           INTEGER      NOUT           Declare variables
!           COMPLEX     Y, Z, W
!
!           W = CMPLX(0.49,0.49)       Compute
!           Y = CBI(W)
!
!           CALL UMACH (2, NOUT)       Print the results
!           WRITE (NOUT,99998) W, Y
!
99998  FORMAT(12x,"CBI(",F6.3 " , ",F6.3 ") = ( ",F6.3, " , ",F6.3," )" )
End

```

Output

```
CBI( 0.490, 0.490) = ( 0.802, 0.243 )
```

CAID

This function evaluates the derivative of the Airy function of the first kind for complex arguments.

Function Return Value

CAID — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the Airy function value is desired. (Input)

Optional Arguments

SCALING — Logical argument specifying whether or not the scaling function will be applied to the $Ai'(z)$ function value. (Input)
Default: `SCALING = .false.`

FORTRAN 90 Interface

Generic: `C_CAID(Z)`

Specific: The specific interface names are `C_CAID` and `Z_CAID`.

Description

The function $Ai'(z)$ is defined to be the derivative of the Airy function, $Ai(z)$ (see [CAI](#)).

An optional argument, `SCALING`, defines a scaling function $s(z)$ that multiplies the results. This scaling function is

Scaling	Action
<code>.false.</code>	$s(z) = 1$
<code>.true.</code>	$s(z) = e^{[2/3]z^{3/2}}$

Comments

Informational Errors

Type	Code	Description
2	1	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially small; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .
2	2	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially large; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .

Example

In this example, $Ai(0.49, 0.49)$ and $Ai'(0.49, 0.49)$ are computed and printed.

```

USE CAID_INT
USE CAI_INT
USE UMACH_INT
IMPLICIT NONE

!
!           INTEGER      NOUT           Declare variables
!           COMPLEX      Y, Z, W, Z
!
!           Compute
W = CMPLX(0.49,0.49)
Y = CAI(W)
Z = CAID(W)
!
!           Print the results
CALL UMACH (2, NOUT)
WRITE (NOUT,99998) W, Y
WRITE (NOUT,99997) W, Z
!
99997  FORMAT(12x,"CAID(",F6.3 ", ",F6.3 ") = ( ",F6.3, ", ",F6.3," )" )
99998  FORMAT(12x,"CAI(",F6.3 ", ",F6.3 ") = ( ",F6.3, ", ",F6.3," )" )
End

```

Output

```

CAI( 0.490, 0.490) = ( 0.219, -0.113 )
CAID( 0.490, 0.490) = ( -0.240, 0.064 )

```

CBID

This function evaluates the derivative of the Airy function of the second kind for complex arguments.

Function Return Value

CBID — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the Airy function value is desired. (Input)

Optional Arguments

SCALING — Logical argument specifying whether or not the scaling function will be applied to the $Ai'(z)$ function value used to compute $Bi'(z)$. (Input)
Default: `SCALING = .false.`

FORTRAN 90 Interface

Generic: `CBID (Z)`
Specific: The specific interface names are `C_CBID` and `Z_CBID`.

Description

The function $Bi'(z)$ is defined to be the derivative of the Airy function of the second kind, $Bi(z)$, (see [CBI](#)), expressed using the connection formula

$$Bi'(z) = e^{-5\pi i/6} Ai'(ze^{-2\pi i/3}) + e^{5\pi i/6} Ai'(ze^{2\pi i/3})$$

using function `CAID` for $Ai'(z)$.

An optional argument, `SCALING`, defines a scaling function $s(z)$ that multiplies the results. This scaling function is

Scaling	Action
<code>.false.</code>	$s(z) = 1$
<code>.true.</code>	$s(z) = e^{[2/3]z^{3/2}}$

The values for $Bi'(z)$ are returned with the scaling for $Ai'(z)$.

Comments

Informational Errors

Type	Code	Description
2	1	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially small; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .
2	2	The real part of $(2/3) \times z^{(3/2)}$ was too large in the region where the function is exponentially large; function values were set to zero to avoid underflow. Try supplying the optional argument <code>SCALING</code> .

Example

In this example, $Bi'(0.49, 0.49)$ is computed and printed.

```

USE CBID_INT
USE UMACH_INT
IMPLICIT NONE

!
!           INTEGER      NOUT           Declare variables
!           COMPLEX     Y, Z, W
!
!           W = CMPLX(0.49,0.49)       Compute
!           Y = CBID(W)
!
!           CALL UMACH (2, NOUT)       Print the results
!           WRITE (NOUT,99998) W, Y
!
99998  FORMAT(12x,"CBID(",F6.3 " , " ,F6.3 ") = ( ",F6.3, " , ",F6.3," )" )
End

```

Output

```
CBID( 0.490, 0.490) = ( 0.411, 0.180 )
```


Elliptic Integrals

Routines

Evaluates the complete elliptic integral of the first kind, $K(x)$	ELK	258
Evaluates the complete elliptic integral of the second kind, $E(x)$	ELE	260
Evaluates Carlson's elliptic integral of the first kind, $R_F(x, y, z)$	ELRF	262
Evaluates Carlson's elliptic integral of the second kind, $R_D(x, y, z)$	ELRD	264
Evaluates Carlson's elliptic integral of the third kind, $R_J(x, y, z)$	ELRJ	266
Evaluates a special case of Carlson's elliptic integral, $R_C(x, y, z)$	ELRC	268

Usage Notes

The notation used in this chapter follows that of Abramowitz and Stegun (1964) and Carlson (1979).

The complete elliptic integral of the first kind is

$$K(m) = \int_0^{\pi/2} (1 - m \sin^2 \theta)^{-1/2} d\theta$$

and the complete elliptic integral of the second kind is

$$E(m) = \int_0^{\pi/2} (1 - m \sin^2 \theta)^{1/2} d\theta$$

Instead of the *parameter* m , the *modular angle* α is sometimes used with $m = \sin^2 \alpha$. Also used is the *modulus* k with $k^2 = m$.

$$\begin{aligned} E(k) &= \int_0^{\pi/2} (1 - k^2 \sin^2 \theta)^{1/2} d\theta \\ &= R_F(0, 1 - k^2, 1) - \frac{1}{3} k^2 R_D(0, 1 - k^2, 1) \end{aligned}$$

Carlson Elliptic Integrals

The Carlson elliptic integrals are defined by Carlson (1979) as follows:

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)]^{1/2}}$$

$$R_C(x, y) = \frac{1}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)^2]^{1/2}}$$

$$R_J(x, y, z, \rho) = \frac{3}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)(t+\rho)^2]^{1/2}}$$

$$R_D(x, y, z) = \frac{3}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)^3]^{1/2}}$$

The standard Legendre elliptic integrals can be written in terms of the Carlson functions as follows (these relations are from Carlson (1979)):

$$\begin{aligned} F(\phi, k) &= \int_0^\phi (1 - k^2 \sin^2 \theta)^{-1/2} d\theta \\ &= (\sin \phi) R_F(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) \end{aligned}$$

$$\begin{aligned} E(\phi, k) &= \int_0^\phi (1 - k^2 \sin^2 \theta)^{1/2} d\theta \\ &= (\sin \phi) R_F(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) - \frac{1}{3} k^2 (\sin \phi)^3 R_D(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) \end{aligned}$$

$$\begin{aligned} \Pi(\phi, k, n) &= \int_0^\phi (1 + n \sin^2 \theta)^{-1} (1 - k^2 \sin^2 \theta)^{-1/2} d\theta \\ &= (\sin \phi) R_F(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) - \frac{n}{3} (\sin \phi)^3 R_J(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1, 1 + n \sin^2 \phi) \end{aligned}$$

$$\begin{aligned} D(\phi, k) &= \int_0^\phi \sin^2 \theta (1 - k^2 \sin^2 \theta)^{-1/2} d\theta \\ &= \frac{1}{3} (\sin \phi)^3 R_D(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) \end{aligned}$$

$$\begin{aligned} K(k) &= \int_0^{\pi/2} (1 - k^2 \sin^2 \theta)^{-1/2} d\theta \\ &= R_F(0, 1 - k^2, 1) \end{aligned}$$

$$\begin{aligned} E(k) &= \int_0^{\pi/2} (1 - k^2 \sin^2 \theta)^{1/2} d\theta \\ &= R_F(0, 1 - k^2, 1) - \frac{1}{3} k^2 R_D(0, 1 - k^2, 1) \end{aligned}$$

The function $R_C(x, y)$ is related to inverse trigonometric and inverse hyperbolic functions.

$$\begin{aligned}
 \ln x &= (x-1) R_c \left[\left(\frac{1+x}{2} \right), x \right] & 0 < x < \infty \\
 \sin^{-1} x &= x R_c(1-x^2, 1) & -1 \leq x \leq 1 \\
 \sinh^{-1} x &= x R_c(1+x^2, 1) & -\infty < x < \infty \\
 \cos^{-1} x &= \sqrt{1-x^2} R_c(x^2, 1) & 0 \leq x \leq 1 \\
 \cosh^{-1} x &= \sqrt{x^2-1} R_c(x^2, 1) & 1 \leq x < \infty \\
 \tan^{-1} x &= x R_c(1, 1+x^2) & -\infty < x < \infty \\
 \tanh^{-1} x &= x R_c(1, 1-x^2) & -1 < x < 1 \\
 \cot^{-1} x &= R_c(x^2, x^2+1) & 0 < x < \infty \\
 \coth^{-1} x &= R_c(x^2, x^2-1) & 1 < x < \infty
 \end{aligned}$$

ELK

This function evaluates the complete elliptic integral of the kind $K(x)$.

Function Return Value

ELK — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
X must be greater than or equal to 0 and less than 1.

FORTRAN 90 Interface

Generic: **ELK (X)**
 Specific: The specific interface names are **S_ELK** and **D_ELK**.

FORTRAN 77 Interface

Single: **ELK (X)**
 Double: The double precision name is **DELK**.

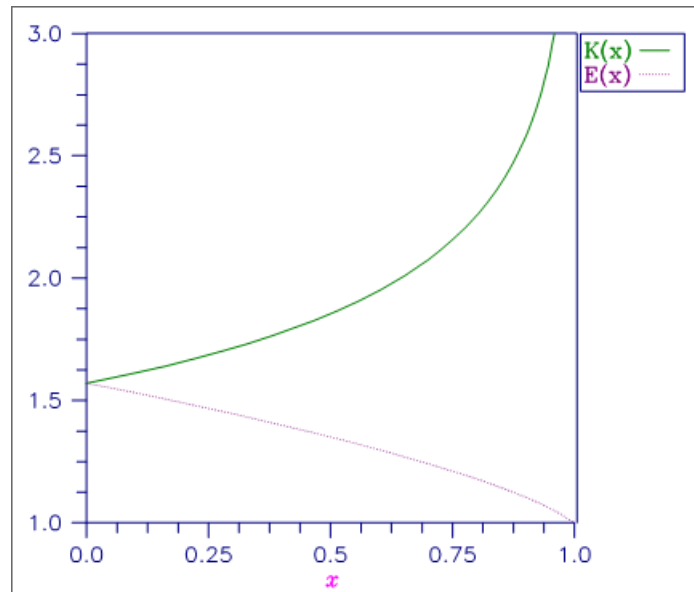
Description

The complete elliptic integral of the first kind is defined to be

$$K(x) = \int_0^{\pi/2} \frac{d\theta}{[1 - x \sin^2 \theta]^{1/2}} \quad \text{for } 0 \leq x < 1$$

The argument x must satisfy $0 \leq x < 1$; otherwise, **ELK** is set to $b = \mathbf{AMACH}(2)$, the largest representable floating-point number.

The function $K(x)$ is computed using the routine **ELRF** and the relation $K(x) = R_{\mathbf{F}}(0, 1 - x, 1)$.

Figure 17, Plot of $K(x)$ and $E(x)$

Example

In this example, $K(0)$ is computed and printed.

```

USE ELK_INT
USE UMACH_INT

IMPLICIT NONE                                Declare variables
!
INTEGER NOUT
REAL VALUE, X
!
X = 0.0
VALUE = ELK(X)                               Compute
!
CALL UMACH (2, NOUT)                          Print the results
WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ELK(', F6.3, ') = ', F6.3)
END

```

Output

```
ELK( 0.000) = 1.571
```

ELE

This function evaluates the complete elliptic integral of the second kind $E(x)$.

Function Return Value

ELE — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)
X must be greater than or equal to 0 and less than or equal to 1.

FORTRAN 90 Interface

Generic: **ELE** (**X**)
 Specific: The specific interface names are **S_ELE** and **D_ELE**.

FORTRAN 77 Interface

Single: **ELE** (**X**)
 Double: The double precision name is **DELE**.

Description

The complete elliptic integral of the second kind is defined to be

$$E(x) = \int_0^{\pi/2} [1 - x \sin^2 \theta]^{1/2} d\theta \quad \text{for } 0 \leq x < 1$$

The argument x must satisfy $0 \leq x < 1$; otherwise, **ELE** is set to $b = \text{AMACH}(2)$, the largest representable floating-point number.

The function $E(x)$ is computed using the routines **ELRF** and **ELRD**. The computation is done using the relation

$$E(x) = R_F(0, 1-x, 1) - \frac{x}{3} R_D(0, 1-x, 1)$$

For a plot of $E(x)$, see [Figure 17, "Plot of \$K\(x\)\$ and \$E\(x\)\$."](#)

Example

In this example, $E(0.33)$ is computed and printed.

```
      USE ELE_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X
!      Compute
      X = 0.33
      VALUE = ELE(X)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' ELE(', F6.3, ') = ', F6.3)
      END
```

Output

```
ELE( 0.330) = 1.432
```

ELRF

This function evaluates Carlson's incomplete elliptic integral of the first kind $R_F(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$.

Function Return Value

ELRF — Function value. (Output)

Required Arguments

- X** — First variable of the incomplete elliptic integral. (Input)
It must be nonnegative
- Y** — Second variable of the incomplete elliptic integral. (Input)
It must be nonnegative.
- Z** — Third variable of the incomplete elliptic integral. (Input)
It must be nonnegative.

FORTRAN 90 Interface

Generic: **ELRF** (**X**, **Y**, **Z**)
Specific: The specific interface names are **S_ELRF** and **D_ELRF**.

FORTRAN 77 Interface

Single: **ELRF** (**X**, **Y**, **Z**)
Double: The double precision name is **DELRF**.

Description

The Carlson's complete elliptic integral of the first kind is defined to be

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{[(t+x)(t+y)(t+z)]^{1/2}}$$

The arguments must be nonnegative and less than or equal to $b/5$. In addition, $x+y$, $x+z$, and $y+z$ must be greater than or equal to $5s$. Should any of these conditions fail, **ELRF** is set to b . Here, $b = \mathbf{AMACH}(2)$ is the largest and $s = \mathbf{AMACH}(1)$ is the smallest representable floating-point number.

The function **ELRF** is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

In this example, $R_F(0, 1, 2)$ is computed and printed.

```

      USE ELRF_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X, Y, Z
!      Compute
      X = 0.0
      Y = 1.0
      Z = 2.0
      VALUE = ELRF(X, Y, Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, Y, Z, VALUE
99999 FORMAT (' ELRF(', F6.3, ', ', F6.3, ', ', F6.3, ') = ', F6.3)
      END

```

Output

```
ELRF( 0.000, 1.000, 2.000) = 1.311
```

ELRD

This function evaluates Carlson's incomplete elliptic integral of the second kind $R_D(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

Function Return Value

ELRD — Function value. (Output)

Required Arguments

X — First variable of the incomplete elliptic integral. (Input)
It must be nonnegative.

Y — Second variable of the incomplete elliptic integral. (Input)
It must be nonnegative.

Z — Third variable of the incomplete elliptic integral. (Input)
It must be positive.

FORTRAN 90 Interface

Generic: **ELRD** (**X**, **Y**, **Z**)

Specific: The specific interface names are **S_ELRD** and **D_ELRD**.

FORTRAN 77 Interface

Single: **ELRD** (**X**, **Y**, **Z**)

Double: The double precision name is **DELRD**.

Description

The Carlson's complete elliptic integral of the second kind is defined to be

$$R_D(x, y, z) = \frac{3}{2} \int_0^\infty \frac{dt}{\left[(t+x)(t+y)(t+z)^3 \right]^{1/2}}$$

The arguments must be nonnegative and less than or equal to $0.69(-\ln \epsilon)^{1/9} s^{-2/3}$ where $\epsilon = \text{AMACH}(4)$ is the machine precision, $s = \text{AMACH}(1)$ is the smallest representable positive number. Furthermore, $x + y$ and z must be greater than $\max\{3s^{2/3}, 3/b^{2/3}\}$, where $b = \text{AMACH}(2)$ is the largest floating-point number. If any of these conditions are false, then **ELRD** is set to b .

The function **ELRD** is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

In this example, $R_D(0, 2, 1)$ is computed and printed.

```

      USE ELRD_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      REAL VALUE, X, Y, Z
!      Compute
      X = 0.0
      Y = 2.0
      Z = 1.0
      VALUE = ELRD(X, Y, Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, Y, Z, VALUE
99999 FORMAT (' ELRD( ', F6.3, ', ', F6.3, ', ', F6.3, ') = ', F6.3)
      END

```

Output

```
ELRD( 0.000, 2.000, 1.000) = 1.797
```

ELRJ

This function evaluates Carlson's incomplete elliptic integral of the third kind $R_f(\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{RHO})$

Function Return Value

ELRJ — Function value. (Output)

Required Arguments

X — First variable of the incomplete elliptic integral. (Input)

It must be nonnegative.

Y — Second variable of the incomplete elliptic integral. (Input)

It must be nonnegative.

Z — Third variable of the incomplete elliptic integral. (Input)

It must be nonnegative.

RHO — Fourth variable of the incomplete elliptic integral. (Input)

It must be positive.

FORTRAN 90 Interface

Generic: `ELRJ (X, Y, Z, RHO)`

Specific: The specific interface names are `S_ELRJ` and `D_ELRJ`.

FORTRAN 77 Interface

Single: `ELRJ (X, Y, Z, RHO)`

Double: The double precision name is `DELRJ`.

Description

The Carlson's complete elliptic integral of the third kind is defined to be

$$R_J(x, y, z, \rho) = \frac{3}{2} \int_0^{\infty} \frac{dt}{[(t+x)(t+y)(t+z)(t+\rho)^2]^{1/2}}$$

The arguments must be nonnegative. In addition, $x+y$, $x+z$, $y+z$ and ρ must be greater than or equal to $(5s)^{1/3}$ and less than or equal to $.3(b/5)^{1/3}$, where $s = \text{AMACH}(1)$ is the smallest representable floating-point number. Should any of these conditions fail, **ELRJ** is set to $b = \text{AMACH}(2)$, the largest floating-point number.

The function **ELRJ** is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Example

In this example, $R_J(2, 3, 4, 5)$ is computed and printed.

```

      USE ELRJ_INT
      USE UMACH_INT

      IMPLICIT NONE
!           Declare variables
      INTEGER NOUT
      REAL RHO, VALUE, X, Y, Z
!           Compute
      X = 2.0
      Y = 3.0
      Z = 4.0
      RHO = 5.0
      VALUE = ELRJ(X, Y, Z, RHO)
!           Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, Y, Z, RHO, VALUE
99999 FORMAT (' ELRJ(', F6.3, ', ', F6.3, ', ', F6.3, ', ', F6.3, &
             ' ) = ', F6.3)
      END

```

Output

```
ELRJ( 2.000, 3.000, 4.000, 5.000) = 0.143
```

ELRC

This function evaluates an elementary integral from which inverse circular functions, logarithms and inverse hyperbolic functions can be computed.

Function Return Value

ELRC — Function value. (Output)

Required Arguments

- X** — First variable of the incomplete elliptic integral. (Input)
It must be nonnegative and satisfy the conditions given in Comments.
- Y** — Second variable of the incomplete elliptic integral. (Input)
It must be positive and satisfy the conditions given in Comments.

FORTRAN 90 Interface

Generic: **ELRC (X, Y)**
Specific: The specific interface names are **S_ELRC** and **D_ELRC**.

FORTRAN 77 Interface

Single: **ELRC (X, Y)**
Double: The double precision name is **DELRC**.

Description

The special case of Carlson's complete elliptic integral of the first kind is defined to be

$$R_C(x, y) = \frac{1}{2} \int_0^{\infty} \frac{dt}{[(t+x)(t+y)^2]^{1/2}}$$

The argument x must be nonnegative, y must be positive, and $x + y$ must be less than or equal to $b/5$ and greater than or equal to $5s$. If any of these conditions are false, then **ELRC** is set to b . Here, $b = \mathbf{AMACH}(2)$ is the largest and $s = \mathbf{AMACH}(1)$ is the smallest representable floating-point number.

The function `ELRC` is based on the code by Carlson and Notis (1981) and the work of Carlson (1979).

Comments

The sum $X + Y$ must be greater than or equal to `ARGMIN` and both X and Y must be less than or equal to `ARGMAX`. `ARGMIN` = $s * 5$ and `ARGMAX` = $b/5$, where s is the machine minimum (`AMACH(1)`) and b is the machine maximum (`AMACH(2)`).

Example

In this example, $R_c(2.25, 2.0)$ is computed and printed.

```
      USE ELRC_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      REAL VALUE, X, Y
!
      X = 0.0
      Y = 1.0
      VALUE = ELRC(X, Y)                          Compute
!
      CALL UMACH (2, NOUT)                         Print the results
      WRITE (NOUT,99999) X, Y, VALUE
99999 FORMAT (' ELRC(', F6.3, ', ', F6.3, ') = ', F6.3)
      END
```

Output

```
ELRC( 0.000, 1.000) = 1.571
```


Elliptic and Related Functions

Routines

10.1 Weierstrass Elliptic and Related Functions

Lemniscatic case	CWPL	272
Lemniscatic case derivative	CWPLD	274
Equianharmonic case	CWPQ	276
Equianharmonic case derivative	CWPQD	278

10.2 Jacobi Elliptic Functions

Jacobi function $\operatorname{sn}(x, m)$ (real argument)	EJSN	280
Jacobi function $\operatorname{cn}(x, m)$ (real argument)	EJCN	283
Jacobi function $\operatorname{dn}(x, m)$ (real argument)	EJDN	286

Usage Notes

Elliptic functions are doubly periodic, single-valued complex functions of a single variable that are analytic, except at a finite number of poles. Because of the periodicity, we need consider only the fundamental period parallelogram. The irreducible number of poles, counting multiplicities, is the *order* of the elliptic function. The simplest, non-trivial, elliptic functions are of order two.

The Weierstrass elliptic functions, $\wp(z, \omega, \omega')$ have a double pole at $z = 0$ and so are of order two. Here, 2ω and $2\omega'$ are the periods.

The Jacobi elliptic functions each have two simple poles and so are also of order two. The period of the functions is as follows:

Function	Periods
$\operatorname{sn}(x, m)$	$4K(m) \quad 2iK'(m)$
$\operatorname{cn}(x, m)$	$4K(m) \quad 4iK'(m)$
$\operatorname{dn}(x, m)$	$2K(m) \quad 4iK'(m)$

The function $K(m)$ is the complete elliptic integral, see [ELK](#), and $K'(m) = K(1 - m)$.

CWPL

This function evaluates the Weierstrass' \wp function in the lemniscatic case for complex argument with unit period parallelogram.

Function Return Value

CWPL — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **CWPL** (Z)

Specific: The specific interface names are **C_CWPL** and **Z_CWPL**.

FORTRAN 77 Interface

Complex: **CWPL** (Z)

Double complex: The double complex name is **ZWPL**.

Description

The Weierstrass' \wp function, $\wp(z) = \wp(z | \omega, \omega')$, is an elliptic function of order two with periods 2ω and $2\omega'$ and a double pole at $z = 0$. **CWPL**(Z) computes $\wp(z | \omega, \omega')$ with $2\omega = 1$ and $2\omega' = i$.

The input argument is first reduced to the fundamental parallelogram of all z satisfying $-1/2 \leq \Re z \leq 1/2$ and $-1/2 \leq \Im z \leq 1/2$. Then, a rational approximation is used.

All arguments are valid with the exception of the lattice points $z = m + ni$, which are the poles of **CWPL**. If the argument is a lattice point, then $b = \mathbf{AMACH}(2)$, the largest floating-point number, is returned. If the argument has modulus greater than $10\epsilon^{-1}$, then NaN (not a number) is returned. Here, $\epsilon = \mathbf{AMACH}(4)$ is the machine precision.

Function **CWPL** is based on code by Eckhardt (1980). Also, see Eckhardt (1977).

Example

In this example, $\wp(0.25 + 0.25i)$ is computed and printed.

```
      USE CWPL_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (0.25, 0.25)
      VALUE = CWPL(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CWPL(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
CWPL( 0.250, 0.250) = ( 0.000,-6.875)
```

CWPLD

This function evaluates the first derivative of the Weierstrass' \wp function in the lemniscatic case for complex argument with unit period parallelogram.

Function Return Value

CWPLD — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **CWPLD** (Z)

Specific: The specific interface names are **C_CWPLD** and **Z_CWPLD**.

FORTRAN 77 Interface

Complex: **CWPLD** (Z)

Double complex: The double complex name is **ZWPLD**.

Description

The Weierstrass' \wp function, $\wp(z) = \wp(z | \omega, \omega')$, is an elliptic function of order two with periods 2ω and $2\omega'$ and a double pole at $z = 0$. **CWPLD**(Z) computes the derivative of $\wp(z | \omega, \omega')$ with $2\omega = 1$ and $2\omega' = i$. **CWPL** computes $\wp(z | \omega, \omega')$.

The input argument is first reduced to the fundamental parallelogram of all z satisfying $-1/2 \leq \Re z \leq 1/2$ and $-1/2 \leq \Im z \leq 1/2$. Then, a rational approximation is used.

All arguments are valid with the exception of the lattice points $z = m + ni$, which are the poles of **CWPL**. If the argument is a lattice point, then $b = \mathbf{AMACH}(2)$, the largest floating-point number, is returned.

Function **CWPLD** is based on code by Eckhardt (1980). Also, see Eckhardt (1977).

Example

In this example, $\wp(0.25 + 0.25i)$ is computed and printed.

```
      USE CWPLD_INT
      USE UMACH_INT

      IMPLICIT NONE
!      Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!      Compute
      Z = (0.25, 0.25)
      VALUE = CWPLD(Z)
!      Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CWPLD(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
CWPLD( 0.250, 0.250) = (36.054,36.054)
```

CWPQ

This function evaluates the Weierstrass' \wp function in the equianharmonic case for complex argument with unit period parallelogram.

Function Return Value

CWPQ — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: CWPQ (Z)

Specific: The specific interface names are C_CWPQ and Z_CWPQ.

FORTRAN 77 Interface

Complex: CWPQ (Z)

Double complex: The double complex name is ZWPQ.

Description

The Weierstrass' \wp function, $\wp(z) = \wp(z \mid \omega, \omega')$, is an elliptic function of order two with periods 2ω and $2\omega'$ and a double pole at $z = 0$. CWPQ(Z) computes $\wp(z \mid \omega, \omega')$ with

$$4\omega = 1 - i\sqrt{3} \quad \text{and} \quad 4\omega' = 1 + i\sqrt{3}$$

The input argument is first reduced to the fundamental parallelogram of all z satisfying

$$-1/2 \leq \Re z \leq 1/2 \quad \text{and} \quad -\sqrt{3}/4 \leq \Im z \leq \sqrt{3}/4$$

Then, a rational approximation is used.

All arguments are valid with the exception of the lattice points

$$z = m(1 - i\sqrt{3}) + n(1 + i\sqrt{3})$$

which are the poles of `CWPQ`. If the argument is a lattice point, then $b = \text{AMACH}(2)$, the largest floating-point number, is returned. If the argument has modulus greater than $10\epsilon^{-1}$, then NaN (not a number) is returned. Here, $\epsilon = \text{AMACH}(4)$ is the machine precision.

Function `CWPQ` is based on code by Eckhardt (1980). Also, see Eckhardt (1977).

Example

In this example, $\wp(0.25 + 0.14437567i)$ is computed and printed.

```

      USE CWPQ_INT
      USE UMACH_INT

      IMPLICIT NONE                                Declare variables
!
      INTEGER NOUT
      COMPLEX VALUE, Z
!
      Z = (0.25, 0.14437567)                       Compute
      VALUE = CWPQ(Z)
!
      CALL UMACH (2, NOUT)                          Print the results
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CWPQ(', F6.3, ', ', F6.3, ') = (', &
             F7.3, ', ', F7.3, ')')
      END

```

Output

```
CWPQ( 0.250, 0.144) = ( 5.895,-10.216)
```


CWPQD

This function evaluates the first derivative of the Weierstrass' \wp function in the equianharmonic case for complex argument with unit period parallelogram.

Function Return Value

CWPQD — Complex function value. (Output)

Required Arguments

Z — Complex argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: CWPQD (Z)

Specific: The specific interface names are C_CWPQD and Z_CWPQD.

FORTRAN 77 Interface

Complex: CWPQD (Z)

Double complex: The double complex name is ZWPQD.

Description

The Weierstrass' \wp function, $\wp(z) = \wp(z | \omega, \omega')$, is an elliptic function of order two with periods 2ω and $2\omega'$ and a double pole at $z = 0$. CWPQD(Z) computes the derivative of $\wp(z | \omega, \omega')$ with

$$4\omega = 1 - i\sqrt{3} \quad \text{and} \quad 4\omega' = 1 + i\sqrt{3}$$

CWPQ computes $\wp(z | \omega, \omega')$.

The input argument is first reduced to the fundamental parallelogram of all z satisfying

$$-1/2 \leq \Re z \leq 1/2 \quad \text{and} \quad -\sqrt{3}/4 \leq \Im z \leq \sqrt{3}/4$$

Then, a rational approximation is used.

All arguments are valid with the exception of the lattice points

$$z = m(1 - i\sqrt{3}) + n(1 + i\sqrt{3})$$

which are the poles of `CWPQD`. If the argument is a lattice point, then $b = \text{AMACH}(2)$, the largest floating-point number, is returned.

Function `CWPQD` is based on code by Eckhardt (1980). Also, see Eckhardt (1977).

Example

In this example, $\wp(0.25 + 0.14437567i)$ is computed and printed.

```

      USE CWPQD_INT
      USE UMACH_INT

      IMPLICIT NONE
!          Declare variables
      INTEGER NOUT
      COMPLEX VALUE, Z
!          Compute
      Z = (0.25, 0.14437567)
      VALUE = CWPQD(Z)
!          Print the results
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, VALUE
99999 FORMAT (' CWPQD(', F6.3, ',', F6.3, ') = (', &
             F6.3, ',', F6.3, ')')
      END

```

Output

```
CWPQD( 0.250, 0.144) = ( 0.028,85.934)
```

EJSN

This function evaluates the Jacobi elliptic function $\operatorname{sn}(x, m)$.

Function Return Value

EJSN — Real or complex function value. (Output)

Required Arguments

X — Real or complex argument for which the function value is desired. (Input)

AM — Parameter of the elliptic function ($m = k^2$). (Input)

FORTRAN 90 Interface

Generic: **EJSN (X, AM)**

Specific: The specific interface names are **S_EJSN**, **D_EJSN**, **C_EJSN**, and **Z_EJSN**

FORTRAN 77 Interface

Single: **EJSN (X, AM)**

Double: The double precision name is **DEJSN**.

Complex: The complex name is **CEJSN**.

Double Complex: The double complex name is **ZEJSN**.

Description

The Jacobi elliptic function $\operatorname{sn}(x, m) = \sin \phi$, where the amplitude ϕ is defined by the following:

$$x = \int_0^{\phi} \frac{d\theta}{(1 - m \sin^2 \theta)^{1/2}}$$

The function $\operatorname{sn}(x, m)$ is computed by first applying, if necessary, a Jacobi transformation so that the parameter, m , is between zero and one. Then, a descending Landen (Gauss) transform is applied until the parameter is small. The small parameter approximation is then applied.

Comments

Informational errors

Type	Code	Description
3	2	The result is accurate to less than one half precision because $ x $ is too large.
3	2	The result is accurate to less than one half precision because $ \text{REAL}(z) $ is too large.
3	3	The result is accurate to less than one half precision because $ \text{AIMAG}(z) $ is too large.
3	5	Landen transform did not converge. Result may not be accurate. This should never occur.

Examples

Example 1

In this example, $\text{sn}(1.5, 0.5)$ is computed and printed.

```

      USE EJSN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM, VALUE, X
!
      AM = 0.5
      X = 1.5
      VALUE = EJSN(X, AM)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, AM, VALUE
99999 FORMAT (' EJSN(', F6.3, ', ', F6.3, ') = ', F6.3)
      END

```

Output

```
EJSN( 1.500, 0.500) = 0.968
```

Example 2

In this example, $\text{sn}(1.5 + 0.3i, 0.5)$ is computed and printed.

```

      USE EJSN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM

```

```
!      COMPLEX      VALUE, Z
      Z      = (1.5, 0.3)
      AM     = 0.5
      VALUE  = EJSN(Z, AM)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, AM, VALUE
99999 FORMAT (' EJSN(', F6.3, ', ', F6.3, ') = (', &
           F6.3, ', ', F6.3, ')')
      END
```

Output

```
EJSN(( 1.500, 0.300), 0.500) = ( 0.993, 0.054)
```

EJC�

This function evaluates the Jacobi elliptic function $\text{cn}(x, m)$.

Function Return Value

EJC� — Real or complex function value. (Output)

Required Arguments

X — Real or complex argument for which the function value is desired. (Input)

AM — Parameter of the elliptic function ($m = k^2$). (Input)

FORTRAN 90 Interface

Generic: **EJC� (X, AM)**

Specific: The specific interface names are **S_EJC�**, **D_EJC�**, **C_EJC�**, and **Z_EJC�**.

FORTRAN 77 Interface

Single: **EJC� (X, AM)**

Double: The double precision name is **DEJC�**.

Complex: The complex name is **CEJC�**.

Double Complex: The double complex name is **ZEJC�**.

Description

The Jacobi elliptic function $\text{cn}(x, m) = \cos \phi$, where the amplitude ϕ is defined by the following:

$$x = \int_0^{\phi} \frac{d\theta}{(1 - m \sin^2 \theta)^{1/2}}$$

The function $\text{cn}(x, m)$ is computed by first applying, if necessary, a Jacobi transformation so that the parameter, m , is between zero and one. Then, a descending Landen (Gauss) transform is applied until the parameter is small. The small parameter approximation is then applied.

Comments

Informational errors

Type	Code	Description
3	2	The result is accurate to less than one half precision because $ x $ is too large.
3	2	The result is accurate to less than one half precision because $ \text{REAL}(z) $ is too large.
3	3	The result is accurate to less than one half precision because $ \text{AIMAG}(z) $ is too large.
3	5	Landen transform did not converge. Result may not be accurate. This should never occur.

Examples

Example 1

In this example, $\text{cn}(1.5, 0.5)$ is computed and printed.

```

      USE EJCN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM, VALUE, X
!
      AM = 0.5
      X = 1.5
      VALUE = EJCN(X, AM)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, AM, VALUE
99999 FORMAT (' EJCN(', F6.3, ', ', F6.3, ') = ', F6.3)
      END

```

Output

```
EJCN( 1.500, 0.500) = 0.250
```

Example 2

In this example, $\text{cn}(1.5 + 0.3i, 0.5)$ is computed and printed.

```

      USE EJCN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM

```

```
!      COMPLEX      VALUE, Z
!      Z          = (1.5, 0.3)
!      AM         = 0.5
!      VALUE     = EJC�(Z, AM)
!
!      CALL UMACH (2, NOUT)
!      WRITE (NOUT,99999) Z, AM, VALUE
99999 FORMAT (' EJC�(', F6.3, ', ', F6.3, '), ', F6.3, ') = (', &
!      F6.3, ', ', F6.3, ')')
!      END
```

Output

```
EJC�(( 1.500, 0.300), 0.500) = ( 0.251,-0.212)
```


EJDN

This function evaluates the Jacobi elliptic function $\operatorname{dn}(x, m)$.

Function Return Value

EJDN — Real or complex function value. (Output)

Required Arguments

X — Real or complex argument for which the function value is desired. (Input)

AM — Parameter of the elliptic function ($m = k^2$). (Input)

FORTRAN 90 Interface

Generic: **EJDN (X, AM)**

Specific: The specific interface names are **S_EJDN**, **D_EJDN**, **C_EJDN**, and **Z_EJDN**.

FORTRAN 77 Interface

Single: **EJDN (X, AM)**

Double: The double precision name is **DEJDN**.

Complex: The complex precision name is **CEJDN**.

Double Complex: The double complex precision name is **ZEJDN**.

Description

The Jacobi elliptic function $\operatorname{dn}(x, m) = (1 - m \sin^2 \phi)^{1/2}$, where the amplitude ϕ is defined by the following:

$$x = \int_0^{\phi} \frac{d\theta}{(1 - m \sin^2 \theta)^{1/2}}$$

The function $\operatorname{dn}(x, m)$ is computed by first applying, if necessary, a Jacobi transformation so that the parameter, m , is between zero and one. Then, a descending Landen (Gauss) transform is applied until the parameter is small. The small parameter approximation is then applied.

Comments

Informational errors

Type	Code	Description
3	2	The result is accurate to less than one half precision because $ x $ is too large.
3	2	The result is accurate to less than one half precision because $ \text{REAL}(z) $ is too large.
3	3	The result is accurate to less than one half precision because $ \text{AIMAG}(z) $ is too large.
3	5	Landen transform did not converge. Result may not be accurate. This should never occur.

Examples

Example 1

In this example, $\text{dn}(1.5, 0.5)$ is computed and printed.

