Advanced Math Library Contents

Introduction
Higher mathematical functions
Statistics and data reduction
Signal processing
Numerical analysis
System Requirements
Installing the HTBasic Math Library
Loading the Subroutines
Function Reference

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Copyright © 2003 TransEra

Introduction

The HTBasic Math Library is a collection of subroutines that give users of the HTBasic programming language access to fast versions of higher mathematical and signal processing functions. Most of these routines are compiled, so they run at a much higher speed than equivalent BASIC subroutines.

The routines are meant to be incorporated into user BASIC programs to enhance their speed, and to save the user's writing the subroutines himself.

The subroutines included in the HTBasic Math Library include:

Higher mathematical functions

- Cylindrical and spherical Bessel and Hankel functions of real arguments of integer and half-integer order.
- Airy and Kelvin functions of real arguments.
- Error function and complementary error function of real and complex arguments, and Dawson's integral.
- Elliptical integrals of real arguments.
- Fresnel integrals.
- Exponential, log, sine, and cosine integrals of real arguments.
- Gamma and beta functions of real and complex arguments.
- Incomplete gamma and beta functions of real arguments.
- LeGendre, Hermite, and Chebyshev polynomials of real arguments and integer orders.

Statistics and data reduction

- Probability density functions and probability integrals for many probability distributions.
- Mean, median, standard deviation, and variance of sets of data.
- Curve fitting using both linear regression and higher-order polynomial functions.
- Polar/rectangular conversion of sequences of complex numbers.

Signal processing

- Fourier transforms and inverse Fourier transforms of both real and complex sequences.
- Digital filtering, correlation, convolution, autocorrelation, and power spectral density of sequences of data.
- Windowing using cosine, triangular, and Bessel windows.
- Built-in waveforms.

Numerical analysis

- Solutions to linear systems of equations having both real and complex coefficients.
- Polynomial evaluation.
- Numerical integration.
- Roots of equations of the form $f(x) = 0$.
- Derivatives and antiderivatives of polynomials.

System Requirements

The HTBasic Math Library requires HTBasic 386 release 3.2 or above, a computer having a 386 processor with an 387 numeric coprocessor (or compatible) or a 486 processor and at least 1.5 Mbytes of memory.

Installing the HTBasic Math Library

To install the Math Library Component of the HTBasic Legacy Workshop, first place the distribution CD-ROM into the CD drive. If you are in a Windows environment, the CD will autoplay and will give you a graphical setup menu to guide you through the installation. If you are in a DOS environment, at the root of the CD type SETUP.

Follow the instructions in the installation program to install the Math Library Component of HTBasic Legacy Workshop to your hard drive. You will be given an opportunity to choose which components to install.

During installation you will be asked to input your unique serial number. This eleven digit number can be found on your CD jewel case. Correct input of this number is required to complete installation. This number will be stored on your system.

If you are installing from windows, the install program will create a program group and icons for running the Workshop components.

HTBasic Legacy Workshop is distributed on CD-ROM. If you need to install to a machine with out a CD-ROM, a make disk utility is provided on the CD-ROM. From the root of the CD type DISKETTE, and follow the instructions in the make disk program.Before installing the HTBasic Math Library, be sure that HTBasic is properly installed on your computer, as described in its instruction manual.

Loading the Subroutines

The subroutines in this library are grouped into *modules*. A module is a group of subroutines that share some common program sections. When a program loads a routine from a module, HTBasic loads the entire module. For example, the *FNAi* Airy function is in the module AIRY. When this module is loaded, all the other functions in the module, *FNAie*, *FNBi*, and *FNBie*, become available to the program also.

As shipped, the diskette containing the Math Library contains two copies of each module. One copy is in a file having the same name as the module, with the ".HTS" extension, and one is in the large file MATHLIB.HTS, which contains all the modules. This is done for convenience only; only one copy of each module needs to reside on a computer for the library to be usable. The Loading topic in each manual entry in the following section gives two or more ways to load the module containing each routine, one from the file having just the module containing the routine and one or more from the MATHLIB.HTS file.

The ordinary way to load a subroutine into an HTBasic program is to first enter the program into the computer from the keyboard or load it from a file. After this is done, type the appropriate form of the HTBasic LOADSUB command on the computer, as explained later in this section. The module containing the subroutine then becomes a part of the program in the computer. If the subroutine is a compiled subroutine (as are most of those in the math library), it will be displayed in program listings as a single CSUB line listing its name and arguments. After the subroutine is loaded, the program may be run or modified as needed. After the subroutine is loaded, the program should be saved to disk using the HTBasic RE-STORE command. After this is done, the subroutine is part of the program and will be loaded together with the program when the file is loaded into HTBasic. Using the HTBasic SAVE command to save the program in an ASCII file will remove any compiled subroutines from the program.

One or more modules may be loaded under program control as well, by placing the appropriate LOADSUB line in the program. When this is done, the program makes provisions for not Loading the modules a second time if it is run more than once.

If a module is loaded more than once into a program, it will still function, but the memory used by the additional copies of the module is wasted until HTBasic terminates.

There are two sets of the math library subroutines on the distribution diskette. One set of routines is in files having the same names as the modules they contain, with the ".HTS" extension. For example, the *Airy* module is contained in the file AIRY.HTS. Another set of the math library functions is in the file MATHLIB.HTS. This file contains all the functions in a single file. The Loading topic in each manual entry in the following section gives two or more ways to load the module containing each routine, one from the file having just the module containing the routine and one or more from the MATHLIB.HTS file. Loading the subroutines from the MATHLIB.HTS file is more automatic; the HTBasic interpreter will search the program for unloaded subroutines and functions and try to find them in MATHLIB.HTS. It will then search the functions and subroutines it loaded from MATHLIB.HTS to see if they in turn call other functions and subroutines, and, if so, will search MATHLIB.HTS for those also. This procedure can be quite slow, as MATHLIB.HTS contains many subroutines. Loading from the separate files is faster, but requires that the user know which files contain the modules it needs.

Although there are many ways to organize files on a hard disk drive, there are two methods widely used with HTBasic installations. The first of these is to place all the HTBasic program files and user program files in a single directory, usually named C:\HTB386. HTBasic is then always run from that directory. The documentation in this manual assumes this organization of the disk. In particular, the Loading topic in each entry in the manual contains a file name without any directory name, implying that the file containing the subroutine is in the current directory.

Another way to organize the hard disk is to put all the HTBasic files in a single directory but to put the user programs in one or more other directories. If this is done, add the appropriate directory name to the file name given in the Loading topic in the manual entry. For example, to load the module containing the Airy function *Ai*, the manual says to use the statement

```
LOADSUB ALL FROM "AIRY.HTS"
```

or

```
LOADSUB FROM "MATHLIB.HTS"
```

to load the module containing the *FNAi* function. If the directory containing the file AIRY.HTS or MATHLIB.HTS is C:\HTB386 and this is not the current directory when a program needing the AIRY module is run, use the statement

```
LOADSUB ALL FROM "C:\HTB386\MATHLIB\AIRY.HTS"
```

or

```
LOADSUB FROM "C:\HTB386\MATHLIB\MATHLIB.HTS"
```

instead of that listed in the Loading topic.

In those situations where the program is saved as an ASCII file, for example, when using a PC text editor to develop an HTBasic program, the appropriate LOADSUB command may be enclosed in an IF block to ensure that the math subroutines are loaded only once, as shown in following program section:

```
10 COM /Math/ INTEGER Loaded
20 IF Loaded=0 THEN
30 LOADSUB FROM "MATHLIB.HTS"
40 Loaded=1
50 END IF
```

This program section works because COM variables such as *Loaded* are initialized to zero when created and after that time they retain their former values.

Many of the functions in the Math Library use helper routines. The names of these routines begin with "F_". A program incorporating functions from the Math Library should avoid definining subroutines with names that begin with "F_". This restriction may be eliminated in future versions of the Math Library.

Function Reference

In this portion of the manual, each function or subroutine is listed in alphabetical order. In the case of functions, the "FN" at the beginning of the function name is not used in the alphabetization. Following the name and a one-line Description of the routine is a Loading topic that tells the program statements to use in order to load the subroutine for use in a program, as discussed earlier. Only one of these statements should be used. The next topic, Usage, describes the types of variables used to CALL, or run, the subroutine. Arrays are shown with "(*)" in place of their actual dimensions. An actual program declaring an array would contain a number in parenthesis instead of the asterisk shown in the Usage topic. Also, the variable type declarations, such as INTEGER, REAL, or COMPLEX, generally appear at the beginning of the program, while the CALL statement appears later in the program, even though they are shown together in the manual entries.

The main portion of each manual entry is the Description topic, which describes what each subroutine does. This is followed by an Errors topic, which explains any HTBasic Errors that the subroutine can cause and why they happen. There may also be a See Also topic listing related subroutines, a Notes topic giving additional details about the subroutine, and a graph of some of the values returned by the subroutine.

Many function descriptions mention the value MAXREAL. As explained in the HTBasic manual, this is the largest value that can be represented in the computer's real number notation. In computers that use the IEEE double precision floating point standard, this value is about 1.7 × 10309.

Ai

Airy function of the first kind.