```

      USE EJDN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM, VALUE, X
!
      AM = 0.5
      X = 1.5
      VALUE = EJDN(X, AM)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, AM, VALUE
99999 FORMAT (' EJDN(', F6.3, ', ', F6.3, ') = ', F6.3)
      END

```

Output

```
EJDN( 1.500, 0.500) = 0.729
```

Example 2

In this example, $\text{dn}(1.5 + 0.3j, 0.5)$ is computed and printed.

```

      USE EJDN_INT
      USE UMACH_INT

      IMPLICIT NONE
!
      INTEGER NOUT
      REAL AM

```

```
      COMPLEX      VALUE, Z
!
      Z          = (1.5, 0.3)
      AM         = 0.5
      VALUE     = EJDN(Z, AM)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) Z, AM, VALUE
99999 FORMAT (' EJDN(', F6.3, ', ', F6.3, ') = (', &
             F6.3, ', ', F6.3, ')')
      END
```

Output

```
EJDN(( 1.500, 0.300), 0.500) = ( 0.714,-0.037)
```

Probability Distribution Functions and Inverses

Routines

11.1 Discrete Random Variables: Cumulative Distribution Functions and Probability Density Function

Binomial cumulative distribution function	BINDF	296
Binomial probability density function	BINPR	298
Geometric cumulative distribution function	GEODF	301
Inverse of Geometric cumulative distribution function	GEOIN	303
Geometric probability density function	GEOPR	305
Hypergeometric cumulative distribution function	HYPDF	307
Hypergeometric probability density function	HYPPR	310
Poisson cumulative distribution function	POIDF	313
Poisson probability density function	POIPR	315
Discrete uniform cumulative distribution function	UNDDF	318
Inverse of discrete uniform cumulative distribution function	UNDIN	320
Discrete uniform probability density function	UNDPR	322

11.2 Continuous Random Variables: Distribution Functions and Their Inverses

Kolmogorov-Smirnov one-sided statistic cumulative distribution function	AKS1DF	324
Kolmogorov-Smirnov two-sided statistic cumulative distribution function	AKS2DF	327
Lognormal cumulative distribution function	ALNDF	330
Inverse of the lognormal cumulative distribution function	ALNIN	332
Lognormal probability density function	ALNPR	334
Normal (Gaussian) cumulative distribution function	ANORDF	336
Inverse of the normal cumulative distribution function	ANORIN	339
Normal (Gaussian) probability density function	ANORPR	341
Beta cumulative distribution function	BETDF	343
Inverse of the beta cumulative distribution function	BETIN	346
Beta probability density function	BETPR	348
Noncentral beta cumulative distribution function	BETNDF	350
Inverse of the noncentral beta cumulative distribution function	BETNIN	353
Noncentral beta probability density function	BETNPR	356
Bivariate normal cumulative distribution function	BNRDF	359
Chi-squared cumulative distribution function	CHIDF	361

Inverse of the chi-squared cumulative distribution function	CHIIN	364
Chi-squared probability density function	CHIPR	366
Noncentral chi-squared cumulative distribution function	CSNDF	368
Inverse of the noncentral chi-squared cumulative distribution function	CSNIN	371
Noncentral chi-squared probability density function	CSNPR	373
Exponential distribution cumulative function	EXPDF	376
Inverse of the exponential cumulative distribution function	EXPIN	378
Exponential probability density function	EXPPR	380
Extreme value cumulative distribution function	EXVDF	382
Inverse of the Extreme value cumulative distribution function	EXVIN	384
Extreme value probability density function	EXVPR	386
F cumulative distribution function	FDF	388
Inverse of the F cumulative distribution function	FIN	391
F probability density function	FPR	393
Noncentral F cumulative distribution function	FNDF	395
Inverse of the noncentral F cumulative distribution function	FNIN	398
Noncentral F probability density function	FNPR	401
Gamma cumulative distribution function	GAMDF	404
Inverse of the gamma cumulative distribution function	GAMIN	407
Gamma probability density function	GAMPR	409
Rayleigh's cumulative distribution function	RALDF	411
Inverse of the Rayleigh's cumulative distribution function	RALIN	413
Rayleigh's probability density function	RALPR	415
Student's t cumulative distribution function	TDF	417
Inverse of the Student's t cumulative distribution function	TIN	420
Student's t probability density function	TPR	422
Noncentral Student's t cumulative distribution function	TNDF	424
Inverse of the noncentral Student's t cumulative distribution function . .	TNIN	427
Noncentral Student's t probability density function	TNPR	429
Uniform cumulative distribution function	UNDF	431
Inverse of the uniform cumulative distribution function	UNIN	433
Uniform probability density function	UNPR	435
Weibull cumulative distribution function	WBLDF	437
Inverse of the Weibull cumulative distribution function	WBLIN	439
Weibull probability density function	WBLPR	441
11.3 General Continuous Random Variables		
Distribution function given ordinates of density	GCDF	443
Inverse of distribution function given ordinates of density	GCIN	447
Inverse of distribution function given subprogram	GFNIN	451

Usage Notes

Definitions and discussions of the terms basic to this chapter can be found in Johnson and Kotz (1969, 1970a, 1970b). These are also good references for the specific distributions.

In order to keep the calling sequences simple, whenever possible, the routines in this chapter are written for standard forms of statistical distributions. Hence, the number of parameters for any given distribution may be fewer than the number often associated with the distribution. For example, while a gamma distribution is often characterized by two parameters (or even a third, "location"), there is only one parameter that is necessary, the "shape." The "scale" parameter can be used to scale the variable to the *standard* gamma distribution. For another example, the functions relating to the normal distribution, [ANORDF](#) and [ANORIN](#), are for a normal distribution with mean equal to zero and variance equal to one. For other means and variances, it is very easy for the user to standardize the variables by subtracting the mean and dividing by the square root of the variance.

The *distribution function* for the (real, single-valued) random variable X is the function F defined for all real x by

$$F(x) = \text{Prob}(X \leq x)$$

where $\text{Prob}(\cdot)$ denotes the probability of an event. The distribution function is often called the *cumulative distribution function* (CDF).

For distributions with finite ranges, such as the beta distribution, the CDF is 0 for values less than the left endpoint and 1 for values greater than the right endpoint. The routines in this chapter return the correct values for the distribution functions when values outside of the range of the random variable are input, but warning error conditions are set in these cases.

Discrete Random Variables

For discrete distributions, the function giving the probability that the random variable takes on specific values is called the *probability function*, defined by

$$p(x) = \text{Prob}(X = x)$$

The "PR" routines in this chapter evaluate probability functions.

The CDF for a discrete random variable is

$$F(x) = \sum_A p(k)$$

$$\sqrt{a^2 + b^2}$$

where A is the set such that $k \leq x$. The “DF” routines in this chapter evaluate cumulative distribution functions. Since the distribution function is a step function, its inverse does not exist uniquely.

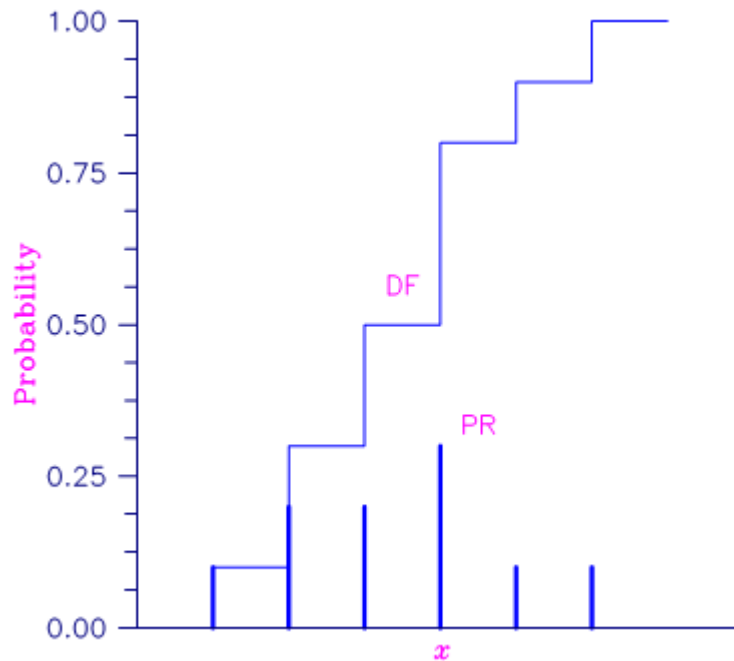


Figure 18, Discrete Random Variable

In the plot above, a routine like `BINPR` in this chapter evaluates the individual probability, given X . A routine like `BINDF` would evaluate the sum of the probabilities up to and including the probability at X .

Continuous Distributions

For continuous distributions, a probability function, as defined above, would not be useful because the probability of any given point is 0. For such distributions, the useful analog is the *probability density function* (PDF). The integral of the PDF is the probability over the interval; if the continuous random variable X has PDF f , then

$$\text{Prob}(a < X \leq b) = \int_a^b f(x) dx$$

The relationship between the CDF and the PDF is

$$F(x) = \int_{-\infty}^x f(t) dt$$

as shown below.

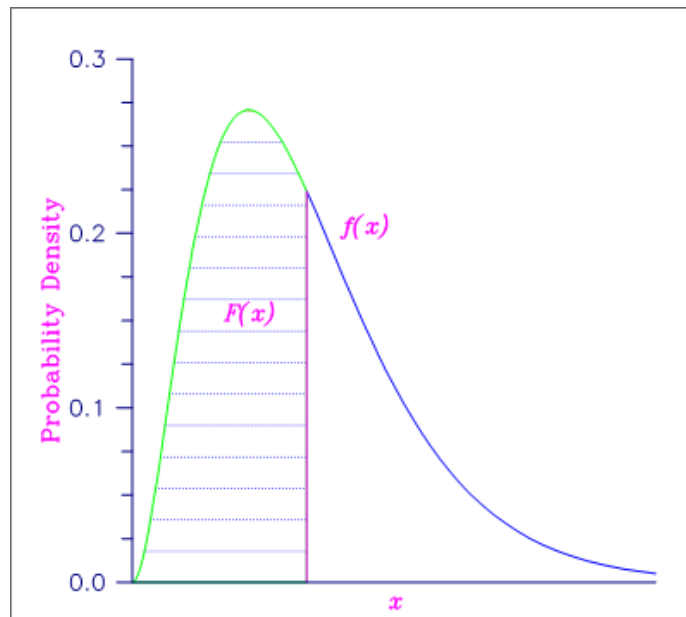


Figure 19, Probability Density Function

The “**DF**” routines for continuous distributions in this chapter evaluate cumulative distribution functions, just as the ones for discrete distributions.

For (absolutely) continuous distributions, the value of $F(x)$ uniquely determines x within the support of the distribution. The “**IN**” routines in this chapter compute the inverses of the distribution functions; that is, given $F(x)$ (called “**P**” for “probability”), a routine like **BETIN** computes x . The inverses are defined only over the open interval $(0, 1)$.

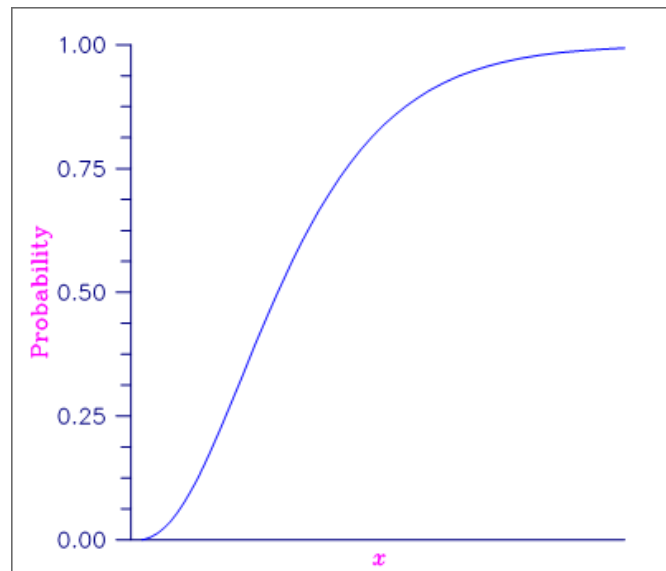


Figure 20, Cumulative Probability Distribution Function

There are two routines in this chapter that deal with general continuous distribution functions. The routine `GCDF` computes a distribution function using values of the density function, and the routine `GCIN` computes the inverse. These two routines may be useful when the user has an estimate of a probability density.

Additional Comments

Whenever a probability close to 1.0 results from a call to a distribution function or is to be input to an inverse function, it is often impossible to achieve good accuracy because of the nature of the representation of numeric values. In this case, it may be better to work with the complementary distribution function (one minus the distribution function). If the distribution is symmetric about some point (as the normal distribution, for example) or is reflective about some point (as the beta distribution, for example), the complementary distribution function has a simple relationship with the distribution function. For example, to evaluate the standard normal distribution at 4.0, using `ANORIN` directly, the result to six places is 0.999968. Only two of those digits are really useful, however. A more useful result may be 1.000000 minus this value, which can be obtained to six significant figures as 3.16713E-05 by evaluating `ANORIN` at -4.0. For the normal distribution, the two values are related by $\Phi(x) = 1 - \Phi(-x)$, where $\Phi(\cdot)$ is the normal distribution function. Another example is the beta distribution with parameters 2 and 10. This distribution is skewed to the right; so evaluating `BETDF` at 0.7, we obtain 0.999953. A more precise result is obtained by evaluating `BETDF` with parameters 10 and 2 at 0.3. This yields 4.72392E-5. (In both of these examples, it is wise not to trust the last digit.)

Many of the algorithms used by routines in this chapter are discussed by Abramowitz and Stegun (1964). The algorithms make use of various expansions and recursive relationships, and often use different methods in different regions.

Cumulative distribution functions are defined for all real arguments; however, if the input to one of the distribution functions in this chapter is outside the range of the random variable, an error of Type 1 is issued, and the output is set to zero or one, as appropriate. A Type 1 error is of lowest severity, a “note;” and, by default, no printing or stopping of the program occurs. The other common errors that occur in the routines of this chapter are Type 2, “alert;” for a function value being set to zero due to underflow; Type 3, “warning;” for considerable loss of accuracy in the result returned; and Type 5, “terminal;” for incorrect and/ or inconsistent input, complete loss of accuracy in the result returned, or inability to represent the result (because of overflow). When a Type 5 error occurs, the result is set to NaN (not a number, also used as a missing value code, obtained by IMSL routine **AMACH**(6). (See the section [User Errors](#) in the Reference Material.)

BINDF

This function evaluates the binomial cumulative distribution function.

Function Return Value

BINDF — Function value, the probability that a binomial random variable takes a value less than or equal to K . (Output)

BINDF is the probability that K or fewer successes occur in N independent Bernoulli trials, each of which has a PIN probability of success.

Required Arguments

K — Argument for which the binomial distribution function is to be evaluated. (Input)

N — Number of Bernoulli trials. (Input)

PIN — Probability of success on each independent trial. (Input)

FORTRAN 90 Interface

Generic: **BINDF** (**K**, **N**, **PIN**)

Specific: The specific interface names are **S_BINDF** and **D_BINDF**.

FORTRAN 77 Interface

Single: **BINDF** (**K**, **N**, **PIN**)

Double: The double precision name is **DBINDF**.

Description

Function **BINDF** evaluates the cumulative distribution function of a binomial random variable with parameters n and p where $n = N$ and $p = PIN$. It does this by summing probabilities of the random variable taking on the specific values in its range. These probabilities are computed by the recursive relationship

$$\Pr(X = j) = \frac{(n+1-j)p}{j(1-p)} \Pr(X = j-1)$$

To avoid the possibility of underflow, the probabilities are computed forward from 0, if k is not greater than n times p , and are computed backward from n , otherwise. The smallest positive machine number, ϵ , is used as the starting value for summing the probabilities, which are rescaled by $(1-p)^n \epsilon$ if forward computation is performed and by $p^n \epsilon$ if backward computation is done. For the special case of $p = 0$, **BINDF** is set to 1; and for the case $p = 1$, **BINDF** is set to 1 if $k = n$ and to 0 otherwise.

Comments

Informational errors

Type	Code	Description
1	3	The input argument, κ , is less than zero.
1	4	The input argument, κ , is greater than the number of Bernoulli trials, n .

Example

Suppose X is a binomial random variable with $n = 5$ and $p = 0.95$. In this example, we find the probability that X is less than or equal to 3.

```

USE UMACH_INT
USE BINDF_INT

IMPLICIT NONE
INTEGER K, N, NOUT
REAL PIN, PR
!
CALL UMACH (2, NOUT)
K = 3
N = 5
PIN = 0.95
PR = BINDF(K,N, PIN)
WRITE (NOUT,99999) PR
99999 FORMAT (' The probability that X is less than or equal to 3 is ' &
, F6.4)
END

```

Output

```
The probability that X is less than or equal to 3 is 0.0226
```

BINPR

This function evaluates the binomial probability density function.

Function Return Value

BINPR Function value, the probability that a binomial random variable takes a value equal to k . (Output)

Required Arguments

K — Argument for which the binomial probability function is to be evaluated. (Input)

N — Number of Bernoulli trials. (Input)

PIN — Probability of success on each independent trial. (Input)

FORTRAN 90 Interface

Generic: `BINPR (K, N, PIN)`

Specific: The specific interface names are `S_BINPR` and `D_BINPR`.

FORTRAN 77 Interface

Single: `BINPR (K, N, PIN)`

Double: The double precision name is `DBINPR`.

Description

The function **BINPR** evaluates the probability that a binomial random variable with parameters n and p where $p = \text{PIN}$ takes on the value k . It does this by computing probabilities of the random variable taking on the values in its range less than (or the values greater than) k . These probabilities are computed by the recursive relationship

$$\Pr(X = j) = \frac{(n + 1 - j)p}{j(1 - p)} \Pr(X = j - 1)$$

To avoid the possibility of underflow, the probabilities are computed forward from 0, if k is not greater than n times p , and are computed backward from n , otherwise. The smallest positive machine number, ϵ , is used as the starting value for computing the probabilities, which are rescaled by $(1 - p)^n \epsilon$ if forward computation is performed and by $p^n \epsilon$ if backward computation is done.

For the special case of $p = 0$, BINPR is set to 0 if k is greater than 0 and to 1 otherwise; and for the case $p = 1$, BINPR is set to 0 if k is less than n and to 1 otherwise.

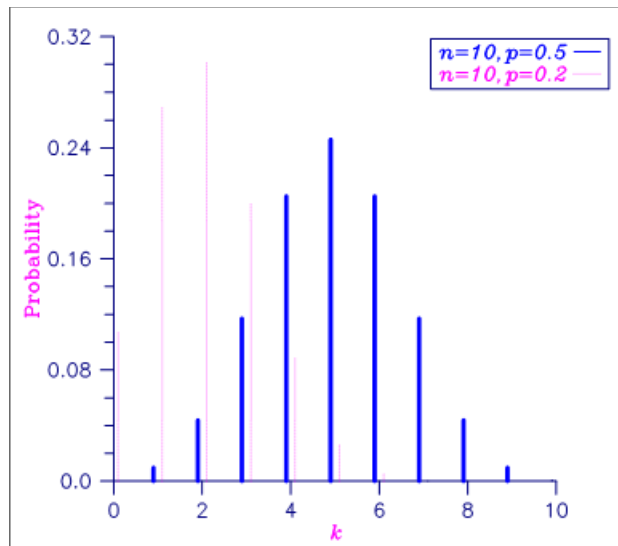


Figure 21, Binomial Probability Function

Comments

Informational errors

Type	Code	Description
1	3	The input argument, κ , is less than zero.
1	4	The input argument, κ , is greater than the number of Bernoulli trials, \mathfrak{n} .

Example

Suppose X is a binomial random variable with $N = 5$ and $PIN = 0.95$. In this example, we find the probability that X is equal to 3.

```
      USE UMACH_INT
      USE BINPR_INT
      IMPLICIT NONE
      INTEGER K, N, NOUT
      REAL PIN, PR
!
      CALL UMACH (2, NOUT)
      K = 3
      N = 5
      PIN = 0.95
      PR = BINPR(K,N,PIN)
      WRITE (NOUT,99999) PR
99999 FORMAT (' The probability that X is equal to 3 is ', F6.4)
      END
```

Output

```
The probability that X is equal to 3 is 0.0214
```

GEODF

This function evaluates the discrete geometric cumulative probability distribution function.

Function Return Value

GEODF — Function value, the probability that a geometric random variable takes a value less than or equal to **IX**. (Output)

Required Arguments

IX — Argument for which the geometric cumulative distribution function is to be evaluated. (Input)

PIN — Probability parameter for each independent trial (the probability of success for each independent trial). **PIN** must be in the open interval (0, 1). (Input)

FORTRAN 90 Interface

Generic: **GEODF (IX, PIN)**

Specific: The specific interface names are **S_GEODF** and **D_GEODF**.

FORTRAN 77 Interface

Single: **GEODF (IX, PIN)**

Double: The double precision name is **DGEODF**.

Description

The function **GEODF** evaluates the discrete geometric cumulative probability distribution function with parameter $p = \text{PIN}$, defined

$$F(x | p) = \sum_{i=0}^{\lfloor x \rfloor} pq^i, \quad q = 1 - p, \quad 0 < p < 1$$

The return value is the probability that up to x trials would be observed before observing a success.

Example

In this example, we evaluate the probability function at $IX = 3$, $PIN = 0.25$.

```
USE UMACH_INT
USE GEODF_INT
IMPLICIT NONE
INTEGER NOUT, IX
REAL PIN, PR
CALL UMACH(2, NOUT)

IX = 3
PIN = 0.25e0
PR = GEODF(IX, PIN)
WRITE (NOUT, 99999) IX, PIN, PR
99999 FORMAT (' GEODF(' , I2, ', ', ' ', F4.2, ') = ', F10.6)
END
```

Output

```
GEODF( 3, 0.25) = 0.683594
```

GEOIN

This function evaluates the inverse of the geometric cumulative probability distribution function.

Function Return Value

GEOIN — Integer function value. The probability that a geometric random variable takes a value less than or equal to the returned value is the input probability, P . (Output)

Required Arguments

P — Probability for which the inverse of the discrete geometric cumulative distribution function is to be evaluated. P must be in the open interval $(0, 1)$. (Input)

PIN — Probability parameter for each independent trial (the probability of success for each independent trial). PIN must be in the open interval $(0, 1)$. (Input)

FORTRAN 90 Interface

Generic: **GEOIN** (P , PIN)
Specific: The specific interface names are **S_GEOIN** and **D_GEOIN**.

FORTRAN 77 Interface

Single: **GEOIN** (P , PIN)
Double: The double precision name is **DGEOIN**.

Description

The function **GEOIN** evaluates the inverse distribution function of a geometric random variable with parameter PIN . The inverse of the CDF is defined as the smallest integer x such that the geometric CDF is not less than a given value P , $0 < P < 1$.

Example

In this example, we evaluate the inverse probability function at $PIN = 0.25$, $P = 0.6835$.

```
USE UMACH_INT
USE GEOIN_INT
IMPLICIT NONE
INTEGER NOUT, IX
REAL P, PIN
CALL UMACH(2, NOUT)
PIN = 0.25
P = 0.6835
IX = GEOIN(P, PIN)
WRITE (NOUT, 99999) P, PIN, IX
99999 FORMAT (' GEOIN(', F4.2, ', ', ' ', F6.4 ') = ', I2)
END
```

Output

```
GEOIN(0.6835, 0.25) = 3
```

GEOPR

This function evaluates the discrete geometric probability density function.

Function Return Value

GEOPR — Function value, the probability that a random variable from a geometric distribution having parameter **PIN** will be equal to **IX**. (Output)

Required Arguments

IX — Argument for which the discrete geometric probability density function is to be evaluated. **IX** must be greater than or equal to 0. (Input)

PIN — Probability parameter of the geometric probability function (the probability of success for each independent trial). **PIN** must be in the open interval (0, 1). (Input)

FORTRAN 90 Interface

Generic: **GEOPR (IX, PIN)**
Specific: The specific interface names are **S_GEOPR** and **D_GEOPR**.

FORTRAN 77 Interface

Single: **GEOPR (IX, PIN)**
Double: The double precision name is **DGEOPR**.

Description

The function **GEOPR** evaluates the discrete geometric probability density function, defined

$$f(x | p) = pq^x, \quad q = 1 - p, \quad 0 < p < 1, \quad x = 0, 1, \dots, \text{HUGE}(1), \quad \text{where } p = \text{PIN}.$$

Example

In this example, we evaluate the probability density function at **IX** = 3, **PIN** = 0.25.

```
USE UMACH_INT
USE GEOPR_INT
IMPLICIT NONE
INTEGER NOUT, IX
REAL PIN, PR
CALL UMACH(2, NOUT)
IX = 3
PIN = 0.25e0
PR = GEOPR(IX, PIN)
WRITE (NOUT, 99999) IX, PIN, PR
99999 FORMAT (' GEOPR(', I2, ', ', ' ', F4.2, ') = ', F6.4)
END
```

Output

```
GEOPR( 3, 0.25) = 0.1055
```

HYPDF

This function evaluates the hypergeometric cumulative distribution function.

Function Return Value

HYPDF — Function value, the probability that a hypergeometric random variable takes a value less than or equal to K . (Output)

HYPDF is the probability that K or fewer defectives occur in a sample of size N drawn from a lot of size L that contains M defectives.

See [Comment 1](#).

Required Arguments

K — Argument for which the hypergeometric cumulative distribution function is to be evaluated. (Input)

N — Sample size. (Input)

N must be greater than zero and greater than or equal to K .

M — Number of defectives in the lot. (Input)

L — Lot size. (Input)

L must be greater than or equal to N and M .

FORTRAN 90 Interface

Generic: **HYPDF** (K , N , M , L)

Specific: The specific interface names are **S_HYPDF** and **D_HYPDF**.

FORTRAN 77 Interface

Single: **HYPDF** (K , N , M , L)

Double: The double precision name is **DHYPDF**.

Description

The function `HYPDF` evaluates the cumulative distribution function of a hypergeometric random variable with parameters n , l , and m . The hypergeometric random variable X can be thought of as the number of items of a given type in a random sample of size n that is drawn without replacement from a population of size l containing m items of this type. The probability function is

$$\Pr(X = j) = \frac{\binom{m}{j} \binom{l-m}{n-j}}{\binom{l}{n}} \quad \text{for } j = i, i+1, i+2, \dots, \min(n, m)$$

where $i = \max(0, n - l + m)$.

If k is greater than or equal to i and less than or equal to $\min(n, m)$, `HYPDF` sums the terms in this expression for j going from i up to k . Otherwise, `HYPDF` returns 0 or 1, as appropriate. So, as to avoid rounding in the accumulation, `HYPDF` performs the summation differently depending on whether or not k is greater than the mode of the distribution, which is the greatest integer less than or equal to $(m+1)(n+1)/(l+2)$.

Comments

1. If the generic version of this function is used, the immediate result must be stored in a variable before use in an expression. For example:

```
X = HYPDF (K, N, M, L)
```

```
Y = SQRT(X)
```

must be used rather than

```
Y = SQRT(HYPDF(K, N, M, L))
```

If this is too much of a restriction on the programmer, then the specific name can be used without this restriction.

2. Informational errors

Type	Code	Description
1	5	The input argument, κ , is less than zero.
1	6	The input argument, κ , is greater than the sample size.

Example

Suppose X is a hypergeometric random variable with $\mathbf{N} = 100$, $\mathbf{L} = 1000$, and $\mathbf{M} = 70$. In this example, we evaluate the distribution function at 7.

```
      USE UMACH_INT
      USE HYPDF_INT
      IMPLICIT NONE
      INTEGER K, L, M, N, NOUT
      REAL DF
!
      CALL UMACH (2, NOUT)
      K = 7
      N = 100
      L = 1000
      M = 70
      DF = HYPDF(K,N,M,L)
      WRITE (NOUT,99999) DF
99999 FORMAT (' The probability that X is less than or equal to 7 is ' &
             , F6.4)
      END
```

Output

The probability that X is less than or equal to 7 is 0.5995

HYPPR

This function evaluates the hypergeometric probability density function.

Function Return Value

HYPPR — Function value, the probability that a hypergeometric random variable takes a value equal to K .

(Output)

HYPPR is the probability that exactly K defectives occur in a sample of size N drawn from a lot of size L that contains M defectives.

See [Comment 1](#).

Required Arguments

K — Argument for which the hypergeometric probability function is to be evaluated. (Input)

N — Sample size. (Input)

N must be greater than zero and greater than or equal to K .

M — Number of defectives in the lot. (Input)

L — Lot size. (Input)

L must be greater than or equal to N and M .

FORTRAN 90 Interface

Generic: **HYPPR** (K , N , M , L)

Specific: The specific interface names are **S_HYPPR** and **D_HYPPR**.

FORTRAN 77 Interface

Single: **HYPPR** (K , N , M , L)

Double: The double precision name is **DHYPPR**.

Description

The function `HYPPR` evaluates the probability density function of a hypergeometric random variable with parameters n , l , and m . The hypergeometric random variable X can be thought of as the number of items of a given type in a random sample of size n that is drawn without replacement from a population of size l containing m items of this type. The probability density function is

$$\Pr(X = k) = \frac{\binom{m}{k} \binom{l-m}{n-k}}{\binom{l}{n}} \quad \text{for } k = i, i+1, i+2, \dots, \min(n, m)$$

where $i = \max(0, n - l + m)$. `HYPPR` evaluates the expression using log gamma functions.

Comments

1. If the generic version of this function is used, the immediate result must be stored in a variable before use in an expression. For example:

```
X = HYPPR(K, N, M, L)
```

```
Y = SQRT(X)
```

must be used rather than

```
Y = SQRT(HYPPR(K, N, M, L))
```

If this is too much of a restriction on the programmer, then the specific name can be used without this restriction.

2. Informational errors

Type	Code	Description
1	5	The input argument, κ , is less than zero.
1	6	The input argument, κ , is greater than the sample size.

Example

Suppose X is a hypergeometric random variable with $\mathbf{N} = 100$, $\mathbf{L} = 1000$, and $\mathbf{M} = 70$. In this example, we evaluate the probability function at 7.

```
USE UMACH_INT
USE HYPPR_INT

IMPLICIT NONE
INTEGER K, L, M, N, NOUT
REAL PR
```

```
!  
  CALL UMACH (2, NOUT)  
  K = 7  
  N = 100  
  L = 1000  
  M = 70  
  PR = HYPPR(K,N,M,L)  
  WRITE (NOUT,99999) PR  
99999 FORMAT (' The probability that X is equal to 7 is ', F6.4)  
  END
```

Output

```
The probability that X is equal to 7 is 0.1628
```

POIDF

This function evaluates the Poisson cumulative distribution function.

Function Return Value

POIDF — Function value, the probability that a Poisson random variable takes a value less than or equal to K . (Output)

Required Arguments

K — Argument for which the Poisson cumulative distribution function is to be evaluated. (Input)

THETA — Mean of the Poisson distribution. (Input)
THETA must be positive.

FORTRAN 90 Interface

Generic: `POIDF (K, THETA)`

Specific: The specific interface names are `S_POIDF` and `D_POIDF`.

FORTRAN 77 Interface

Single: `POIDF (K, THETA)`

Double: The double precision name is `DPOIDF`.

Description

The function `POIDF` evaluates the cumulative distribution function of a Poisson random variable with parameter **THETA**. **THETA**, which is the mean of the Poisson random variable, must be positive. The probability function (with $\theta = \text{THETA}$) is

$$f(x) = e^{-\theta} \theta^x / x!, \quad \text{for } x = 0, 1, 2, \dots$$

The individual terms are calculated from the tails of the distribution to the mode of the distribution and summed. `POIDF` uses the recursive relationship

$$f(x + 1) = f(x) \theta / (x + 1), \quad \text{for } x = 0, 1, 2, \dots, k - 1,$$

with $f(0) = e^{-\theta}$.

Comments

Informational error

Type	Code	Description
1	1	The input argument, κ , is less than zero.

Example

Suppose X is a Poisson random variable with $\theta = 10$. In this example, we evaluate the distribution function at 7.

```
USE UMACH_INT
USE POIDF_INT
IMPLICIT NONE
INTEGER K, NOUT
REAL DF, THETA
!
CALL UMACH (2, NOUT)
K = 7
THETA = 10.0
DF = POIDF(K, THETA)
WRITE (NOUT, 99999) DF
99999 FORMAT (' The probability that X is less than or equal to ', &
             '7 is ', F6.4)
END
```

Output

```
The probability that X is less than or equal to 7 is 0.2202
```

POIPR

This function evaluates the Poisson probability density function.

Function Return Value

POIPR — Function value, the probability that a Poisson random variable takes a value equal to κ .
(Output)

Required Arguments

K — Argument for which the Poisson probability density function is to be evaluated. (Input)

THETA — Mean of the Poisson distribution. (Input)
THETA must be positive.

FORTRAN 90 Interface

Generic: **POIPR (K, THETA)**
Specific: The specific interface names are **S_POIPR** and **D_POIPR**.

FORTRAN 77 Interface

Single: **POIPR (K, THETA)**
Double: The double precision name is **DPOIPR**.

Description

The function **POIPR** evaluates the probability density function of a Poisson random variable with parameter **THETA**. **THETA**, which is the mean of the Poisson random variable, must be positive. The probability function (with $\theta = \text{THETA}$) is

$$f(x) = e^{-\theta} \theta^k / k!, \quad \text{for } k = 0, 1, 2, \dots$$

POIPR evaluates this function directly, taking logarithms and using the log gamma function.

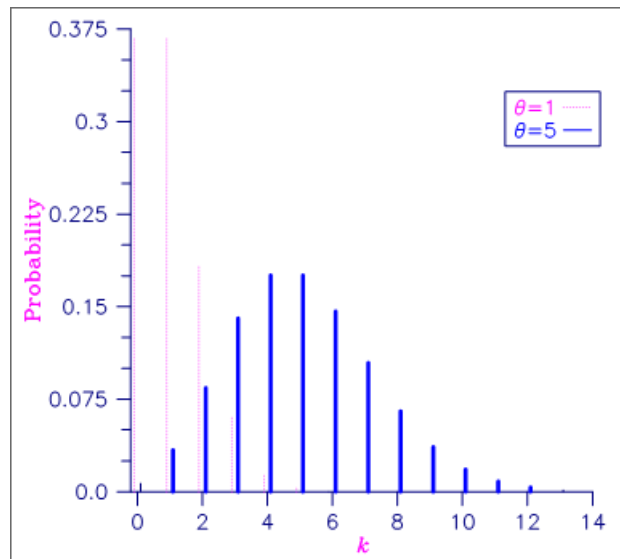


Figure 22, Poisson Probability Function

Comments

Informational error

Type	Code	Description
1	1	The input argument, k , is less than zero.

Example

Suppose X is a Poisson random variable with $\theta = 10$. In this example, we evaluate the probability function at 7.

```

USE UMACH_INT
USE POIPR_INT
IMPLICIT NONE

INTEGER K, NOUT
REAL PR, THETA

!
CALL UMACH (2, NOUT)
K = 7
THETA = 10.0
PR = POIPR(K, THETA)
WRITE (NOUT, 99999) PR
99999 FORMAT (' The probability that X is equal to 7 is ', F6.4)
END

```

Output

The probability that X is equal to 7 is 0.0901

UNDDF

This function evaluates the discrete uniform cumulative distribution function.

Function Return Value

UNDDF — Function value, the probability that a uniform random variable takes a value less than or equal to **IX**. (Output)

Required Arguments

IX — Argument for which the discrete uniform cumulative distribution function is to be evaluated. (Input)

N — Scale parameter. **N** must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: **UNDDF** (**IX**, **N**)

Specific: The specific interface names are **S_UNDDF** and **D_UNDDF**.

FORTRAN 77 Interface

Single: **UNDDF** (**IX**, **N**)

Double: The double precision name is **DUNDDF**.

Description

The notation below uses the floor and ceiling function notation, $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$.

The function **UNDDF** evaluates the discrete uniform cumulative probability distribution function with scale parameter **N**, defined

$$F(x|N) = \frac{\lfloor x \rfloor}{N}, \quad 1 \leq x \leq N$$

Example

In this example, we evaluate the probability function at **IX** = 3, **N** = 5.

```
USE UMACH_INT
USE UNDDF_INT
IMPLICIT NONE
INTEGER NOUT, IX, N
REAL PR
CALL UMACH(2, NOUT)
IX = 3
N = 5
PR = UNDDF(IX, N)
WRITE (NOUT, 99999) IX, N, PR
99999 FORMAT (' UNDDF(' , I2, ' , ' , I2, ') = ' , F6.4)
END
```

Output

```
UNDDF( 3, 5) = 0.6000
```

UNDIN

This function evaluates the inverse of the discrete uniform cumulative distribution function.

Function Return Value

UNDIN — Integer function value. The probability that a uniform random variable takes a value less than or equal to the returned value is the input probability, P . (Output)

Required Arguments

P — Probability for which the inverse of the discrete uniform cumulative distribution function is to be evaluated. P must be nonnegative and less than or equal to 1.0. (Input)

N — Scale parameter. N must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: `UNDIN (P, N)`

Specific: The specific interface names are `S_UNDIN` and `D_UNDIN`.

FORTRAN 77 Interface

Single: `UNDIN (P, N)`

Double: The double precision name is `DUNDIN`.

Description

The notation below uses the floor and ceiling function notation, $\lfloor . \rfloor$ and $\lceil . \rceil$.

The function `UNDIN` evaluates the inverse distribution function of a discrete uniform random variable with scale parameter N , defined

$$x = \lceil pN \rceil, \quad 0 \leq p \leq 1$$

Example

In this example, we evaluate the inverse probability function at $P = 0.6$, $N = 5$.