Loading LOADSUB ALL FROM "AIRY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNAi(X)

Description

FNAi returns the value of the Airy function of the first kind of *x*. Note that sometimes Airy functions are written with an order, as in Ai3(*x*). In this notation, the function *FNAi* returns the value of Ai0(*x*). Ai(*x*) is defined for any real value *x*.

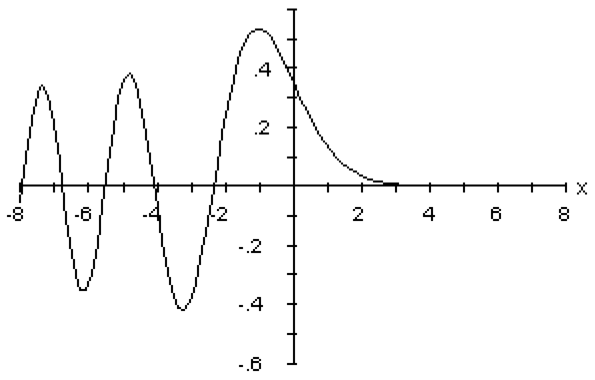
Errors

FNAi causes a BASIC error if its argument is not of type REAL.

See Also

Aie, Ai_Bi, Bi

Ai(*x*)



Aie

Scaled Airy function of the first kind.

Loading

```
LOADSUB ALL FROM "AIRY.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
REAL X,Y
Y=FNAie(X)
```

Description

FNAie returns the value of *e*2*xx*/3Ai(*x*). *X* must be positive or zero. *Ai* is the Airy function of the first kind. This subroutine is useful for determining the value of Ai(*x*) for large positive values of *x*, where the related *FNAi* function returns values near zero.

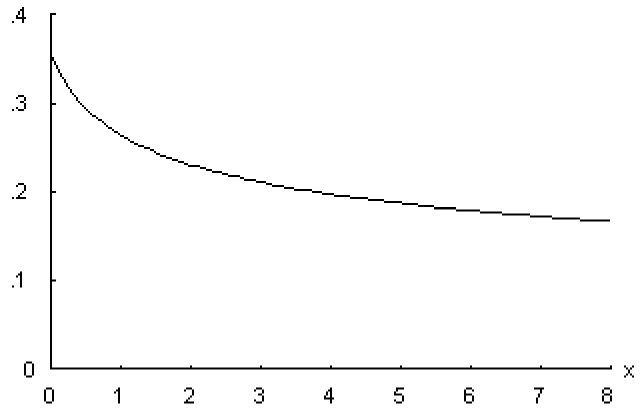
Errors

FNAie causes a BASIC error if its argument is not of type REAL or if *x* is negative.

See Also

Ai

*e*2*xx*/3Ai(*x*)



Ai_bi

Airy functions of the first and second kinds.

Loading LOADSUB ALL FROM "AIRY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNAi_bi(X)
```

Description

FNAi_bi returns the values of the Airy functions of the first and second kinds of the real value *x*. The real part of the return value *C* is the value of Ai(*x*) and the imaginary part of *C* is the value of Bi(*x*). Although it is defined for all real values of *x*, this function is usually used with negative values of *x*, since the Airy functions of the first and second kinds behave somewhat like damped cosine and sine functions in this region.

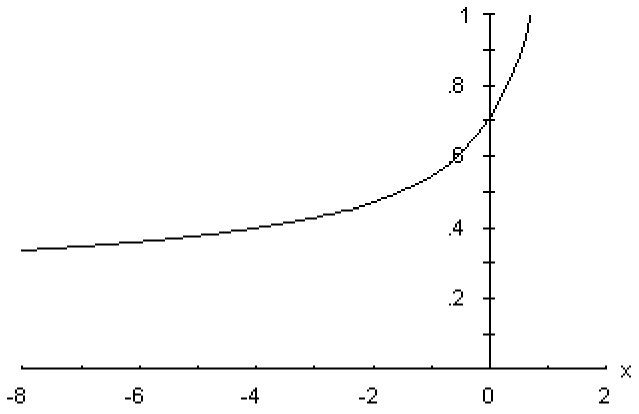
Errors

FNAi_bi causes a BASIC error if its first argument is not of type COMPLEX or its second argument is not of type REAL. It also causes a BASIC error if the value of the imaginary component of the value returned (Bi) would be larger than MAXREAL, the largest number representable.

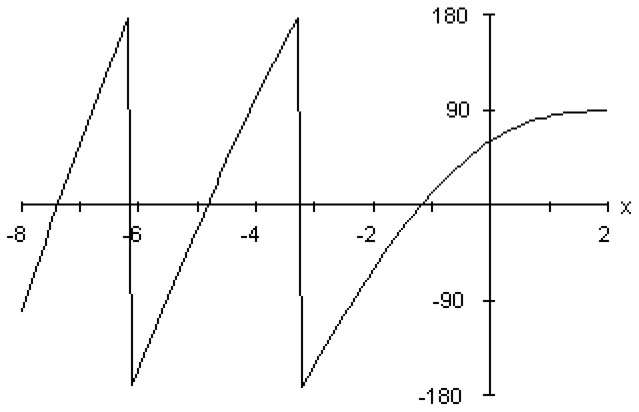
See Also

Ai, Bi

Ai(x) + iBi(x)



Arg[Ai(*x*) + *i*Bi(*x*)], degrees



Autocorrelate

Autocorrelation of a sequence.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB Autocorrelate FROM "MATHLIB.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage INTEGER Logn
REAL A(*),C(*)
CALL Autocorrelate(Logn,A(*),C(*))

Description

Autocorrelate calculates the autocorrelation of the sequence in the array *A* and returns the result in the array *C*. *Logn* is the base-2 logarithm of the number of points in the sequence to be correlated. The array *A* must have at least 2*Logn* elements and the array *C* must have at least 2*Logn*+1 elements; if they have more than the required number of elements, the extra elements are ignored and unmodified. The number of elements in *A* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

The number of elements required in *C* for each value of *Logn* is twice the value given in the table above.

The autocorrelation is a measure of a function's similarity to itself as the abscissa is shifted. If *a*(*x*) is the function being tested and the interval of interest is *x* (0,*T*), and if *a* is zero outside this interval, then the autocorrelation, *c*(*x*), is defined by the relation

$$c(x) = \frac{1}{T} \int_0^T a(t)\, a(t-x)\, dt.$$

Note that, while *a*(*x*) is nonzero on the interval *x* (0,*T*), *c*(*x*) is nonzero on the interval x (-*T*,*T*).

If the function *a* is only defined at regularly-spaced discrete points *x* = (*k*+½)*T*/*N*, *k* = {0,1,2,...,*N*-1}, the integration can be approximated by assuming that *a*(*x*) is constant and equal to *a*([*k*+½]*T*/*N*) between *x* = *kT*/*N* and *x* = (*k*+1)*T*/*N*. The expression above can then be replaced by

$$c([k+½]T/N) = \frac{1}{N}\sum_{m=0}^{N-1} a([m+½]T/N)\, a([m-k+½]T/N).$$

In this case, *c*([*k*+½]*T*/*N*) is defined for *k* = {-*N*,-(*N*-1),...,0,...,*N*-1}.

Autocorrelate returns the values of *c*([*k*+½]*T*/*N*) in the array *C*. The first *N* elements of *C* represent *k* = {0,1,...,*N*-1} and the last *N* elements in *C* represent *k* = {-*N*,-(*N*-1),...,-1}.

Errors

Autocorrelate causes a BASIC error if its arguments are not of the types shown in the **USAGE** section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *C* is smaller than the values described above.

See Also

Correlate, Convolve, Fft, Power_spectrum, Fft

Be

Complex Kelvin function of the first kind of a real argument.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNBe(X)
```

Description

FNBe returns the values of the real and imaginary Kelvin functions of the first kind of the value *x*. The real part of the value returned is the value of ber(*x*) and the imaginary part is the value of bei(*x*). Although ber(*x*) and bei(*x*) are defined for all real values of *x*, large positive values of *x* may produce results greater than MAXREAL, the largest value representable.

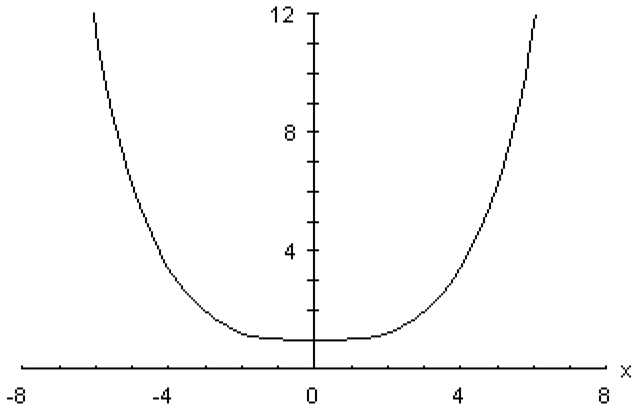
Errors

FNBe causes a BASIC error if it argument is not of type REAL. It also causes a BASIC error if the value of either component of the value returned would be larger than MAXREAL.