```
USE UMACH_INT
USE UNDIN_INT
IMPLICIT NONE
INTEGER NOUT, N, IX
REAL P
CALL UMACH(2, NOUT)
P = 0.60
N = 5
IX = UNDIN(P, N)
WRITE (NOUT, 99999) P, N, IX
99999 FORMAT (' UNDIN(', F4.2, ', ', I2, ') = ', I2)
END
```

Output

```
UNDIN(0.60, 5) = 3
```

UNDPR

This function evaluates the discrete uniform probability density function.

Function Return Value

UNDPR — Function value, the probability that a random variable from a uniform distribution having scale parameter N will be equal to IX . (Output)

Required Arguments

IX — Argument for which the discrete uniform probability density function is to be evaluated. (Input)

N — Scale parameter. N must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: `UNDPR (IX, N)`

Specific: The specific interface names are `S_UNDPR` and `D_UNDPR`.

FORTRAN 77 Interface

Single: `UNDPR (IX, N)`

Double: The double precision name is `DUNDPR`.

Description

The discrete uniform PDF is defined for positive integers x in the range $1, \dots, N$, $N > 0$. It has the value

$y = f(x|N) = \frac{1}{N}$, $1 \leq x \leq N$, and $y = 0$, $x > N$. Allowing values of x resulting in $y = 0$, $x > N$ is a convenience.

Example

In this example, we evaluate the discrete uniform probability density function at $IX = 3$, $N = 5$.

```
USE UMACH_INT
USE UNDPR_INT
IMPLICIT NONE
```

```
INTEGER NOUT, IX, N
REAL PR
CALL UMACH(2, NOUT)
IX = 3
N = 5
PR = UN DPR(IX, N)
WRITE (NOUT, 99999) IX, N, PR
99999 FORMAT (' UN DPR(' , I2, ' , ' , I2, ') = ' , F6.4)
END
```

Output

```
UN DPR( 3, 5) = 0.2000
```

AKS1DF

This function evaluates the cumulative distribution function of the one-sided Kolmogorov-Smirnov goodness of fit D^+ or D^- test statistic based on continuous data for one sample.

Function Return Value

AKS1DF — The probability of a smaller D . (Output)

Required Arguments

NOBS — The total number of observations in the sample. (Input)

D — The D^+ or D^- test statistic. (Input)

D is the maximum positive difference of the empirical cumulative distribution function (CDF) minus the hypothetical CDF or the maximum positive difference of the hypothetical CDF minus the empirical CDF.

FORTRAN 90 Interface

Generic: **AKS1DF** (NOBS, D)

Specific: The specific interface names are **S_AKS1DF** and **D_AKS1DF**.

FORTRAN 77 Interface

Single: **AKS1DF** (NOBS, D)

Double: The double precision name is **DKS1DF**.

Description

Routine **AKS1DF** computes the cumulative distribution function (CDF) for the one-sided Kolmogorov-Smirnov one-sample D^+ or D^- statistic when the theoretical CDF is strictly continuous. Let $F(x)$ denote the theoretical distribution function, and let $S_n(x)$ denote the empirical distribution function obtained from a sample of size **NOBS**.

Then, the D^+ statistic is computed as

$$D^+ = \sup_x [F(x) - S_n(x)]$$

while the one-sided D^- statistic is computed as

$$D^- = \sup_x [S_n(x) - F(x)]$$

Exact probabilities are computed according to a method given by Conover (1980, page 350) for sample sizes of 80 or less. For sample sizes greater than 80, Smirnov's asymptotic result is used, that is, the value of the CDF is taken as $1 - e^{-2nd^2}$, where d is D^+ or D^- (Kendall and Stuart, 1979, page 482). This asymptotic expression is conservative (the value returned by **AKS1DF** is smaller than the exact value, when the sample size exceeds 80).

Comments

1. Workspace may be explicitly provided, if desired, by use of **AK21DF** / **DK21DF**. The reference is:

AK2DF (**NOBS**, **D**, **WK**)

The additional argument is:

WK — Work vector of length $3 * \text{NOBS} + 3$ if $\text{NOBS} \leq 80$. **WK** is not used if **NOBS** is greater than 80.

2. Informational errors

Type	Code	Description
1	2	Since the D test statistic is less than zero, the distribution function is zero at D .
1	3	Since the D test statistic is greater than one, the distribution function is one at D .

3. If $\text{NOBS} \leq 80$, then exact one-sided probabilities are computed. In this case, on the order of NOBS^2 operations are required. For $\text{NOBS} > 80$, approximate one-sided probabilities are computed. These approximate probabilities require very few computations.
4. An approximate two-sided probability for the $\text{D} = \max(D^+, D^-)$ statistic can be computed as twice the **AKS1DF** probability for **D**(minus one, if the probability from **AKS1DF** is greater than 0.5).

Programming Notes

Routine **AKS1DF** requires on the order of NOBS^2 operations to compute the exact probabilities, where an operation consists of taking ten or so logarithms. Because so much computation is occurring within each “operation,” **AKS1DF** is much slower than its two-sample counterpart, function **AKS2DF**.

Example

In this example, the exact one-sided probabilities for the tabled values of D^+ or D^- , given, for example, in Conover (1980, page 462), are computed. Tabled values at the 10% level of significance are used as input to **AKS1DF** for sample sizes of 5 to 50 in increments of 5 (the last two tabled values are obtained using the asymptotic critical values of

$$1.07 / \sqrt{\text{NOBS}}$$

The resulting probabilities should all be close to 0.90.

```

USE UMACH_INT
USE AKS1DF_INT
IMPLICIT NONE
INTEGER I, NOBS, NOUT
REAL D(10)
!
DATA D/0.447, 0.323, 0.266, 0.232, 0.208, 0.190, 0.177, 0.165, &
0.160, 0.151/
!
CALL UMACH (2, NOUT)
!
DO 10 I=1, 10
NOBS = 5*I
!
WRITE (NOUT,99999) D(I), NOBS, AKS1DF(NOBS,D(I))
!
99999 FORMAT (' One-sided Probability for D = ', F8.3, ' with NOBS ' &
, '= ', I2, ' is ', F8.4)
10 CONTINUE
END

```

Output

```

One-sided Probability for D = 0.447 with NOBS = 5 is 0.9000
One-sided Probability for D = 0.323 with NOBS = 10 is 0.9006
One-sided Probability for D = 0.266 with NOBS = 15 is 0.9002
One-sided Probability for D = 0.232 with NOBS = 20 is 0.9009
One-sided Probability for D = 0.208 with NOBS = 25 is 0.9002
One-sided Probability for D = 0.190 with NOBS = 30 is 0.8992
One-sided Probability for D = 0.177 with NOBS = 35 is 0.9011
One-sided Probability for D = 0.165 with NOBS = 40 is 0.8987
One-sided Probability for D = 0.160 with NOBS = 45 is 0.9105
One-sided Probability for D = 0.151 with NOBS = 50 is 0.9077

```

AKS2DF

This function evaluates the cumulative distribution function of the Kolmogorov-Smirnov goodness of fit D test statistic based on continuous data for two samples.

Function Return Value

AKS2DF — The probability of a smaller D . (Output)

Required Arguments

NOBSX — The total number of observations in the first sample. (Input)

NOBSY — The total number of observations in the second sample. (Input)

D — The D test statistic. (Input)

D is the maximum absolute difference between empirical cumulative distribution functions (CDFs) of the two samples.

FORTRAN 90 Interface

Generic: **AKS2DF** (**NOBSX**, **NOBSY**, **D**)

Specific: The specific interface names are **S_AKS2DF** and **D_AKS2DF**.

FORTRAN 77 Interface

Single: **AKS2DF** (**NOBSX**, **NOBSY**, **D**)

Double: The double precision name is **DKS2DF**.

Description

Function **AKS2DF** computes the cumulative distribution function (CDF) for the two-sided Kolmogorov-Smirnov two-sample D statistic when the theoretical CDF is strictly continuous. Exact probabilities are computed according to a method given by Kim and Jennrich (1973). Approximate asymptotic probabilities are computed according to methods also given in this reference.

Let $F_n(x)$ and $G_m(x)$ denote the empirical distribution functions for the two samples, based on $n = \text{NOBSX}$ and $m = \text{NOBSY}$ observations. Then, the D statistic is computed as

$$D = \sup_x | F_n(x) - G_m(x) |$$

Comments

1. Workspace may be explicitly provided, if desired, by use of `AK22DF`/`DK22DF`. The reference is:

`AK22DF (NOBSX, NOBSY, D, WK)`

The additional argument is:

WK — Work vector of length $\max(\text{NOBSX}, \text{NOBSY}) + 1$.

2. Informational errors

Type	Code	Description
1	2	Since the D test statistic is less than zero, then the distribution function is zero at D.
1	3	Since the D test statistic is greater than one, then the distribution function is one at D.

Programming Notes

Function `AKS2DF` requires on the order of $\text{NOBSX} * \text{NOBSY}$ operations to compute the exact probabilities, where an operation consists of an addition and a multiplication. For $\text{NOBSX} * \text{NOBSY}$ less than 10000, the exact probability is computed. If this is not the case, then the Smirnov approximation discussed by Kim and Jennrich (1973) is used if the minimum of NOBSX and NOBSY is greater than ten percent of the maximum of NOBSX and NOBSY , or if the minimum is greater than 80. Otherwise, the Kolmogorov approximation discussed by Kim and Jennrich (1973) is used.

Example

Function `AKS2DF` is used to compute the probability of a smaller D statistic for a variety of sample sizes using values close to the 0.95 probability value.

```

USE UMACH_INT
USE AKS2DF_INT

IMPLICIT NONE
INTEGER I, NOBSX(10), NOBSY(10), NOUT
REAL D(10)
!
DATA NOBSX/5, 20, 40, 70, 110, 200, 200, 200, 100, 100/
DATA NOBSY/10, 10, 10, 10, 10, 20, 40, 60, 80, 100/
DATA D/0.7, 0.55, 0.475, 0.4429, 0.4029, 0.2861, 0.2113, 0.1796, &
0.18, 0.18/
!
```

```
CALL UMACH (2, NOUT)
!
DO 10 I=1, 10
!
WRITE (NOUT,99999) D(I), NOBSX(I), NOBSY(I), &
AKS2DF(NOBSX(I),NOBSY(I),D(I))
!
99999 FORMAT (' Probability for D = ', F5.3, ' with NOBSX = ', I3, &
' and NOBSY = ', I3, ' is ', F9.6, '.')
10 CONTINUE
END
```

Output

```
Probability for D = 0.700 with NOBSX = 5 and NOBSY = 10 is 0.980686.
Probability for D = 0.550 with NOBSX = 20 and NOBSY = 10 is 0.987553.
Probability for D = 0.475 with NOBSX = 40 and NOBSY = 10 is 0.972423.
Probability for D = 0.443 with NOBSX = 70 and NOBSY = 10 is 0.961646.
Probability for D = 0.403 with NOBSX = 110 and NOBSY = 10 is 0.928667.
Probability for D = 0.286 with NOBSX = 200 and NOBSY = 20 is 0.921126.
Probability for D = 0.211 with NOBSX = 200 and NOBSY = 40 is 0.917110.
Probability for D = 0.180 with NOBSX = 200 and NOBSY = 60 is 0.914520.
Probability for D = 0.180 with NOBSX = 100 and NOBSY = 80 is 0.908185.
Probability for D = 0.180 with NOBSX = 100 and NOBSY = 100 is 0.946098.
```

ALNDF

This function evaluates the lognormal cumulative probability distribution function.

Function Return Value

ALNDF — Function value, the probability that a standard lognormal random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the lognormal cumulative distribution function is to be evaluated. (Input)

AMU — Location parameter of the lognormal cumulative distribution function. (Input)

SIGMA — Shape parameter of the lognormal cumulative distribution function. **SIGMA** must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: `ALNDF (X, AMU, SIGMA)`

Specific: The specific interface names are `S_ALNDF` and `D_ALNDF`.

FORTRAN 77 Interface

Single: `ALNDF (X, AMU, SIGMA)`

Double: The double precision name is `DLNDF`.

Description

The function `ALNDF` evaluates the lognormal cumulative probability distribution function, defined as

$$\begin{aligned}
 F(x | \mu, \sigma) &= \frac{1}{\sigma\sqrt{2\pi}} \int_0^x \frac{1}{t} e^{-\left(\frac{(\log(t)-\mu)^2}{2\sigma^2}\right)} dt \\
 &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\log(x)} e^{-\left(\frac{u-\mu^2}{\sqrt{2}\sigma}\right)} du
 \end{aligned}$$

Example

In this example, we evaluate the probability distribution function at $x = 0.7137$, $\text{AMU} = 0.0$, $\text{SIGMA} = 0.5$.

```

USE UMACH_INT
USE ALNDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, AMU, SIGMA, PR
CALL UMACH(2, NOUT)
X = .7137
AMU = 0.0
SIGMA = 0.5
PR = ALNDF(X, AMU, SIGMA)
WRITE (NOUT, 99999) X, AMU, SIGMA, PR
99999 FORMAT (' ALNDF(' , F6.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END

```

Output

```
ALNDF( 0.71, 0.00, 0.50) = 0.2500
```

ALNIN

This function evaluates the inverse of the lognormal cumulative probability distribution function.

Function Return Value

ALNIN — Function value, the probability that a lognormal random variable takes a value less than or equal to the returned value is the input probability P . (Output)

Required Arguments

P — Probability for which the inverse of the lognormal distribution function is to be evaluated. (Input)

AMU — Location parameter of the lognormal cumulative distribution function. (Input)

SIGMA — Shape parameter of the lognormal cumulative distribution function. **SIGMA** must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: `ALNIN (P, AMU, SIGMA)`

Specific: The specific interface names are `S_ALNIN` and `D_ALNIN`.

FORTRAN 77 Interface

Single: `ALNIN (P, AMU, SIGMA)`

Double: The double precision name is `DLNIN`.

Description

The function **ALNIN** evaluates the inverse distribution function of a lognormal random variable with location parameter **AMU** and scale parameter **SIGMA**. The probability that a standard lognormal random variable takes a value less than or equal to the returned value is P .

Example

In this example, we evaluate the inverse probability function at $P = 0.25$, $AMU = 0.0$, $SIGMA = 0.5$.

```
USE UMACH_INT
USE ALNIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, AMU, SIGMA, P
CALL UMACH(2, NOUT)
P = .25
AMU = 0.0
SIGMA = 0.5
X = ALNIN(P, AMU, SIGMA)
WRITE (NOUT, 99999) P, AMU, SIGMA, X
99999 FORMAT (' ALNIN(', F6.3, ', ', ', F4.2, ', ', ', F4.2, ') = ', F6.4)
END
```

Output

```
ALNIN( 0.250, 0.00, 0.50) = 0.7137
```


ALNPR

This function evaluates the lognormal probability density function.

Function Return Value

ALNPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the lognormal probability density function is to be evaluated. (Input)

AMU — Location parameter of the lognormal probability function. (Input)

SIGMA — Shape parameter of the lognormal probability function. **SIGMA** must be greater than 0. (Input)

FORTRAN 90 Interface

Generic: **ALNPR (X, AMU, SIGMA)**

Specific: The specific interface names are **S_ALNPR** and **D_ALNPR**.

FORTRAN 77 Interface

Single: **ALNPR (X, AMU, SIGMA)**

Double: The double precision name is **DLNPR**.

Description

The function **ALNPR** evaluates the lognormal probability density function, defined as

$$f(x | \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\left(\frac{(\log(x)-\mu)^2}{2\sigma^2}\right)}$$

Example

In this example, we evaluate the probability function at $X = 1.0$, $AMU = 0.0$, $SIGMA = 0.5$.

```
USE UMACH_INT
USE ALNPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, AMU, SIGMA, PR
CALL UMACH(2, NOUT)
X = 1.0
AMU = 0.0
SIGMA = 0.5
PR = ALNPR(X, AMU, SIGMA)
WRITE (NOUT, 99999) X, AMU, SIGMA, PR
99999 FORMAT (' ALNPR(' , F6.2, ', ', ' , F4.2, ', ', ' , F4.2, ') = ', F6.4)
END
```

Output

```
ALNPR( 1.00, 0.00, 0.50) = 0.7979
```

ANORDF

This function evaluates the standard normal (Gaussian) cumulative distribution function.

Function Return Value

ANORDF — Function value, the probability that a normal random variable takes a value less than or equal to x . (Output)

Required Arguments

x — Argument for which the normal cumulative distribution function is to be evaluated. (Input)

FORTRAN 90 Interface

Generic: **ANORDF** (x)

Specific: The specific interface names are **S_ANORDF** and **D_ANORDF**.

FORTRAN 77 Interface

Single: **ANORDF** (x)

Double: The double precision name is **DNORDF**.

Description

Function **ANORDF** evaluates the cumulative distribution function, Φ , of a standard normal (Gaussian) random variable, that is,

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The standard normal distribution (for which **ANORDF** is the distribution function) has mean of 0 and variance of 1. The probability that a normal random variable with mean and variance σ^2 is less than y is given by **ANORDF** evaluated at $(y - \mu)/\sigma$.

$\Phi(x)$ is evaluated by use of the complementary error function, `erfc`. (See [ERFC](#), IMSL MATH/LIBRARY Special Functions). The relationship is:

$$\Phi(x) = \text{erfc}(-x/\sqrt{2.0}) / 2$$

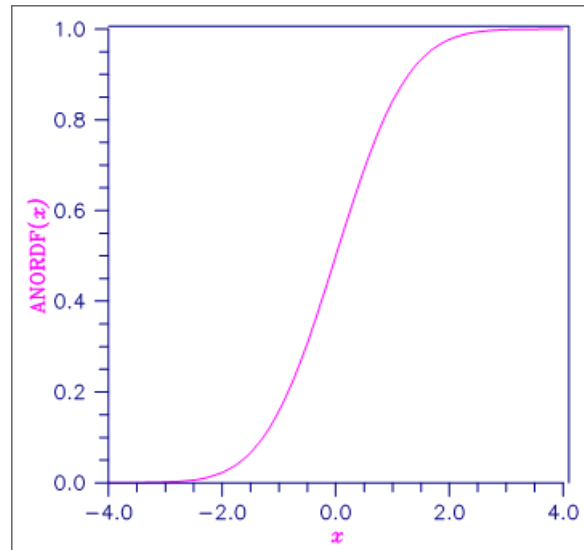


Figure 23, Standard Normal Distribution Function

Example

Suppose X is a normal random variable with mean 100 and variance 225. In this example, we find the probability that X is less than 90, and the probability that X is between 105 and 110.

```

USE UMACH_INT
USE ANORDF_INT

IMPLICIT NONE
INTEGER NOUT
REAL P, X1, X2

!
CALL UMACH (2, NOUT)
X1 = (90.0-100.0)/15.0
P = ANORDF(X1)
WRITE (NOUT,99998) P
99998 FORMAT (' The probability that X is less than 90 is ', F6.4)
X1 = (105.0-100.0)/15.0
X2 = (110.0-100.0)/15.0
P = ANORDF(X2) - ANORDF(X1)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that X is between 105 and 110 is ', &
F6.4)
END

```

Output

```
The probability that X is less than 90 is 0.2525  
The probability that X is between 105 and 110 is 0.1169
```

ANORIN

This function evaluates the inverse of the standard normal (Gaussian) cumulative distribution function.

Function Return Value

ANORIN — Function value. (Output)

The probability that a standard normal random variable takes a value less than or equal to **ANORIN** is P .

Required Arguments

P — Probability for which the inverse of the normal cumulative distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

FORTRAN 90 Interface

Generic: **ANORIN** (P)

Specific: The specific interface names are **S_ANORIN** and **D_ANORIN**.

FORTRAN 77 Interface

Single: **ANORIN** (P)

Double: The double precision name is **DNORIN**.

Description

Function **ANORIN** evaluates the inverse of the cumulative distribution function, Φ , of a standard normal (Gaussian) random variable, that is, $\text{ANORIN}(P) = \Phi^{-1}(p)$, where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$$

The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x . The standard normal distribution has a mean of 0 and a variance of 1.

Example

In this example, we compute the point such that the probability is 0.9 that a standard normal random variable is less than or equal to this point.

```
USE UMACH_INT
USE ANORIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL P, X
!
CALL UMACH (2, NOUT)
P = 0.9
X = ANORIN(P)
WRITE (NOUT,99999) X
99999 FORMAT (' The 90th percentile of a standard normal is ', F6.4)
END
```

Output

```
The 90th percentile of a standard normal is 1.2816
```

ANORPR

This function evaluates the standard normal probability density function.

Function Return Value

ANORPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the normal probability density function is to be evaluated. (Input)

FORTRAN 90 Interface

Generic: ANORPR (X)

Specific: The specific interface names are S_NORPR and D_NORPR.

FORTRAN 77 Interface

Single: ANORPR (X)

Double: The double precision name is DNORPR.

Description

The function **ANORPR** evaluates the normal probability density function, defined as

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\left(\frac{x^2}{2}\right)}$$

Example

In this example, we evaluate the probability function at $x = 0.5$.

```
USE UMACH_INT
USE ANORPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, PR
CALL UMACH(2, NOUT)
```



```
X = 0.5
PR = ANORPR(X)
WRITE (NOUT, 99999) X, PR
99999 FORMAT (' ANORPR(', F4.2, ') = ', F6.4)
END
```

Output

```
ANORPR(0.50) = 0.3521
```

BETDF

This function evaluates the beta cumulative distribution function.

Function Return Value

BETDF — Probability that a random variable from a beta distribution having parameters **PIN** and **QIN** will be less than or equal to **X**. (Output)

Required Arguments

X — Argument for which the beta distribution function is to be evaluated. (Input)

PIN — First beta distribution parameter. (Input)
PIN must be positive.

QIN — Second beta distribution parameter. (Input)
QIN must be positive.

FORTRAN 90 Interface

Generic: **BETDF (X, PIN, QIN)**

Specific: The specific interface names are **S_BETDF** and **D_BETDF**.

FORTRAN 77 Interface

Single: **BETDF (X, PIN, QIN)**

Double: The double precision name is **DBETDF**.

Description

Function **BETDF** evaluates the cumulative distribution function of a beta random variable with parameters **PIN** and **QIN**. This function is sometimes called the *incomplete beta ratio* and, with $p = \text{PIN}$ and $q = \text{QIN}$, is denoted by $I_x(p, q)$. It is given by

$$I_x(p,q) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} \int_0^x t^{p-1}(1-t)^{q-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function $I_x(p, q)$ is the probability that the random variable takes a value less than or equal to x .

The integral in the expression above is called the *incomplete beta function* and is denoted by $\beta_x(p, q)$. The constant in the expression is the reciprocal of the *beta function* (the incomplete function evaluated at one) and is denoted by $\beta(p, q)$.

Function **BETDF** uses the method of Bosten and Battiste (1974).

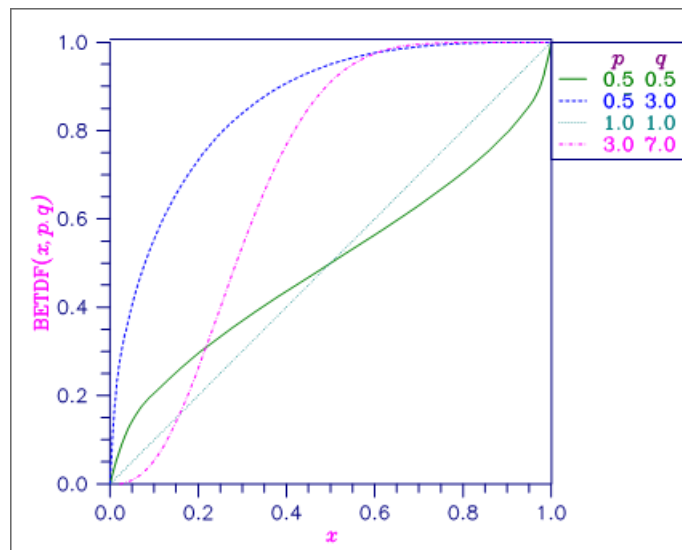


Figure 24, Beta Distribution Function

Comments

Informational errors

Type	Code	Description
1	1	Since the input argument x is less than or equal to zero, the distribution function is equal to zero at x .
1	2	Since the input argument x is greater than or equal to one, the distribution function is equal to one at x .

Example

Suppose X is a beta random variable with parameters 12 and 12. (X has a symmetric distribution.) In this example, we find the probability that X is less than 0.6 and the probability that X is between 0.5 and 0.6. (Since X is a symmetric beta random variable, the probability that it is less than 0.5 is 0.5.)

```
      USE UMACH_INT
      USE BETDF_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL P, PIN, QIN, X
!
      CALL UMACH (2, NOUT)
      PIN = 12.0
      QIN = 12.0
      X = 0.6
      P = BETDF(X,PIN,QIN)
      WRITE (NOUT,99998) P
99998 FORMAT (' The probability that X is less than 0.6 is ', F6.4)
      X = 0.5
      P = P - BETDF(X,PIN,QIN)
      WRITE (NOUT,99999) P
99999 FORMAT (' The probability that X is between 0.5 and 0.6 is ', &
             F6.4)
      END
```

Output

```
The probability that X is less than 0.6 is 0.8364
The probability that X is between 0.5 and 0.6 is 0.3364
```

BETIN

This function evaluates the inverse of the beta cumulative distribution function.

Function Return Value

BETIN — Function value. (Output)

The probability that a beta random variable takes a value less than or equal to **BETIN** is P .

Required Arguments

P — Probability for which the inverse of the beta distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

PIN — First beta distribution parameter. (Input)

PIN must be positive.

QIN — Second beta distribution parameter. (Input)

QIN must be positive.

FORTRAN 90 Interface

Generic: **BETIN** (P , PIN , QIN)

Specific: The specific interface names are **S_BETIN** and **D_BETIN**.

FORTRAN 77 Interface

Single: **BETIN** (P , PIN , QIN)

Double: The double precision name is **DBETIN**.

Description

The function **BETIN** evaluates the inverse distribution function of a beta random variable with parameters PIN and QIN , that is, with $P = P$, $p = PIN$, and $q = QIN$, it determines x (equal to **BETIN** (P , PIN , QIN)), such that

$$P = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} \int_0^x t^{p-1}(1-t)^{q-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to x is P .

Comments

Informational errors

Type	Code	Description
3	1	The value for the inverse Beta distribution could not be found in 100 iterations. The best approximation is used.

Example

Suppose X is a beta random variable with parameters 12 and 12. (X has a symmetric distribution.) In this example, we find the value x_0 such that the probability that $X \leq x_0$ is 0.9.

```

USE UMACH_INT
USE BETIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL P, PIN, QIN, X
!
CALL UMACH (2, NOUT)
PIN = 12.0
QIN = 12.0
P = 0.9
X = BETIN(P,PIN,QIN)
WRITE (NOUT,99999) X
99999 FORMAT (' X is less than ', F6.4, ' with probability 0.9.')
END

```

Output

```
X is less than 0.6299 with probability 0.9.
```

BETPR

This function evaluates the beta probability density function.

Function Return Value

BETPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the beta probability density function is to be evaluated. (Input)

PIN — First beta distribution parameter. (Input)

PIN must be positive.

QIN — Second beta distribution parameter. (Input)

QIN must be positive.

FORTRAN 90 Interface

Generic: **BETPR (X, PIN, QIN)**

Specific: The specific interface names are **S_BETPR** and **D_BETPR**.

FORTRAN 77 Interface

Single: **BETPR (X, PIN, QIN)**

Double: The double precision name is **DBETPR**.

Description

The function **BETPR** evaluates the beta probability density function with parameters **PIN** and **QIN**. Using $x = X$, $a = PIN$ and $b = QIN$, the beta distribution is defined as

$$f(x | a, b) = \frac{1}{B(a, b)}(1-x)^{b-1}x^{a-1}, \quad a, b > 0, \quad 0 \leq x \leq 1$$

The reciprocal of the beta function used as the normalizing factor is computed using IMSL function [BETA](#) (see [Chapter 4, "Gamma Functions and Related Functions"](#)).

Example

In this example, we evaluate the probability function at $X = 0.75$, $PIN = 2.0$, $QIN = 0.5$.

```
USE UMACH_INT
USE BETPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, PIN, QIN, PR
CALL UMACH(2, NOUT)
X = .75
PIN = 2.0
QIN = 0.5
PR = BETPR(X, PIN, QIN)
WRITE (NOUT, 99999) X, PIN, QIN, PR
99999 FORMAT (' BETPR(', F4.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
BETPR(0.75, 2.00, 0.50) = 1.1250
```


BETNDF

This function evaluates the noncentral beta cumulative distribution function (CDF).

Function Return Value

BETNDF — Probability that a random variable from a beta distribution having shape parameters **SHAPE1** and **SHAPE2** and noncentrality parameter **LAMBDA** will be less than or equal to **X**. (Output)

Required Arguments

X — Argument for which the noncentral beta cumulative distribution function is to be evaluated. (Input)
X must be non-negative and less than or equal to 1.

SHAPE1 — First shape parameter of the noncentral beta distribution. (Input)
SHAPE1 must be positive.

SHAPE2 — Second shape parameter of the noncentral beta distribution. (Input)
SHAPE2 must be positive.

LAMBDA — Noncentrality parameter. (Input)
LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: `BETNDF (X, SHAPE1, SHAPE2, LAMBDA)`

Specific: The specific interface names are `S_BETNDF` and `D_BETNDF`.

Description

The noncentral beta distribution is a generalization of the beta distribution. If Z is a noncentral chi-square random variable with noncentrality parameter λ and $2\alpha_1$ degrees of freedom, and Y is a chi-square random variable with $2\alpha_2$ degrees of freedom which is statistically independent of Z , then

$$X = \frac{Z}{Z+Y} = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

is a noncentral beta-distributed random variable and

$$F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}$$

is a noncentral F -distributed random variable. The CDF for noncentral beta variable X can thus be simply defined in terms of the noncentral F CDF:

$$CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$$

where $CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda)$ is a noncentral beta CDF with $x = \mathbf{x}$, $\alpha_1 = \text{SHAPE1}$, $\alpha_2 = \text{SHAPE2}$, and noncentrality parameter $\lambda = \text{LAMBDA}$; $CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$ is a noncentral F CDF with argument f , numerator and denominator degrees of freedom $2\alpha_1$ and $2\alpha_2$ respectively, and noncentrality parameter λ and:

$$f = \frac{\alpha_2}{\alpha_1} \frac{x}{1-x}; \quad x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

(See documentation for function [FNDF](#) for a discussion of how the noncentral F CDF is defined and calculated.)

With a noncentrality parameter of zero, the noncentral beta distribution is the same as the beta distribution.

Example

This example traces out a portion of a noncentral beta distribution with parameters $\text{SHAPE1} = 50$, $\text{SHAPE2} = 5$, and $\text{LAMBDA} = 10$.

```

USE UMACH_INT
USE BETNDF_INT
USE FNDF_INT
IMPLICIT NONE
INTEGER NOUT, I
REAL X, LAMBDA, SHAPE1, SHAPE2, &
      BCFV, FCFV, F(8)

DATA F /0.0, 0.4, 0.8, 1.2, &
      1.6, 2.0, 2.8, 4.0 /

CALL UMACH (2, NOUT)
SHAPE1 = 50.0
SHAPE2 = 5.0
LAMBDA = 10.0

WRITE (NOUT, '(/" SHAPE1: ", F4.0, &
  & "; SHAPE2: ", F4.0, &
  & "; LAMBDA: ", F4.0 // &
  & 6x, "X", 6x, "NCBETCDF(X)", 3x, "NCBETCDF(X)" / &
  & 14x, "expected"')) SHAPE1, SHAPE2, LAMBDA

DO I = 1, 8
  X = (SHAPE1*F(I)) / (SHAPE1*F(I) + SHAPE2)
  FCFV = FNDF(F(I), 2*SHAPE1, 2*SHAPE2, LAMBDA)
  BCFV = BETNDF(X, SHAPE1, SHAPE2, LAMBDA)
  WRITE (NOUT, '(2X, F8.6, 2(2X, E12.6))') &
    X, FCFV, BCFV

```

```
END DO  
END
```

Output

```
SHAPE1: 50.; SHAPE2: 5.; LAMBDA: 10.
```

X	NCBETCDF(X) expected	NCBETCDF(X)
0.000000	0.000000E+00	0.000000E+00
0.800000	0.488790E-02	0.488790E-02
0.888889	0.202633E+00	0.202633E+00
0.923077	0.521143E+00	0.521143E+00
0.941176	0.733853E+00	0.733853E+00
0.952381	0.850413E+00	0.850413E+00
0.965517	0.947125E+00	0.947125E+00
0.975610	0.985358E+00	0.985358E+00

BETNIN

This function evaluates the inverse of the noncentral beta cumulative distribution function (CDF).

Function Return Value

BETNIN — Function value, the value of the inverse of the cumulative distribution function evaluated at P . The probability that a noncentral beta random variable takes a value less than or equal to **BETNIN** is P . (Output)

Required Arguments

P — Probability for which the inverse of the noncentral beta cumulative distribution function is to be evaluated. (Input)
 P must be non-negative and less than or equal to 1.

SHAPE1 — First shape parameter of the noncentral beta distribution. (Input)
 $SHAPE1$ must be positive.

SHAPE2 — Second shape parameter of the noncentral beta distribution. (Input)
 $SHAPE2$ must be positive.

LAMBDA — Noncentrality parameter. (Input)
 $LAMBDA$ must be non-negative.

FORTRAN 90 Interface

Generic: `BETNIN (P, SHAPE1, SHAPE2, LAMBDA)`

Specific: The specific interface names are `S_BETNIN` and `D_BETNIN`.

Description

The noncentral beta distribution is a generalization of the beta distribution. If Z is a noncentral chi-square random variable with noncentrality parameter λ and $2\alpha_1$ degrees of freedom, and Y is a chi-square random variable with $2\alpha_2$ degrees of freedom which is statistically independent of Z , then

$$X = \frac{Z}{Z+Y} = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

is a noncentral beta-distributed random variable and

$$F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}$$

is a noncentral F -distributed random variable. The CDF for noncentral beta variable X can thus be simply defined in terms of the noncentral F CDF:

$$p = CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$$

where $CDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda)$ is a noncentral beta CDF with $x = \mathbf{x}$, $\alpha_1 = \text{SHAPE1}$, $\alpha_2 = \text{SHAPE2}$, and noncentrality parameter $\lambda = \text{LAMBDA}$; $CDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$ is a noncentral F CDF with argument f , numerator and denominator degrees of freedom $2\alpha_1$ and $2\alpha_2$ respectively, and noncentrality parameter λ ; $p =$ the probability that $F \leq f$ = the probability that $X \leq x$ and:

$$f = \frac{\alpha_2}{\alpha_1} \frac{x}{1-x}; \quad x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

(See the documentation for function [FNDF](#) for a discussion of how the noncentral F CDF is defined and calculated.) The correspondence between the arguments of function `BETNIN(P, SHAPE1, SHAPE2, LAMBDA)` and the variables in the above equations is as follows: $\alpha_1 = \text{SHAPE1}$, $\alpha_2 = \text{SHAPE2}$, $\lambda = \text{LAMBDA}$, and $p = P$.

Function `BETNIN` evaluates

$$x = CDF_{nc\beta}^{-1}(p, \alpha_1, \alpha_2, \lambda)$$

by first evaluating

$$f = CDF_{ncF}^{-1}(p, 2\alpha_1, 2\alpha_2, \lambda)$$

and then solving for x using

$$x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

(See the documentation for function [FNIN](#) for a discussion of how the inverse noncentral F CDF is calculated.)

Example

This example traces out a portion of an inverse noncentral beta distribution with parameters $\text{SHAPE1} = 50$, $\text{SHAPE2} = 5$, and $\text{LAMBDA} = 10$.

```

USE UMACH_INT
USE BETNDF_INT
USE BETNIN_INT
USE UMACH_INT
IMPLICIT NONE

INTEGER  :: NOUT, I
REAL     :: SHAPE1 = 50.0, SHAPE2=5.0, LAMBDA=10.0
REAL     :: X, CDF, CDFINV
REAL     :: F0(8)=(/ 0.0, .4, .8, 1.2, 1.6, 2.0, 2.8, 4.0 /)

CALL UMACH (2, NOUT)
WRITE (NOUT, '(/" SHAPE1: ", F4.0, " SHAPE2: ", F4.0, '// &
' " LAMBDA: ", F4.0 // ' // &
' " X P = CDF(X) CDFINV(P)')' ) &
SHAPE1, SHAPE2, LAMBDA
DO I = 1, 8
X = (SHAPE1*F0(I))/(SHAPE2 + SHAPE1*F0(I))
CDF = BETNDF(X, SHAPE1, SHAPE2, LAMBDA)
CDFINV = BETNIN(CDF, SHAPE1, SHAPE2, LAMBDA)
WRITE (NOUT, '(3(2X, E12.6))') X, CDF, CDFINV
END DO
END

```

Output

```

SHAPE1:  50.  SHAPE2:  5.  LAMBDA:  10.