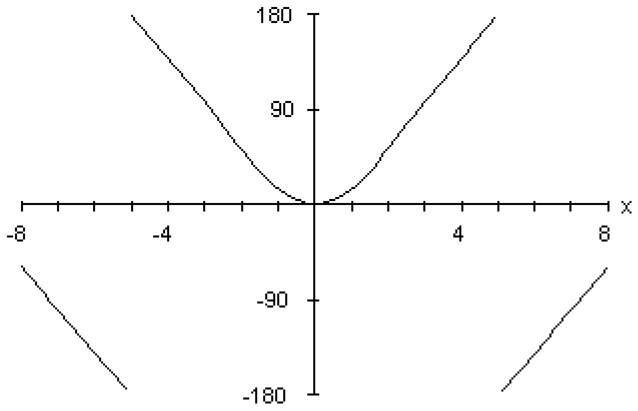
See Also

Ber, Bei, Ke

ber(*x*) + *i*bei(*x*)



Arg[ber(*x*) + *i*bei(*x*)], degrees


180
90
-90
-180
x
-8
-4
4
8

Bei

Imaginary Kelvin function of the first kind of a real argument.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNBei(X)

Description

FNBei returns the value of the imaginary Kelvin function of the first kind of the value *x*. Although bei(*x*) is defined for all real values of *x*, large positive or negative values of *x* may produce results greater in magnitude than MAXREAL, the largest value representable. Note that sometimes Kelvin functions are written with an order, as in bei3(*x*). In this notation, the function *FNBei* returns the value of bei0(*x*).

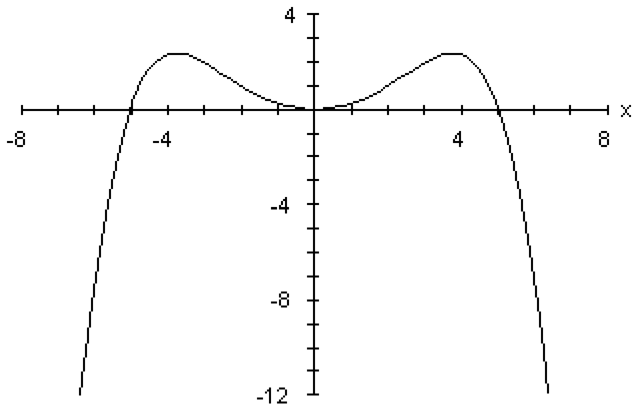
Errors

FNBei causes a BASIC error if its argument is not of type REAL or if the value returned would be larger in magnitude than MAXREAL.

See Also

Be, Ber

bei(*x*)



Ber

Real Kelvin function of the first kind of a real argument.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNBer(X)
```

Description

FNBer returns the value of the real Kelvin function of the first kind of the value *x*. Although ber(*x*) is defined for all real values of *x*, large positive or negative values of *x* may produce results greater in magnitude than MAXREAL, the largest value representable. Note that sometimes Kelvin functions are written with an order, as in ber3(*x*). In this notation, the function *FNBer* returns the value of ber0(*x*).

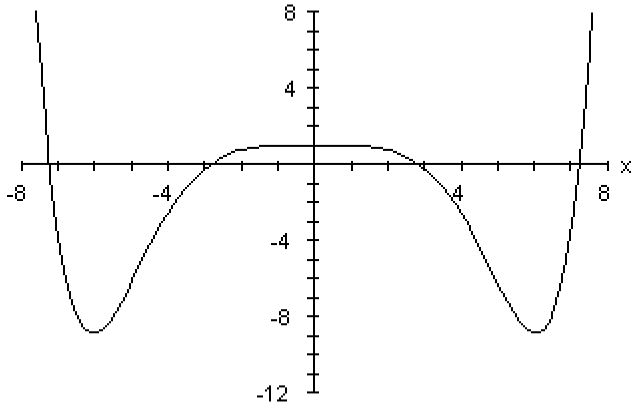
Errors

FNBer causes a BASIC error if its argument is not of type REAL or if the value returned would be larger in magnitude than MAXREAL.

See Also

Be, Bei

ber(*x*)



Beta

Beta function.

Loading LOADSUB ALL FROM "GAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL A,B,Y
Y=FNBeta(A,B)
```

Description

FNBeta returns the value of *B*(*a*,*b*), where *B* represents the *beta* function. *B*(*a*,*b*) is defined as (*a*)(*b*)/(*a*+*b*) (see *Gamma*). *B*(*a*,*b*) is only defined for *a* > 0 and *b* > 0.

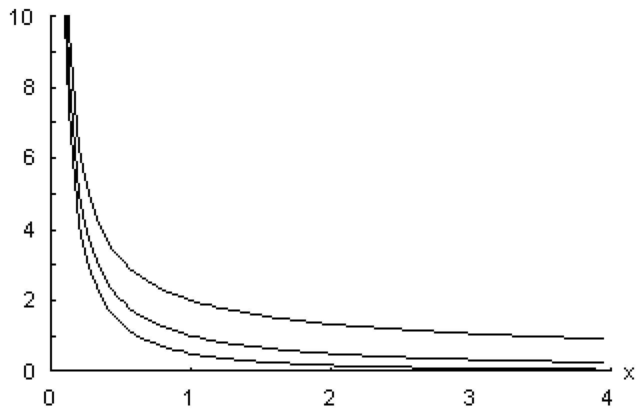
Errors

FNBeta causes a BASIC error if its arguments are not all of type REAL or if either *a* or *b* is negative or zero.

See Also

Cbeta, Gamma, Logbeta

B(*a*,*b*)



Bi

Airy function of the second kind.

Loading LOADSUB ALL FROM "AIRY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNBi(X)
```

Description

FNBi returns the value of the Airy function of the second kind of *x*. Note that sometimes Airy functions are written with an order, as in Bi3(*x*). In this notation, the function *FNBi* returns the value of Bi0(*x*). Although Bi(*x*) is defined for all real values of *x*, large positive values of *x* may produce results greater than MAXREAL, the largest value representable.

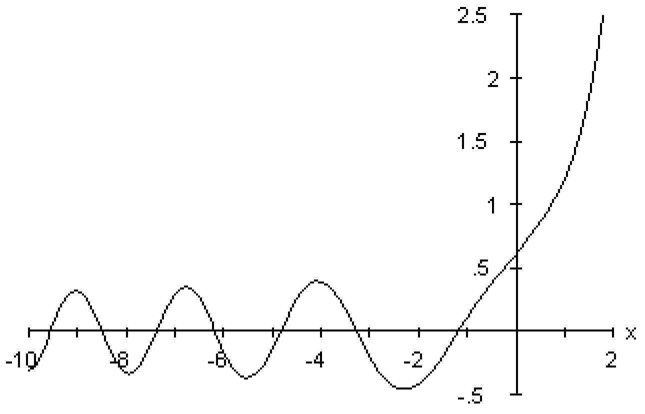
Errors

FNBi causes a BASIC error if its argument is not of type REAL or if its result would be larger than MAXREAL.

See Also

Ai, Ai_Bi, Bie

Bi(*x*)



Bie

Scaled Airy function of the second kind.

Loading LOADSUB ALL FROM "AIRY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNBie(X)
```

Description

FNBie returns the value of *e*-2*xx*/3Bi(*x*). *X* must be positive or zero. *Bi* is the Airy function of the second kind. This subroutine is useful for determining the value of Bi(*x*) for large positive values of *x*, where the related *Bi* function returns large values or produces BASIC **Errors** for values too large to represent.

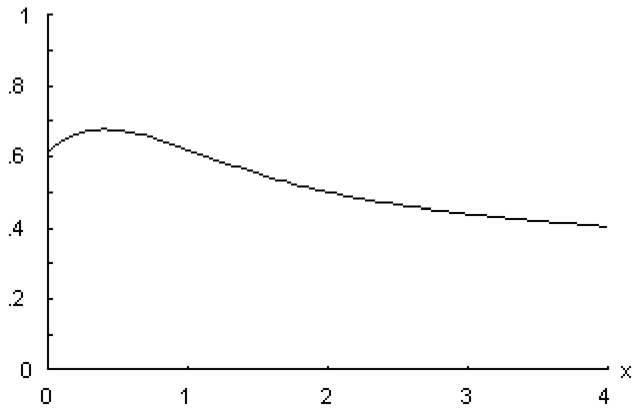
Errors

FNBie causes a BASIC error if its argument is not of type REAL or if *x* is negative.

See Also

Bi

e-2*xx*/3Bi(*x*)



Binom

Binomial coefficients.

Loading LOADSUB ALL FROM "FACT.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER M,N
REAL Y
Y=FNBinom(N,M)
```

Description

FNBinom returns the binomial coefficient

$$\binom{n}{m}.$$

In terms of factorials,

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

Errors

FNBinom causes a BASIC error if its arguments are not of type INTEGER or if the value returned would have a magnitude larger than MAXREAL, the largest value representable.

See Also

Fact

C

Fresnel cosine integral of a real argument.

Loading LOADSUB ALL FROM "FRESNEL.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNC(X)

Description

FNC returns the value of the Fresnel cosine integral of *x*, *C*(*x*). *C*(*x*) is defined by the relation

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

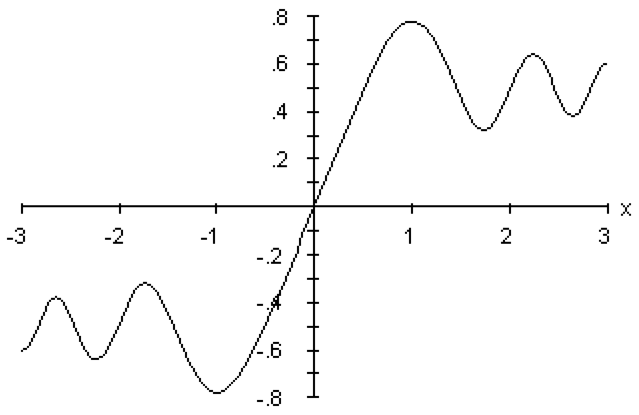
Errors

FNC causes a BASIC error if its argument is not of type REAL.

See Also

S

C(*x*)



Cbeta

Complex beta function of a complex argument.

Loading

LOADSUB ALL FROM "CGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
COMPLEX A,B,C
C=FNCbeta(A,B)
```

Description

FNCbeta returns the value of the beta function of the complex values *a* and *b*, *B*(*a*,*b*). *B*(*a*,*b*) is defined as (*a*)(*b*)/(*a*+*b*) (see *Gamma*). *B*(*a*,*b*) is only defined for *e*(*a*) > 0 and *e*(*b*) > 0.