      X          P = CDF(X)      CDFINV(P)
0.000000E+00  0.000000E+00  0.000000E+00
0.800000E+00  0.488791E-02  0.800000E+00
0.888889E+00  0.202633E+00  0.888889E+00
0.923077E+00  0.521144E+00  0.923077E+00
0.941176E+00  0.733853E+00  0.941176E+00
0.952381E+00  0.850413E+00  0.952381E+00
0.965517E+00  0.947125E+00  0.965517E+00
0.975610E+00  0.985358E+00  0.975610E+00

```

BETNPR

This function evaluates the noncentral beta probability density function.

Function Return Value

BETNPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the noncentral beta probability density function is to be evaluated. (Input)
X must be non-negative and less than or equal to 1.

SHAPE1 — First shape parameter of the noncentral beta distribution. (Input)
SHAPE1 must be positive.

SHAPE2 — Second shape parameter of the noncentral beta distribution. (Input)
SHAPE2 must be positive.

LAMBDA — Noncentrality parameter. (Input)
LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: `BETNPR (X, SHAPE1, SHAPE2, LAMBDA)`

Specific: The specific interface names are `S_BETNPR` and `D_BETNPR`.

Description

The noncentral beta distribution is a generalization of the beta distribution. If Z is a noncentral chi-square random variable with noncentrality parameter λ and $2\alpha_1$ degrees of freedom, and Y is a chi-square random variable with $2\alpha_2$ degrees of freedom which is statistically independent of Z , then

$$X = \frac{Z}{Z+Y} = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2}$$

is a noncentral beta-distributed random variable and

$$F = \frac{\alpha_2 Z}{\alpha_1 Y} = \frac{\alpha_2 X}{\alpha_1 (1 - X)}$$

is a noncentral F -distributed random variable. The PDF for noncentral beta variable X can thus be simply defined in terms of the noncentral F PDF:

$$PDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda) = PDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda) \frac{df}{dx}$$

Where $PDF_{nc\beta}(x, \alpha_1, \alpha_2, \lambda)$ is a noncentral beta PDF with $x = \mathbf{x}$, $\alpha_1 = \text{SHAPE1}$, $\alpha_2 = \text{SHAPE2}$, and noncentrality parameter $\lambda = \text{LAMBDA}$; $PDF_{ncF}(f, 2\alpha_1, 2\alpha_2, \lambda)$ is a noncentral F PDF with argument f , numerator and denominator degrees of freedom $2\alpha_1$ and $2\alpha_2$ respectively, and noncentrality parameter λ ; and:

$$f = \frac{\alpha_2 x}{\alpha_1 (1-x)}; \quad x = \frac{\alpha_1 f}{\alpha_1 f + \alpha_2};$$

$$\frac{df}{dx} = \frac{(\alpha_2 + \alpha_1 f)^2}{\alpha_1 \alpha_2} = \frac{\alpha_2}{\alpha_1 (1-x)^2}$$

(See the documentation for function [FNPR](#) for a discussion of how the noncentral F PDF is defined and calculated.)

With a noncentrality parameter of zero, the noncentral beta distribution is the same as the beta distribution.

Example

This example traces out a portion of a noncentral beta distribution with parameters $\text{SHAPE1} = 50$, $\text{SHAPE2} = 5$, and $\text{LAMBDA} = 10$.

```

USE UMACH_INT
USE BETNPR_INT
USE FNPR_INT
IMPLICIT NONE

INTEGER NOUT, I
REAL X, LAMBDA, SHAPE1, SHAPE2, &
      BPDFV, FPDFV, DBETNPR, DFNPR, F(8), &
      BPDFVEXPECT, DFDX

DATA F /0.0, 0.4, 0.8, 3.2, 5.6, 8.8, 14.0, 18.0/

CALL UMACH (2, NOUT)
SHAPE1 = 50.0
SHAPE2 = 5.0
LAMBDA = 10.0

WRITE (NOUT, '(/" SHAPE1: ", F4.0, "; SHAPE2: ", F4.0, "; ' // &
      'LAMBDA: ", F4.0 // 6x, "X", 6x, "NCBETPDF(X)", 3x, "NCBETPDF' // &
      '(X)", /      14x, "expected")') SHAPE1, SHAPE2, LAMBDA

```



```
DO I = 1, 8
  X = (SHAPE1*F(I)) / (SHAPE1*F(I) + SHAPE2)
  DFDX = (SHAPE2/SHAPE1) / (1.0 - X)**2
  FPDFV = FNPR(F(I), 2*SHAPE1, 2*SHAPE2, LAMBDA)
  BPDFVEXPECT = DFDX * FPDFV
  BPDFV = BETNPR(X, SHAPE1, SHAPE2, LAMBDA)
  WRITE (NOUT, '(2X, F8.6, 2(2X, E12.6))') X, BPDFVEXPECT, BPDFV
END DO
END
```

Output

```
SHAPE1: 50.; SHAPE2: 5.; LAMBDA: 10.
```

X	NCBETPDF(X) expected	NCBETPDF(X)
0.000000	0.000000E+00	0.000000E+00
0.800000	0.243720E+00	0.243720E+00
0.888889	0.658624E+01	0.658624E+01
0.969697	0.402367E+01	0.402365E+01
0.982456	0.919544E+00	0.919542E+00
0.988764	0.219100E+00	0.219100E+00
0.992908	0.436654E-01	0.436647E-01
0.994475	0.175215E-01	0.175217E-01

BNRDF

This function evaluates the bivariate normal cumulative distribution function.

Function Return Value

BNRDF — Function value, the probability that a bivariate normal random variable with correlation **RHO** takes a value less than or equal to **X** and less than or equal to **Y**. (Output)

Required Arguments

X — One argument for which the bivariate normal distribution function is to be evaluated. (Input)

Y — The other argument for which the bivariate normal distribution function is to be evaluated. (Input)

RHO — Correlation coefficient. (Input)

FORTRAN 90 Interface

Generic: **BNRDF** (**X**, **Y**, **RHO**)

Specific: The specific interface names are **S_BNRDF** and **D_BNRDF**.

FORTRAN 77 Interface

Single: **BNRDF** (**X**, **Y**, **RHO**)

Double: The double precision name is **DBNRDF**.

Description

Function **BNRDF** evaluates the cumulative distribution function F of a bivariate normal distribution with means of zero, variances of one, and correlation of **RHO**; that is, with $\rho = \text{RHO}$, and $|\rho| < 1$,

$$F(x,y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^x \int_{-\infty}^y \exp\left(-\frac{u^2 - 2\rho uv + v^2}{2(1-\rho^2)}\right) du dv$$

To determine the probability that $U \leq u_0$ and $V \leq v_0$, where $(U, V)^T$ is a bivariate normal random variable with mean $\boldsymbol{\mu} = (\mu_U, \mu_V)^T$ and variance-covariance matrix

$$\Sigma = \begin{pmatrix} \sigma_U^2 & \sigma_{UV} \\ \sigma_{UV} & \sigma_V^2 \end{pmatrix}$$

transform $(U, V)^T$ to a vector with zero means and unit variances. The input to **BNRDF** would be

$$\mathbf{X} = (u_0 - \mu_U) / \sigma_U, \mathbf{Y} = (v_0 - \mu_V) / \sigma_V, \text{ and } \rho = \sigma_{UV} / (\sigma_U \sigma_V).$$

Function **BNRDF** uses the method of Owen (1962, 1965). Computation of Owen's T-function is based on code by M. Patefield and D. Tandy (2000). For $|\rho| = 1$, the distribution function is computed based on the univariate statistic, $Z = \min(x, y)$, and on the normal distribution function **ANORDF**.

Example

Suppose (X, Y) is a bivariate normal random variable with mean $(0, 0)$ and variance-covariance matrix

$$\begin{pmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{pmatrix}$$

In this example, we find the probability that X is less than -2.0 and Y is less than 0.0 .

```

USE BNRDF_INT
USE UMACH_INT
IMPLICIT NONE
INTEGER NOUT
REAL P, RHO, X, Y
!
CALL UMACH (2, NOUT)
X = -2.0
Y = 0.0
RHO = 0.9
P = BNRDF(X,Y,RHO)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that X is less than -2.0 and Y ', &
             'is less than 0.0 is ', F6.4)
END

```

Output

```
The probability that X is less than -2.0 and Y is less than 0.0 is 0.0228
```

CHIDF

This function evaluates the chi-squared cumulative distribution function.

Function Return Value

CHIDF — Function value, the probability that a chi-squared random variable takes a value less than or equal to **CHSQ**. (Output)

Required Arguments

CHSQ — Argument for which the chi-squared distribution function is to be evaluated. (Input)

DF — Number of degrees of freedom of the chi-squared distribution. (Input)
DF must be positive.

Optional Arguments

COMPLEMENT — Logical. If **.TRUE.**, the complement of the chi-squared cumulative distribution function is evaluated. If **.FALSE.**, the chi-squared cumulative distribution function is evaluated. (Input)
See the [Description](#) section for further details on the use of **COMPLEMENT**.
Default: **COMPLEMENT = .FALSE.**

FORTRAN 90 Interface

Generic: **CHIDF (CHSQ, DF [, ...])**

Specific: The specific interface names are **S_CHIDF** and **D_CHIDF**.

FORTRAN 77 Interface

Single: **CHIDF (CHSQ, DF)**

Double: The double precision name is **DCHIDF**.

Description

Function **CHIDF** evaluates the cumulative distribution function, F , of a chi-squared random variable with **DF** degrees of freedom, that is, with $\nu = \mathbf{DF}$, and $x = \mathbf{CHSQ}$,

$$F(x, \nu) = \frac{1}{2^{\nu/2} \Gamma(\nu/2)} \int_0^x e^{-t/2} t^{\nu/2-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

For $\nu > \nu_{max}$ = {343 for double precision, 171 for single precision}, **CHIDF** uses the Wilson-Hilferty approximation (Abramowitz and Stegun [A&S] 1964, equation 26.4.17) for p in terms of the normal CDF, which is evaluated using function **ANORDF**.

For $\nu \leq \nu_{max}$, **CHIDF** uses series expansions to evaluate p : for $x < \nu$, **CHIDF** calculates p using A&S series 6.5.29, and for $x \geq \nu$, **CHIDF** calculates p using the continued fraction expansion of the incomplete gamma function given in A&S equation 6.5.31.

If **COMPLEMENT** = **.TRUE.**, the value of **CHIDF** at the point x is $1 - p$, where $1 - p$ is the probability that the random variable takes a value greater than x . In those situations where the desired end result is $1 - p$, the user can achieve greater accuracy in the right tail region by using the result returned by **CHIDF** with the optional argument **COMPLEMENT** set to **.TRUE.** rather than by using $1 - p$ where p is the result returned by **CHIDF** with **COMPLEMENT** set to **.FALSE.**

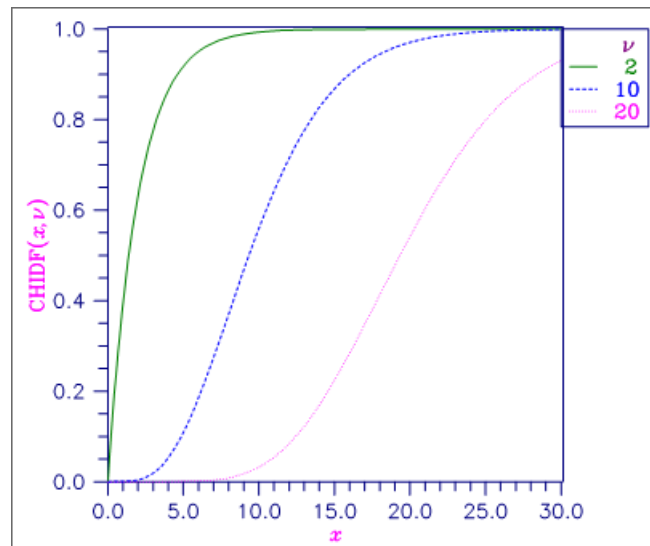


Figure 25, Chi-Squared Distribution Function

Comments

Informational error

Type	Code	Description
1	1	Since the input argument, <code>CHSQ</code> , is less than zero, the distribution function is zero at <code>CHSQ</code> .
2	3	The normal distribution is used for large degrees of freedom. However, it has produced underflow. Therefore, the probability, <code>CHIDF</code> , is set to zero.

Example

Suppose X is a chi-squared random variable with 2 degrees of freedom. In this example, we find the probability that X is less than 0.15 and the probability that X is greater than 3.0.

```

USE CHIDF_INT
USE UMACH_INT
IMPLICIT NONE

INTEGER NOUT
REAL CHSQ, DF, P

CALL UMACH (2, NOUT)
DF = 2.0
CHSQ = 0.15
P = CHIDF(CHSQ,DF)
WRITE (NOUT,99998) P
99998 FORMAT (' The probability that chi-squared with 2 df is less ', &
             'than 0.15 is ', F6.4)
CHSQ = 3.0
P = CHIDF(CHSQ,DF, complement=.true.)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that chi-squared with 2 df is greater ' &
             ', 'than 3.0 is ', F6.4)
END

```

Output

```

The probability that chi-squared with 2 df is less than 0.15 is 0.0723
The probability that chi-squared with 2 df is greater than 3.0 is 0.2231

```

CHIIN

This function evaluates the inverse of the chi-squared cumulative distribution function.

Function Return Value

CHIIN — Function value. (Output)

The probability that a chi-squared random variable takes a value less than or equal to **CHIIN** is **P**.

Required Arguments

P — Probability for which the inverse of the chi-squared distribution function is to be evaluated. (Input)
P must be in the open interval (0.0, 1.0).

DF — Number of degrees of freedom of the chi-squared distribution. (Input)
DF must be greater than or equal to 0.5.

FORTRAN 90 Interface

Generic: **CHIIN** (**P**, **DF**)

Specific: The specific interface names are **S_CHIIN** and **D_CHIIN**.

FORTRAN 77 Interface

Single: **CHIIN** (**P**, **DF**)

Double: The double precision name is **DCHIIN**.

Description

Function **CHIIN** evaluates the inverse distribution function of a chi-squared random variable with **DF** degrees of freedom, that is, with $P = \mathbf{P}$ and $\nu = \mathbf{DF}$, it determines x (equal to **CHIIN**(**P**, **DF**)), such that

$$P = \frac{1}{2^{\nu/2}\Gamma(\nu/2)} \int_0^x e^{-t/2} t^{\nu/2-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to x is P .

For $\nu < 40$, **CHIIN** uses bisection (if $\nu \leq 2$ or $P > 0.98$) or regula falsi to find the point at which the chi-squared distribution function is equal to P . The distribution function is evaluated using routine **CHIDF**.

For $40 \leq \nu < 100$, a modified Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.18) to the normal distribution is used, and routine **ANORIN** is used to evaluate the inverse of the normal distribution function. For $\nu \geq 100$, the ordinary Wilson-Hilferty approximation (Abramowitz and Stegun 1964, equation 26.4.17) is used.

Comments

Informational error

Type	Code	Description
4	1	Over 100 iterations have occurred without convergence. Convergence is assumed.

Example

In this example, we find the 99-th percentage points of a chi-squared random variable with 2 degrees of freedom and of one with 64 degrees of freedom.

```

      USE UMACH_INT
      USE CHIIN_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL DF, P, X
!
      CALL UMACH (2, NOUT)
      P = 0.99
      DF = 2.0
      X = CHIIN(P,DF)
      WRITE (NOUT,99998) X
99998 FORMAT (' The 99-th percentage point of chi-squared with 2 df ' &
             , 'is ', F7.3)
      DF = 64.0
      X = CHIIN(P,DF)
      WRITE (NOUT,99999) X
99999 FORMAT (' The 99-th percentage point of chi-squared with 64 df ' &
             , 'is ', F7.3)
      END

```

Output

```

The 99-th percentage point of chi-squared with 2 df is 9.210
The 99-th percentage point of chi-squared with 64 df is 93.217

```


CHIPR

This function evaluates the chi-squared probability density function.

Function Return Value

CHIPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the chi-squared probability density function is to be evaluated. (Input)

DF — Number of degrees of freedom of the chi-squared distribution. (Input)

FORTRAN 90 Interface

Generic: **CHIPR (X, DF)**

Specific: The specific interface names are **S_CHIPR** and **D_CHIPR**.

FORTRAN 77 Interface

Single: **CHIPR (X, DF)**

Double: The double precision name is **DCHIPR**.

Description

The function **CHIPR** evaluates the chi-squared probability density function. The chi-squared distribution is a special case of the gamma distribution and is defined as

$$f(x|v) = \Gamma(x|v/2, 2) = \frac{1}{2^{v/2}\Gamma(v/2)}(x)^{v/2-1}e^{-\frac{x}{2}}, \quad x, v > 0$$

Example

In this example, we evaluate the probability function at **X = 3.0**, **DF = 5.0**.

```
USE UMACH_INT
USE CHIPR_INT
IMPLICIT NONE
```

```
INTEGER NOUT
REAL X, DF, PR
CALL UMACH(2, NOUT)
X = 3.0
DF = 5.0
PR = CHIPR(X, DF)
WRITE (NOUT, 99999) X, DF, PR
99999 FORMAT (' CHIPR(', F4.2, ', ', ' ', F4.2, ') = ', F6.4)
END
```

Output

```
CHIPR(3.00, 5.00) = 0.1542
```

CSNDF

This function evaluates the noncentral chi-squared cumulative distribution function.

Function Return Value

CSNDF — Function value, the probability that a noncentral chi-squared random variable takes a value less than or equal to **CHSQ**. (Output)

Required Arguments

CHSQ — Argument for which the noncentral chi-squared cumulative distribution function is to be evaluated. (Input)

DF — Number of degrees of freedom of the noncentral chi-squared cumulative distribution. (Input)
DF must be positive and less than or equal to 200,000.

ALAM — The noncentrality parameter. (Input)
ALAM must be nonnegative, and **ALAM** + **DF** must be less than or equal to 200,000.

FORTRAN 90 Interface

Generic: **CSNDF** (**CHSQ**, **DF**, **ALAM**)
Specific: The specific interface names are **S_CSNDFF** and **D_CSNDFF**.

FORTRAN 77 Interface

Single: **CSNDF** (**CHSQ**, **DF**, **ALAM**)
Double: The double precision name is **DCSNDF**.

Description

Function **CSNDF** evaluates the cumulative distribution function of a noncentral chi-squared random variable with **DF** degrees of freedom and noncentrality parameter **ALAM**, that is, with $\nu = \mathbf{DF}$, $\lambda = \mathbf{ALAM}$, and $x = \mathbf{CHSQ}$,

$$F(x | v, \lambda) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^x \frac{t^{(v+2i)/2-1} e^{-t/2}}{2^{(v+2i)/2} \Gamma(\frac{v+2i}{2})} dt$$

where $\Gamma(\cdot)$ is the gamma function. This is a series of central chi-squared distribution functions with Poisson weights. The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The noncentral chi-squared random variable can be defined by the distribution function above, or alternatively and equivalently, as the sum of squares of independent normal random variables. If Y_i have independent normal distributions with means μ_i and variances equal to one and

$$X = \sum_{i=1}^n Y_i^2$$

then X has a noncentral chi-squared distribution with n degrees of freedom and noncentrality parameter equal to

$$\sum_{i=1}^n \mu_i^2$$

With a noncentrality parameter of zero, the noncentral chi-squared distribution is the same as the chi-squared distribution.

Function **CSNDF** determines the point at which the Poisson weight is greatest, and then sums forward and backward from that point, terminating when the additional terms are sufficiently small or when a maximum of 1000 terms have been accumulated. The recurrence relation 26.4.8 of Abramowitz and Stegun (1964) is used to speed the evaluation of the central chi-squared distribution functions.

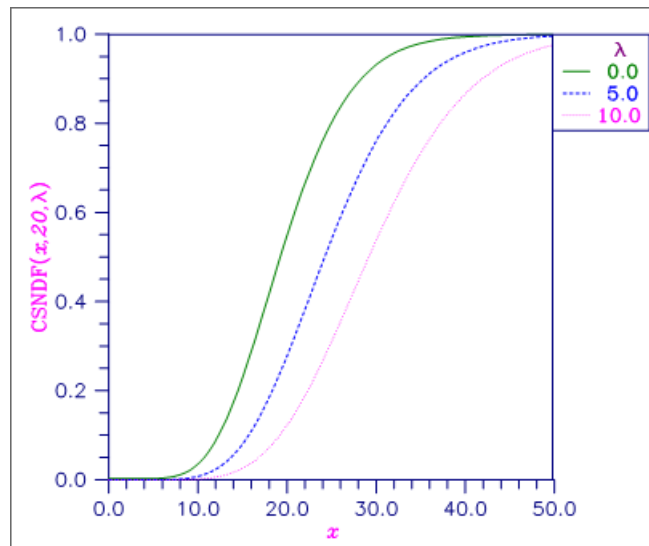


Figure 26, Noncentral Chi-squared Distribution Function

Example

In this example, CSNDF is used to compute the probability that a random variable that follows the noncentral chi-squared distribution with noncentrality parameter of 1 and with 2 degrees of freedom is less than or equal to 8.642.

```

USE UMACH_INT
USE CSNDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL ALAM, CHSQ, DF, P
!
CALL UMACH (2, NOUT)
DF = 2.0
ALAM = 1.0
CHSQ = 8.642
P = CSNDF(CHSQ,DF,ALAM)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that a noncentral chi-squared random', &
/, ' variable with 2 df and noncentrality 1.0 is less', &
/, ' than 8.642 is ', F5.3)
END

```

Output

```

The probability that a noncentral chi-squared random
variable with 2 df and noncentrality 1.0 is less
than 8.642 is 0.950

```

CSNIN

This function evaluates the inverse of the noncentral chi-squared cumulative function.

Function Return Value

CSNIN — Function value. (Output)

The probability that a noncentral chi-squared random variable takes a value less than or equal to **CSNIN** is **P**.

Required Arguments

P — Probability for which the inverse of the noncentral chi-squared cumulative distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

DF — Number of degrees of freedom of the noncentral chi-squared distribution. (Input)

DF must be greater than or equal to 0.5 and less than or equal to 200,000.

ALAM — The noncentrality parameter. (Input)

ALAM must be nonnegative, and **ALAM** + **DF** must be less than or equal to 200,000.

FORTRAN 90 Interface

Generic: **CSNIN** (**P**, **DF**, **ALAM**)

Specific: The specific interface names are **S_CSININ** and **D_CSININ**.

FORTRAN 77 Interface

Single: **CSNIN** (**P**, **DF**, **ALAM**)

Double: The double precision name is **DCSNIN**.

Description

Function **CSNIN** evaluates the inverse distribution function of a noncentral chi-squared random variable with **DF** degrees of freedom and noncentrality parameter **ALAM**; that is, with $P = \mathbf{P}$, $\nu = \mathbf{DF}$, and $\lambda = \mathbf{ALAM}$, it determines $c_0 (= \mathbf{CSNIN}(\mathbf{P}, \mathbf{DF}, \mathbf{ALAM}))$, such that

$$P = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} \int_0^{c_0} \frac{x^{(v+2i)/2-1} e^{-x/2}}{2^{(v+2i)/2} \Gamma(\frac{v+2i}{2})} dx$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to c_0 is P .

Function **CSNIN** uses bisection and modified regula falsi to invert the distribution function, which is evaluated using routine **CSNDF**. See **CSNDF** for an alternative definition of the noncentral chi-squared random variable in terms of normal random variables.

Comments

Informational error

Type	Code	Description
4	1	Over 100 iterations have occurred without convergence. Convergence is assumed.

Example

In this example, we find the 95-th percentage point for a noncentral chi-squared random variable with 2 degrees of freedom and noncentrality parameter 1.

```

USE CSNIN_INT
USE UMACH_INT
IMPLICIT NONE
INTEGER NOUT
REAL ALAM, CHSQ, DF, P
!
CALL UMACH (2, NOUT)
DF = 2.0
ALAM = 1.0
P = 0.95
CHSQ = CSNIN(P, DF, ALAM)
WRITE (NOUT, 99999) CHSQ
!
99999 FORMAT (' The 0.05 noncentral chi-squared critical value is ', &
             F6.3, '.')
!
END

```

Output

```
The 0.05 noncentral chi-squared critical value is 8.642.
```

CSNPR

This function evaluates the noncentral chi-squared probability density function.

Function Return Value

CSNPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the noncentral chi-squared probability density function is to be evaluated. (Input)

X must be non-negative.

DF — Number of degrees of freedom of the noncentral chi-squared distribution. (Input)

DF must be positive.

LAMBDA — Noncentrality parameter. (Input)

LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: CSNPR (X, DF, LAMBDA)

Specific: The specific interface names are S_CSNPR and D_CSNPR.

Description

The noncentral chi-squared distribution is a generalization of the chi-squared distribution. If $\{X_i\}$ are k independent, normally distributed random variables with means μ_i and variances σ_i^2 , then the random variable:

$$X = \sum_{i=1}^k \left(\frac{X_i}{\sigma_i} \right)^2$$

is distributed according to the noncentral chi-squared distribution. The noncentral chi-squared distribution has two parameters: k which specifies the number of degrees of freedom (i.e. the number of X_i), and λ which is related to the mean of the random variables X_i by:

$$\lambda = \sum_{i=1}^k \left(\frac{\mu_i}{\sigma_i} \right)^2$$

The noncentral chi-squared distribution is equivalent to a (central) chi-squared distribution with $k + 2i$ degrees of freedom, where i is the value of a Poisson distributed random variable with parameter $\lambda/2$. Thus, the probability density function is given by:

$$F(x, k, \lambda) = \sum_{i=0}^{\infty} \frac{e^{-\lambda/2} (\lambda/2)^i}{i!} f(x, k + 2i)$$

where the (central) chi-squared PDF $f(x, k)$ is given by:

$$f(x, k) = \frac{(x/2)^{k/2} e^{-x/2}}{x \Gamma(k/2)} \quad \text{for } x > 0, \text{ else } 0$$

where $\Gamma(\cdot)$ is the gamma function. The above representation of $F(x, k, \lambda)$ can be shown to be equivalent to the representation:

$$F(x, k, \lambda) = \frac{e^{-(\lambda+x)/2} (x/2)^{k/2}}{x} \sum_{i=0}^{\infty} \phi_i$$

$$\phi_i = \frac{(\lambda x / 4)^i}{i! \Gamma(k/2 + i)}$$

Function `CSNPR (X, DF, LAMBDA)` evaluates the probability density function of a noncentral chi-squared random variable with `DF` degrees of freedom and noncentrality parameter `LAMBDA`, corresponding to $k = \text{DF}$, $\lambda = \text{LAMBDA}$, and $x = X$.

Function `CSNDF (X, DF, LAMBDA)` evaluates the cumulative distribution function incorporating the above probability density function.

With a noncentrality parameter of zero, the noncentral chi-squared distribution is the same as the central chi-squared distribution.

Example

This example calculates the noncentral chi-squared distribution for a distribution with 100 degrees of freedom and noncentrality parameter $\lambda = 40$.

```
USE UMACH_INT
USE CSNPR_INT
```

```
IMPLICIT NONE

INTEGER :: NOUT, I
REAL    :: X(6)=(/ 0.0, 8.0, 40.0, 136.0, 280.0, 400.0 /)
REAL    :: LAMBDA=40.0, DF=100.0, PDFV

CALL UMACH (2, NOUT)
WRITE (NOUT, '(// "DF: ", F4.0, "  LAMBDA: ", F4.0 //') &
      ' " X          PDF(X)') DF, LAMBDA
DO I = 1, 6
  PDFV = CSNPR(X(I), DF, LAMBDA)
  WRITE (NOUT, '(1X, F5.0, 2X, E12.5)') X(I), PDFV
END DO
END
```

Output

```
DF: 100.  LAMBDA:  40.

 X          PDF(X)
 0.  0.00000E+00
 8.  0.00000E+00
40.  0.34621E-13
136. 0.21092E-01
280. 0.40027E-09
400. 0.11250E-21
```

EXPDF

This function evaluates the exponential cumulative distribution function.

Function Return Value

EXPDF — Function value, the probability that an exponential random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the exponential cumulative distribution function is to be evaluated. (Input)

B — Scale parameter of the exponential distribution function. (Input)

FORTRAN 90 Interface

Generic: **EXPDF** (**X**, **B**)

Specific: The specific interface names are **S_EXPDF** and **D_EXPDF**.

FORTRAN 77 Interface

Single: **EXPDF** (**X**, **B**)

Double: The double precision name is **DEXPDF**.

Description

The function **EXPDF** evaluates the exponential cumulative distribution function (CDF), defined:

$$F(x|b) = \int_0^x f(t|b) dt = 1 - e^{-\frac{x}{b}}$$

where

$$f(x|b) = \frac{1}{b} e^{-\frac{x}{b}}$$

is the exponential probability density function (PDF).

Example

In this example, we evaluate the probability function at $X = 2.0$, $B = 1.0$.

```
USE UMACH_INT
USE EXPDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, B, PR
CALL UMACH(2, NOUT)
X = 2.0
B = 1.0
PR = EXPDF(X, B)
WRITE (NOUT, 99999) X, B, PR
99999 FORMAT (' EXPDF(', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
EXPDF(2.00, 1.00) = 0.8647
```

EXPIN

This function evaluates the inverse of the exponential cumulative distribution function.

Function Return Value

EXPIN — Function value, the value of the inverse of the cumulative distribution function. (Output)

Required Arguments

P — Probability for which the inverse of the exponential distribution function is to be evaluated. (Input)

B — Scale parameter of the exponential distribution function. (Input)

FORTRAN 90 Interface

Generic: `EXPIN (P, B)`

Specific: The specific interface names are `S_EXPIN` and `D_EXPIN`.

FORTRAN 77 Interface

Single: `EXPIN (P, B)`

Double: The double precision name is `DEXPIN`.

Description

The function `EXPIN` evaluates the inverse distribution function of an exponential random variable with scale parameter $b = B$.

Example

In this example, we evaluate the inverse probability function at $P = 0.8647$, $B = 1.0$.

```
USE UMACH_INT
USE EXPIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, B, P
CALL UMACH(2, NOUT)
```

```
P = 0.8647
B = 1.0
X = EXPIN(P, B)
WRITE (NOUT, 99999) P, B, X
99999 FORMAT (' EXPIN(', F6.4, ', ', ', F4.2, ') = ', F6.4)
END
```

Output

```
EXPIN(0.8647, 1.00) = 2.0003
```

EXPPR

This function evaluates the exponential probability density function.

Function Return Value

EXPPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the exponential probability density function is to be evaluated. (Input)

B — Scale parameter of the exponential probability density function. (Input)

FORTRAN 90 Interface

Generic: **EXPPR (X, B)**

Specific: The specific interface names are **S_EXPPR** and **D_EXPPR**.

FORTRAN 77 Interface

Single: **EXPPR (X, B)**

Double: The double precision name is **DEXPPR**.

Description

The function **EXPPR** evaluates the exponential probability density function. The exponential distribution is a special case of the gamma distribution and is defined as

$$f(x | b) = \Gamma(x | 1, b) = \frac{1}{b} e^{-\frac{x}{b}}, \quad x, b > 0$$

This relationship is used in the computation of $f(x|b)$.

Example

In this example, we evaluate the probability function at $X = 2.0$, $B = 1.0$.

```
USE UMACH_INT
USE EXPPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, B, PR
CALL UMACH(2, NOUT)
X = 2.0
B = 1.0
PR = EXPPR(X, B)
WRITE (NOUT, 99999) X, B, PR
99999 FORMAT (' EXPPR(', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
EXPPR(2.00, 1.00) = 0.1353
```

EXVDF

This function evaluates the extreme value cumulative distribution function.

Function Return Value

EXVDF — Function value, the probability that an extreme value random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the extreme value cumulative distribution function is to be evaluated. (Input)

AMU — Location parameter of the extreme value probability distribution function. (Input)

BETA — Scale parameter of the extreme value probability distribution function. (Input)

FORTRAN 90 Interface

Generic: **EXVDF** (**X**, **AMU**, **BETA**)

Specific: The specific interface names are **S_EXVDF** and **D_EXVDF**.

FORTRAN 77 Interface

Single: **EXVDF** (**X**, **AMU**, **BETA**)

Double: The double precision name is **DEXVDF**.

Description

The function **EXVDF** evaluates the extreme value cumulative distribution function, defined as

$$F(x | \mu, \beta) = 1 - e^{-e^{\frac{x-\mu}{\beta}}}$$

The extreme value distribution is also known as the Gumbel minimum distribution.

Example

In this example, we evaluate the probability function at $X = 1.0$, $AMU = 0.0$, $BETA = 1.0$.

```
USE UMACH_INT
USE EXVDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, AMU, B, PR
CALL UMACH(2, NOUT)
X = 1.0
AMU = 0.0
B = 1.0
PR = EXVDF(X, AMU, B)
WRITE (NOUT, 99999) X, AMU, B, PR
99999 FORMAT (' EXVDF(' , F6.2, ', ', ' , F4.2, ', ', ' , F4.2, ') = ', F6.4)
END
```

Output

```
EXVDF( 1.00, 0.00, 1.00) = 0.9340
```

EXVIN

This function evaluates the inverse of the extreme value cumulative distribution function.

Function Return Value

EXVIN — Function value, the value of the inverse of the extreme value cumulative distribution function.
(Output)

Required Arguments

P — Probability for which the inverse of the extreme value distribution function is to be evaluated. (Input)

AMU — Location parameter of the extreme value probability function. (Input)

BETA — Scale parameter of the extreme value probability function. (Input)

FORTRAN 90 Interface

Generic: `EXVIN (P, AMU, BETA)`

Specific: The specific interface names are `S_EXVIN` and `D_EXVIN`.

FORTRAN 77 Interface

Single: `EXVIN (P, AMU, BETA)`

Double: The double precision name is `DEXVIN`.

Description

The function **EXVIN** evaluates the inverse distribution function of an extreme value random variable with location parameter **AMU** and scale parameter **BETA**.

Example

In this example, we evaluate the inverse probability function at $P = 0.934$, $AMU = 1.0$, $BETA = 1.0$

```
USE UMACH_INT
USE EXVIN_INT
IMPLICIT NONE
```

```
INTEGER NOUT
REAL X, AMU, B, PR
CALL UMACH(2, NOUT)
PR = .934
AMU = 0.0
B = 1.0
X = EXVIN(PR, AMU, B)
WRITE (NOUT, 99999) PR, AMU, B, X
99999 FORMAT (' EXVIN(', F6.3, ', ', ', F4.2, ', ', ', F4.2, ') = ', F6.4)
END
```

Output

```
EXVIN( 0.934, 0.00, 1.00) = 0.9999
```

EXVPR

This function evaluates the extreme value probability density function.

Function Return Value

EXVPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the extreme value probability density function is to be evaluated. (Input)

AMU — Location parameter of the extreme value probability density function. (Input)

BETA — Scale parameter of the extreme value probability density function. (Input)

FORTRAN 90 Interface

Generic: **EXVPR (X, AMU, BETA)**

Specific: The specific interface names are **S_EXVPR** and **D_EXVPR**.

FORTRAN 77 Interface

Single: **EXVPR (X, AMU, BETA)**

Double: The double precision name is **DEXVPR**.

Description

The function **EXVPR** evaluates the extreme value probability density function, defined as

$$f(x | \mu, \beta) = \beta^{-1} e^{\frac{x-\mu}{\beta}} e^{-e^{\frac{x-\mu}{\beta}}}, \quad -\infty < x, \mu < +\infty, \beta > 0$$

The extreme value distribution is also known as the Gumbel minimum distribution.

Example

In this example, we evaluate the extreme value probability density function at **X = 2.0**, **AMU = 0.0**, **BETA = 1.0**.

```
USE UMACH_INT
USE EXVPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, AMU, B, PR
CALL UMACH(2, NOUT)
X = -2.0
AMU = 0.0
B = 1.0
PR = EXVPR(X, AMU, B)
WRITE (NOUT, 99999) X, AMU, B, PR
99999 FORMAT (' EXVPR(', F6.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
EXVPR( -2.00, 0.00, 1.00) = 0.1182
```

FDF

This function evaluates the F cumulative distribution function.

Function Return Value

FDF — Function value, the probability that an F random variable takes a value less than or equal to the input F . (Output)

Required Arguments

F — Argument for which the F cumulative distribution function is to be evaluated. (Input)

DFN — Numerator degrees of freedom. (Input)
DFN must be positive.

DFD — Denominator degrees of freedom. (Input)
DFD must be positive.

Optional Arguments

COMPLEMENT — Logical. If `.TRUE.`, the complement of the F cumulative distribution function is evaluated. If `.FALSE.`, the F cumulative distribution function is evaluated. (Input)
See the [Description](#) section for further details on the use of **COMPLEMENT**.
Default: `COMPLEMENT = .FALSE.`

FORTRAN 90 Interface

Generic: **FDF** (**F**, **DFN**, **DFD** [, ...])
Specific: The specific interface names are **S_FDF** and **D_FDF**.

FORTRAN 77 Interface

Single: **FDF** (**F**, **DFN**, **DFD**)
Double: The double precision name is **DFDF**.

Description

Function **FDF** evaluates the distribution function of a Snedecor's F random variable with **DFN** numerator degrees of freedom and **DFD** denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then using the routine **BETDF**. If X is an F variate with ν_1 and ν_2 degrees of freedom and $Y = \nu_1 X / (\nu_2 + \nu_1 X)$, then Y is a beta variate with parameters $p = \nu_1/2$ and $q = \nu_2/2$. The function **FDF** also uses a relationship between F random variables that can be expressed as follows.

$$\text{FDF}(X, \text{DFN}, \text{DFD}) = 1.0 - \text{FDF}(1.0/X, \text{DFD}, \text{DFN})$$

If **COMPLEMENT** = **.TRUE.**, the value of **FDF** at the point x is $1 - p$, where $1 - p$ is the probability that the random variable takes a value greater than x . In those situations where the desired end result is $1 - p$, the user can achieve greater accuracy in the right tail region by using the result returned by **FDF** with the optional argument **COMPLEMENT** set to **.TRUE.** rather than by using $1 - p$ where p is the result returned by **FDF** with **COMPLEMENT** set to **.FALSE.**

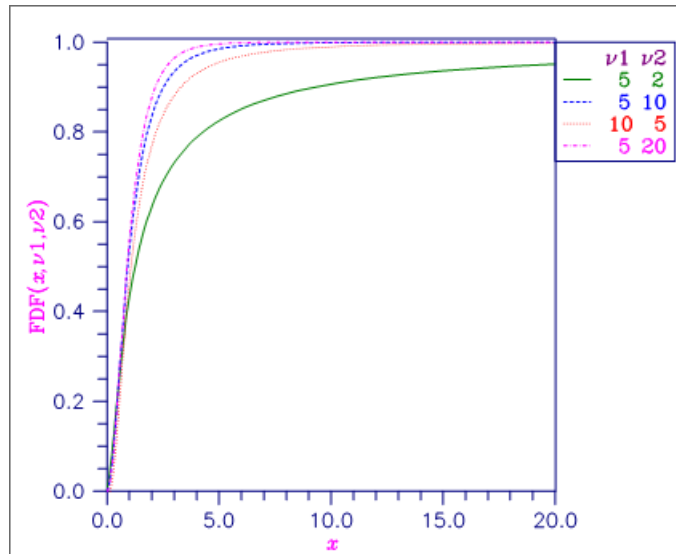


Figure 27, F Distribution Function

Comments

Informational error

Type	Code	Description
1	3	Since the input argument F is not positive, the distribution function is zero at F .

Example

In this example, we find the probability that an F random variable with one numerator and one denominator degree of freedom is greater than 648.