Errors

FNCbeta causes a BASIC error if its arguments are not all of type COMPLEX, or if the real part of either *a* or *b* is negative or zero.

See Also

Beta, Cgamma, Clogbeta

Cdigamma

Complex digamma function of a complex argument.

Loading

```
LOADSUB ALL FROM "CDIGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
COMPLEX Z,C
C=Cdigamma(Z)
```

Description

FNCdigamma returns the value of the digamma function (sometimes called the *psi* function) of the complex value z, $\Psi(z)$. The value of $\Psi(z)$ approaches ± as z approaches a real negative integer value or zero.

Errors

FNCdigamma causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of either the real or imaginary component of $\Psi(z)$ exceeds MAXREAL, the largest number representable.

See Also

Cgamma, Digamma

Cerf

Complex error function of a complex argument.

Loading

LOADSUB ALL FROM "CERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

COMPLEX Z,C
C=FNCerf(Z)

Description

FNCerf returns the value of the error function of the complex value *z*, erf(*z*). The imaginary part of the value of erf(*z*) approaches ± if the real part of *z* is zero and the magnitude of the imaginary part of *z* becomes large.

Errors

FNCerf causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of either the real or imaginary component of erf(*z*) exceeds MAXREAL, the largest number representable.

See Also

Cerfc, Erf

Cerfc

Complex complementary error function of a complex argument.

Loading LOADSUB ALL FROM "CERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage COMPLEX Z,C
C=FNCerfc(Z)

Description

FNCerfc returns the value of the complementary error function of the complex value *z*, erfc(*z*). The imaginary part of the value of erfc(*z*) approaches if the real part of *z* is zero and the magnitude of the imaginary part of *z* becomes large.

Erfc(*z*) is related to the error function returned by the *FNCerf* function, erf(*z*), by the expression

$$\mathrm{erfc}(z) = 1 - \mathrm{erf}(z).$$

Errors

FNCerfc causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of erfc(*z*) exceeds MAXREAL, the largest number representable.

See Also

Cerf, Erf

Cfft

Complex discrete Fourier transform.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB Cfft FROM "MATHLIB.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER Logn
COMPLEX A(*),C(*)
CALL Cfft(Logn,A(*),C(*))
```

Description

Cfft calculates the discrete Fourier transform of the sequence in the array *A* and stores the result in the array *C*. *Logn* is the base-2 log of the number of points in the sequences. The arrays *A* and *C* must contain at least 2*Logn* elements; if they have more than this number of elements, the extra elements are ignored and unmodified. The number of elements denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

If the values in *A* are taken to be values of a continuous complex signal, *a*(*t*), sampled at constant intervals of *T* (time, distance, or whatever units apply), and if the signal sampled contained no terms at or above the frequency 1/2*T*, then the coefficients in the array *C* are the coefficients of the complex Fourier series that describes *a*(*t*). *A*(*t*) can be reconstructed from the elements of *C* through the following formula:

$$a(t) = \sum_{k=-N/2}^{N/2-1} c_k e^{-i\frac{2\pi kt}{NT}}$$

where

$$N = 2^{Logn}.$$

The first *N*/2 elements in the array *C* represent *k* = {0,1,...,*N*/2-1} and the last *N*/2 elements in *C* represent k = {-*N*/2,-(*N*/2-1),...,-1}.

If the signal *a*(*t*) contains components at or above the frequency 1/2*T*, the situation is complicated by *aliasing*, which is explained in most signal processing textbooks.

Some of the more common operations done using discrete Fourier transforms, such as convolution, correlation, and filtering, are available as separate CSUBs; see the entries for *Autocorrelate*, *Convolve*, *Correlate*, *Filter*, *Rfilter*, and *Power_Spectrum* for details on their use. The inverse of *Cfft* is computed by the *Icfft* subroutine. A discrete Fourier transform for real sequences is done by the *Fft* subroutine. *Fft* is approximately twice as fast as *Cfft* for a given real sequence and uses half the storage.

Errors

Cfft causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *C* is smaller than 2*Logn*.

Examples

The manual entry for the *Fft* routine contains two examples that explain some of the uses and limitations of the discrete Fourier transform. Although the programs in the examples use the real discrete Fourier transform calculated by the *Fft* subroutine, the principles explained there are valid for the complex discrete Fourier transform also.

See Also

Convolve, Correlate, Fft, Filter, Icfft, Power_spectrum, Rfilter

Note

Some discrete Fourier transforms view the input array as a series of multipliers of Dirac delta operators. The values output from such transforms are the same as those output by *Cfft* except that each value is multiplied by *N*.

Cgamma

Complex gamma function of a complex argument.

Loading LOADSUB ALL FROM "CGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
COMPLEX Z,C
C=FNCgamma(Z)
```

Description

FNCgamma returns the value of the gamma function of the complex value *z*, $\Gamma(z)$. The value of $\Gamma(z)$ approaches ± as *z* approaches a real negative integer or zero. The gamma function is related to the factorial of a nonnegative integer *n*, by the relation

$$n! = \Gamma(n+1).$$

This relation is often used to define a factorial function for all complex numbers except negative real integers and zero, by replacing *n* in the above expression with a complex variable.

Errors

FNCgamma causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of either the real or the imaginary component of $\Gamma(z)$ exceeds MAXREAL, the largest number representable.

See Also

Cloggamma, Gamma

Chi

Hyperbolic cosine integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNChi(X)

Description

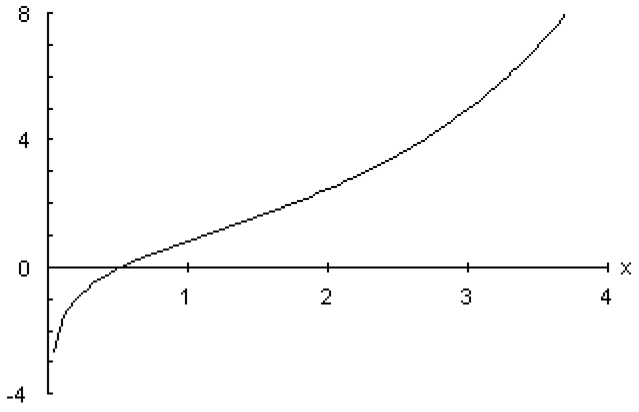
FNChi returns the value of the hyperbolic cosine integral of *x*, Chi(*x*). Chi(*x*) is defined by the relation

$$\mathrm{Chi}(x) = \gamma + \log(x) + \int_0^x \frac{\cosh(t) - 1}{t}\,dt.$$

where γ is Euler's number; γ = 0.57721566490153...

The real version of Chi(*x*) is only defined for positive values of *x*. Large positive values of *x* may produce results greater than MAXREAL, the largest value representable.

Chi(*x*)



Errors

FNChi causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is negative or zero or if Chi(*x*) would be greater than MAXREAL.

See Also

Ei, Ci, Shi, Si

Ci

Cosine integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNCi(X)

Description

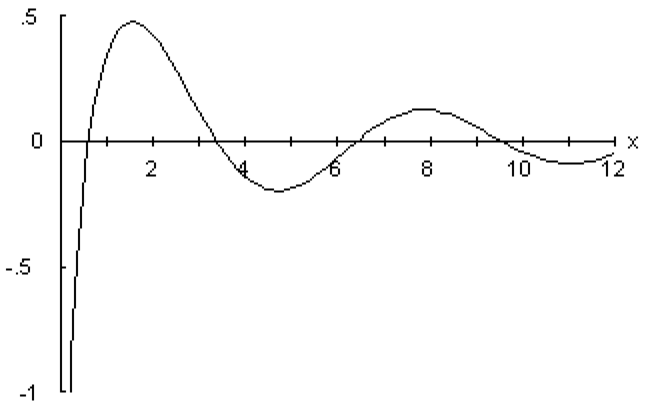
FNCi returns the value of the cosine integral of *x*, Ci(*x*). Ci(*x*) is defined by the relation

$$\mathrm{Ci}(x) = \gamma + \log(x) + \int_0^x \frac{\cos(t) - 1}{t}\, dt.$$

where γ is Euler's number; γ = 0.57721566490153...

The real version of Ci(*x*) is only defined for positive values of *x*. Large positive values of *x* may produce results greater than MAXREAL, the largest value representable.

Ci(*x*)



Errors

FNCi causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is negative or zero or if Ci(*x*) would be greater than MAXREAL.

See Also

Ei, Ci, Shi, Si

Clogbeta

Complex logarithm of the beta function of a complex argument.

Loading

```
LOADSUB ALL FROM "CGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
COMPLEX A,B,C
C=FNClogbeta(A,B)
```

Description

FNClogbeta returns the value of the logarithm of the beta function of the complex values a and b, $\log[B(a,b)]$. $B(a,b)$ is defined as $\Gamma(a)\Gamma(b)/\Gamma(a+b)$ (see *Gamma*). $B(a,b)$ is only defined for $\Re(a) > 0$ and $\Re(b) > 0$.

Errors

FNClogbeta causes a BASIC error if its arguments are not both of type COMPLEX or if the real part of either a or b is negative or zero.

See Also

Beta, Cbeta, Cgamma, Logbeta

Cloggamma

Complex logarithm of the gamma function of a complex argument.

Loading

LOADSUB ALL FROM "CGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

COMPLEX Z,C
C=FNCloggamma(Z)

Description

FNCloggamma returns the value of the logarithm of the gamma function of the complex value z, $\log[\Gamma(z)]$. The value of $\log[\Gamma(z)]$ approaches ± as z approaches a negative real integer or zero.

Errors

FNCloggamma causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of either the real or imaginary component of $\log[\Gamma(z)]$ exceeds MAXREAL, the largest number representable.

See Also

Cgamma, Gamma, Loggamma

Cmul2

Multiply outputs of *Fft* function.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Cmul2 FROM "MATHLIB.HTS"

Usage

```
COMPLEX A(*),B(*),C(*)
CALL Cmul2(A(*),B(*),C(*))
```

Description

Cmul2 multiplies each element in *A* by the corresponding element in *B* and stores the result in the corresponding element of *C*. The elements are in the form of the Fourier series coefficients output by the *Fft* subroutine. A special routine for multiplying these coefficients is necessary because the basis functions of the Fourier sine series are not *normal*, since

$$\int_0^1 \cos^2(2\pi kt)\,dt = \frac{1}{2}$$

and

$$\int_0^1 \sin^2(2\pi kt)\,dt = \frac{1}{2}$$

for k a positive integer. When the coefficients of two such series are multiplied, the result for each term having $k > 0$ needs to be scaled by dividing by 2 to make the resultant series have the same basis functions as the original series.

Two series output by the related *Cfft* subroutine can be multiplied using the HTBasic matrix dot (".") operator, since the basis function for *Cfft*, *e*2Πik*t* (*i* =√ -1), is normal.

Errors

Cmul2 causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if the size of *A*, *B*, or *C* is smaller than 2*Logn*.

See Also

Fft

Convolve

Convolution of two sequences.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Convolve FROM "MATHLIB.HTS"

Usage

```
INTEGER Logn
REAL A(*),B(*),C(*),S(*)
CALL Convolve(Logn,A(*),B(*),C(*),S(*))
```

Description

Convolve calculates the convolution of the sequences in the arrays *A* and *B* and places the result in the array *C*. *Logn* is the base-2 log of the number of points in the sequences to be convolved. The arrays *A* and *B* must have at least 2*Logn* elements and the arrays *C* and *S* must have at least 2*Logn*+1 elements; if they have more than the required number of elements, the extra elements are ignored and unmodified. S is a scratch array of at least the size C. It contains nothing useful after the function has run, but is needed to store intermediate results within the function. The number of elements in *A* and *B* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

The number of elements required in *C* and *S* for each value of *Logn* is twice the value given in the table above. *S* is used internally by *Convolve* and contains no usefule data after *Convolve* has run.

If *a*(*x*) and *b*(*x*) are the functions being tested, if the interval of interest is *x* (0,*T*), and if *a* and *b* are zero outside this interval, the convolution of *a* and *b*, *c*(*x*), is defined by the relation

$$c(x) = \frac{1}{T}\int_0^T a(t)\,b(x-t)\,dt.$$

Note that, while *f*(*x*) and *b*(*x*) are nonzero on the interval *x* (0,*T*), *c*(*x*) is nonzero on the interval *x* (0,2*T*).

If the functions *a* and *b* are only defined at regularly-spaced discrete points *x* = [*k*+½]*T*/*N*, *k* = {0,1,2,...,*N*-1}, the integration can be approximated by assuming that *a*(*x*) and *b*(*x*) are constant and equal to *a*([*k*+½]*T*/*N*) and *b*([*k*+½]*T*/*N*) between *x* = *kT*/*N* and *x* = (*k*+1)*T*/*N*.

The expression above can then be replaced by

$$c([k+½]T/N) = \frac{1}{N}\sum_{m=0}^{N-1} a([m+½]T/N)\, b([k-m+½]T/N).$$

In this case, $c([k+½]T/N)$ is defined for $k = \{0,...,2N-1\}$.

Convolve returns the values of $c([k+½]T/N)$ in the array *C*.

Errors

Convolve causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A*, *B*, or *C* is smaller than the values described above.

See Also

Conv, Correlate, Filter, Fft, Power_spectrum

Correlate

Correlation of two sequences.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Correlate FROM "MATHLIB.HTS"

Usage INTEGER Logn
REAL A(*),B(*),C(*),S(*)
CALL Correlate(Logn,A(*),B(*),C(*),S(*))

Description

Correlate calculates the correlation of the sequences in the arrays *A* and *B* and places the result in the array *C*. *Logn* is the base-2 log of the number of points in the sequences to be correlated. The arrays *A* and *B* must have at least 2*Logn* elements and the array *C* must have at least 2*Logn*+1 elements; if they have more than the required number of elements, the extra elements are ignored and unmodified. S is a scratch array of at least the size of C. It contains nothing useful after the function has run, but is needed to store intermediate results within the function. The number of elements in *A* and *B* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

The number of elements required in *C* and *S* for each value of *Logn* is twice the value given in the table above. *S* is used internally by *Correlate* and contains no usefule data after *Correlate* has run.

Correlation is a measure of two functions' similarities to each other as the abscissa is shifted. If *a*(*x*) and *b*(*x*) are the functions being tested, if the interval of interest is *x* (0,*T*), and if *a* and *b* are zero outside this interval, the correlation, *c*(*x*) of *f* and *g* is defined by the relation

$$c(x) = \frac{1}{T}\int_0^T a(t)\,b(t-x)\,dt.$$

Note that, while *a*(*x*) and *b*(*x*) are nonzero on the interval *x* (0,*T*), *c*(*x*) is nonzero on the interval *x* (-*T*,*T*).

If the functions *a* and *b* are only defined at regularly-spaced discrete points *x* = [*k*+½]*T*/*N*, *k* = {0,1,2,...,*N*-1}, the integration can be approximated by assuming that *a*(*x*) and *b*(*x*) are constant and equal to *a*([*k*+½]*T*/*N*) and *b*([*k*+½]*T*/*N*) between *x* = *kT*/*N* and *x* = (*k*+1)*T*/*N*. The expression above can then be replaced by

$$c([k+\tfrac{1}{2}]T/N) = \frac{1}{N}\sum_{m=0}^{N-1} a([m+\tfrac{1}{2}]T/N)\, b([m-k+\tfrac{1}{2}]T/N).$$

In this case, *c*([*k*+½]*T*/*N*) is defined for *k* = {-(*N*-1),...,0,...,*N*-1}.

Correlate returns the values of *c*([*k*+½]*T*/*N*) in the array *C*. The first *N* elements in *C* represent *k* = {0,1,...,*N*-1} and the last *N* elements in *C* represent *k* = {-*N*,-(*N*-1),..,-1}.

Errors

Correlate causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A*, *B*, or *C* is smaller than the values described above.

See Also

Autocorrelate, Convolve, Corr, Fft, Power_spectrum

Cpoly

Evaluate a polynomial.

Loading

```
LOADSUB ALL FROM "CPOLY.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
COMPLEX Z,Y,C(*)
Y=FNCPoly(N,C(*),Z)
```

Description

FNCpoly evaluates a polynomial of degree *n* whose coefficients are given in the elements of *C* at argument *z*. The first element in *C* is the constant term in the polynomial, the second element is the first-degree term (the multiplier of z^2), the third element is the second-degree term (the multiplier of z^2), etc. *C* must contain at least *n*+1 elements; if it contains more than *n*+1 elements, the extra elements are ignored.

Errors

FNCpoly causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the array *C* has fewer than *n*+1 elements.

See Also

Poly

Crossing

Find the point in an array that crosses a threshold.

Loading

```
LOADSUB ALL FROM "CROSS.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER K,N,S
REAL A(*),T
K=FNCrossing(A(*),T,S,N)
```

Description

FNCrossing finds the *n*th time that the values in the array *A*, beginning with element *s*, cross the threshold value *t* and returns the index of the crossing. If *n* is negative, the search is done backwards from the *s*th element for the *n*th crossing; otherwise, the search is done forward from the *s*th element.

The values in *A* are considered to cross *t* when an element of *A* is equal to *t* and the previous element was not equal to *t*, when the first element in *A* equals *t*, when an element in *A* is greater than *t* and the previous element was less than *t*, or when an element in *A* is less than *t* and the previous element was greater than *t*. In the latter two cases, the crossing occurs between two elements in *A*; the value returned is the index of the element *after* the crossing.