```
      USE UMACH_INT
      USE FDF_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL DFD, DFN, F, P
!
      CALL UMACH (2, NOUT)
      F = 648.0
      DFN = 1.0
      DFD = 1.0
      P = FDF(F,DFN,DFD, COMPLEMENT=.TRUE.)
      WRITE (NOUT,99999) P
99999 FORMAT (' The probability that an F(1,1) variate is greater ', &
             'than 648 is ', F6.4)
      END
```

Output

```
The probability that an F(1, 1) variate is greater than 648 is 0.0250
```

FIN

This function evaluates the inverse of the F cumulative distribution function.

Function Return Value

FIN — Function value. (Output)

The probability that an F random variable takes a value less than or equal to **FIN** is P .

Required Arguments

P — Probability for which the inverse of the F distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

DFN — Numerator degrees of freedom. (Input)

DFN must be positive.

DFD — Denominator degrees of freedom. (Input)

DFD must be positive.

FORTRAN 90 Interface

Generic: **FIN** (**P**, **DFN**, **DFD**)

Specific: The specific interface names are **S_FDF** and **D_FDF**.

FORTRAN 77 Interface

Single: **FIN** (**P**, **DFN**, **DFD**)

Double: The double precision name is **DFDF**.

Description

Function **FIN** evaluates the inverse distribution function of a Snedecor's F random variable with **DFN** numerator degrees of freedom and **DFD** denominator degrees of freedom. The function is evaluated by making a transformation to a beta random variable and then using the routine **BETIN**. If X is an F variate with ν_1 and ν_2 degrees of

freedom and $Y = \nu_1 X / (\nu_2 + \nu_1 X)$, then Y is a beta variate with parameters $p = \nu_1/2$ and $q = \nu_2/2$. If $P \leq 0.5$, **FIN** uses this relationship directly, otherwise, it also uses a relationship between F random variables that can be expressed as follows, using routine **FDF**, which is the F cumulative distribution function:

$$\text{FDF}(F, \text{DFN}, \text{DFD}) = 1.0 - \text{FDF}(1.0/F, \text{DFD}, \text{DFN}).$$

Comments

Informational error

Type	Code	Description
4	4	FIN is set to machine infinity since overflow would occur upon modifying the inverse value for the F distribution with the result obtained from the inverse beta distribution.

Example

In this example, we find the 99-th percentage point for an F random variable with 1 and 7 degrees of freedom.

```

      USE UMACH_INT
      USE FIN_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL DFD, DFN, F, P
!
      CALL UMACH (2, NOUT)
      P = 0.99
      DFN = 1.0
      DFD = 7.0
      F = FIN(P,DFN,DFD)
      WRITE (NOUT,99999) F
99999 FORMAT (' The F(1,7) 0.01 critical value is ', F6.3)
      END

```

Output

```
The F(1, 7) 0.01 critical value is 12.246
```

FPR

This function evaluates the F probability density function.

Function Return Value

FPR — Function value, the value of the probability density function. (Output)

Required Arguments

F — Argument for which the F probability density function is to be evaluated. (Input)

DFN — Numerator degrees of freedom. (Input)

DFN must be positive.

DFD — Denominator degrees of freedom. (Input)

DFD must be positive.

FORTRAN 90 Interface

Generic: **FPR (F, DFN, DFD)**

Specific: The specific interface names are **S_FPR** and **D_FDPR**

FORTRAN 77 Interface

Single: **FPR (F, DFN, DFD)**

Double: The double precision name is **DFPR**.

Description

The function **FPR** evaluates the F probability density function, defined as

$$f(x|v_1, v_2) = n(v_1, v_2) x^{\frac{v_1-2}{2}} \left(1 + \frac{v_1 x}{v_2}\right)^{-\frac{(v_1+v_2)}{2}},$$

$$n(v_1, v_2) = \frac{\Gamma\left(\frac{v_1+v_2}{2}\right)}{\Gamma\left(\frac{v_1}{2}\right)\Gamma\left(\frac{v_2}{2}\right)} \left(\frac{v_1}{v_2}\right)^{\frac{v_1}{2}}, \quad x > 0, v_i > 0, \quad i = 1, 2$$

The parameters v_1 and v_2 correspond to the arguments DFN and DFD.

Example

In this example, we evaluate the probability function at $F = 2.0$, $DFN = 10.0$, $DFD = 1.0$.

```

USE UMACH_INT
USE FPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL F, DFN, DFD, PR
CALL UMACH(2, NOUT)
F = 2.0
DFN = 10.0
DFD = 1.0
PR = FPR(F, DFN, DFD)
WRITE (NOUT, 99999) F, DFN, DFD, PR
99999 FORMAT (' FPR( ', F6.2, ', ', F6.2, ', ', F6.2, ') = ', F6.4)
END

```

Output

```
FPR( 2.00, 10.00, 1.00) = 0.1052
```

FPDF

This function evaluates the noncentral F cumulative distribution function (CDF).

Function Return Value

FPDF — Probability that a random variable from an F distribution having noncentrality parameter **LAMBDA** takes a value less than or equal to the input **F**. (Output)

Required Arguments

F — Argument for which the noncentral F cumulative distribution function is to be evaluated. (Input)
F must be non-negative.

DF1 — Number of numerator degrees of freedom of the noncentral F distribution. (Input)
DF1 must be positive.

DF2 — Number of denominator degrees of freedom of the noncentral F distribution. (Input)
DF2 must be positive.

LAMBDA — Noncentrality parameter. (Input)
LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: FPDF (F, DF1, DF2, LAMBDA)

Specific: The specific interface names are S_FPDF and D_FPDF.

Description

If X is a noncentral chi-square random variable with noncentrality parameter λ and ν_1 degrees of freedom, and Y is a chi-square random variable with ν_2 degrees of freedom which is statistically independent of X , then

$$F = (X / \nu_1) / (Y / \nu_2)$$

is a noncentral F -distributed random variable whose CDF is given by

$$CDF(f, v_1, v_2, \lambda) = \sum_{j=0}^{\infty} c_j$$

where

$$c_j = \omega_j I_x\left(\frac{v_1}{2} + j, \frac{v_2}{2}\right)$$

$$\omega_j = e^{-\lambda/2} (\lambda/2)^j / j! = \frac{\lambda}{2j} \omega_{j-1}$$

$$I_x(a, b) = B_x(a, b) / B(a, b)$$

$$B_x(a, b) = \int_0^x t^{a-1} (1-t)^{b-1} dt = x^a \sum_{j=0}^{\infty} \frac{\Gamma(j+1-b)}{(a+j)\Gamma(1-b)j!} x^j$$

$$x = v_1 f / (v_2 + v_1 f)$$

$$B(a, b) = B_1(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

$$I_x(a+1, b) = I_x(a, b) - T_x(a, b)$$

$$T_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^b = T_x(a-1, b) \frac{a-1+b}{a} x$$

and $\Gamma(\cdot)$ is the gamma function. The above series expansion for the noncentral F CDF was taken from Butler and Paoletta (1999) (see [Paoletta.pdf](#)), with the correction for the recursion relation given below:

$$I_x(a+1, b) = I_x(a, b) - T_x(a, b)$$

extracted from the AS 63 algorithm for calculating the incomplete beta function as described by Majumder and Bhattacharjee (1973).

The correspondence between the arguments of function `FPDF(F, DF1, DF2, LAMBDA)` and the variables in the above equations is as follows: $v_1 = DF1$, $v_2 = DF2$, $\lambda = LAMBDA$, and $f = F$.

For $\lambda = 0$, the noncentral F distribution is the same as the F distribution.

Example

This example traces out a portion of a noncentral F distribution with parameters $DF1 = 100$, $DF2 = 10$, and $LAMBDA = 10$.

```

USE UMACH_INT
USE FPDF_INT
IMPLICIT NONE
INTEGER NOUT, I
REAL X, LAMBDA, DF1, DF2, CDFV, X0(8)
DATA X0 / 0.0, .4, .8, 1.2, 1.6, 2.0, 2.8, 4.0 /

CALL UMACH (2, NOUT)
DF1 = 100.0
DF2 = 10.0
LAMBDA = 10.0
WRITE (NOUT, '( "DF1: ", F4.0, "; DF2: ", F4.0, &
" ; LAMBDA: ", F4.0 // " X          CDF(X)" )') &
DF1, DF2, LAMBDA
DO I = 1, 8
X = X0(I)
CDFV = FPDF(X, DF1, DF2, LAMBDA)
WRITE (NOUT, '(1X, F5.1, 2X, E12.6)') X, CDFV
END DO
END

```

Output

```
DF1: 100.; DF2: 10.; LAMBDA: 10.
```

X	CDF(X)
0.0	0.000000E+00
0.4	0.488790E-02
0.8	0.202633E+00
1.2	0.521143E+00
1.6	0.733853E+00
2.0	0.850413E+00
2.8	0.947125E+00
4.0	0.985358E+00

FNIN

This function evaluates the inverse of the noncentral F cumulative distribution function (CDF).

Function Return Value

FNIN — Function value, the value of the inverse of the cumulative distribution function evaluated at P . The probability that a noncentral F random variable takes a value less than or equal to **FNIN** is P . (Output)

Required Arguments

P — Probability for which the inverse of the noncentral F cumulative distribution function is to be evaluated. (Input)

P must be non-negative and less than 1.

DF1 — Number of numerator degrees of freedom of the noncentral F distribution. (Input)

DF1 must be positive.

DF2 — Number of denominator degrees of freedom of the noncentral F distribution. (Input)

DF2 must be positive.

LAMBDA — Noncentrality parameter. (Input)

LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: `FNIN (P, DF1, DF2, LAMBDA)`

Specific: The specific interface names are `S_FNIN` and `D_FNIN`.

Description

If X is a noncentral chi-square random variable with noncentrality parameter λ and ν_1 degrees of freedom, and Y is a chi-square random variable with ν_2 degrees of freedom which is statistically independent of X , then

$$F = (X / \nu_1) / (Y / \nu_2)$$

is a noncentral F -distributed random variable whose CDF is given by

$$p = CDF(f, v_1, v_2, \lambda) = \sum_{j=0}^{\infty} c_j$$

where:

$$c_j = \omega_j I_x\left(\frac{v_1}{2} + j, \frac{v_2}{2}\right)$$

$$\omega_j = e^{-\lambda/2} (\lambda/2)^j / j! = \frac{\lambda}{2j} \omega_{j-1}$$

$$I_x(a, b) = B_x(a, b) / B(a, b)$$

$$B_x(a, b) = \int_0^x t^{a-1} (1-t)^{b-1} dt = x^a \sum_{j=0}^{\infty} \frac{\Gamma(j+1-b)}{(a+j)\Gamma(1-b)j!} x^j$$

$$x = v_1 f / (v_2 + v_1 f)$$

$$B(a, b) = B_1(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

$$I_x(a+1, b) = I_x(a, b) - T_x(a, b)$$

$$T_x(a, b) = \frac{\Gamma(a+b)}{\Gamma(a+1)\Gamma(b)} x^a (1-x)^b = T_x(a-1, b) \frac{a-1+b}{a} x$$

and $\Gamma(\cdot)$ is the gamma function, and $p = CDF(f)$ is the probability that $F \leq f$. The correspondence between the arguments of function `FNIN(P,DF1,DF2,LAMBDA)` and the variables in the above equations is as follows: $v_1 = DF1$, $v_2 = DF2$, $\lambda = LAMBDA$, and $p = P$.

Function `FNIN` evaluates

$$f = CDF^{-1}(p, v_1, v_1, \lambda)$$

Function `FNIN` uses bisection and modified regula falsi search algorithms to invert the distribution function $CDF(f)$, which is evaluated using function `FNDF`. For sufficiently small p , an accurate approximation of $CDF^{-1}(p)$ can be used which requires no such inverse search algorithms.

Example

This example traces out a portion of an inverse noncentral F distribution with parameters $DF1 = 100$, $DF2 = 10$, and $LAMBDA = 10$.

```

USE UMACH_INT
USE FNDF_INT
USE FNIN_INT
IMPLICIT NONE
INTEGER NOUT, I
REAL F, LAMBDA, DF1, DF2, CDF, CDFINV, F0(8)
DATA F0 / 0.0, .4, .8, 1.2, 1.6, 2.0, 2.8, 4.0 /

CALL UMACH (2, NOUT)
DF1 = 100.0
DF2 = 10.0
LAMBDA = 10.0
WRITE (NOUT, '( "DF1: ", F4.0, "; DF2: ", F4.0, &
" ; LAMBDA: ", F4.0 // " F      P = CDF(F)      CDFINV(P) ") ' ) &
DF1, DF2, LAMBDA
DO I = 1, 8
F = F0(I)
CDF = FNDF(F, DF1, DF2, LAMBDA)
CDFINV = FNIN(CDF, DF1, DF2, LAMBDA)
WRITE (NOUT, '(1X, F5.1, 2(2X, E12.6))') F, CDF, CDFINV
END DO
END

```

Output

```
DF1: 100.; DF2: 10.; LAMBDA: 10.
```

F	P = CDF(F)	CDFINV(P)
0.0	0.000000E+00	0.000000E+00
0.4	0.488790E-02	0.400000E+00
0.8	0.202633E+00	0.800000E+00
1.2	0.521143E+00	0.120000E+01
1.6	0.733853E+00	0.160000E+01
2.0	0.850413E+00	0.200000E+01
2.8	0.947125E+00	0.280000E+01
4.0	0.985358E+00	0.400000E+01

FNPR

This function evaluates the noncentral F probability density function.

Function Return Value

FNPR — Function value, the value of the probability density function. (Output)

Required Arguments

F — Argument for which the noncentral F probability density function is to be evaluated. (Input)
F must be non-negative.

DF1 — Number of numerator degrees of freedom of the noncentral F distribution. (Input)
DF1 must be positive.

DF2 — Number of denominator degrees of freedom of the noncentral F distribution. (Input)
DF2 must be positive.

LAMBDA — Noncentrality parameter. (Input)
LAMBDA must be non-negative.

FORTRAN 90 Interface

Generic: **FNPR** (**F**, **DF1**, **DF2**, **LAMBDA**)

Specific: The specific interface names are **S_FNPR** and **D_FNPR**.

Description

If X is a noncentral chi-square random variable with noncentrality parameter λ and ν_1 degrees of freedom, and Y is a chi-square random variable with ν_2 degrees of freedom which is statistically independent of X , then

$$F = (X / \nu_1) / (Y / \nu_2)$$

is a noncentral F -distributed random variable whose PDF is given by

$$PDF(f, v_1, v_2, \lambda) = \Psi \sum_{k=0}^{\infty} \Phi_k$$

where

$$\Psi = \frac{e^{-\lambda/2} (v_1 f)^{v_1/2} (v_2)^{v_2/2}}{f(v_1 f + v_2)^{(v_1+v_2)/2} \Gamma(v_2/2)}$$

$$\Phi_k = \frac{R^k \Gamma\left(\frac{v_1+v_2}{2} + k\right)}{k! \Gamma\left(\frac{v_1}{2} + k\right)}$$

$$R = \frac{\lambda v_1 f}{2(v_1 f + v_2)}$$

and $\Gamma(\cdot)$ is the gamma function, $v_1 = DF1$, $v_2 = DF2$, $\lambda = LAMBDA$, and $f = F$.

With a noncentrality parameter of zero, the noncentral F distribution is the same as the F distribution.

The efficiency of the calculation of the above series is enhanced by:

- calculating each term Φ_k in the series recursively in terms of either the term Φ_{k-1} preceding it or the term Φ_{k+1} following it, and
- initializing the sum with the largest series term and adding the subsequent terms in order of decreasing magnitude.

Special cases:

For $R = \lambda f = 0$:

$$PDF(f, v_1, v_2, \lambda) = \Psi \Phi_0 = \Psi \frac{\Gamma([v_1 + v_2] / 2)}{\Gamma(v_1 / 2)}$$

For $\lambda = 0$:

$$PDF(f, v_1, v_2, \lambda) = \frac{(v_1 f)^{v_1/2} (v_2)^{v_2/2} \Gamma([v_1 + v_2] / 2)}{f(v_1 f + v_2)^{(v_1+v_2)/2} \Gamma(v_1 / 2) \Gamma(v_2 / 2)}$$

For $f = 0$:

$$PDF(f, v_1, v_2, \lambda) = \frac{e^{-\lambda/2} f^{v_1/2-1} (v_1/v_2)^{v_1/2} \Gamma([v_1 + v_2]/2)}{\Gamma(v_1/2) \Gamma(v_2/2)} = \begin{cases} 0 & \text{if } v_1 > 2; \\ e^{-\lambda/2} & \text{if } v_1 = 2; \\ \infty & \text{if } v_1 < 2 \end{cases}$$

Example

This example traces out a portion of a noncentral F distribution with parameters $DF1 = 100$, $DF2 = 10$, and $LAMBDA = 10$.

```

USE UMACH_INT
USE FNPR_INT
IMPLICIT NONE

INTEGER NOUT, I
REAL F, LAMBDA, DF1, DF2, PDFV, X0(8)
DATA X0 /0.0, 0.4, 0.8, 3.2, 5.6,8.8, 14.0, 18.0/

CALL UMACH (2, NOUT)
DF1 = 100.0
DF2 = 10.0
LAMBDA = 10.0

WRITE (NOUT, '( "DF1: ", F4.0, "; DF2: ", F4.0, "; LAMBDA'// &
' : ", F4.0 // " F PDF(F)" )' ) DF1, DF2, LAMBDA

DO I = 1, 8
  F = X0(I)
  PDFV = FNPR(F, DF1, DF2, LAMBDA)
  WRITE (NOUT, '(1X, F5.1, 2X, E12.6)' ) F, PDFV
END DO
END

```

Output

```

DF1: 100.; DF2: 10.; LAMBDA: 10.

F PDF(F)
0.0 0.000000E+00
0.4 0.974879E-01
0.8 0.813115E+00
3.2 0.369482E-01
5.6 0.283023E-02
8.8 0.276607E-03
14.0 0.219632E-04
18.0 0.534831E-05

```

GAMDF

This function evaluates the gamma cumulative distribution function.

Function Return Value

GAMDF — Function value, the probability that a gamma random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the gamma distribution function is to be evaluated. (Input)

A — The shape parameter of the gamma distribution. (Input)
This parameter must be positive.

FORTRAN 90 Interface

Generic: **GAMDF** (**X**, **A**)

Specific: The specific interface names are **S_GAMDF** and **D_GAMDF**.

FORTRAN 77 Interface

Single: **GAMDF** (**X**, **A**)

Double: The double precision name is **DGAMDF**.

Description

Function **GAMDF** evaluates the distribution function, F , of a gamma random variable with shape parameter a ; that is,

$$F(x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. (The gamma function is the integral from 0 to ∞ of the same integrand as above). The value of the distribution function at the point x is the probability that the random variable takes a value less than or equal to x .

The gamma distribution is often defined as a two-parameter distribution with a scale parameter b (which must be positive), or even as a three-parameter distribution in which the third parameter c is a location parameter. In the most general case, the probability density function over (c, ∞) is

$$f(t) = \frac{1}{b^a \Gamma(a)} e^{-(t-c)/b} (x-c)^{a-1}$$

If T is such a random variable with parameters $a, b,$ and $c,$ the probability that $T \leq t_0$ can be obtained from **GAMDF** by setting $\mathbf{x} = (t_0 - c)/b.$

If \mathbf{x} is less than a or if \mathbf{x} is less than or equal to 1.0, **GAMDF** uses a series expansion. Otherwise, a continued fraction expansion is used. (See Abramowitz and Stegun, 1964.)

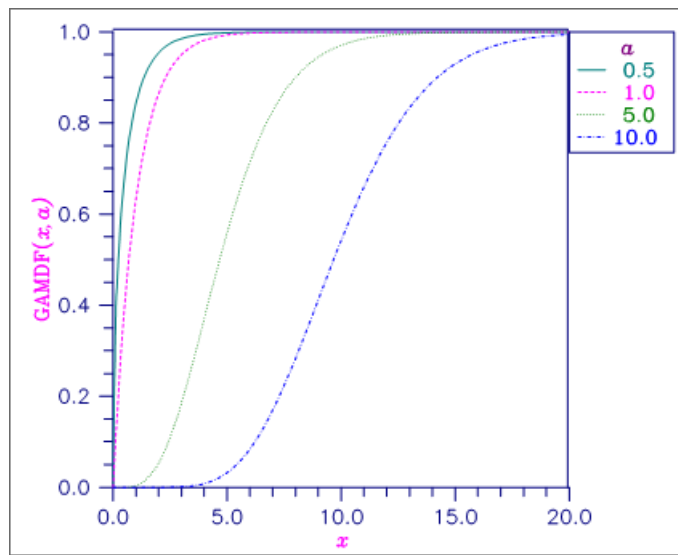


Figure 28, Gamma Distribution Function

Comments

Informational error

Type	Code	Description
1	2	Since the input argument x is less than zero, the distribution function is set to zero.

Example

Suppose X is a gamma random variable with a shape parameter of 4. (In this case, it has an *Erlang distribution* since the shape parameter is an integer.) In this example, we find the probability that X is less than 0.5 and the probability that X is between 0.5 and 1.0.

```
      USE UMACH_INT
      USE GAMDF_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL A, P, X
!
      CALL UMACH (2, NOUT)
      A = 4.0
      X = 0.5
      P = GAMDF(X,A)
      WRITE (NOUT,99998) P
99998 FORMAT (' The probability that X is less than 0.5 is ', F6.4)
      X = 1.0
      P = GAMDF(X,A) - P
      WRITE (NOUT,99999) P
99999 FORMAT (' The probability that X is between 0.5 and 1.0 is ', &
             F6.4)
      END
```

Output

```
The probability that X is less than 0.5 is 0.0018
The probability that X is between 0.5 and 1.0 is 0.0172
```

GAMIN

This function evaluates the inverse of the gamma cumulative distribution function.

Function Return Value

GAMIN — Function value. (Output)

The probability that a gamma random variable takes a value less than or equal to **GAMIN** is **P**.

Required Arguments

P — Probability for which the inverse of the gamma cumulative distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

A — The shape parameter of the gamma distribution. (Input)

This parameter must be positive.

FORTRAN 90 Interface

Generic: **GAMIN** (**P**, **A**)

Specific: The specific interface names are **S_GAMIN** and **D_GAMIN**.

FORTRAN 77 Interface

Single: **GAMIN** (**P**, **A**)

Double: The double precision name is **DGAMIN**.

Description

Function **GAMIN** evaluates the inverse distribution function of a gamma random variable with shape parameter a , that is, it determines x ($= \text{GAMIN}(\text{P}, \text{A})$), such that

$$P = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to x is P . See the documentation for routine **GAMDF** for further discussion of the gamma distribution.

Function **GAMIN** uses bisection and modified regula falsi to invert the distribution function, which is evaluated using routine **GAMDF**.

Comments

Informational error

Type	Code	Description
4	1	Over 100 iterations have occurred without convergence. Convergence is assumed.

Example

In this example, we find the 95-th percentage point for a gamma random variable with shape parameter of 4.

```
      USE UMACH_INT
      USE GAMIN_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL A, P, X
!
      CALL UMACH (2, NOUT)
      A = 4.0
      P = 0.95
      X = GAMIN(P,A)
      WRITE (NOUT,99999) X
!
99999 FORMAT (' The 0.05 gamma(4) critical value is ', F6.3, &
            ' .')
!
      END
```

Output

```
The 0.05 gamma(4) critical value is 7.754.
```

GAMPR

This function evaluates the gamma probability density function.

Function Return Value

GAMPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the gamma probability density function is to be evaluated. (Input)

A — The shape parameter of the gamma distribution. (Input)
This parameter must be positive.

FORTRAN 90 Interface

Generic: `GAMPR (X, A)`

Specific: The specific interface names are `S_GAMPR` and `D_GAMPR`.

FORTRAN 77 Interface

Single: `GAMPR (X, A)`

Double: The double precision name is `DGAMPR`.

Description

The function `GAMPR` evaluates the gamma probability density function, defined as

$$\Gamma(x|a) = \frac{1}{\Gamma(a)}(x)^{a-1}e^{-x}, \quad x, a > 0$$

Example

In this example, we evaluate the probability function at `X = 4.0`, `A = 5.0`.

```
USE UMACH_INT
USE GAMPR_INT
IMPLICIT NONE
```

```
INTEGER NOUT
REAL X, A, PR
CALL UMACH(2, NOUT)
X = 4.0
A = 5.0
PR = GAMPR(X, A)
WRITE (NOUT, 99999) X, A, PR
99999 FORMAT (' GAMPR(', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
GAMPR(4.00, 5.00) = 0.1954
```

RALDF

This function evaluates the Rayleigh cumulative distribution function.

Function Return Value

RALDF — Function value, the probability that a Rayleigh random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the Rayleigh cumulative distribution function is to be evaluated. (Input)

ALPHA — Scale parameter of the Rayleigh cumulative distribution function. (Input)

FORTRAN 90 Interface

Generic: `RALDF (X, ALPHA)`

Specific: The specific interface names are `S_RALDF` and `D_RALDF`.

FORTRAN 77 Interface

Single: `RALDF (X, ALPHA)`

Double: The double precision name is `DRALDF`.

Description

The function **RALDF** evaluates the Rayleigh cumulative probability distribution function, which is a special case of the Weibull cumulative probability distribution function, where the shape parameter **GAMMA** is 2.0

$$F(x) = 1 - e^{-\frac{x^2}{2\alpha^2}}$$

RALDF evaluates the Rayleigh cumulative probability distribution function using the relationship

$$\text{RALDF}(X, \text{ALPHA}) = \text{WBLDF}(X, \text{SQRT}(2.0) * \text{ALPHA}, 2.0).$$

Example

In this example, we evaluate the Rayleigh cumulative distribution function at $x = 0.25$, $\text{ALPHA} = 0.5$.

```
USE UMACH_INT
USE RALDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, ALPHA, PR
CALL UMACH(2, NOUT)
X = 0.25
ALPHA = 0.5
PR = RALDF(X, ALPHA)
WRITE (NOUT, 99999) X, ALPHA, PR
99999 FORMAT (' RALDF(', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
RALDF(0.25, 0.50) = 0.1175
```

RALIN

This function evaluates the inverse of the Rayleigh cumulative distribution function.

Function Return Value

RALIN — Function value, the value of the inverse of the cumulative distribution function. (Output)

Required Arguments

P — Probability for which the inverse of the Rayleigh distribution function is to be evaluated. (Input)

ALPHA — Scale parameter of the Rayleigh cumulative distribution function. (Input)

FORTRAN 90 Interface

Generic: `RALIN (P, ALPHA)`

Specific: The specific interface names are `S_RALIN` and `D_RALIN`.

FORTRAN 77 Interface

Single: `RALIN (P, ALPHA)`

Double: The double precision name is `DRALIN`.

Description

The function `RALIN` evaluates the inverse distribution function of a Rayleigh random variable with scale parameter `ALPHA`.

Example

In this example, we evaluate the inverse probability function at `P = 0.1175`, `ALPHA = 0.5`.

```
USE UMACH_INT
USE RALIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, ALPHA, P
CALL UMACH(2, NOUT)
```



```
P = 0.1175
ALPHA = 0.5
X = RALIN(P, ALPHA)
WRITE (NOUT, 99999) P, ALPHA, X
99999 FORMAT (' RALIN(', F6.4, ', ', ', F4.2, ') = ', F6.4)
END
```

Output

```
RALIN(0.1175, 0.50) = 0.2500
```

RALPR

This function evaluates the Rayleigh probability density function.

Function Return Value

RALPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the Rayleigh probability density function is to be evaluated. (Input)

ALPHA — Scale parameter of the Rayleigh probability function. (Input)

FORTRAN 90 Interface

Generic: `RALPR (X, ALPHA)`

Specific: The specific interface names are `S_RALPR` and `D_RALPR`.

FORTRAN 77 Interface

Single: `RALPR (X, ALPHA)`

Double: The double precision name is `DRALPR`.

Description

The function `RALPR` evaluates the Rayleigh probability density function, which is a special case of the Weibull probability density function where `GAMMA` is equal to 2.0, and is defined as

$$f(x|\alpha) = \frac{x}{\alpha^2} e^{-\left(\frac{x^2}{2\alpha^2}\right)}, \quad x > 0$$

Example

In this example, we evaluate the Rayleigh probability density function at `X = 0.25`, `ALPHA = 0.5`.

```
USE UMACH_INT
USE RALPR_INT
```

```
IMPLICIT NONE
INTEGER NOUT
REAL X, ALPHA, PR
CALL UMACH(2, NOUT)
X = 0.25
ALPHA = 0.5
PR = RALPR(X, ALPHA)
WRITE (NOUT, 99999) X, ALPHA, PR
99999 FORMAT (' RALPR(', F4.2, ', ', ' ', F4.2, ') = ', F6.4)
END
```

Output

```
RALPR(0.25, 0.50) = 0.8825
```

TDF

This function evaluates the Student's t cumulative distribution function.

Function Return Value

TDF — Function value, the probability that a Student's t random variable takes a value less than or equal to the input **T**. (Output)

Required Arguments

T — Argument for which the Student's t distribution function is to be evaluated. (Input)

DF — Degrees of freedom. (Input)
DF must be greater than or equal to 1.0.

Optional Arguments

COMPLEMENT — Logical. If **.TRUE.**, the complement of the Student's t cumulative distribution function is evaluated. If **.FALSE.**, the Student's t cumulative distribution function is evaluated. (Input)
See the [Description](#) section for further details on the use of **COMPLEMENT**.
Default: **COMPLEMENT = .FALSE.**

FORTRAN 90 Interface

Generic: **TDF** (**T**, **DF** [, ...])
Specific: The specific interface names are **S_TDF** and **D_TDF**.

FORTRAN 77 Interface

Single: **TDF** (**T**, **DF**)
Double: The double precision name is **DTDF**.

Description

Function **TDF** evaluates the cumulative distribution function of a Student's t random variable with **DF** degrees of freedom. If the square of **T** is greater than or equal to **DF**, the relationship of a t to an F random variable (and subsequently, to a beta random variable) is exploited, and routine **BETDF** is used. Otherwise, the method described by Hill (1970) is used. Let $\mathbf{v} = \mathbf{DF}$. If \mathbf{v} is not an integer, if \mathbf{v} is greater than 19, or if \mathbf{v} is greater than 200, a Cornish-Fisher expansion is used to evaluate the distribution function. If \mathbf{v} is less than 20 and $\mathbf{ABS}(\mathbf{T})$ is less than 2.0, a trigonometric series (see Abramowitz and Stegun 1964, equations 26.7.3 and 26.7.4, with some rearrangement) is used. For the remaining cases, a series given by Hill (1970) that converges well for large values of \mathbf{T} is used.

If **COMPLEMENT** = **.TRUE.**, the value of **TDF** at the point x is $1 - p$, where $1 - p$ is the probability that the random variable takes a value greater than x . In those situations where the desired end result is $1 - p$, the user can achieve greater accuracy in the right tail region by using the result returned by **TDF** with the optional argument **COMPLEMENT** set to **.TRUE.** rather than by using $1 - p$ where p is the result returned by **TDF** with **COMPLEMENT** set to **.FALSE.**

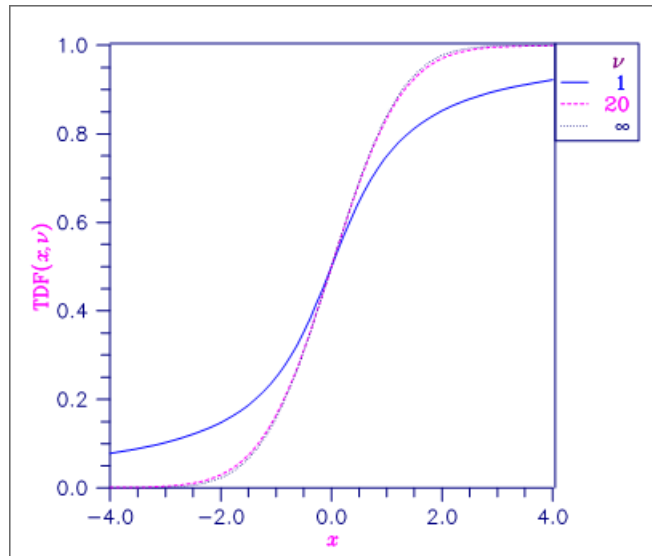


Figure 29, Student's t Distribution Function

Example

In this example, we find the probability that a t random variable with 6 degrees of freedom is greater in absolute value than 2.447. We use the fact that t is symmetric about 0.

```
USE TDF_INT
USE UMACH_INT
IMPLICIT NONE
INTEGER NOUT
REAL DF, P, T
!
CALL UMACH (2, NOUT)
T = 2.447
DF = 6.0
P = 2.0*TDF(-T,DF)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that a t(6) variate is greater ', &
             'than 2.447 in', /, ' absolute value is ', F6.4)
END
```

Output

```
The probability that a t(6) variate is greater than 2.447 in absolute value is 0.0500
```

TIN

This function evaluates the inverse of the Student's t cumulative distribution function.

Function Return Value

TIN — Function value. (Output)

The probability that a Student's t random variable takes a value less than or equal to **TIN** is **P**.

Required Arguments

P — Probability for which the inverse of the Student's t cumulative distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

DF — Degrees of freedom. (Input)

DF must be greater than or equal to 1.0.

FORTRAN 90 Interface

Generic: **TIN** (**P**, **DF**)

Specific: The specific interface names are **S_TIN** and **D_TIN**.

FORTRAN 77 Interface

Single: **TIN** (**P**, **DF**)

Double: The double precision name is **DTIN**.

Description

Function **TIN** evaluates the inverse distribution function of a Student's t random variable with **DF** degrees of freedom. Let $\nu = \text{DF}$. If ν equals 1 or 2, the inverse can be obtained in closed form, if ν is between 1 and 2, the relationship of a t to a beta random variable is exploited and routine **BETIN** is used to evaluate the inverse; otherwise the algorithm of Hill (1970) is used. For small values of ν greater than 2, Hill's algorithm inverts an integrated expansion in $1/(1 + t^2/\nu)$ of the t density. For larger values, an asymptotic inverse Cornish-Fisher type expansion about normal deviates is used.

Comments

Informational error

Type	Code	Description
4	3	TIN is set to machine infinity since overflow would occur upon modifying the inverse value for the F distribution with the result obtained from the inverse β distribution.

Example

In this example, we find the 0.05 critical value for a two-sided t test with 6 degrees of freedom.

```
USE TIN_INT
USE UMACH_INT
IMPLICIT NONE
INTEGER NOUT
REAL DF, P, T
!
CALL UMACH (2, NOUT)
P = 0.975
DF = 6.0
T = TIN(P,DF)
WRITE (NOUT,99999) T
99999 FORMAT (' The two-sided t(6) 0.05 critical value is ', F6.3)
END
```

Output

```
The two-sided t(6) 0.05 critical value is 2.447
```


TPR

This function evaluates the Student's t probability density function.

Function Return Value

TPR — Function value, the value of the probability density function. (Output)

Required Arguments

T — Argument for which the Student's t probability density function is to be evaluated. (Input)

DF — Degrees of freedom. (Input)

DF must be greater than or equal to 1.0.

FORTRAN 90 Interface

Generic: TPR (T, DF)

Specific: The specific interface names are S_TPR and D_TPR

FORTRAN 77 Interface

Single: TPR (T, DF)

Double: The double precision name is DTPR.

Description

The function **TPR** evaluates the Student's t probability density function, defined as

$$f(t | \nu) = (\beta(0.5, 0.5\nu) \sqrt{\nu})^{-1} \left(1 + \frac{t^2}{\nu}\right)^{-(\nu+1)/2}, \quad -\infty < t < +\infty, \nu \geq 1$$

Where $\nu = \mathbf{DF}$.

The normalizing factor uses the Beta function, **BETA** (see *Chapter 4, "Gamma Functions and Related Functions"*).

Example

In this example, we evaluate the probability function at $T = 1.5$, $DF = 10.0$.