The value of *s* and the value returned are with reference to the lower bound specified when *A* was dimensioned or the value specified in the OPTION BASE in effect when *A* was dimensioned, if no lower bound was specified.

If the portion of *A* from *s* to the end of *A* ($n > 0$) or the portion of *A* from *s* to the beginning of *A* ($n < 0$) contains fewer than *n* crossings of the value of *t*, -1 is returned.

Errors

FNCrossing causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *s* is not in the range of the subscripts of *A*.

Csolve

Solve a system of linear equations with complex coefficients.

Loading

```
LOADSUB ALL FROM "CSOLVE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Csolve FROM "MATHLIB.HTS"
```

Usage

```
COMPLEX A(*),B(*)
CALL Csolve(A(*),B(*))
```

Description

Csolve finds the solution to the system of linear equations represented by *A* and *B* and returns the solution in *B*. *A* must be square, that is, it must have the same number of rows as columns. *B* must have the same number of rows as *A* and usually is a one-dimensional array (a vector). If A represents the matrix whose entries are stored in *A* and b represents the vector whose entries are stored in *B*, *Csolve* finds the solution vector, z, for the matrix equation

$$Az = b$$

and returns the solution in *B*, replacing the former contents of *B*. The contents of the array *A* are also destroyed by *Csolve*.

The array *B* may be two-dimensional. In this case, after *Csolve* executes, each column in *B* contains the solution vector for the case when the input values in that column were used as b in the above equation.

Csolve is equivalent to the BASIC lines

```
MAT Temp=INV(A)
MAT Z=Temp*B
MAT B=Z
```

except that the arrays *Temp* and *Z* are not needed; the intermediate results overwrite some of the elements of *A*. *Csolve* is faster than the above BASIC fragment, because the matrix inversion is not needed.

Errors

Csolve causes a BASIC error if its arguments are not both of type COMPLEX, if *A* is not square or *B* doesn't have the same number of rows as *A*, or if *A* is singular.

See Also

Solve

Cw

Complex alternate error function of a complex argument.

Loading LOADSUB ALL FROM "CERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage COMPLEX Z,C
C=FNCw(Z)

Description

FNCw returns the value of the alternate error function of the complex value *z*, *w*(*z*). *W*(*z*) is defined by the relation

$$w(z) = e^{-z^2}\left(1 + \frac{2i}{\sqrt{\pi}}\int_0^z e^{-t^2}dt\right).$$

The value of *w*(*z*) approaches + if the real part of *z* is zero and the magnitude of the imaginary part of *z* becomes large in the negative direction.

The alternate error function is related to the complementary error function evaluated by the *FNCerfc* function by the expression

$$w(z) = e^{-z^2}\mathrm{erfc}(-iz),$$

where *i* = -1.

Errors

FNCw causes a BASIC error if its argument is not of type COMPLEX or if the magnitude of *w*(*z*) exceeds MAXREAL, the largest number representable.

See Also

Cerf, Dawson, Erf

Dawson

Dawson's integral.

Loading LOADSUB ALL FROM "DAWSON.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNDawson(X)

Description

FNDawson returns the value of Dawson's integral of *x*, Daws(*x*). Dawson's integral is defined by the formula

$$\mathrm{Daws}(x) = e^{-x^2}\int_0^x e^{t^2}\,dt.$$

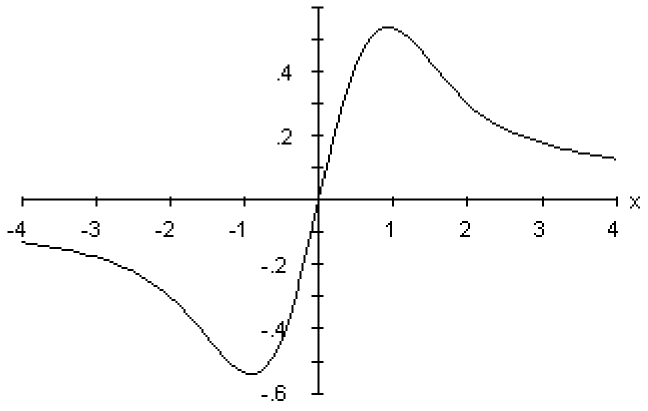
Dawson's integral is related to the alternate error function computed by the *FNCw* function, *w*(*x*), by the formula

$$\mathrm{Daws}(x) = \frac{\sqrt{\pi}}{2}\, i\,[\,w(x) - e^{-x^2}\,],$$

where *i* = -1.

Dawson's integral is defined for all values of *x*.

Daws(*x*)



Errors

FNDawson causes a BASIC error if its argument is not of type REAL.

See Also

Cw, Erf

Digamma

Digamma function of a real argument.

Loading LOADSUB ALL FROM "DIGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL Y,X
Y=FNDigamma(X)
```

Description

FNDigamma returns the value of the digamma function (sometimes called the *psi* function) of *x*, ψ(*x*). The value of ψ(*x*) approaches ± as *x* approaches a negative integer value or zero.

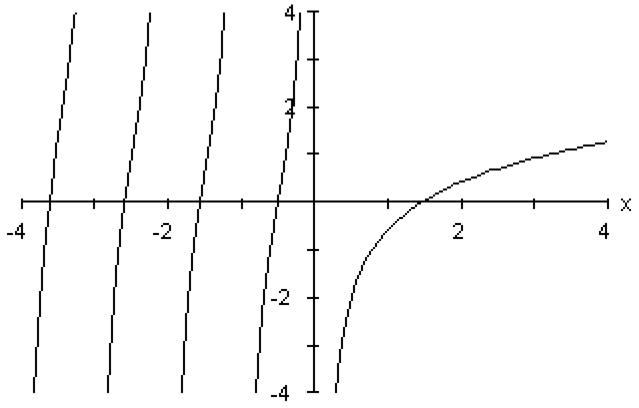
Errors

FNDigamma causes a BASIC error if its argument is not of type REAL or if the magnitude of ψ(*x*) exceeds MAXREAL, the largest number representable.

See Also

Cdigamma, Gamma

(*x*)



E1

Exponential integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNE1(X)
```

Description

FNE1 returns the value of the first-order exponential integral of *x*, *E*1(*x*). *E*1(*x*) is defined by the relation

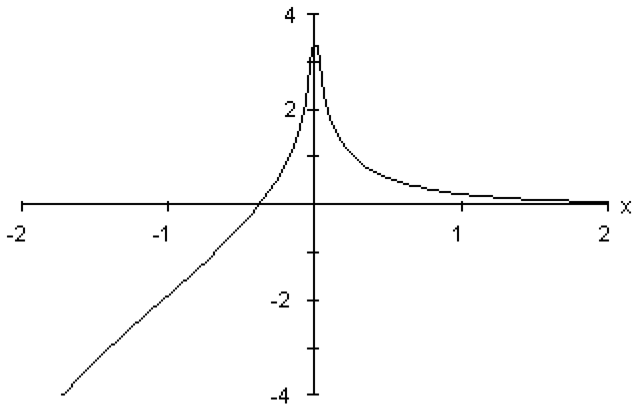
$$E_1 = \int_x^\infty \frac{e^{-t}}{t}\,dt.$$

The integration represents the value obtained by integrating in the complex plane along a path that excludes the origin and that does not cross the negative part of the real axis. The value of *E*1(*x*) is infinite at *x* = 0.

*E*1(*x*) is related to the exponential integral computed by the *FNEi* function, Ei(*x*), by the expression

$$E_1(x) = -\mathrm{Ei}(-x).$$

*E*1(*x*)



Errors

FNE1 causes a BASIC error if its argument is not of type REAL or if the magnitude of *E*1(*x*) would be greater than MAXREAL, the largest value that can be represented.

See Also

Ei

Ei

Exponential integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNEi(X)

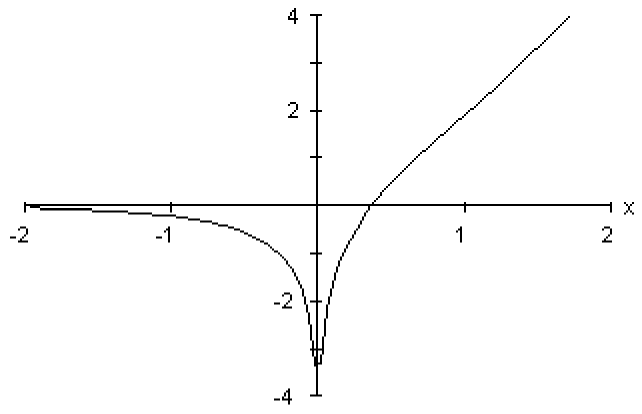
Description

FNEi returns the value of the exponential integral of *x*, Ei(*x*). Ei(*x*) is defined by the relation

$$\mathrm{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}\, dt.$$

The integration represents the value obtained by integrating in the complex plane along a path that excludes the origin and that does not cross the negative part of the real axis. The value of Ei(*x*) is - at *x* = 0 and becomes large for large positive values of *x*.

Ei(*x*)



Errors

FNEi causes a BASIC error if its argument is not of type REAL or if the magnitude of Ei(*x*) would be greater than MAXREAL.