```
USE UMACH_INT
USE TPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL T, DF, PR
CALL UMACH(2, NOUT)
T = 1.5
DF = 10.0
PR = TPR(T, DF)
WRITE (NOUT, 99999) T, DF, PR
99999 FORMAT (' TPR(' , F4.2, ' , ' , F6.2, ' ) = ' , F6.4)
END
```

Output

```
TPR(1.50, 10.00) = 0.1274
```

TPDF

This function evaluates the noncentral Student's t cumulative distribution function.

Function Return Value

TPDF — Function value, the probability that a noncentral Student's t random variable takes a value less than or equal to T . (Output)

Required Arguments

T — Argument for which the noncentral Student's t cumulative distribution function is to be evaluated. (Input)

IDF — Number of degrees of freedom of the noncentral Student's t cumulative distribution. (Input)
IDF must be positive.

DELTA — The noncentrality parameter. (Input)

FORTRAN 90 Interface

Generic: **TPDF** (**T**, **IDF**, **DELTA**)

Specific: The specific interface names are **S_TPDF** and **D_TPDF**.

FORTRAN 77 Interface

Single: **TPDF** (**T**, **IDF**, **DELTA**)

Double: The double precision name is **DTPDF**.

Description

Function **TPDF** evaluates the cumulative distribution function F of a noncentral t random variable with **IDF** degrees of freedom and noncentrality parameter **DELTA**; that is, with $\nu = \text{IDF}$, $\delta = \text{DELTA}$, and $t_0 = T$,

$$F(t_0) = \int_{-\infty}^{t_0} \frac{v^{v/2} e^{-\delta^2/2}}{\sqrt{\pi} \Gamma(v/2) (v+x^2)^{(v+1)/2}} \sum_{i=0}^{\infty} \Gamma((v+i+1)/2) \left(\frac{\delta^i}{i!}\right) \left(\frac{2x^2}{v+x^2}\right)^{i/2} dx$$

where $\Gamma(\cdot)$ is the gamma function. The value of the distribution function at the point t_0 is the probability that the random variable takes a value less than or equal to t_0 .

The noncentral t random variable can be defined by the distribution function above, or alternatively and equivalently, as the ratio of a normal random variable and an independent chi-squared random variable. If w has a normal distribution with mean δ and variance equal to one, u has an independent chi-squared distribution with v degrees of freedom, and

$$x = w / \sqrt{u/v}$$

then x has a noncentral t distribution with degrees of freedom and noncentrality parameter δ .

The distribution function of the noncentral t can also be expressed as a double integral involving a normal density function (see, for example, Owen 1962, page 108). The function **TPDF** uses the method of Owen (1962, 1965), which uses repeated integration by parts on that alternate expression for the distribution function.

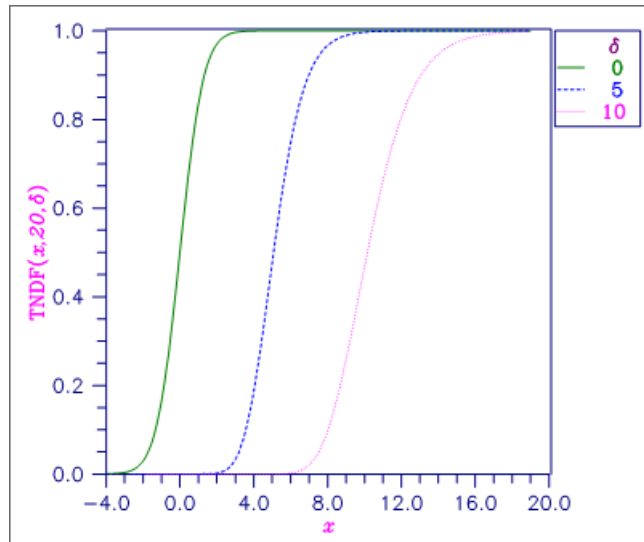


Figure 30, Noncentral Student's t Distribution Function

Comments

Informational error

Type	Code	Description
4	2	An accurate result cannot be computed due to possible underflow for the machine precision available. $\text{DELTA} * \text{SQRT}(\text{IDF} / (\text{IDF} + \text{T}^2))$ must be less than $\text{SQRT}(-1.9 * \text{ALOG}(S))$, where $S = \text{AMACH}(1)$.

Example

Suppose T is a noncentral t random variable with 6 degrees of freedom and noncentrality parameter 6. In this example, we find the probability that T is less than 12.0. (This can be checked using the table on page 111 of Owen 1962, with $\eta = 0.866$, which yields $\lambda = 1.664$.)

```

USE UMACH_INT
USE TPDF_INT
IMPLICIT NONE
INTEGER    IDF, NOUT
REAL      DELTA, P, T
!
CALL UMACH (2, NOUT)
IDF      = 6
DELTA    = 6.0
T        = 12.0
P        = TPDF(T, IDF, DELTA)
WRITE (NOUT, 99999) P
99999 FORMAT (' The probability that T is less than 12.0 is ', F6.4)
END

```

Output

```
The probability that T is less than 12.0 is 0.9501
```

TNIN

This function evaluates the inverse of the noncentral Student's t cumulative distribution function.

Function Return Value

TNIN — Function value. (Output)

The probability that a noncentral Student's t random variable takes a value less than or equal to **TNIN** is **P**.

Required Arguments

P — Probability for which the inverse of the noncentral Student's t cumulative distribution function is to be evaluated. (Input)

P must be in the open interval (0.0, 1.0).

IDF — Number of degrees of freedom of the noncentral Student's t cumulative distribution. (Input) **IDF** must be positive.

DELTA — The noncentrality parameter. (Input)

FORTRAN 90 Interface

Generic: **TNIN** (**P**, **IDF**, **DELTA**)

Specific: The specific interface names are **S_TNIN** and **D_TNIN**.

FORTRAN 77 Interface

Single: **TNIN** (**P**, **IDF**, **DELTA**)

Double: The double precision name is **DTNIN**.

Description

Function **TNIN** evaluates the inverse distribution function of a noncentral t random variable with **IDF** degrees of freedom and noncentrality parameter **DELTA**; that is, with $P = \mathbf{P}$, $\nu = \mathbf{IDF}$, and $\delta = \mathbf{DELTA}$, it determines t_0 ($= \mathbf{TNIN}(\mathbf{P}, \mathbf{IDF}, \mathbf{DELTA})$), such that

$$P = \int_{-\infty}^{t_0} \frac{v^{v/2} e^{-\delta^2/2}}{\sqrt{\pi} \Gamma(v/2) (v+x^2)^{(v+1)/2}} \sum_{i=0}^{\infty} \Gamma((v+i+1)/2) \left(\frac{\delta^i}{i!}\right) \left(\frac{2x^2}{v+x^2}\right)^{i/2} dx$$

where $\Gamma(\cdot)$ is the gamma function. The probability that the random variable takes a value less than or equal to t_0 is P . See [TNDF](#) for an alternative definition in terms of normal and chi-squared random variables. The function `TNIN` uses bisection and modified regula falsi to invert the distribution function, which is evaluated using routine `TNDF`.

Comments

Informational error

Type	Code	Description
4	1	Over 100 iterations have occurred without convergence. Convergence is assumed.

Example

In this example, we find the 95-th percentage point for a noncentral t random variable with 6 degrees of freedom and noncentrality parameter 6.

```

USE TNIN_INT
USE UMACH_INT
IMPLICIT NONE
INTEGER IDF, NOUT
REAL DELTA, P, T
!
CALL UMACH (2, NOUT)
IDF = 6
DELTA = 6.0
P = 0.95
T = TNIN(P, IDF, DELTA)
WRITE (NOUT, 99999) T
!
99999 FORMAT (' The 0.05 noncentral t critical value is ', F6.3, &
            ' .')
!
END

```

Output

```
The 0.05 noncentral t critical value is 11.995.
```

TNPR

This function evaluates the noncentral Student's t probability density function.

Function Return Value

TNPR — Function value, the value of the probability density function. (Output)

Required Arguments

T — Argument for which the noncentral Student's t probability density function is to be evaluated. (Input)

DF — Number of degrees of freedom of the noncentral Student's t distribution. (Input)
DF must be positive.

DELTA — Noncentrality parameter. (Input)

FORTRAN 90 Interface

Generic: TNPR (T, DF, DELTA)

Specific: The specific interface names are S_TNPR and D_TNPR.

Description

The noncentral Student's t distribution is a generalization of the Student's t distribution.

If w is a *normally distributed* random variable with unit variance and mean δ and u is a chi-square random variable with ν degrees of freedom that is statistically independent of w , then

$$T = w / \sqrt{u/\nu}$$

is a noncentral t -distributed random variable with ν degrees of freedom and noncentrality parameter δ , that is, with $\nu = \text{DF}$, and $\delta = \text{DELTA}$. The probability density function for the noncentral t -distribution is:

$$f(t, \nu, \delta) = \frac{\nu^{\nu/2} e^{-\delta^2/2}}{\sqrt{\pi} \Gamma(\nu/2) (\nu + t^2)^{(\nu+1)/2}} \sum_{i=0}^{\infty} \Phi_i$$

where

$$\Phi_i = \frac{\Gamma((\nu + i + 1)/2) [\delta t]^i (2 / (\nu + t^2))^{i/2}}{i!}$$

and $t = T$.

For $\delta = 0$, the PDF reduces to the (central) Student's t PDF:

$$f(t, \nu, 0) = \frac{\Gamma((\nu + 1)/2) (1 + (t^2/\nu))^{-(\nu+1)/2}}{\sqrt{\nu\pi} \Gamma(\nu/2)}$$

and, for $t = 0$, the PDF becomes:

$$f(0, \nu, \delta) = \frac{\Gamma((\nu + 1)/2) e^{-\delta^2/2}}{\sqrt{\nu\pi} \Gamma(\nu/2)}$$

Example

This example calculates the noncentral Student's t PDF for a distribution with 2 degrees of freedom and noncentrality parameter $\delta = 10$.

```

USE TNPR_INT
USE UMACH_INT
IMPLICIT NONE

INTEGER :: NOUT, I
REAL    :: X(6)=(/ -0.5, 1.5, 3.5, 7.5, 51.5, 99.5 /)
REAL    :: DF, DELTA, PDFV

CALL UMACH (2, NOUT)
DF = 2.0
DELTA = 10.0

WRITE (NOUT, '( "DF: ", F4.0, " DELTA: ", F4.0 // ' // &
' " X PDF(X) ") ') DF, DELTA

DO I = 1, 6
    PDFV = TNPR(X(I), DF, DELTA)
    WRITE (NOUT, '(1X, F4.1, 2X, E12.5)') X(I), PDFV
END DO
END

```

Output

```

DF:  2.  DELTA: 10.

  X      PDF(X)
-0.5  0.16399E-23
 1.5  0.74417E-09
 3.5  0.28972E-02
 7.5  0.78853E-01
51.5  0.14215E-02
99.5  0.20290E-03

```

UNDF

This function evaluates the uniform cumulative distribution function.

Function Return Value

UNDF — Function value, the probability that a uniform random variable takes a value less than or equal to X . (Output)

Required Arguments

X — Argument for which the uniform cumulative distribution function is to be evaluated. (Input)

A — Location parameter of the uniform cumulative distribution function. (Input)

B — Value used to compute the scale parameter ($B - A$) of the uniform cumulative distribution function. (Input)

FORTRAN 90 Interface

Generic: `UNDF (X, A, B)`

Specific: The specific interface names are `S_UNDF` and `D_UNDF`.

FORTRAN 77 Interface

Single: `UNDF (X, A, B)`

Double: The double precision name is `DUNDF`.

Description

The function **UNDF** evaluates the uniform cumulative distribution function with location parameter **A** and scale parameter (**B - A**). The function definition is

$$F(x|A,B) = \begin{cases} 0, & \text{if } x < A \\ \frac{x-A}{B-A}, & \text{if } A \leq x \leq B \\ 1, & \text{if } x > B \end{cases}$$

Example

In this example, we evaluate the probability function at $x = 0.65$, $A = 0.25$, $B = 0.75$.

```
USE UMACH_INT
USE UNDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, PR
CALL UMACH(2, NOUT)
X = 0.65
A = 0.25
B = 0.75
PR = UNDF(X, A, B)
WRITE (NOUT, 99999) X, A, B, PR
99999 FORMAT (' UNDF(', F4.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
UNDF(0.65, 0.25, 0.75) = 0.8000
```

UNIN

This function evaluates the inverse of the uniform cumulative distribution function.

Function Return Value

UNIN — Function value, the value of the inverse of the cumulative distribution function. (Output)

Required Arguments

- P** — Probability for which the inverse of the uniform cumulative distribution function is to be evaluated. (Input)
- A** — Location parameter of the uniform cumulative distribution function. (Input)
- B** — Value used to compute the scale parameter ($B - A$) of the uniform cumulative distribution function. (Input)

FORTRAN 90 Interface

- Generic: `UNIN (P, A, B)`
- Specific: The specific interface names are `S_UNIN` and `D_UNIN`.

FORTRAN 77 Interface

- Single: `UNIN (P, A, B)`
- Double: The double precision name is `DUNIN`.

Description

The function **UNIN** evaluates the inverse distribution function of a uniform random variable with location parameter **A** and scale parameter ($B - A$).

Example

In this example, we evaluate the inverse probability function at $P = 0.80$, $A = 0.25$, $B = 0.75$.

```
USE UMACH_INT
USE UNIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, P
CALL UMACH(2, NOUT)
P = 0.80
A = 0.25
B = 0.75
X = UNIN(P, A, B)
WRITE (NOUT, 99999) P, A, B, X
99999 FORMAT (' UNIN(', F4.2, ', ', ' ', F4.2, ', ', ' ', F4.2, ') = ', F6.4)
END
```

Output

```
UNIN(0.80, 0.25, 0.75) = 0.6500
```

UNPR

This function evaluates the uniform probability density function.

Function Return Value

UNPR — Function value, the value of the probability density function. (Output)

Required Arguments

- X** — Argument for which the uniform probability density function is to be evaluated. (Input)
- A** — Location parameter of the uniform probability function. (Input)
- B** — Value used to compute the scale parameter ($B - A$) of the uniform probability density function. (Input)

FORTRAN 90 Interface

- Generic: **UNPR (X, A, B)**
- Specific: The specific interface names are **S_UNPR** and **D_UNPR**.

FORTRAN 77 Interface

- Single: **UNPR (X, A, B)**
- Double: The double precision name is **DUNPR**.

Description

The function **UNPR** evaluates the uniform probability density function with location parameter **A** and scale parameter ($B - A$), defined

$$f(x|A,B) = \begin{cases} \frac{1}{B-A} & \text{for } A \leq x \leq B \\ 0 & \text{otherwise} \end{cases}$$

Example

In this example, we evaluate the uniform probability density function at $x = 0.65$, $A = 0.25$, $B = 0.75$.

```
USE UMACH_INT
USE UNPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, PR
CALL UMACH(2, NOUT)
X = 0.65
A = 0.25
B = 0.75
PR = UNPR(X, A, B)
WRITE (NOUT, 99999) X, A, B, PR
99999 FORMAT (' UNPR(', F4.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
UNPR(0.65, 0.25, 0.75) = 2.0000
```

WBLDF

This function evaluates the Weibull cumulative distribution function.

Function Return Value

WBLDF — Function value, the probability that a Weibull random variable takes a value less than or equal to x . (Output)

Required Arguments

X — Argument for which the Weibull cumulative distribution function is to be evaluated. (Input)

A — Scale parameter. (Input)

B — Shape parameter. (Input)

FORTRAN 90 Interface

Generic: **WBLDF** (**X**, **A**, **B**)

Specific: The specific interface names are **S_WBLDF** and **D_WBLDF**.

FORTRAN 77 Interface

Single: **WBLDF** (**X**, **A**, **B**)

Double: The double precision name is **DWBLDF**.

Description

The function **WBLDF** evaluates the Weibull cumulative distribution function with scale parameter **A** and shape parameter **B**, defined

$$F(x|a,b) = 1 - e^{-\left(\frac{x}{a}\right)^b}$$

To deal with potential loss of precision for small values of $\left(\frac{x}{a}\right)^b$, the difference expression for p is re-written as

$$u = \left(\frac{x}{a}\right)^b, \quad p = u \left[\frac{(e^{-u} - 1)}{-u} \right]$$

and the right factor is accurately evaluated using `EXPRL`.

Example

In this example, we evaluate the Weibull cumulative distribution function at $X = 1.5$, $A = 1.0$, $B = 2.0$.

```
USE UMACH_INT
USE WBLDF_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, PR
CALL UMACH(2, NOUT)
X = 1.5
A = 1.0
B = 2.0
PR = WBLDF(X, A, B)
WRITE (NOUT, 99999) X, A, B, PR
99999 FORMAT (' WBLDF(', F4.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
WBLDF(1.50, 1.00, 2.00) = 0.8946
```

WBLIN

This function evaluates the inverse of the Weibull cumulative distribution function.

Function Return Value

WBLIN — Function value, the value of the inverse of the Weibull cumulative distribution function.
(Output)

Required Arguments

P — Probability for which the inverse of the Weibull cumulative distribution function is to be evaluated.
(Input)

A — Scale parameter. (Input)

B — Shape parameter. (Input)

FORTRAN 90 Interface

Generic: `WBLIN (P, A, B)`

Specific: The specific interface names are `S_WBLIN` and `D_WBLIN`.

FORTRAN 77 Interface

Single: `WBLIN (P, A, B)`

Double: The double precision name is `DWBLIN`.

Description

The function `WBLIN` evaluates the inverse distribution function of a Weibull random variable with scale parameter `A` and shape parameter `B`.

Example

In this example, we evaluate the inverse probability function at `P = 0.8946`, `A = 1.0`, `B = 2.0`.

```
USE UMACH_INT
USE WBLIN_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, P
CALL UMACH(2, NOUT)
P = 0.8946
A = 1.0
B = 2.0
X = WBLIN(P, A, B)
WRITE (NOUT, 99999) P, A, B, X
99999 FORMAT (' WBLIN(', F4.2, ', ', F4.2, ', ', F4.2, ') = ', F6.4)
END
```

Output

```
WBLIN(0.8946, 1.00, 2.00) = 1.5000
```

WBLPR

This function evaluates the Weibull probability density function.

Function Return Value

WBLPR — Function value, the value of the probability density function. (Output)

Required Arguments

X — Argument for which the Weibull probability density function is to be evaluated. (Input)

A — Scale parameter. (Input)

B — Shape parameter. (Input)

FORTRAN 90 Interface

Generic: **WBLPR (X, A, B)**

Specific: The specific interface names are **S_WBLPR** and **D_WBLPR**.

FORTRAN 77 Interface

Single: **WBLPR (X, A, B)**

Double: The double precision name is **DWBLPR**.

Description

The function **WBLPR** evaluates the Weibull probability density function with scale parameter **A** and shape parameter **B**, defined

$$f(x|a,b) = \frac{b}{a} \left(\frac{x}{a}\right)^{b-1} e^{-\left(\frac{x}{a}\right)^b}, \quad a, b > 0$$

Example

In this example, we evaluate the Weibull probability density function at **X** = 1.5, **A** = 1.0, **B** = 2.0.

```
USE UMACH_INT
USE WBLPR_INT
IMPLICIT NONE
INTEGER NOUT
REAL X, A, B, PR`
CALL UMACH(2, NOUT)
X = 1.5
A = 1.0
B = 2.0
PR = WBLPR(X, A, B)
WRITE (NOUT, 99999) X, A, B, PR
99999 FORMAT (' WBLPR(', F4.2, ', ', ', F4.2, ', ', ', F4.2, ') = ', F6.4)
END
```

Output

```
WBLPR(1.50, 1.00, 2.00) = 0.3162
```

GPDF

This function evaluates a general continuous cumulative distribution function given ordinates of the density.

Function Return Value

GPDF — Function value, the probability that a random variable whose density is given in **F** takes a value less than or equal to **X0**. (Output)

Required Arguments

X0 — Point at which the cumulative distribution function is to be evaluated. (Input)

X — Array containing the abscissas or the endpoints. (Input)

If **IOPT** = 1 or 3, **X** is of length 2. If **IOPT** = 2 or 4, **X** is of length **M**. For **IOPT** = 1 or 3, **X(1)** contains the lower endpoint of the support of the distribution and **X(2)** is the upper endpoint. For **IOPT** = 2 or 4, **X** contains, in strictly increasing order, the abscissas such that **X(I)** corresponds to **F(I)**.

F — Vector of length **M** containing the probability density ordinates corresponding to increasing abscissas. (Input)

If **IOPT** = 1 or 3, for **I** = 1, 2, ..., **M**, **F(I)** corresponds to $X(1) + (I - 1) * (X(2) - X(1)) / (M - 1)$; otherwise, **F** and **X** correspond one for one.

Optional Arguments

IOPT — Indicator of the method of interpolation. (Input)

Default: **IOPT** = 1.

IOPT Interpolation Method

- 1 Linear interpolation with equally spaced abscissas.
- 2 Linear interpolation with possibly unequally spaced abscissas.
- 3 A cubic spline is fitted to equally spaced abscissas.
- 4 A cubic spline is fitted to possibly unequally spaced abscissas.

M—Number of ordinates of the density supplied. (Input)

M must be greater than 1 for linear interpolation (**IOPT** = 1 or 2) and greater than 3 if a curve is fitted through the ordinates (**IOPT** = 3 or 4).

Default: **M** = size (**F**,1).

FORTRAN 90 Interface

Generic: **GCDF** (**X0**, **X**, **F** [, ...])

Specific: The specific interface names are **S_GCDF** and **D_GCDF**.

FORTRAN 77 Interface

Single: **GCDF** (**X0**, **IOPT**, **M**, **X**, **F**)

Double: The double precision name is **DGCDF**.

Description

Function **GCDF** evaluates a continuous distribution function, given ordinates of the probability density function. It requires that the range of the distribution be specified in **X**. For distributions with infinite ranges, endpoints must be chosen so that most of the probability content is included. The function **GCDF** first fits a curve to the points given in **X** and **F** with either a piecewise linear interpolant or a C^1 cubic spline interpolant based on a method by Akima (1970). Function **GCDF** then determines the area, *A*, under the curve. (If the distribution were of finite range and if the fit were exact, this area would be 1.0.) Using the same fitted curve, **GCDF** next determines the area up to the point x_0 (= **X0**). The value returned is the area up to x_0 divided by *A*. Because of the scaling by *A*, it is not assumed that the integral of the density defined by **X** and **F** is 1.0. For most distributions, it is likely that better approximations to the distribution function are obtained when **IOPT** equals 3 or 4, that is, when a cubic spline is used to approximate the function. It is also likely that better approximations can be obtained when the abscissas are chosen more densely over regions where the density and its derivatives (when they exist) are varying greatly.

Comments

1. If **IOPT** = 3, automatic workspace usage is:

GCDF 6 * **M** units, or

DGCDF 11 * **M** units.

2. If **IOPT** = 4, automatic workspace usage is

GCDF 5 * **M** units, or

DGCDF 9 * M units.

3. Workspace may be explicitly provided, if desired, by the use of **G4DF/DG4DF**. The reference is:

G4DF(P, IOPT, M, X, F, WK, IWK)

The arguments in addition to those of **GCDF** are:

WK — Work vector of length 5 * M if IOPT = 3, and of length 4 * M if IOPT = 4.

IWK — Work vector of length M.

Example

In this example, we evaluate the beta distribution function at the point 0.6. The probability density function of a beta random variable with parameters p and q is

$$f(x) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} x^{p-1} (1-x)^{q-1} \quad \text{for } 0 \leq x \leq 1$$

where $\Gamma(\cdot)$ is the gamma function. The density is equal to 0 outside the interval [0, 1]. We compute a constant multiple (we can ignore the constant gamma functions) of the density at 300 equally spaced points and input this information in **X** and **F**. Knowing that the probability density of this distribution is very peaked in the vicinity of 0.5, we could perhaps get a better fit by using unequally spaced abscissas, but we will keep it simple. Note that this is the same example as one used in the description of routine **BETDF**. The result from **BETDF** would be expected to be more accurate than that from **GCDF** since **BETDF** is designed specifically for this distribution.

```

      USE UMACH_INT
      USE GCDF_INT

      IMPLICIT NONE
      INTEGER M
      PARAMETER (M=300)

!
      INTEGER I, IOPT, NOUT
      REAL F(M), H, P, PIN1, QIN1, X(2), X0, XI

!
      CALL UMACH (2, NOUT)
      X0 = 0.6
      IOPT = 3

!
!                                     Initializations for a beta(12,12)
!                                     distribution.
      PIN1 = 11.0
      QIN1 = 11.0
      XI = 0.0
      H = 1.0/(M-1.0)
      X(1) = XI
      F(1) = 0.0
      XI = XI + H

!
!                                     Compute ordinates of the probability
!                                     density function.
      DO 10 I=2, M - 1
         F(I) = XI**PIN1*(1.0-XI)**QIN1
         XI = XI + H
10 CONTINUE

```



```
X(2) = 1.0
F(M) = 0.0
P      = GCDF(X0, X, F, IOPT=IOPT)
WRITE (NOUT,99999) P
99999 FORMAT (' The probability that X is less than 0.6 is ', F6.4)
END
```

Output

```
The probability that X is less than 0.6 is 0.8364
```

GCIN

Evaluates the inverse of a general continuous cumulative distribution function given ordinates of the density.

Required Arguments

- P** — Probability for which the inverse of the distribution function is to be evaluated. (Input)
P must be in the open interval (0.0, 1.0).
- X** — Array containing the abscissas or the endpoints. (Input)
 If **IOPT** = 1 or 3, **X** is of length 2. If **IOPT** = 2 or 4, **X** is of length **M**. For **IOPT** = 1 or 3, **X**(1) contains the lower endpoint of the support of the distribution and **X**(2) is the upper endpoint. For **IOPT** = 2 or 4, **X** contains, in strictly increasing order, the abscissas such that **X**(**I**) corresponds to **F**(**I**).
- F** — Vector of length **M** containing the probability density ordinates corresponding to increasing abscissas. (Input)
 If **IOPT** = 1 or 3, for **I** = 1, 2, ..., **M**, **F**(**I**) corresponds to $\mathbf{X}(1) + (\mathbf{I} - 1) * (\mathbf{X}(2) - \mathbf{X}(1)) / (\mathbf{M} - 1)$; otherwise, **F** and **X** correspond one for one.
- GCIN** — Function value. (Output)
 The probability that a random variable whose density is given in **F** takes a value less than or equal to **GCIN** is **P**.

Optional Arguments

- IOPT** — Indicator of the method of interpolation. (Input)
 Default: **IOPT** = 1.

IOPT	Interpolation Method
1	Linear interpolation with equally spaced abscissas.
2	Linear interpolation with possibly unequally spaced abscissas.
3	A cubic spline is fitted to equally spaced abscissas.
4	A cubic spline is fitted to possibly unequally spaced abscissas.

M — Number of ordinates of the density supplied. (Input)

M must be greater than 1 for linear interpolation (**IOPT** = 1 or 2) and greater than 3 if a curve is fitted through the ordinates (**IOPT** = 3 or 4).

Default: **M** = size (**F**,1).

FORTRAN 90 Interface

Generic: `CALL GCIN (P, X, F [, ...])`

Specific: The specific interface names are `S_GCIN` and `D_GCIN`.

FORTRAN 77 Interface

Single: `CALL GCIN (P, IOPT, M, X, F)`

Double: The double precision function name is `DGCIN`.

Description

Function `GCIN` evaluates the inverse of a continuous distribution function, given ordinates of the probability density function. The range of the distribution must be specified in **X**. For distributions with infinite ranges, endpoints must be chosen so that most of the probability content is included.

The function `GCIN` first fits a curve to the points given in **X** and **F** with either a piecewise linear interpolant or a C^1 cubic spline interpolant based on a method by Akima (1970). Function `GCIN` then determines the area, *A*, under the curve. (If the distribution were of finite range and if the fit were exact, this area would be 1.0.) It next finds the maximum abscissa up to which the area is less than *AP* and the minimum abscissa up to which the area is greater than *AP*. The routine then interpolates for the point corresponding to *AP*. Because of the scaling by *A*, it is not assumed that the integral of the density defined by **X** and **F** is 1.0.

For most distributions, it is likely that better approximations to the distribution function are obtained when **IOPT** equals 3 or 4, that is, when a cubic spline is used to approximate the function. It is also likely that better approximations can be obtained when the abscissas are chosen more densely over regions where the density and its derivatives (when they exist) are varying greatly.

Comments

Workspace may be explicitly provided, if desired, by the use of `G3IN/DG3IN`. The reference is

`G3IN (P, IOPT, M, X, F, WK, IWK)`

The arguments in addition to those of `GCIN` are:

WK — Work vector of length $5 * M$ if **IOPT** = 3, and of length $4 * M$ if **IOPT** = 4.

IWK — Work vector of length M.

Example

In this example, we find the 90-th percentage point for a beta random variable with parameters 12 and 12. The probability density function of a beta random variable with parameters p and q is

$$f(x) = \frac{\Gamma(p+q)}{\Gamma(p)\Gamma(q)} x^{p-1} (1-x)^{q-1} \quad \text{for } 0 \leq x \leq 1$$

where $\Gamma(\cdot)$ is the gamma function. The density is equal to 0 outside the interval $[0, 1]$. With $p = q$, this is a symmetric distribution. Knowing that the probability density of this distribution is very peaked in the vicinity of 0.5, we could perhaps get a better fit by using unequally spaced abscissas, but we will keep it simple and use 300 equally spaced points. Note that this is the same example that is used in the description of routine **BETIN**. The result from **BETIN** would be expected to be more accurate than that from **GCIN** since **BETIN** is designed specifically for this distribution.

```

USE GCIN_INT
USE UMACH_INT
USE BETA_INT

IMPLICIT NONE
INTEGER M
PARAMETER (M=300)

!
INTEGER I, IOPT, NOUT
REAL C, F(M), H, P, PIN, PIN1, QIN, QIN1, &
X(2), X0, XI
!
CALL UMACH (2, NOUT)
P = 0.9
IOPT = 3

!
! Initializations for a beta(12,12)
! distribution.
PIN = 12.0
QIN = 12.0
PIN1 = PIN - 1.0
QIN1 = QIN - 1.0
C = 1.0/BETA(PIN,QIN)
XI = 0.0
H = 1.0/(M-1.0)
X(1) = XI
F(1) = 0.0
XI = XI + H

!
! Compute ordinates of the probability
! density function.
DO 10 I=2, M - 1
F(I) = C*XI**PIN1*(1.0-XI)**QIN1
XI = XI + H
10 CONTINUE
X(2) = 1.0
F(M) = 0.0
X0 = GCIN(P,X,F, IOPT=IOPT)
WRITE (NOUT,99999) X0
99999 FORMAT (' X is less than ', F6.4, ' with probability 0.9.')
```

END

Output

X is less than 0.6304 with probability 0.9.

GFNIN

This function evaluates the inverse of a general continuous cumulative distribution function given in a subprogram.

Function Return Value

GFNIN — The inverse of the function **F** at the point **P**. (Output)
 $F(\text{GFNIN})$ is “close” to **P**.

Required Arguments

F — User-supplied **FUNCTION** to be inverted. **F** must be continuous and strictly monotone. The form is $F(X)$, where

X — The argument to the function. (Input)

F — The value of the function at **X**. (Output)

F must be declared **EXTERNAL** in the calling program.

P — The point at which the inverse of **F** is desired. (Input)

GUESS — An initial estimate of the inverse of **F** at **P**. (Input)

Optional Arguments

EPS — Convergence criterion. (Input)

When the relative change in **GFNIN** from one iteration to the next is less than **EPS**, convergence is assumed. A common value for **EPS** is 0.0001. Another common value is 100 times the machine epsilon.

Default: **EPS** = 100 times the machine epsilon.

FORTRAN 90 Interface

Generic: **GFNIN** (**F**, **P**, **GUESS** [, ...])

Specific: The specific interface names are **S_GFNIN** and **D_GFNIN**.

FORTRAN 77 Interface

Single: **GFNIN** (**F**, **P**, **EPS**, **GUESS**)
 Double: The double precision name is **DGFNIN**.

Description

Function **GFNIN** evaluates the inverse of a continuous, strictly monotone function. Its most obvious use is in evaluating inverses of continuous distribution functions that can be defined by a FORTRAN function. If the distribution function cannot be specified in a FORTRAN function, but the density function can be evaluated at a number of points, then routine **GCIN** can be used.

Function **GFNIN** uses regula falsi and/or bisection, possibly with the Illinois modification (see Dahlquist and Bjorck 1974). A maximum of 100 iterations are performed.

Comments

1. Informational errors

Type	Code	Description
4	1	After 100 attempts, a bound for the inverse cannot be determined. Try again with a different initial estimate.
4	2	No unique inverse exists.
4	3	Over 100 iterations have occurred without convergence. Convergence is assumed.

2. The function to be inverted need not be a distribution function, it can be any continuous, monotonic function.

Example

In this example, we find the 99–th percentage point for an F random variable with 1 and 7 degrees of freedom. (This problem could be solved easily using routine **FIN**. Compare the example for **FIN**). The function to be inverted is the F distribution function, for which we use routine **FDF**. Since **FDF** requires the degrees of freedom in addition to the point at which the function is evaluated, we write another function **F** that receives the degrees of freedom via a common block and then calls **FDF**. The starting point (initial guess) is taken as two standard deviations above the mean (since this would be a good guess for a normal distribution). It is not necessary to supply such a good guess. In this particular case, an initial estimate of 1.0, for example, yields the same answer in essentially the same number of iterations. (In fact, since the F distribution is skewed, the initial guess, 7.0, is really not that close to the final answer.)

```

      USE UMACH_INT
      USE GFNIN_INT
      IMPLICIT NONE
      INTEGER NOUT
      REAL DFD, DFN, F, F0, GUESS, P, SQRT
      COMMON /FCOM/ DFN, DFD
      INTRINSIC SQRT
      EXTERNAL F
!
      CALL UMACH (2, NOUT)
      P = 0.99
      DFN = 1.0
      DFD = 7.0
!
!           Compute GUESS as two standard
!           deviations above the mean.
      GUESS = DFD/(DFD-2.0) + 2.0*SQRT(2.0*DFD*DFD*(DFN+DFD-2.0)/(DFN* &
      (DFD-2.0)**2*(DFD-4.0)))
      F0 = GFNIN(F,P,GUESS)
      WRITE (NOUT,99999) F0
99999 FORMAT (' The F(1,7) 0.01 critical value is ', F6.3)
      END
!
      REAL FUNCTION F (X)
      REAL X
!
      REAL DFD, DFN, FDF
      COMMON /FCOM/ DFN, DFD
      EXTERNAL FDF
!
      F = FDF(X,DFN,DFD)
      RETURN
      END

```

Output

```
The F(1,7) 0.01 critical value is 12.246
```


Mathieu Functions

Routines

Evaluate the eigenvalues for the periodic Mathieu functions	MATEE	456
Evaluate even, periodic Mathieu functions	MATCE	459
Evaluate odd, periodic Mathieu functions	MATSE	463

Usage Notes

Mathieu's equation is

$$\frac{d^2y}{dv^2} + (a - 2q \cos 2v)y = 0$$

It arises from the solution, by separation of variables, of Laplace's equation in elliptical coordinates, where a is the separation constant and q is related to the ellipticity of the coordinate system. If we let $t = \cos v$, then Mathieu's equation can be written as

$$(1 - t^2) \frac{d^2y}{dt^2} - t \frac{dy}{dt} + (a + 2q - 4qt^2)y = 0$$

For various physically important problems, the solution $y(t)$ must be periodic. There exist, for particular values of a , periodic solutions to Mathieu's equation of period $k\pi$ for any integer k . These particular values of a are called *eigenvalues* or *characteristic values*. They are computed using the routine [MATEE](#).

There exist sequences of both even and odd periodic solutions to Mathieu's equation. The even solutions are computed by [MATCE](#). The odd solutions are computed by [MATSE](#).

MATEE

Evaluates the eigenvalues for the periodic Mathieu functions.

Required Arguments

Q — Parameter. (Input)

ISYM — Symmetry indicator. (Input)

ISYM	Meaning
0	Even
1	Odd

IPER — Periodicity indicator. (Input)

ISYM	Meaning
0	pi
1	2 * pi

EVAL — Vector of length **N** containing the eigenvalues. (Output)

Optional Arguments

N — Number of eigenvalues to be computed. (Input)

Default: **N** = size (**EVAL**,1)

FORTRAN 90 Interface

Generic: `CALL MATEE (Q, ISYM, IPER, EVAL [, ...])`

Specific: The specific interface names are `S_MATEE` and `D_MATEE`.

FORTRAN 77 Interface

Single: `CALL MATEE (Q, N, ISYM, IPER, EVAL)`

Double: The double precision function name is `DMATEE`.

Description

The eigenvalues of Mathieu's equation are computed by a method due to Hodge (1972). The desired eigenvalues are the same as the eigenvalues of the following symmetric, tridiagonal matrix:

$$\begin{bmatrix} W_0 & qX_0 & 0 & 0 & \dots \\ qX_0 & W_2 & qX_2 & 0 & \dots \\ 0 & qX_2 & W_4 & qX_4 & \dots \\ 0 & 0 & qX_4 & W_6 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Here,

$$X_m = \begin{cases} \sqrt{2} & \text{if } \text{ISYM} = \text{IPER} = m = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$W_m = [m + \text{IPER} + 2(1 - \text{IPER})\text{ISYM}]^2 + V_m$$

where

$$V_m = \begin{cases} +q & \text{if } \text{IPER} = 1, \text{ISYM} = 0 \text{ and } m = 0 \\ -q & \text{if } \text{IPER} = 1, \text{ISYM} = 1 \text{ and } m = 0 \\ 0 & \text{otherwise} \end{cases}$$

Since the above matrix is semi-infinite, it must be truncated before its eigenvalues can be computed. Routine **MATEE** computes an estimate of the number of terms needed to get accurate results. This estimate can be overridden by calling **M2TEE** with **NORDER** equal to the desired order of the truncated matrix.

The eigenvalues of this matrix are computed using the routine **EVLSB** found in the IMSL Fortran Math Library, *Chapter 2, "Eigensystem Analysis"*.

Comments

1. Workspace may be explicitly provided, if desired, by use of **M2TEE/DM2TEE**. The reference is

```
CALL M2TEE (Q, N, ISYM, IPER, EVAL, NORDER, WORKD, WORKE)
```

The additional arguments are as follows:

NORDER — Order of the matrix whose eigenvalues are computed. (Input)

WORKD — Work vector of size **NORDER**. (Input/Output)

If **EVAL** is large enough then **EVAL** and **WORKD** can be the same vector.

WORKE — Work vector of size **NORDER**. (Input/Output)

2. Informational error

Type	Code	Description
4	1	The iteration for the eigenvalues did not converge.

Example

In this example, the eigenvalues for $Q = 5$, even symmetry, and π periodicity are computed and printed.

```

      USE UMACH_INT
      USE MATEE_INT

      IMPLICIT      NONE
!
!           Declare variables
      INTEGER      N
      PARAMETER    (N=10)
!
      INTEGER      ISYM, IPER, K, NOUT
      REAL         Q, EVAL(N)
!
!           Compute
      Q           = 5.0
      ISYM        = 0
      IPER        = 0
      CALL MATEE (Q, ISYM, IPER, EVAL)
!
!           Print the results
      CALL UMACH (2, NOUT)
      DO 10 K=1, N
         WRITE (NOUT,99999) 2*K-2, EVAL(K)
10 CONTINUE
99999 FORMAT (' Eigenvalue', I2, ' = ', F9.4)
      END

```

Output

```

Eigenvalue 0 = -5.8000
Eigenvalue 2 =  7.4491
Eigenvalue 4 = 17.0966
Eigenvalue 6 = 36.3609
Eigenvalue 8 = 64.1989
Eigenvalue10 = 100.1264
Eigenvalue12 = 144.0874
Eigenvalue14 = 196.0641
Eigenvalue16 = 256.0491
Eigenvalue18 = 324.0386

```

MATCE

Evaluates a sequence of even, periodic, integer order, real Mathieu functions.

Required Arguments

- X** — Argument for which the sequence of Mathieu functions is to be evaluated. (Input)
- Q** — Parameter. (Input)
The parameter **Q** must be positive.
- N** — Number of elements in the sequence. (Input)
- CE** — Vector of length **N** containing the values of the function through the series. (Output)
CE(I) contains the value of the Mathieu function of order **I - 1** at **X** for **I = 1** to **N**.

FORTRAN 90 Interface

- Generic: `CALL MATCE (X, Q, N, CE)`
- Specific: The specific interface names are `S_MATCE` and `D_MATCE`.

FORTRAN 77 Interface

- Single: `CALL MATCE (X, Q, N, CE)`
- Double: The double precision name is `DMATCE`.

Description

The eigenvalues of Mathieu's equation are computed using [MATEE](#). The function values are then computed using a sum of Bessel functions, see Gradshteyn and Ryzhik (1965), equation 8.661.

Comments

1. Workspace may be explicitly provided, if desired, by use of `M2TCE/DM2TCE`. The reference is
`CALL M2TCE (X, Q, N, CE, NORDEP, NEEDEV, EVAL0, EVAL1, COEF, WORK, BSJ)`
The additional arguments are as follows:

NORDER — Order of the matrix used to compute the eigenvalues. (Input)
It must be greater than **N**. Routine **MATSE** computes **NORDER** by the following call to **M3TEE**.

CALL M3TEE(Q, N, NORDER)

NEDEV — Logical variable, if **.TRUE.**, the eigenvalues must be computed. (Input)

EVAL0 — Real work vector of length **NORDER** containing the eigenvalues computed by **MATEE** with **ISYM = 0** and **IPER = 0**. (Input/Output)
If **NEDEV** is **.TRUE.**, then **EVAL0** is computed by **M2TCE**; otherwise, it must be set as an input value.

EVAL1 — Real work vector of length **NORDER** containing the eigenvalues computed by **MATEE** with **ISYM = 0** and **IPER = 1**. (Input/Output)
If **NEDEV** is **.TRUE.**, then **EVAL1** is computed by **M2TCE**; otherwise, it must be set as an input value.

COEF — Real work vector of length **NORDER + 4**.

WORK — Real work vector of length **NORDER + 4**.

BSJ — Real work vector of length $2 * \text{NORDER} - 2$.

2. Informational error

Type	Code	Description
4	1	The iteration for the eigenvalues did not converge.

Examples

Example 1

In this example, $ce_n(x = \pi/4, q = 1)$, $n = 0, \dots, 9$ is computed and printed.

```

USE CONST_INT
USE MATCE_INT
USE UMACH_INT

!
IMPLICIT      NONE                                Declare variables
INTEGER      N
PARAMETER    (N=10)

!
INTEGER      K, NOUT
REAL         CE(N), Q, X                          Compute
!
Q = 1.0
X = CONST('PI')
X = 0.25* X
CALL MATCE (X, Q, N, CE)

!
CALL UMACH (2, NOUT)                               Print the results
DO 10 K=1, N
  WRITE (NOUT,99999) K-1, X, Q, CE(K)

```

```

10 CONTINUE
99999 FORMAT (' ce sub', I2, ' (', F6.3, ', ', F6.3, ') = ', F6.3)
END

```

Output

```

ce sub 0 ( 0.785, 1.000) = 0.654
ce sub 1 ( 0.785, 1.000) = 0.794
ce sub 2 ( 0.785, 1.000) = 0.299
ce sub 3 ( 0.785, 1.000) = -0.555
ce sub 4 ( 0.785, 1.000) = -0.989
ce sub 5 ( 0.785, 1.000) = -0.776
ce sub 6 ( 0.785, 1.000) = -0.086
ce sub 7 ( 0.785, 1.000) = 0.654
ce sub 8 ( 0.785, 1.000) = 0.998
ce sub 9 ( 0.785, 1.000) = 0.746

```

Example 2

In this example, we compute $ce_n(x, q)$ for various values of n and x and a fixed value of q . To avoid having to recompute the eigenvalues, which depend on q but not on x , we compute the eigenvalues once and pass in their value to `M2TCE`. The eigenvalues are computed using `MATEE`. The routine `M3TEE` is used to compute `NORDER` based on `Q` and `N`. The arrays `BSJ`, `COEF` and `WORK` are used as temporary storage in `M2TCE`.

```

USE IMSL_LIBRARIES

IMPLICIT NONE

!
!                               Declare variables
INTEGER      MAXORD, N, NX
PARAMETER    (MAXORD=100, N=4, NX=5)
!
INTEGER      ISYM, K, NORDER, NOUT
REAL         BSJ(2*MAXORD-2), CE(N), COEF(MAXORD+4)
REAL         EVAL0(MAXORD), EVAL1(MAXORD), PI, Q, WORK(MAXORD+4), X
!
!                               Compute NORDER
Q = 1.0
CALL M3TEE (Q, N, NORDER)
!
CALL UMACH (2, NOUT)
WRITE (NOUT, 99997) NORDER
!
!                               Compute eigenvalues
ISYM = 0
CALL MATEE (Q, ISYM, 0, EVAL0)
CALL MATEE (Q, ISYM, 1, EVAL1)
!
!                               Compute function values
PI = CONST('PI')
!
WRITE (NOUT, 99998)
DO 10 K=0, NX
  X = (K*PI)/NX
  CALL M2TCE(X, Q, N, CE, NORDER, .FALSE., EVAL0, EVAL1, &
    COEF, WORK, BSJ)
  WRITE (NOUT, 99999) X, CE(1), CE(2), CE(3), CE(4)
10 CONTINUE
!
99997 FORMAT (' NORDER = ', I3)
99998 FORMAT (' /, 28X, 'Order', /, 20X, '0', 7X, '1', 7X, &
  '2', 7X, '3')
99999 FORMAT (' ce(', F6.3, ') = ', 4F8.3)
END

```


Output

```
NORDER = 23

Order
0      1      2      3
ce( 0.000) = 0.385  0.857  1.086  1.067
ce( 0.628) = 0.564  0.838  0.574 -0.131
ce( 1.257) = 0.926  0.425 -0.575 -0.820
ce( 1.885) = 0.926 -0.425 -0.575  0.820
ce( 2.513) = 0.564 -0.838  0.574  0.131
ce( 3.142) = 0.385 -0.857  1.086 -1.067
```

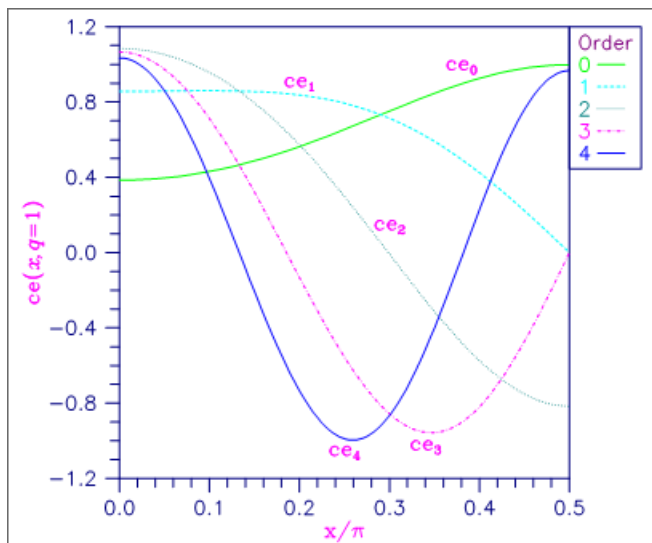


Figure 31, Plot of $ce_n(x, q = 1)$

MATSE

Evaluates a sequence of odd, periodic, integer order, real Mathieu functions.

Required Arguments

- X** — Argument for which the sequence of Mathieu functions is to be evaluated. (Input)
- Q** — Parameter. (Input)
The parameter **Q** must be positive.
- N** — Number of elements in the sequence. (Input)
- SE** — Vector of length **N** containing the values of the function through the series. (Output)
SE(I) contains the value of the Mathieu function of order **I** at **X** for **I** = 1 to **N**.

FORTRAN 90 Interface

- Generic: `CALL MATSE (X, Q, N, SE)`
- Specific: The specific interface names are `S_MATSE` and `D_MATSE`.

FORTRAN 77 Interface

- Single: `CALL MATSE (X, Q, N, SE)`
- Double: The double precision function name is `DMATSE`.

Description

The eigenvalues of Mathieu's equation are computed using [MATEE](#). The function values are then computed using a sum of Bessel functions, see Gradshteyn and Ryzhik (1965), equation 8.661.

Comments

1. Workspace may be explicitly provided, if desired, by use of `M2TSE/DM2TSE`. The reference is
`CALL M2TSE (X, Q, N, SE, NORDER, NEEDEV, EVAL0, EVAL1, COEF, WORK, BSJ)`
The additional arguments are as follows:

NORDER — Order of the matrix used to compute the eigenvalues. (Input)
It must be greater than **N**. Routine **MATSE** computes **NORDER** by the following call to **M3TEE**.

CALL M3TEE (Q, N, NORDER)

NEEDEV — Logical variable, if **.TRUE.**, the eigenvalues must be computed. (Input)

EVAL0 — Real work vector of length **NORDER** containing the eigenvalues computed by **MATEE** with **ISYM = 1** and **IPER = 0**. (Input/Output)
If **NEEDEV** is **.TRUE.**, then **EVAL0** is computed by **M2TSE**; otherwise, it must be set as an input value.

EVAL1 — Real work vector of length **NORDER** containing the eigenvalues computed by **MATEE** with **ISYM = 1** and **IPER = 1**. (Input/Output)
If **NEEDEV** is **.TRUE.**, then **EVAL1** is computed by **M2TSE**; otherwise, it must be set as an input value.

COEF — Real work vector of length **NORDER + 4**.

WORK — Real work vector of length **NORDER + 4**.

BSI — Real work vector of length $2 * \text{NORDER} + 1$.

2. Informational error

Type	Code	Description
4	1	The iteration for the eigenvalues did not converge.

Example

In this example, $se_n(x = \pi/4, q = 10), n = 0, \dots, 9$ is computed and printed.

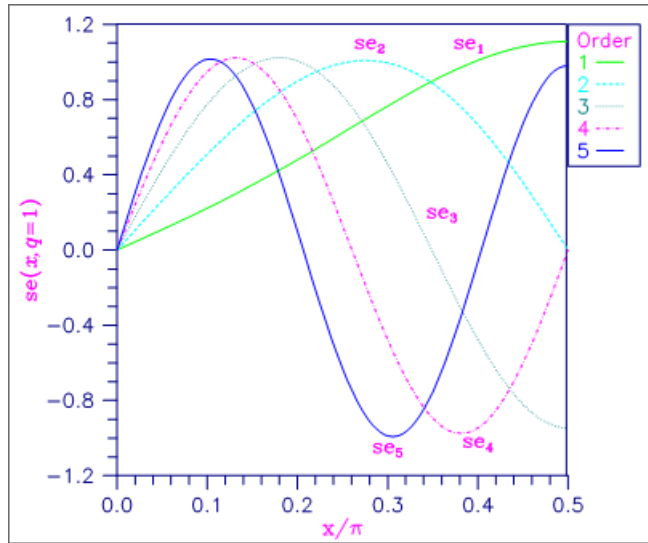


Figure 32, Plot of $se_n(x, q = 1)$

```

USE CONST_INT
USE MATSE_INT
USE UMACH_INT

IMPLICIT NONE
!
! Declare variables
INTEGER N
PARAMETER (N=10)
!
INTEGER K, NOUT
REAL SE(N), Q, X
!
! Compute
Q = 10.0
X = CONST('PI')
X = 0.25* X
CALL MATSE (X, Q, N, SE)
!
! Print the results
CALL UMACH (2, NOUT)
DO 10 K=1, N
WRITE (NOUT,99999) K-1, X, Q, SE(K)
10 CONTINUE
99999 FORMAT (' se sub', I2, ' (', F6.3, ', ', F6.3, ') = ', F6.3)
END

```

Output

```

se sub 0 ( 0.785,10.000) = 0.250
se sub 1 ( 0.785,10.000) = 0.692
se sub 2 ( 0.785,10.000) = 1.082
se sub 3 ( 0.785,10.000) = 0.960
se sub 4 ( 0.785,10.000) = 0.230
se sub 5 ( 0.785,10.000) = -0.634
se sub 6 ( 0.785,10.000) = -0.981
se sub 7 ( 0.785,10.000) = -0.588

```

```
se sub 8 ( 0.785,10.000) = 0.219  
se sub 9 ( 0.785,10.000) = 0.871
```

Miscellaneous Functions

Routines

Spence dilogarithm	SPENC	470
Initialize a Chebyshev series	INITS	472
Evaluate a Chebyshev series	CSEVL	474

Usage Notes

Many functions of one variable can be numerically computed using a Chebyshev series,

$$f(x) = \sum_{n=0}^{\infty} A_n T_n(x) \quad -1 \leq x \leq 1$$

A Chebyshev series is better for numerical computation than a Taylor series since the Chebyshev polynomials, $T_n(x)$, are better behaved than the monomials, x^n .

A Taylor series can be converted into a Chebyshev series using an algorithm of Fields and Wimp, (see Luke (1969), page 292).

Let

$$f(x) = \sum_{n=0}^{\infty} \xi_n x^n$$

be a Taylor series expansion valid for $|x| < 1$. Define

$$A_n = \frac{2}{4^n} \sum_{k=0}^{\infty} \frac{(n + \frac{1}{2})_k (n + 1)_k \xi_{n+k}}{(2n + 1)_k k!}$$

where $(a)_k = \Gamma(a + k)/\Gamma(a)$ is Pochhammer's symbol.

(Note that $(a)_{k+1} = (a + k)(a)_k$). Then,

$$f(x) = \frac{1}{2} T_0^*(x) + \sum_{n=1}^{\infty} A_n T_n^*(x) \quad 0 \leq x \leq 1$$

where

$$T_n^*(x)$$

are the shifted Chebyshev polynomials,

$$T_n^*(x) = T_n(2x - 1)$$

In an actual implementation of this algorithm, the number of terms in the Taylor series and the number of terms in the Chebyshev series must both be finite. If the Taylor series is an alternating series, then the error in using only the first M terms is less than $|\xi_{M+1}|$. The error in truncating the Chebyshev series to N terms is no more than

$$\sum_{n=N+1}^{\infty} |c_n|$$

If the Taylor series is valid on $|x| < R$, then we can write

$$f(x) = \sum_{n=0}^{\infty} \xi_n R^n (x/R)^n$$

and use $\xi_n R^n$ instead of ξ_n in the algorithm to obtain a Chebyshev series in x/R valid for $0 < x < R$. Unfortunately, if R is large, then the Chebyshev series converges more slowly.

The Taylor series centered at zero can be shifted to a Taylor series centered at c . Let $t = x - c$, so

$$\begin{aligned} f(x) = f(t+c) &= \sum_{n=0}^{\infty} \xi_n (t+c)^n = \sum_{n=0}^{\infty} \sum_{j=0}^n \xi_n \binom{n}{j} c^{n-j} t^j \\ &= \sum_{n=0}^{\infty} \hat{\xi}_n t^n = \sum_{n=0}^{\infty} \hat{\xi}_n (x-c)^n \end{aligned}$$

By interchanging the order of the double sum, it can easily be shown that

$$\hat{\xi}_j = \sum_{n=j}^{\infty} \binom{n}{j} c^{n-j} \xi_n$$

By combining scaling and shifting, we can obtain a Chebyshev series valid over any interval $[a, b]$ for which the original Taylor series converges.

The algorithm can also be applied to asymptotic series,

$$f(x) \sim \sum_{n=0}^{\infty} \xi_n x^{-n} \text{ as } |x| \rightarrow \infty$$

by treating the series truncated to M terms as a polynomial in $1/x$. The asymptotic series is usually divergent; but if it is alternating, the error in truncating the series to M terms is less than $|\xi_{M+1}| / R^{M+1}$ for $R \leq x < \infty$. Normally, as M increases, the error initially decreases to a small value and then increases without a bound. Therefore, there is a limit to the accuracy that can be obtained by increasing M . More accuracy can be obtained by increasing R . The optimal value of M depends on both the sequence ξ_j and R . For R fixed, the optimal value of M can be found by finding the value of M at which $|\xi_M| / R^M$ starts to increase.

Since we want a routine accurate to near machine precision, the algorithm must be implemented using somewhat higher precision than is normally used. This is best done using a symbolic computation package.

SPENC

This function evaluates a form of Spence's integral.

Function Return Value

SPENC — Function value. (Output)

Required Arguments

X — Argument for which the function value is desired. (Input)

FORTRAN 90 Interface

Generic: **SPENC (X)**

Specific: The specific interface names are **S_SPENC** and **D_SPENC**.

FORTRAN 77 Interface

Single: **SPENC (X)**

Double: The double precision function name is **DSPENC**.

Description

The Spence dilogarithm function, $s(x)$, is defined to be

$$s(x) = - \int_0^x \frac{\ln|1-y|}{y} dy$$

For $|x| \leq 1$, the uniformly convergent expansion

$$s(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^2}$$

is valid.

Spence's function can be used to evaluate much more general integral forms. For example,

$$c \int_0^z \frac{\log(ax+b)}{cx+d} dx = \log \left| \frac{a(cz+d)}{ad-bc} \right| - s \left(\frac{a(cz+d)}{ad-bc} \right)$$

Example

In this example, $s(0.2)$ is computed and printed.

```

      USE SPENC_INT
      USE UMACH_INT

      IMPLICIT      NONE
!
      INTEGER      NOUT
      REAL         VALUE, X
!
      X           = 0.2
      VALUE = SPENC(X)
!
      CALL UMACH (2, NOUT)
      WRITE (NOUT,99999) X, VALUE
99999 FORMAT (' SPENC(', F6.3, ') = ', F6.3)
      END

```

Output

```

SPENC( 0.200) = 0.211

```

INITS

This function initializes the orthogonal Chebyshev series so the function value is the number of terms needed to insure the error is no larger than the requested accuracy.

Function Return Value

INITS — Number of terms needed to insure the error is no larger than **ETA**. (Output)

Required Arguments

OS — Vector of length **NOS** containing coefficients in an orthogonal series. (Input)

NOS — Number of coefficients in **OS**. (Input)

ETA — Requested accuracy of the series. (Input)

Contrary to the usual convention, **ETA** is a **REAL** argument to **INITDS**.

FORTRAN 90 Interface

Generic: **INITS** (**OS**, **NOS**, **ETA**)

Specific: The specific interface names are **INITS** and **INITDS**.

FORTRAN 77 Interface

Single: **INITS** (**OS**, **NOS**, **ETA**)

Double: The double precision function name is **INITDS**.

Description

Function **INITS** initializes a Chebyshev series. The function **INITS** returns the number of terms in the series s of length n needed to insure that the error of the evaluated series is everywhere less than **ETA**. The number of input terms n must be greater than 1, so that a series of at least one term and an error estimate can be obtained. In addition, **ETA** should be larger than the absolute value of the last coefficient. If it is not, then all the terms of the series must be used, and no error estimate is available.

Comments

ETA will usually be chosen to be one tenth of machine precision.

CSEVL

This function evaluates the N -term Chebyshev series.

Function Return Value

CSEVL — Function value. (Output)

Required Arguments

X — Argument at which the series is to be evaluated. (Input)

CS — Vector of length N containing the terms of a Chebyshev series. (Input)
In evaluating **CS**, only half of the first coefficient is summed.

Optional Arguments

N — Number of terms in the vector **CS**. (Input)
Default: $N = \text{size}(\text{CS}, 1)$

FORTRAN 90 Interface

Generic: **CSEVL** (**X**, **CS** [, ...])

Specific: The specific interface names are **S_CSEVL** and **D_CSEVL**.

FORTRAN 77 Interface

Single: **CSEVL** (**X**, **CS**, **N**)

Double: The double precision function name is **DCSEVL**.

Description

Function **CSEVL** evaluates a Chebyshev series whose coefficients are stored in the array s of length n at the point x . The argument x must lie in the interval $[-1, +1]$. Other finite intervals can be linearly transformed to this canonical interval. Also, the number of terms in the series must be greater than zero but less than 1000. This latter limit is purely arbitrary; it is imposed in order to guard against the possibility of a floating point number being passed as an argument for n .

Comments

Informational error

Type	Code	Description
3	7	x is outside the interval $(-1.1, +1.1)$

Reference Material

Routines/Topics

User Errors.....	477
ERSET	480
IERCD and N1RTY	481
Machine-Dependent Constants.....	482
IMACH.....	483
AMACH	485
IFNAN(X)	488
UMACH.....	490
Reserved Names	492
Deprecated Features and Deleted Routines	493

User Errors

IMSL routines attempt to detect user errors and handle them in a way that provides as much information to the user as possible. To do this, we recognize various levels of severity of errors, and we also consider the extent of the error in the context of the purpose of the routine; a trivial error in one situation may be serious in another. IMSL routines attempt to report as many errors as they can reasonably detect. Multiple errors present a difficult problem in error detection because input is interpreted in an uncertain context after the first error is detected.

What Determines Error Severity

In some cases, the user's input may be mathematically correct, but because of limitations of the computer arithmetic and of the algorithm used, it is not possible to compute an answer accurately. In this case, the assessed degree of accuracy determines the severity of the error. In cases where the routine computes several output quantities, if some are not computable but most are, an error condition exists. The severity depends on an assessment of the overall impact of the error.

Terminal errors

If the user's input is regarded as meaningless, such as $N = -1$ when " N " is the number of equations, the routine prints a message giving the value of the erroneous input argument(s) and the reason for the erroneous input. The routine will then cause the user's program to stop. An error in which the user's input is meaningless is the most severe error and is called a *terminal error*. Multiple terminal error messages may be printed from a single routine.

Informational errors

In many cases, the best way to respond to an error condition is simply to correct the input and rerun the program. In other cases, the user may want to take actions in the program itself based on errors that occur. An error that may be used as the basis for corrective action within the program is called an *informational error*. If an informational error occurs, a user-retrievable code is set. A routine can return at most one informational error for a single reference to the routine. The codes for the informational error codes are printed in the error messages.

Other errors

In addition to informational errors, IMSL routines issue error messages for which no user-retrievable code is set. Multiple error messages for this kind of error may be printed. These errors, which generally are not described in the documentation, include terminal errors as well as less serious errors. Corrective action within the calling program is not possible for these errors.

Kinds of Errors and Default Actions

Five levels of severity of errors are defined in the MATH/LIBRARY Special Functions. Each level has an associated PRINT attribute and a STOP attribute. These attributes have default settings (YES or NO), but they may also be set by the user. The purpose of having multiple error severity levels is to provide independent control of actions to be taken for errors of different severity. Upon return from an IMSL routine, exactly one error state exists. (A code 0 "error" is no informational error.) Even if more than one informational error occurs, only one message is printed (if the PRINT attribute is YES). Multiple errors for which no corrective action within the calling program is reasonable or necessary result in the printing of multiple messages (if the PRINT attribute for their severity level is YES). Errors of any of the severity levels except level 5 may be informational errors.

- **Level 1: Note.** A *note* is issued to indicate the possibility of a trivial error or simply to provide information about the computations. Default attributes: PRINT=NO, STOP=NO
- **Level 2: Alert.** An *alert* indicates that the user should be advised about events occurring in the software. Default attributes: PRINT=NO, STOP=NO
- **Level 3: Warning.** A *warning* indicates the existence of a condition that may require corrective action by the user or calling routine. A warning error may be issued because the results are accurate to only a few decimal places, because some of the output may be erroneous but most of the output is correct, or because some assumptions underlying the analysis technique are violated. Often no corrective action is necessary and the condition can be ignored. Default attributes: PRINT=YES, STOP=NO
- **Level 4: Fatal.** A *fatal* error indicates the existence of a condition that may be serious. In most cases, the user or calling routine must take corrective action to recover. Default attributes: PRINT=YES, STOP=YES
- **Level 5: Terminal.** A *terminal* error is serious. It usually is the result of an incorrect specification, such as specifying a negative number as the number of equations. These errors may also be caused by various programming errors impossible to diagnose correctly in FORTRAN. The resulting error message may be perplexing to the user. In such cases, the user is advised to compare carefully the actual arguments passed to the routine with the dummy argument descriptions given in the documentation. Special attention should be given to checking argument order and data types.

A terminal error is not an informational error because corrective action within the program is generally not reasonable. In normal usage, execution is terminated immediately when a terminal error occurs. Messages relating to more than one terminal error are printed if they occur. Default attributes: PRINT=YES, STOP=YES

The user can set PRINT and STOP attributes by calling **ERSET** as described in "Routines for Error Handling."

Errors in Lower-Level Routines

It is possible that a user's program may call an IMSL routine that in turn calls a nested sequence of lower-level IMSL routines. If an error occurs at a lower level in such a nest of routines and if the lower-level routine cannot pass the information up to the original user-called routine, then a traceback of the routines is produced. The only common situation in which this can occur is when an IMSL routine calls a user-supplied routine that in turn calls another IMSL routine.

Routines for Error Handling

There are three ways in which the user may interact with the IMSL error handling system: (1) to change the default actions, (2) to retrieve the integer code of an informational error so as to take corrective action, and (3) to determine the severity level of an error. The routines to use are ERSET, IERCD and N1RTY, respectively.

ERSET

Change the default printing or stopping actions when errors of a particular error severity level occur.

Required Arguments

IERSVR — Error severity level indicator. (Input)

If **IERSVR** = 0, actions are set for levels 1 to 5. If **IERSVR** is 1 to 5, actions are set for errors of the specified severity level.

IPACT — Printing action. (Input)

IPACT	Action
-1	Do not change current setting(s).
0	Do not print.
1	Print.
2	Restore the default setting(s).

ISACT — Stopping action. (Input)

ISACT	Action
-1	Do not change current setting(s).
0	Do not stop.
1	Stop.
2	Restore the default setting(s).

FORTRAN 90 Interface

Generic: `CALL ERSET (IERSVR, IPACT, ISACT)`

Specific: The specific interface name is `ERSET`.

FORTRAN 77 Interface

Single: `CALL ERSET (IERSVR, IPACT, ISACT)`

IERCD and N1RTY

The last two routines for interacting with the error handling system, **IERCD** and **N1RTY**, are **INTEGER** functions and are described in the following material.

IERCD retrieves the integer code for an informational error. Since it has no arguments, it may be used in the following way:

```
ICODE = IERCD()
```

The function retrieves the code set by the most recently called IMSL routine.

N1RTY retrieves the error type set by the most recently called IMSL routine. It is used in the following way:

```
ITYPE = N1RTY(1)
```

ITYPE = 1, 2, 4, and 5 correspond to error severity levels 1, 2, 4, and 5, respectively. **ITYPE** = 3 and **ITYPE** = 6 are both warning errors, error severity level 3. While **ITYPE** = 3 errors are informational errors (**IERCD**() \neq 0), **ITYPE** = 6 errors are not informational errors (**IERCD**() = 0).

For software developers requiring additional interaction with the IMSL error handling system, see Aird and Howell (1991).

Examples

Changes to Default Actions

Some possible changes to the default actions are illustrated below. The default actions remain in effect for the kinds of errors not included in the call to **ERSET**.

To turn off printing of warning error messages:

```
CALL ERSET ( 3, 0, -1 )
```

To stop if warning errors occur:

```
CALL ERSET ( 3, -1, 1 )
```

To print all error messages:

```
CALL ERSET ( 0, 1, -1 )
```

To restore all default settings:

```
CALL ERSET ( 0, 2, 2 )
```

Machine-Dependent Constants

The function subprograms in this section return machine-dependent information and can be used to enhance portability of programs between different computers. The routines [IMACH](#), and [AMACH](#) describe the computer's arithmetic. The routine [UMACH](#) describes the input, output, and error output unit numbers.

IMACH

This function retrieves machine integer constants that define the arithmetic used by the computer.

Function Return Value

IMACH(1) = Number of bits per integer storage unit.

IMACH(2) = Number of characters per integer storage unit:

Integers are represented in M -digit, base A form as

$$\sigma \sum_{k=0}^M x_k A^k$$

where σ is the sign and $0 \leq x_k < A$, $k = 0, \dots, M$.

Then,

IMACH(3) = A , the base.

IMACH(4) = M , the number of base- A digits.

IMACH(5) = $A^M - 1$, the largest integer.

The machine model assumes that floating-point numbers are represented in normalized N -digit, base B form as

$$\sigma B^E \sum_{k=1}^N x_k B^{-k}$$

where σ is the sign, $0 < x_1 < B$, $0 \leq x_k < B$, $k = 2, \dots, N$ and $E_{\min} \leq E \leq E_{\max}$. Then,

IMACH(6) = B , the base.

IMACH(7) = N_s , the number base- B -digits in single precision.

IMACH(8) = E_{\min_s} , the smallest single precision exponent.

IMACH(9) = E_{\max_s} , the largest single precision exponent.

IMACH(10) = N_d , the number base- B -digits in double precision.

IMACH(11) = E_{\min_d} , the smallest double precision exponent.

IMACH(12) = E_{\max_d} , largest double precision exponent.

Required Arguments

I — Index of the desired constant. (Input)

FORTRAN 90 Interface

Generic: `IMACH (I)`

Specific: The specific interface name is `IMACH`.

FORTRAN 77 Interface

Single: `IMACH (I)`

AMACH

The function subprogram **AMACH** retrieves machine constants that define the computer's single-precision or double precision arithmetic. Such floating-point numbers are represented in normalized N -digit, base B form as

$$\sigma B^E \sum_{k=1}^N x_k B^{-k}$$

where σ is the sign, $0 < x_1 < B$, $0 \leq x_k < B$, $k = 2, \dots, N$ and

$$E_{\min} \leq E \leq E_{\max}$$

Function Return Value

AMACH(1) = $B^{E_{\min}-1}$, the smallest normalized positive number.

AMACH(2) = $B^{E_{\max}-1} (1 - B^{-N})$, the largest number.

AMACH(3) = B^{-N} , the smallest relative spacing.

AMACH(4) = B^{1-N} , the largest relative spacing.

AMACH(5) = $\log_{10}(B)$.

AMACH(6) = NaN (non-signaling not a number).

AMACH(7) = positive machine infinity.

AMACH(8) = negative machine infinity.

See [Comment 1](#) for a description of the use of the generic version of this function.

See [Comment 2](#) for a description of **min**, **max**, and **N**.

Required Arguments

I — Index of the desired constant. (Input)

FORTRAN 90 Interface

Generic: **AMACH** (I)

Specific: The specific interface names are **S_AMACH** and **D_AMACH**.

FORTRAN 77 Interface

Single: `AMACH(I)`

Double: The double precision name is `DMACH`.

Comments

1. If the generic version of this function is used, the immediate result must be stored in a variable before use in an expression. For example:

```
X = AMACH(I)
```

```
Y = SQRT(X)
```

must be used rather than

```
Y = SQRT(AMACH(I)).
```

If this is too much of a restriction on the programmer, then the specific name can be used without this restriction.

2. Note that for single precision $B = \text{IMACH}(6)$, $N = \text{IMACH}(7)$.
 $E_{\min} = \text{IMACH}(8)$, and $E_{\max} = \text{IMACH}(9)$.
For double precision $B = \text{IMACH}(6)$, $N = \text{IMACH}(10)$.
 $E_{\min} = \text{IMACH}(11)$, and $E_{\max} = \text{IMACH}(12)$.
3. The IEEE standard for binary arithmetic (see IEEE 1985) specifies *quiet* NaN (not a number) as the result of various invalid or ambiguous operations, such as 0/0. The intent is that `AMACH(6)` return a *quiet* NaN. On IEEE format computers that do not support a quiet NaN, a special value near `AMACH(2)` is returned for `AMACH(6)`. On computers that do not have a special representation for infinity, `AMACH(7)` returns the same value as `AMACH(2)`.

DMACH

See [AMACH](#).

IFNAN(X)

This logical function checks if the argument **X** is NaN (not a number).

Function Return Value

IFNAN - Logical function value. True is returned if the input argument is a **NAN**. Otherwise, False is returned. (Output)

Required Arguments

X - Argument for which the test for **NAN** is desired. (Input)

FORTRAN 90 Interface

Generic: **IFNAN (X)**

Specific: The specific interface names are **S_IFNAN** and **D_IFNAN**.

FORTRAN 77 Interface

Single: **IFNAN (X)**

Double: The double precision name is **DIFNAN**.

Description

The logical function **IFNAN** checks if the single or double precision argument **X** is **NAN** (not a number). The function **IFNAN** is provided to facilitate the transfer of programs across computer systems. This is because the check for NaN can be tricky and not portable across computer systems that do not adhere to the IEEE standard. For example, on computers that support the IEEE standard for binary arithmetic (see IEEE 1985), NaN is specified as a bit format not equal to itself. Thus, the check is performed as

```
IFNAN = X .NE. X
```

On other computers that do not use IEEE floating-point format, the check can be performed as:

```
IFNAN = X .EQ. AMACH(6)
```

The function `IFNAN` is equivalent to the specification of the function `Isnan` listed in the Appendix, (IEEE 1985). The above example illustrates the use of `IFNAN`. If `X` is NaN, a message is printed instead of `X`. (Routine `UMACH`, which is described in the following section, is used to retrieve the output unit number for printing the message.)

Example

```
USE IFNAN_INT
USE AMACH_INT
USE UMACH_INT

IMPLICIT NONE
INTEGER NOUT
REAL X

!
CALL UMACH (2, NOUT)
!
X = AMACH(6)
IF (IFNAN(X)) THEN
  WRITE (NOUT,*) ' X is NaN (not a number).'
ELSE
  WRITE (NOUT,*) ' X = ', X
END IF
!
END
```

Output

```
X is NaN (not a number).
```

UMACH

Routine **UMACH** sets or retrieves the input, output, or error output device unit numbers.

Required Arguments

N — Integer value indicating the action desired. If the value of **N** is negative, the input, output, or error output unit number is reset to **NUNIT**. If the value of **N** is positive, the input, output, or error output unit number is returned in **NUNIT**. See the table in argument **NUNIT** for legal values of **N**. (Input)

NUNIT — The unit number that is either retrieved or set, depending on the value of input argument **N**. (Input/Output)

The arguments are summarized by the following table:

N	Effect
1	Retrieves input unit number in NUNIT .
2	Retrieves output unit number in NUNIT .
3	Retrieves error output unit number in NUNIT .
-1	Sets the input unit number to NUNIT .
-2	Sets the output unit number to NUNIT .
-3	Sets the error output unit number to NUNIT .

FORTRAN 90 Interface

Generic: `CALL UMACH (N, NUNIT)`

Specific: The specific interface name is **UMACH**.

FORTRAN 77 Interface

Single: `CALL UMACH (N, NUNIT)`

Description

Routine **UMACH** sets or retrieves the input, output, or error output device unit numbers. **UMACH** is set automatically so that the default FORTRAN unit numbers for standard input, standard output, and standard error are used. These unit numbers can be changed by inserting a call to **UMACH** at the beginning of the main program that calls MATH/LIBRARY routines. If these unit numbers are changed from the standard values, the user should insert an appropriate **OPEN** statement in the calling program.

Example

In the following example, a terminal error is issued from the MATH/LIBRARY **AMACH** function since the argument is invalid. With a call to **UMACH**, the error message will be written to a local file named "**CHECKERR**".