See Also

Ci, E1, Li, Si

Erf

Error function of a real argument.

Loading LOADSUB ALL FROM "ERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

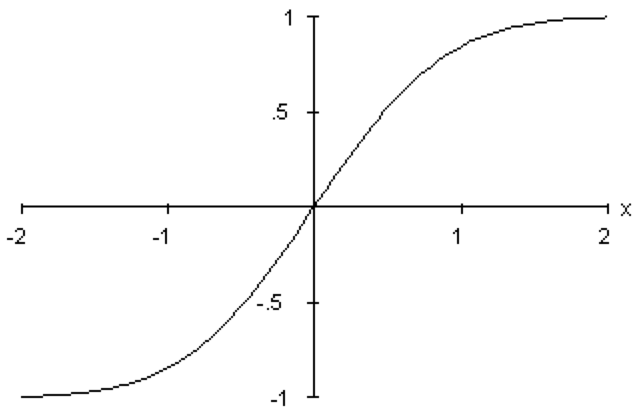
Usage REAL X,Y
Y=FNErf(X)

Description

FNErf returns the value of the error function of *x*, erf(*x*). Erf(x) is defined for all real values of *x* and has values between -1 and +1. Erf(*x*) is defined by the formula

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

erf(*x*)



Errors

FNErf causes a BASIC error if its argument is not of type REAL.

See Also

Cerf, Erfc

Erfc

Complementary error function of a real argument.

Loading LOADSUB ALL FROM "ERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNErfc(X)

Description

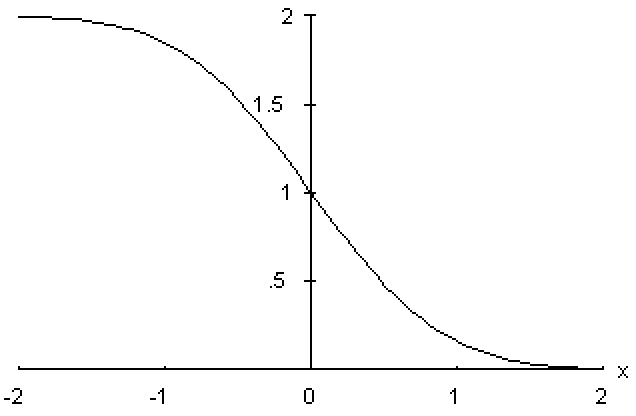
FNErfc returns the value of the complementary error function of *x*, erfc(*x*). Erfc(*x*) is defined and has values between -1 and +1 for all real values of *x*. Erfc(*x*) is defined by the formula

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt.$$

Erfc(*x*) is related to the error function returned by the *FNErf* function, erf(*x*), by the expression

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x).$$

erfc(*x*)



Errors

FNErfc causes a BASIC error if its argument is not of type REAL.

See Also

Cerf, Erfc

Fact

Factorial.

Loading

```
LOADSUB ALL FROM "FACT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL Y
Y=FNFact(N)
```

Description

FNFact returns the value of the factorial of *n*, *n*!. *N* must be a positive integer or zero.

Errors

FNFact causes a BASIC error if its argument is not of types INTEGER or if the value of *n*! is greater than MAXREAL, the largest number representable.

See Also

Binom, Gamma

Ffit

Fit a curve to a function.

Loading LOADSUB ALL FROM "FFIT.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X(*),Y(*),Tol,P(*),Yt(*)
INTEGER M,N,Status
Status=FNFfit(M,N,F$,X(*),Y(*),Tol,P(*),Yt(*))
```

Description

FNFfit attempts to find the values of the parameters to the subroutine named in *F$* that cause the function computed by the subroutine to best fit the data contained in *X* and *Y*. *X* contains the abscissas of the data points and *Y* contains the ordinates. *M* is the number of points in *X* and *Y* and *N* is the number of parameters in *P* to adjust. Before calling *FNFFit*, set *P* to the initial estimates for the parameters and *Tol* to the tolerance to use in determining when the parameters fit the data. After *Ffit* runs, *Yt* contains the values of the function being fit at each of the points in *X* using the final set of parameters. *P* contains the set of parameters.

The return value of *FNFfit* indicates why *FNFfit* finished searching for a solution. These reasons are summarized below:

Return Value	Reason
1	The average square (the *L*-2 norm) of the differ ences between the values in *Yt* and those in *Y* is less than *Tol*.
2	The average absolute value (the *L*-1 norm) of the differences between the values in *Yt* and those in *Y* is less than *Tol*.
3	Both the averages mentioned above are less than *Tol*.
4	The algorithm in *FNFfit* found values in *Yt* that caused the solution to stop growing closer to the values in *Y*.
5	The number of iterations in the algorithm used by *FNFfit* exceeded 200(*N* + 1) before the solution was found.
6 or 7	The solution in *Yt* converged to a value that had both norms described under values 1 and 2, above, greater than *Tol*. Probably *Tol* is too small for the data and function used.

Usually, if *FNFfit* returns a value of 1, 2, or 3, the solution returned in *P* is considered to be correct and if it returns a value of 4, 5, 6, or 7, the solution is considered to be incorrect.

F$* should contain the name of an HTBasic subroutine. The subroutine should take three REAL parameters. The second parameter is an array and the others are scalars. The subroutine should evaluate the function to be fit using the parameters described in the array at the argument in the third parameter and return its value in the first parameter. For example, if *F$ = "Test", then the subroutine *Test* should begin with the definition line

```
SUB Test(REAL Y,P(*),X)
```

where *X*, *P*, and *Y* may be replaced by the names of any REAL variables. The subroutine *Test* should evaluate the desired function of parameters *P* at the value *X* and return the value in *Y*. When the subroutine is called, *P* will be the array *P* mentioned in the description of the *FNFfit* function, above.

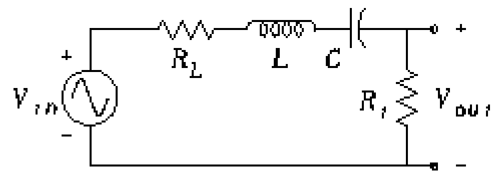
The recommended minimum value for *Tol* is about 1.5 × 10-8.

FNFfit uses the *Levenberg-Marquardt method* as modified by Moré to fit the data to the function. *FNFfit* requires *n* integers and (*m* + 5)*n* + *m* real values of temporary storage. *FNFfit* causes a BASIC error if it cannot allocate this much storage.

Errors

FNFfit causes BASIC Errors if the dimension of *X*, *Y*, or *Yt* is less than *m*, if the dimension of *P* is less than *n*, if *Tol* < 0, or if it cannot allocate enough memory to run. The subroutine named in *F$* may also cause BASIC Errors when called.

Example



An electronic filter circuit is built with an inductor (represented by the symbol *L* in the drawing), a capacitor (represented by *C*), and a 15Ω resistor (represented by *Rt*) in series. The series resistance in the inductor (represented by *RL*) is measured to be 3Ω . A sinusoidal voltage, *V*in with amplitude 10 V (peak-to-peak) is applied to the circuit and the peak-to-peak voltage across *Rt*, *V*out is measured at several frequencies, yielding the data shown in the table below.

Frequency, kHz	*V*out, V
10	1.48591
20	2.67114
30	4.46957
40	6.79728
50	8.06452
60	5.86735
70	4.49784
80	3.55642
90	2.99302
100	2.60036
110	2.29451
120	2.05200
130	1.87589

The values of *L* and *C* are related to the ratio *V*out/*V*in by the expression

$$\frac{V_{\text{out}}}{V_{\text{in}}} = \frac{R_t}{R}\frac{1}{\sqrt{1+\left(\frac{2\pi fL}{R} - \frac{1}{2\pi fRC}\right)^2}},$$

where *R* is the sum of *RL* and *Rt*.

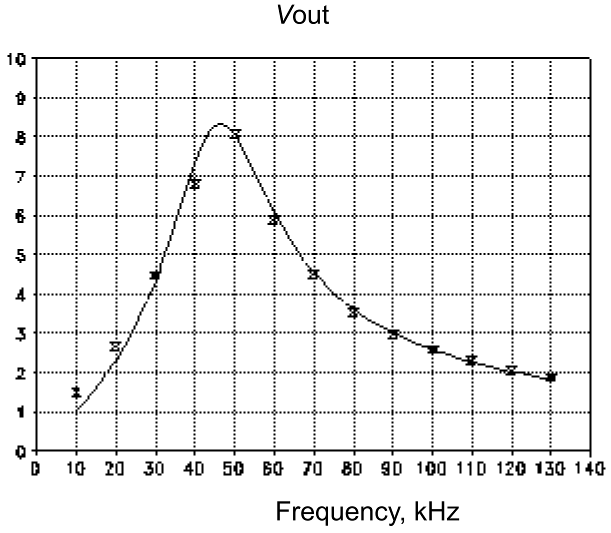
The following program uses *FNFfit* to estimate the values of *L* and *C* from the data in the table. It uses the *FNNorm* function (described under the *Norm* topic in this manual) to calculate the average error in the fit. The data in lines 150 - 170 is the ratio *V*out/*V*in. The initial values for *L* and *C* (represented in the program by *P(1)* and *P(2)*) are the values marked on the components.