```
      USE AMACH_INT
      USE UMACH_INT

      IMPLICIT      NONE
      INTEGER      N, NUNIT
      REAL         X

!                                     Set Parameter
      N = 0

!
      NUNIT = 9
      CALL UMACH (-3, NUNIT)
      OPEN (UNIT=9,FILE='CHECKERR')
      X = AMACH(N)
      END
```

Output

```
The output from this example, written to "CHECKERR" is:
*** TERMINAL ERROR 5 from AMACH.  The argument must be between 1 and 8
***           inclusive. N = 0
```

Reserved Names

When writing programs accessing IMSL MATH/LIBRARY Special Functions, the user should choose FORTRAN names that do not conflict with names of IMSL subroutines, functions, or named common blocks, such as the workspace common block **WORKSP** (see [Automatic Workspace Allocation](#)). The user needs to be aware of two types of name conflicts that can arise. The first type of name conflict occurs when a name (technically a *symbolic name*) is not uniquely defined within a program unit (either a main program or a subprogram). For example, such a name conflict exists when the name **BSJS** is used to refer both to a type **REAL** variable and to the IMSL routine **BSJS** in a single program unit. Such errors are detected during compilation and are easy to correct. The second type of name conflict, which can be more serious, occurs when names of program units and named common blocks are not unique. For example, such a name conflict would be caused by the user defining a routine named **WORKSP** and also referencing a MATH/LIBRARY Special Functions routine that uses the named common block **WORKSP**. Likewise, the user must not define a subprogram with the same name as a subprogram in MATH/LIBRARY Special Functions, that is referenced directly by the user's program or is referenced indirectly by other MATH/LIBRARY Special Functions subprograms.

MATH/LIBRARY Special Functions consists of many routines, some that are described in the *User's Manual* and others that are not intended to be called by the user and, hence, that are not documented. If the choice of names were completely random over the set of valid FORTRAN names and if a program uses only a small subset of MATH/LIBRARY Special Functions, the probability of name conflicts is very small. Since names are usually chosen to be mnemonic, however, the user may wish to take some precautions in choosing FORTRAN names.

Many IMSL names consist of a root name that may have a prefix to indicate the type of the routine. For example, the IMSL single precision routine for computing Bessel functions of the first kind with real order has the name **BSJS**, which is the root name, and the corresponding IMSL double precision routine has the name **DBSJS**. Associated with these two routines are **B2JS** and **DB2JS**. **BSJS** is listed in the Alphabetical Index of Routines, but **DBSJS**, **B2JS**, and **DB2JS** are not. The user of **BSJS** must consider both names **BSJS** and **B2JS** to be reserved; likewise, the user of **DBSJS** must consider both names **DBSJS** and **DB2JS** to be reserved. The root names of *all* routines and named common blocks that are used by MATH/LIBRARY Special Functions and that do not have a numeral in the second position of the root name are listed in the Alphabetical Index of Routines. Some of the routines in this Index are not intended to be called by the user and so are not documented. The careful user can avoid any conflicts with IMSL names if the following rules are observed:

- Do not choose a name that appears in the Alphabetical Summary of Routines in the *User's Manual*, nor one of these names preceded by a **D**, **S**_, **D**_, **C**_, or **Z**_.
- Do not choose a name of three or more characters with a numeral in the second or third position.

These simplified rules include many combinations that are, in fact, allowable. However, if the user selects names that conform to these rules, no conflict will be encountered.

Deprecated Features and Deleted Routines

Automatic Workspace Allocation

FORTRAN subroutines that work with arrays as input and output often require extra arrays for use as workspace while doing computations or moving around data. IMSL routines generally do not require the user explicitly to allocate such arrays for use as workspace. On most systems the workspace allocation is handled transparently. The only limitation is the actual amount of memory available on the system.

On some systems the workspace is allocated out of a stack that is passed as a FORTRAN array in a named common block **WORKSP**. A very similar use of a workspace stack is described by Fox et al. (1978, pages 116–121). (For compatibility with older versions of the IMSL Libraries, space is allocated from the **COMMON** block, if possible.)

The arrays for workspace appear as arguments in lower-level routines. For example, the IMSL Math routine **LSARG** (in *Chapter 1, "Linear Systems"*), which solves systems of linear equations, needs arrays for workspace. **LSARG** allocates arrays from the common area, and passes them to the lower-level routine **L2ARG** which does the computations. In the "Comments" section of the documentation for **LSARG**, the amount of workspace is noted and the call to **L2ARG** is described. This scheme for using lower-level routines is followed throughout the IMSL Libraries. The names of these routines have a "2" in the second position (or in the third position in double precision routines having a "D" prefix). The user can provide workspace explicitly and call directly the "2-level" routine, which is documented along with the main routine. In a very few cases, the 2-level routine allows additional options that the main routine does not allow.

Prior to returning to the calling program, a routine that allocates workspace generally deallocates that space so that it becomes available for use in other routines.

Changing the Amount of Space Allocated

This section is relevant only to those systems on which the transparent workspace allocator is not available.

By default, the total amount of space allocated in the common area for storage of numeric data is 5000 numeric storage units. (A numeric storage unit is the amount of space required to store an integer or a real number. By comparison, a double precision unit is twice this amount. Therefore, the total amount of space allocated in the common area for storage of numeric data is 2500 double precision units.) This space is allocated as needed for **INTEGER**, **REAL**, or other numeric data. For larger problems in which the default amount of workspace is insufficient, the user can change the allocation by supplying the FORTRAN statements to define the array in the named common block and by informing the IMSL workspace allocation system of the new size of the common array. To request 7000 units, the statements are


```
COMMON /WORKSP/ RWKSP
REAL RWKSP(7000)
CALL IWKIN(7000)
```

If an IMSL routine attempts to allocate workspace in excess of the amount available in the common stack, the routine issues a fatal error message that indicates how much space is needed and prints statements like those above to guide the user in allocating the necessary amount. The program below uses IMSL routine [BSJS](#) (See [Chapter 6, "Bessel Functions"](#) of this manual) to illustrate this feature.

This routine requires workspace that is just larger than twice the number of function values requested.

```
INTEGER      N
REAL         BS(10000), X, XNU
EXTERNAL    BSJS
!
!                               Set Parameters
XNU = .5
X   = 1.
N   = 6000
CALL BSJS (XNU, X, N, BS)
END
```

Output

```
*** TERMINAL ERROR from BSJS.  Insufficient workspace for
***      current allocation(s).  Correct by calling
***      IWKIN from main program with the three
***      following statements:  (REGARDLESS OF
***      PRECISION)
***      COMMON /WORKSP/ RWKSP
***      REAL RWKSP(12018)
***      CALL IWKIN(12018)
*** TERMINAL ERROR from BSJS.  The workspace requirement is
***      based on N =6000.
STOP
```

In most cases, the amount of workspace is dependent on the parameters of the problem so the amount needed is known exactly. In a few cases, however, the amount of workspace is dependent on the data (for example, if it is necessary to count all of the unique values in a vector). Thus, the IMSL routine cannot tell in advance exactly how much workspace is needed. In such cases, the error message printed is an estimate of the amount of space required.

Character Workspace

Since character arrays cannot be equivalenced with numeric arrays, a separate named common block **WKSPCH** is provided for character workspace. In most respects, this stack is managed in the same way as the numeric stack. The default size of the character workspace is 2000 character units. (A character unit is the amount of space required to store one character.) The routine analogous to **IWKIN** used to change the default allocation is **IWKICIN**.

The routines in the following list are being deprecated in Version 2.0 of MATH/LIBRARY Special Functions. A deprecated routine is one that is no longer used by anything in the library but is being included in the product for those users who may be currently referencing it in their application. However, any future versions of

MATH/LIBRARY Special Functions will not include these routines. If any of these routines are being called within an application, it is recommended that you change your code or retain the deprecated routine before replacing this library with the next version. Most of these routines were called by users only when they needed to set up their own workspace. Thus, the impact of these changes should be limited.

G2DF
G2IN
G3DF

The following specific FORTRAN intrinsic functions are no longer supplied by IMSL. They can all be found in their manufacturer's FORTRAN runtime libraries. If any change must be made to the user's application as a result of their removal from the IMSL Libraries, it is limited to the redeclaration of the function from "external" to "intrinsic." Argument lists and results should be identical.

ACOS	CEXP	DATAN2	DSQRT
ASIN	CLOG	DCOS	DTAN
ALOG	COS	DCOSH	DTANH
ALOG10	COSH	DEXP	EXP
ASIN	CSIN	DINT	SIN
ATAN	CSQRT	DLOG	SINH
ATAN2	DACOS	DLOG10	SQRT
CABS	DASIN	DSIN	TAN
CCOS	DATAN	DSINH	TANH

Appendix A, Alphabetical Summary of Routines

[A] [B] [C] [D] [E] [F] [G] [H] [I] [L] [M] [N] [P] [R] [S] [T] [U] [W]

A

ACOS	Evaluates the complex arc cosine.
ACOSH	Evaluates the real or complex arc hyperbolic cosine.
AI	Evaluates the Airy function.
AID	Evaluates the derivative of the Airy function.
AIDE	Evaluates the Airy function of the second kind.
AIE	Evaluates the exponentially scaled derivative of the Airy function.
AKEI0	Evaluates the Kelvin function of the second kind, kei, of order zero.
AKEI1	Evaluates the Kelvin function of the second kind, kei, of order one.
AKEIP0	Evaluates the derivative of the Kelvin function of the second kind, kei, of order zero.
AKER0	Evaluates the Kelvin function of the second kind, ker, of order zero.
AKER1	Evaluates the Kelvin function of the second kind, ker, of order one.
AKERP0	Evaluates the derivative of the Kelvin function of the second kind, ker, of order zero.
AKS1DF	Evaluates the cumulative distribution function of the one-sided Kolmogorov-Smirnov goodness of fit D^+ or D^- test statistic based on continuous data for one sample.
AKS2DF	Evaluates the cumulative distribution function of the Kolmogorov-Smirnov goodness of fit D test statistic based on continuous data for two samples.

ALBETA	Evaluates the natural logarithm of the complete beta function for positive arguments.
ALGAMS	Returns the logarithm of the absolute value of the gamma function and the sign of gamma.
ALI	Evaluates the logarithmic integral.
ALNDF	Evaluates the lognormal cumulative probability distribution function
ALNGAM	Evaluates the real or complex function, $\ln \gamma(x) $.
ALNIN	Evaluates the inverse of the lognormal cumulative probability distribution function.
ALNPR	Evaluates the lognormal probability density function.
ALNREL	Evaluates $\ln(x + 1)$ for real or complex x .
AMACH	Retrieves single-precision machine constants.
ANORDF	Evaluates the standard normal (Gaussian) cumulative distribution function.
ANORPR	Evaluates the normal probability density function.
ANORIN	Evaluates the inverse of the standard normal (Gaussian) cumulative distribution function.
ASIN	Evaluates the complex arc sine.
ASINH	Evaluates the \sinh^{-1} arc sine x for real or complex x .
ATAN	Evaluates the complex arc tangent.
ATAN2	Evaluates the complex arc tangent of a ratio.
ATANH	Evaluates $\tanh^{-1} x$ for real or complex x .

B

BEI0	Evaluates the Kelvin function of the first kind, bei , of order zero.
BEI1	Evaluates the Kelvin function of the first kind, bei , of order one.
BEIP0	Evaluates the derivative of the Kelvin function of the first kind, bei , of order zero.
BER0	Evaluates the Kelvin function of the first kind, ber , of order zero.
BER1	Evaluates the Kelvin function of the first kind, ber , of order one.

BERP0	Evaluates the derivative of the Kelvin function of the first kind, ber, of order zero.
BETA	Evaluates the real or complex beta function, $\beta(a,b)$.
BETAI	Evaluates the incomplete beta function ratio.
BETDF	Evaluates the beta cumulative distribution function.
BETIN	Evaluates the inverse of the beta cumulative distribution function.
BETNDF	Evaluates the beta cumulative distribution function.
BETNIN	Evaluates the inverse of the beta cumulative distribution function.
BETNPR	This function evaluates the noncentral beta probability density function.
BETPR	Evaluates the beta probability density function.
BI	Evaluates the Airy function of the second kind.
BID	Evaluates the derivative of the Airy function of the second kind.
BIDE	Evaluates the exponentially scaled derivative of the Airy function of the second kind.
BIE	Evaluates the exponentially scaled Airy function of the second kind.
BINDF	Evaluates the binomial cumulative distribution function.
BINOM	Evaluates the binomial coefficient.
BINPR	Evaluates the binomial probability density function.
BNRDF	Evaluates the bivariate normal cumulative distribution function.
BSI0	Evaluates the modified Bessel function of the first kind of order zero.
BSI0E	Evaluates the exponentially scaled modified Bessel function of the first kind of order zero.
BSI1	Evaluates the modified Bessel function of the first kind of order one.
BSI1E	Evaluates the exponentially scaled modified Bessel function of the first kind of order one.
BSIES	Evaluates a sequence of exponentially scaled modified Bessel functions of the first kind with nonnegative real order and real positive arguments.
BSINS	Evaluates a sequence of modified Bessel functions of the first kind with integer order and real or complex arguments.
BSIS	Evaluates a sequence of modified Bessel functions of the first kind with real order and real positive arguments.
BSJ0	Evaluates the Bessel function of the first kind of order zero.

BSJ1	Evaluates the Bessel function of the first kind of order one.
BSJNS	Evaluates a sequence of Bessel functions of the first kind with integer order and real arguments.
BSJS	Evaluates a sequence of Bessel functions of the first kind with real order and real positive arguments.
BSK0	Evaluates the modified Bessel function of the second kind of order zero.
BSK0E	Evaluates the exponentially scaled modified Bessel function of the second kind of order zero.
BSK1	Evaluates the modified Bessel function of the second kind of order one.
BSK1E	Evaluates the exponentially scaled modified Bessel function of the second kind of order one.
BSKES	Evaluates a sequence of exponentially scaled modified Bessel functions of the second kind of fractional order.
BSKS	Evaluates a sequence of modified Bessel functions of the second kind of fractional order.
BSY0	Evaluates the Bessel function of the second kind of order zero.
BSY1	Evaluates the Bessel function of the second kind of order one.
BSYS	Evaluates a sequence of Bessel functions of the second kind with real nonnegative order and real positive arguments.

C

CAI	Evaluates the Airy function of the first kind for complex arguments.
CAID	Evaluates the derivative of the Airy function of the first kind for complex arguments.
CARG	Evaluates the argument of a complex number.
CBI	Evaluates the Airy function of the second kind for complex arguments.
CBID	Evaluates the derivative of the Airy function of the second kind for complex arguments.
CBIS	Evaluates a sequence of modified Bessel functions of the first kind with real order and complex arguments.
CBJS	Evaluates a sequence of Bessel functions of the first kind with real order and complex arguments.
CBKS	Evaluates a sequence of Modified Bessel functions of the second kind with real order and complex arguments.

CBRT	Evaluates the cube root.
CBYS	Evaluates a sequence of Bessel functions of the second kind with real order and complex arguments.
CERFE	Evaluates the complex scaled complemented error function.
CHI	Evaluates the hyperbolic cosine integral.
CHIDF	Evaluates the chi-squared cumulative distribution function
CHIIN	Evaluates the inverse of the chi-squared cumulative distribution function.
CHIPR	Evaluates the chi-squared probability density function
CI	Evaluates the cosine integral.
CIN	Evaluates a function closely related to the cosine integral.
CINH	Evaluates a function closely related to the hyperbolic cosine integral.
COSDG	Evaluates the cosine for the argument in degrees.
COT	Evaluates the cotangent.
CSEVL	Evaluates the N -term Chebyshev series.
CSNDF	Evaluates the noncentral chi-squared cumulative distribution function.
CSNIN	Evaluates the inverse of the noncentral chi-squared cumulative function.
CSNPR	This function evaluates the noncentral chi-squared probability density function.
CWPL	Evaluates the Weierstrass P -function in the lemniscat case for complex argument with unit period parallelogram.
CWPLD	Evaluate the first derivative of the Weierstrass P -function in the lemniscatic case for complex argum with unit period parallelogram.
CWPQ	Evaluates the Weierstrass P -function in the equianharmonic case for complex argument with unit period parallelogram.
CWPQD	Evaluates the first derivative of the Weierstrass P -function in the equianharmonic case for complex argument with unit period parallelogram.

D

DAWS	Evaluates Dawson function.
DMACH	Retrieves double precision machine constants.

E

E1	Evaluates the exponential integral for arguments greater than zero and the Cauchy principal value of the integral for arguments less than zero.
EI	Evaluates the exponential integral for arguments greater than zero and the Cauchy principal value for arguments less than zero.
EJCN	Evaluates the Jacobi elliptic function $\text{cn}(x, m)$.
EJDN	This function evaluates the Jacobi elliptic function $\text{dn}(x, m)$.
EJSN	Evaluates the Jacobi elliptic function $\text{sn}(x, m)$.
ELE	Evaluates the complete elliptic integral of the second kind $E(x)$.
ELK	Evaluates the complete elliptic integral of the kind $K(x)$.
ELRC	Evaluates an elementary integral from which inverse circular functions, logarithms and inverse hyperbolic functions can be computed.
ELRD	Evaluates Carlson's incomplete elliptic integral of the second kind $\text{RD}(x, y, z)$.
ELRF	Evaluates Carlson's incomplete elliptic integral of the first kind $\text{RF}(x, y, z)$.
ELRJ	Evaluates Carlson's incomplete elliptic integral of the third kind $\text{RJ}(x, y, z, \rho)$.
ENE	Evaluates the exponential integral of integer order for arguments greater than zero scaled by $\text{EXP}(x)$.
ERF	Evaluates the error function.
ERFC	Evaluates the complementary error function.
ERFCE	Evaluates the exponentially scaled complementary error function.
ERFCI	Evaluates the inverse complementary error function.
ERFI	Evaluates the inverse error function.
ERSET	Sets error handler default printer and stop actions.
EXPDF	Evaluates the exponential cumulative distribution function.
EXPIN	Evaluates the inverse of the exponential cumulative distribution function.
EXPPR	Evaluates the exponential probability density function.
EXPRL	Evaluates $(e^x - 1)/x$ for real or complex x .
EXVDF	Evaluates the extreme value cumulative distribution function.

EXVIN	Evaluates the inverse of the extreme value cumulative distribution function.
EXVPR	Evaluates the extreme value probability density function.

F

FAC	Evaluates the factorial of the argument.
FDF	Evaluates the F cumulative distribution function.
FIN	Evaluates the inverse of the F cumulative distribution function.
FNDF	Noncentral F cumulative distribution function.
FNIN	This function evaluates the inverse of the noncentral F cumulative distribution function (CDF).
FNPR	This function evaluates the noncentral F cumulative distribution function (CDF).
FPR	Evaluates the F probability density function.
FRESC	Evaluates the cosine Fresnel integral.
FRESS	Evaluates the sine Fresnel integral.

G

GAMDF	Evaluates the gamma cumulative distribution function.
GAMI	Evaluates the incomplete gamma function.
GAMIC	Evaluates the complementary incomplete gamma function.
GAMIN	This function evaluates the inverse of the gamma cumulative distribution function.
GAMIT	Evaluates the Tricomi form of the incomplete gamma function.
GAMMA	Evaluates the real or complex gamma function, $\Gamma(x)$.
GAMPR	This function evaluates the gamma probability density function.

GAMR	Evaluates the reciprocal of the real or complex gamma function, $1/\Gamma(x)$.
GCDF	Evaluates a general continuous cumulative distribution function given ordinates of the density.
GCIN	Evaluates the inverse of a general continuous cumulative distribution function given ordinates of the density.
GEODF	Evaluates the discrete geometric cumulative probability distribution function.
GEOIN	Evaluates the inverse of the geometric cumulative probability distribution function.
GEOPR	Evaluates the discrete geometric probability density function.
GFNIN	Evaluates the inverse of a general continuous cumulative distribution function given in a subprogram.

H

HYPDF	Evaluates the hypergeometric cumulative distribution function.
HYPPR	Evaluates the hypergeometric probability density function.

I

IERCD and N1RTY	Retrieves the integer code for an informational error.
IFNAN(X)	Checks if a value is NaN (not a number).
IMACH	Retrieves integer machine constants.
INITS	Initializes the orthogonal series so the function value is the number of terms needed to insure the error is no larger than the requested accuracy.

L

LOG10	Evaluates the complex base 10 logarithm, $\log_{10} z$.
-------	----------------------------------------------------------

M

MATCE	Evaluates a sequence of even, periodic, integer order, real Mathieu functions.
MATEE	Evaluates the eigenvalues for the periodic Mathieu functions.
MATSE	Evaluates a sequence of odd, periodic, integer order, real Mathieu functions.

N

IERCD and N1RTY	Retrieves the error type set by the most recently called IMSL routine.
-----------------	------------------------------------------------------------------------

P

POCH	Evaluates a generalization of Pochhammer's symbol.
POCH1	Evaluates a generalization of Pochhammer's symbol starting from the first order.
POIDF	Evaluates the Poisson cumulative distribution function.

POIPR	Evaluates the Poisson probability density function.
PSI	Evaluates the derivative of the log gamma function.
PSI1	Evaluates the second derivative of the log gamma function.

R

RALDF	Evaluates the Rayleigh cumulative distribution function.
RALIN	Evaluates the inverse of the Rayleigh cumulative distribution function.
RALPR	Evaluates the Rayleigh probability density function.

S

SHI	Evaluates the hyperbolic sine integral.
SI	Evaluates the sine integral.
SINDG	Evaluates the sine for the argument in degrees.
SPENC	Evaluates a form of Spence's integral.

T

TAN	Evaluates $\tan z$ for complex z .
TDF	Evaluates the Student's t cumulative distribution function.
TIN	Evaluates the inverse of the Student's t cumulative distribution function.

TNDF	Evaluates the noncentral Student's <i>t</i> cumulative distribution function.
TNIN	Evaluates the inverse of the noncentral Student's <i>t</i> cumulative distribution function.
TNPR	This function evaluates the noncentral Student's <i>t</i> probability density function.
TPR	Evaluates the Student's <i>t</i> probability density function.

U

UMACH	Sets or Retrieves input or output device unit numbers.
UNDF	Evaluates the uniform cumulative distribution function.
UNDDF	Evaluates the discrete uniform cumulative distribution function.
UNDIN	Evaluates the inverse of the discrete uniform cumulative distribution function.
UN DPR	Evaluates the discrete uniform probability density function.
UNIN	Evaluates the inverse of the uniform cumulative distribution function.
UNPR	Evaluates the uniform probability density function.

W

WBLDF	Evaluates the Weibull cumulative distribution function
WBLIN	Evaluates the inverse of the Weibull cumulative distribution function.
WBLPR	Evaluates the Weibull probability density function.

Appendix B, References

Abramowitz and Stegun

Abramowitz, Milton, and Irene A. Stegun (editors) (1964), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, National Bureau of Standards, Washington.

Abramowitz, Milton, and Irene A. Stegun (editors) (1972), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, 10th Edition, US Government Printing Office, Washington, DC, Chapter 9.

Aird and Howell

Aird, Thomas J., and Byron W. Howell (1991), IMSL Technical Report 9103, IMSL, Houston.

Akima

Akima, H. (1970), A new method of interpolation and smooth curve fitting based on local procedures, *Journal of the ACM*, **17**, 589–602.

Barnett

Barnett, A.R. (1981), An algorithm for regular and irregular Coulomb and Bessel functions of real order to machine accuracy, *Computer Physics Communications*, **21**, 297–314.

Boisvert, Howe, Kahaner, and Springmann

Boisvert, Ronald F., Sally E. Howe, David K. Kahaner, and Jeanne L. Springmann (1990), *Guide to Available Mathematical Software*, NISTIR 90-4237, National Institute of Standards and Technology, Gaithersburg, Maryland.

Boisvert, Ronald F., Sally E. Howe, and David K. Kahaner (1985), GAMS: A framework for the management of scientific software, *ACM Transactions on Mathematical Software*, **11**, 313–355.

Bosten and Battiste

Bosten, Nancy E., and E.L. Battiste (1974b), Incomplete beta ratio, *Communications of the ACM*, **17**, 156–157.

Bosten, Nancy E., and E.L. Battiste (1974), Remark on algorithm 179, *Communications of the ACM*, **17**, 153.

Burgoyne

Burgoyne, F.D. (1963), Approximations to Kelvin functions, *Mathematics of Computation* **83**, 295–298.

Butler and Paoella

Butler, R. W., and M. S. Paoella (1999), *Calculating the Density and Distribution Function for the Singly and Doubly Noncentral F*, Preliminary Version, [Paoella.pdf](#), p.10, eq.(30) and ff.

Carlson

Carlson, B.C. (1979), Computing elliptic integrals by duplication, *Numerische Mathematik*, **33**, 1–16.

Carlson and Notis

Carlson, B.C., and E.M. Notis (1981), Algorithms for incomplete elliptic integrals, *ACM Transactions on Mathematical Software*, **7**, 398–403.

Cody

Cody, W.J. (1969) Performance testing of function subroutines, *Proceedings of the Spring Joint Computer Conference*, American Federation for Information Processing Societies Press, Montvale, New Jersey, 759–763.

Cody, W.J. (1983), Algorithm 597: A sequence of modified Bessel functions of the first kind, *ACM Transactions on Mathematical Software*, **9**, 242–245.

Cody et al.

Cody, W.J., R.M. Motley, and L.W. Fullerton (1976), The computation of real fractional order Bessel functions of the second kind, *Applied Mathematics Division Technical Memorandum No. 291*, Argonne National Laboratory, Argonne.

Conover

Conover, W.J. (1980), *Practical Nonparametric Statistics*, 2d ed., John Wiley & Sons, New York.

Cooper

Cooper, B.E. (1968), Algorithm AS4, An auxiliary function for distribution integrals, *Applied Statistics*, **17**, 190–192.

Eckhardt

Eckhardt, Ulrich (1977), A rational approximation to Weierstrass' P-function. II: The Lemniscatic case, *Computing*, **18**, 341–349.

Eckhardt, Ulrich (1980), Algorithm 549: Weierstrass' elliptic functions, *ACM Transactions on Mathematical Software*, **6**, 112–120.

Fabijonas et al.

B. R. Fabijonas, D. W. Lozier, and F. W. J. Olver Computation of Complex Airy Functions and Their Zeros Using Asymptotics and the Differential Equation, *ACM Transactions on Mathematical Software*, Vol. 30, No. 4, December 2004, 471–490.

Fox et al.

Fox, P.A., A.D. Hall, and N.L. Schryer (1978), The PORT mathematical subroutine library, *ACM Transactions on Mathematical Software*, 4, 104–126.

Gautschi

Gautschi, Walter (1964), Bessel functions of the first kind, *Communications of the ACM*, **7**, 187–198.

Gautschi, Walter (1969), Complex error function, *Communications of the ACM*, **12**, 635. Gautschi, Walter (1970), Efficient computation of the complex error function, *SIAM Journal on Mathematical Analysis*, **7**, 187–198.

Gautschi, Walter (1974), Algorithm 471: Exponential integrals, *Collected Algorithms from CACM*, 471.

Gautschi, Walter (1979), A computational procedure for the incomplete gamma function, *ACM Transactions on Mathematical Software*, **5**, 466–481.

Gautschi, Walter (1979), Algorithm 542: Incomplete gamma functions, *ACM Transactions on Mathematical Software*, **5**, 482–489.

Giles and Feng

Giles, David E. and Hui Feng. (2009). "Bias-Corrected Maximum Likelihood Estimation of the Parameters of the Generalized Pareto Distribution." Econometrics Working Paper EWP0902, Department of Economics, University of Victoria.

Gradshteyn and Ryzhik

Gradshteyn, I.S. and I.M. Ryzhik (1965), *Table of Integrals, Series, and Products*, (translated by Scripta Technica, Inc.), Academic Press, New York.

Hart et al.

Hart, John F., E.W. Cheney, Charles L. Lawson, Hans J. Maehly, Charles K. Mesztenyi, John R. Rice, Henry G. Thacher, Jr., and Christoph Witzgall (1968), *Computer Approximations*, John Wiley & Sons, New York.

Hill

Hill, G.W. (1970), Student's t -distribution, *Communications of the ACM*, **13**, 617–619.

Hodge

Hodge, D.B. (1972), *The calculation of the eigenvalues and eigenvectors of Mathieu's equation*, NASA Contractor Report, The Ohio State University, Columbus, Ohio.

Hosking, et al.

Hosking, J.R.M., Wallis, J.R., and E.F. Wood. (1985). "Estimation of the Generalized Extreme Value Distribution by the Method of Probability-Weighted Moments." *Technometrics*. Vol 27. No. 3. pp 251-261.

Hosking and Wallis

Hosking, J.R.M. and J.R. Wallis. (1987). "Parameter and Quantile Estimation for the Generalized Pareto Distribution." *Technometrics*. Vol 29. No. 3. pp 339-349.

IEEE

ANSI/IEEE Std 754-1985 (1985), *IEEE Standard for Binary Floating-Point Arithmetic*, The IEEE, Inc., New York.

Johnson and Kotz

Johnson, Norman L., and Samuel Kotz (1969), *Discrete Distributions*, Houghton Mifflin Company, Boston.

Johnson, Norman L., and Samuel Kotz (1970a), *Continuous Distributions-1*, John Wiley & Sons, New York.

Johnson, Norman L., and Samuel Kotz (1970b), *Continuous Distributions-2*, John Wiley & Sons, New York.

Kendall and Stuart

Kendall, Maurice G., and Alan Stuart (1979), *The Advanced Theory of Statistics, Volume 2: Inference and Relationship*, 4th ed., Oxford University Press, New York.

Kim and Jennrich

Kim, P.J., and Jennrich, R.I. (1973), Tables of the exact sampling distribution of the two sample Kolmogorov-Smirnov criterion D_{mn} ($m < n$), in *Selected Tables in Mathematical Statistics, Volume 1*, (edited by H.L. Harter and D.B. Owen), American Mathematical Society, Providence, Rhode Island.

Kinnucan and Kuki

Kinnucan, P., and H. Kuki (1968), *A single precision inverse error function subroutine*, Computation Center, University of Chicago.

Luke

Luke, Y.L. (1969), *The Special Function and their Approximations*, Volume 1, Academic Press, 34.

Majumder and Bhattacharjee

Majumder, K. L., and G. P. Bhattacharjee (1973), *The Incomplete Beta Integral*, Algorithm AS 63:Journal of the Royal Statistical Society. Series C (Applied Statistics), Vol. 22, No. 3, 409-411, Blackwell Publishing for the Royal Statistical Society, <http://www.jstor.org/stable/2346797>.

NATS FUNPACK

NATS (National Activity to Test Software) FUNPACK (1976), Argonne National Laboratory, Argonne Code Center, Argonne.

Olver and Sookne

Olver, F.W.J., and D.J. Sookne (1972), A note on the backward recurrence algorithms, *Mathematics of Computation*, **26**, 941–947.

Owen

Owen, D.B. (1962), *Handbook of Statistical Tables*, Addison-Wesley Publishing Company, Reading, Mass.

Owen, D.B. (1965), A special case of the bivariate non-central t-distribution, *Biometrika*, **52**, 437–446.

Pennisi

Pennisi, L.L. (1963), *Elements of Complex Variables*, Holt, Rinehart and Winston, New York.

Skovgaard

Skovgaard, Ove (1975), Remark on algorithm 236, *ACM Transactions on Mathematical Software*, **1**, 282–284.

Sookne

Sookne, D.J. (1973a), Bessel functions I and J of complex argument and integer order, *National Bureau of Standards Journal of Research B*, **77B**, 111–114.

Sookne, D.J. (1973b), Bessel functions of real argument and integer order, *National Bureau of Standards Journal of Research B*, **77A**, 125–132.

Stephens

Stephens, M.A., and D'Agostino, R.B (1986), Tests based on EDF statistics., *Goodness-of-Fit Techniques*. Marcel Dekker, New York.

Strecok

Strecok, Anthony J. (1968), On the calculation of the inverse of the error function, *Mathematics of Computation*, **22**, 144–158.

Temme

Temme, N. M. (1975), On the numerical evaluation of the modified Bessel function of the third kind, *Journal of Computational Physics*, **19**, 324–337.

Thompson and Barnett

Thompson, I.J. and A.R. Barnett (1987), Modified Bessel functions $I_{\nu}(z)$ and $K_{\nu}(z)$ of real order and complex argument, to selected accuracy, *Computer Physics Communications*, **47**, 245–257.

Yousif and Melka

Yousif, Hashim A., and Richard Melka (1997), Bessel function of the first kind with complex argument, *Computer Physics Communications*, **vol. 106, no. 3**, 199–206.

Yousif, Hashim A., and Richard Melka (2003), Computing Bessel functions of the second kind in extreme parameter regimes, *Computer Physics Communications*, **151**, 25–34.

Index

A

Airy function 230, 246
derivative 234, 250
exponentially scaled 238
derivative 242
second kind 232, 248
derivative 236, 252
exponentially scaled 240
exponentially scaled
derivative 244
arguments, optional
subprogram 12

B

Bessel functions
first kind
integer order 173
order one 150
order zero 148
real order 179, 191
modified
exponentially scaled 165,
167, 169, 171, 185,
189
first kind, integer
order 176
first kind, nonnegative real
order 185
first kind, order one 158,
167
first kind, order zero 156,
165
first kind, real order 183,
197
second kind, fractional
order 189
second kind, order
one 163, 171
second kind, order
zero 160, 169
second kind, real
order 199
third kind, fractional
order 187

second kind
order one 154
order zero 152
real nonnegative
order 181
real order 194

beta distribution function 346

beta functions
complete 88, 115
natural logarithm 118
incomplete 121

beta probability density 348

beta probability distribution
function 343

binomial coefficient 87

binomial distribution function 296

binomial probability function 298

bivariate normal distribution
function 359

C

Cauchy principal value 62, 64

characteristic values 455

Chebyshev series 468, 474

chi-squared distribution
function 361, 364, 368

chi-squared probability
density 366

complex numbers
evaluating 17

continuous data 324, 327

cosine
arc
hyperbolic 55
complex 41
hyperbolic 49
in degrees 37
integrals 73, 75
hyperbolic 79, 81

cotangent
evaluating 32

cube roots

evaluating 19
cumulative distribution
function 443
cumulative distribution functions
(CDF) 291

D

Dawson's function
evaluating 140
discrete uniform cumulative
probability 318
discrete uniform cumulative prob-
ability distribution 318
discrete uniform probability
density 322
discrete uniform random
variable 320
distribution functions 291
cumulative (CDF) 291
general continuous cumulative
inverse 447
double precision 1
DOUBLE PRECISION types 6

E

eigenvalues 455
elementary functions 3
elliptic integrals
complete 258
second kind 260
first kind
Carlson's incomplete 262
second kind
Carlson's incomplete 264
third kind
Carlson's incomplete 266
error functions 124, 125
complementary 127
complex scaled 132
exponentially scaled 130
inverse 137
inverse 134

- error handling 481
error-handling 9, 13
errors
 alert 295
 informational 477
 note 295
 severity level 13
 terminal 295, 477
 warning 295
exponential cumulative probability distribution 376
exponential functions
 first order 21
exponential integrals 61, 62, 64
 of integer order 66
exponential probability density 380
extreme value cumulative probability distribution 382
extreme value probability density 386
- F**
F distribution function 388, 391
factorial 85
Fresnel integrals 124
 cosine 142
 sine 144
- G**
gamma distribution function 404, 407
gamma distributions
 standard 291
gamma functions 84
 complete 89
 incomplete 99
 complementary 102
 Tricomi form 104
 logarithmic derivative 106, 109
 reciprocal 92
gamma probability density 409
general continuous cumulative distribution function 451
Geometric
 inverse of the geometric cumulative probability distribution 303
 geometric cumulative probability distribution 301
 geometric probability density 305
 getting started 2, 11
- H**
hyperbolic functions 3
hypergeometric distribution function 307
hypergeometric probability function 310
- I**
INTEGER types 6
inverse of the exponential cumulative probability 378
inverse of the geometric cumulative probability distribution 303
inverse of the lognormal cumulative probability distribution 332
inverse of the Rayleigh cumulative probability distribution 413
inverse of the uniform cumulative probability distribution 433
inverse of the Weibull cumulative probability distribution 439
- J**
Jacobi elliptic function 280, 283, 286
- K**
Kelvin function
 first kind
 order one 223
 order zero 205, 207, 213, 215
 second kind
 order one 225, 227
 order zero 209, 211, 217, 219
Kolmogorov-Smirnov goodness of fit 324, 327
- L**
library subprograms 8
logarithmic integrals 68
logarithms
 complex
 common 23
 for gamma functions 94, 97
 natural 25, 118
lognormal cumulative probability distribution 330
lognormal probability density 334
- M**
machine-dependent constants 482
Mathieu functions
 even 459
 integer order 459, 463
 odd 463
 periodic 456, 459, 463
 real 459, 463
- N**
naming conventions 6
noncentral chi-squared function 371
normal probability density 341
- O**
optional argument 12
optional data 11
optional subprogram arguments 12
ordinates of the density 443
orthogonal series 472
overflow 9, 25
- P**
Pochhammer's symbol 111, 113, 468
Poisson distribution function 313
Poisson probability function 315
printing 480
printing results 14
probability density function (PDF) 292
probability functions 291
- R**
Rayleigh cumulative probability

distribution 411, 412
Rayleigh probability density 415
REAL types 6
required arguments 12

S

sine
 arc
 hyperbolic 53
 complex
 arc 39
 hyperbolic 47
 in degrees 35
 integrals 71
 hyperbolic 77
single precision 1
Spence's integral 470
standard normal (Gaussian) distribution function 336, 339
Student's t distribution
 function 417, 420, 424, 427
subprograms
 library 8
 optional arguments 12

T

tangent
 arc
 hyperbolic 57
 complex 30
 arc 43
 arc of a ratio 45
 hyperbolic 51
trigonometric functions 3

U

underflow 9
uniform cumulative probability distribution 431
uniform probability density 435
user interface 1
using library subprograms 8

W

Weibull cumulative probability distribution 437, 438
Weibull cumulative probability distribution function 437

Weibull probability density 441
Weibull random variable 439
Weierstrass' function
 equianharmonic case 276, 278
 lemniscatic case 272, 274