```
10 LOADSUB ALL FROM "FFIT.HTS"
20 LOADSUB ALL FROM "NORM.HTS"
30 REAL P(1:2),Fvec(1:13)
40 REAL Freq(1:13),Vratio(1:13)
50 INTEGER I,Info
60 FOR I=1 TO 13
70 READ Freq(I)
80 NEXT I
90 FOR I=1 TO 13
100 READ Vratio(I)
110 NEXT I
120 DATA 10000.0,20000.0,30000.0,40000.0,50000.0
130 DATA 60000.0,70000.0,80000.0,90000.0,100000.0
140 DATA 110000.0,120000.0,130000.0
150 DATA 0.148591,0.267114,0.446957,0.679728,0.806452
160 DATA 0.586735,0.449784,0.355642,0.299302,0.260036
170 DATA 0.229451,0.205200,0.187589
180 P(1)=1.0E-4 ! initial value for L
190 P(2)=1.0E-7 ! initial value for C
200 Info=FNFfit(13,2,"Rlc",Freq(*),Vratio(*),1.5E-8, P(*),Fvec(*))
210 PRINT "Final average error "; FNNorm(Fvec(*))/SQR(13.0)
220 PRINT "Exit parameter ";Info
230 PRINT "L =";PROUND(P(1)*1.0E+6,0);CHR$(230);"Hy"
240 PRINT "C =";PROUND(P(2)*1.0E+6,-3);CHR$(230);"Fd"
250 END
260 SUB Rlc(REAL Y,P(*),X)
270 REAL Omega
280 INTEGER I
290 Omega=6.28318530717959*X
300 Y=(15.0/18.0)/ABS(CMPLX(1.0,Omega*P(1)/18.0
1.0/(Omega*18.0*P(2))))
310 SUBEND
```

When run, the program produces the output

```
Final average error .0226436510548
Exit parameter 1
L = 112 μHy
C = .105 μFd.
```

The value of *V*out is plotted below over a range of frequencies for the calculated values *L* and *C*. The symbols on the plot are the measured values.



Fft

Discrete Fourier transform of a real sequence.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Fft FROM "MATHLIB.HTS"

Usage

```
INTEGER Logn
REAL A(*)
COMPLEX F(*)
CALL Fft(Logn,A(*),F(*))
```

Description

Fft calculates the discrete Fourier transform of the sequence in the array *A* and stores the result in the array *F*. *Logn* is the base-2 log of the number of points in the sequence. The array *A* must contain at least 2*Logn* elements and the array *F* must contain at least 2*Logn*-1 elements. If they have more than the required number of elements, the extra elements are ignored and unmodified. The number of elements in *A* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

If the values in *A* are taken to be values of a continuous signal, *a*(*t*), sampled at constant intervals of *T* (time, distance, or whatever units apply), and if the signal sampled contained no terms at or above the frequency 1/2*T*, then the coefficients in the array *C* are the coefficients of the Fourier sine series that describes *a*(*t*). *A*(*t*) can be reconstructed from the elements of *F* through the following formula:

$$a(t) = f_0 + \sum_{k=1}^{N/2-1} \Re e(f_k)\cos\left(\frac{2k\pi t}{NT}\right) + \Im m(f_k)\sin\left(\frac{2k\pi t}{NT}\right)$$

where

$$N = 2^{Logn}.$$

If the signal *a*(*t*) contains components at or above the frequency 1/2*T*, the situation is complicated by *aliasing*, which is explained in most signal processing textbooks.

Some of the more common operations done using discrete Fourier transforms, such as convolution, correlation, filtering, and finding power spectral densities are available as separate CSUBs; see the entries for *Autocorrelation*, *Convolve*, *Correlate*, *Filter*, *Rfilter*, and *Power_Spectrum* for details on their use. The inverse of *Fft* is performed by the *Ifft* subroutine. A discrete Fourier transform for complex sequences is computed by the *Cfft* routine.

Errors

Fft causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, if the size of *A* is smaller than 2*Logn*, or if the size of *F* is smaller than 2*Logn*-1.

Examples

Often, the discrete Fourier transform calculated by the *Fft* and *Cft* Math Library subroutines are used to obtain frequency information about continuous signals that have been sampled at discrete points. Such is the case with signal brought into the computer by an A/D converter or by an image scanner. The examples in this section use HTBasic Math Library subroutines to explain some of the pitfalls that arise when a continuous signal is represented by discrete samples and how they are commonly avoided. These pitfalls happen because the samples presented to the computer contain no information about the continuous signal's behavior between the sample points or outside the interval covered by the array of samples.

The frequency spectrum of a continuous, periodic signal, *a*(*t*), of period *P* can be represented by a set of coefficients, *ck*, such that

$$a(t) = c_0 + \sum_{k=1}^{\infty} \Re e(c_k)\cos\left(\frac{2k\pi t}{P}\right) + \sum_{k=1}^{\infty} \Im m(c_k)\sin\left(\frac{2k\pi t}{P}\right).$$

The coefficients *ck* are called the complex Fourier series.

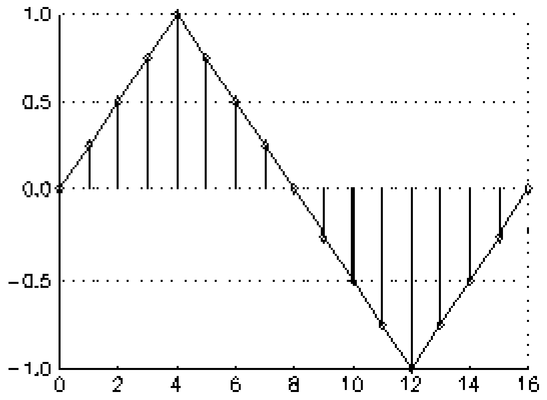


Fig. 1. One period of a sawtooth waveform.

The Nyquist criterion. Consider a periodic sawtooth wave. One period of the wave is shown in the figure 1. If the sampling interval is one period of the waveform, the waveform can be represented in a computer by the 16 samples shown as diamonds in the figure. Note that the samples contain no information about what happens to the wave between samples, so the same samples could represent an infinite number of other waveforms.

The complex Fourier sine/cosine transform of the sawtooth can be calculated or found in a math table. The coefficients, *ck*, of the sine/cosine transform are

$$c_k = \begin{cases} 0, & k \text{ even} \\ (-1)^{(k-1)/2}\dfrac{8}{(\pi k)^2} i, & k \text{ odd} \end{cases}.$$

where *i* is the square root of -1.

The following BASIC program calculates the discrete Fourier transform of the sampled waveform.

```
10 LOADSUB ALL FROM "FFT.CSB"
20 LOADSUB ALL FROM "WAVEFORM.CSB"
30 INTEGER I
40 REAL A(0:15),Anew
50 COMPLEX B(0:7)
60 CALL Waveform(16.0,1.0,0.,0.,3,A(*))
70 CALL Fft(4,A(*),B(*))
80 FOR I=0 TO 7
90 PRINT I,B(I)
100 NEXT I
110 END
```

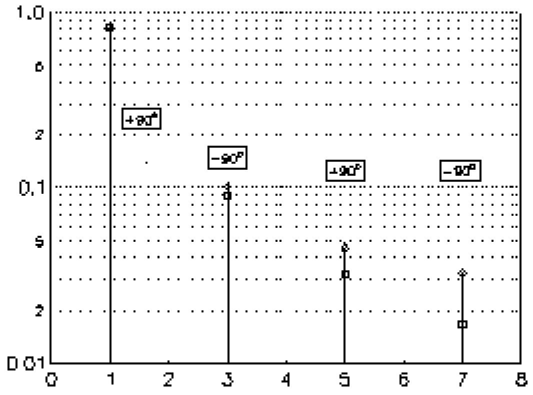


Fig. 2. The magnitude of the discrete and sine/cosine Fourier coefficients of the waveform. The boxed numbers are the phases of the coefficients.

The values produced by the discrete Fourier transform are plotted in figure 2 as diamonds. The first four nonzero values of the complex Fourier sine/cosine transform are plotted in the same figure as squares. Note that the plot is on a logarithmic scale.

The coefficients *fk* of the discrete Fourier transform are related to those of the complex sine/cosine transform by the following formula:

$$f_k = \sum_{m=0}^{\infty} c_{Nm+k} + \sum_{n=1}^{\infty} c^*_{Nn-k} ,$$

where "*" represents the complex conjugate operation and *N* represents the number of samples in the sampling interval (16 in the above example). Therefore, the spectrum calculated by the discrete Fourier transform represents the ordinary Fourier spectrum only when the signal in question has nonzero spectral components for one value of *m* or *n*. This is called the *Nyquist criterion*. Application of this criterion resolves the ambiguity mentioned earlier of which of the infinite number of possible waveforms the samples represent.

Usually, this criterion is satisfied by allowing nonzero spectral components only for *m* = 0. The Nyquist criterion can then be stated as requiring samples to be taken at twice the highest frequency present in the signal being sampled. Note that the sawtooth waveform used in this example does not obey the Nyquist criterion, since is has spectral components for an infinite number of frequencies.

The following BASIC program produces two continuous waveforms that have the same 16 samples as the sawtooth waveform shown at the beginning of this example. The first waveform has nonzero spectral components for *m* = 0 and the second for *n* = 0. The waveforms are plotted after the program listing.

```
10 LOADSUB ALL FROM "FFT.HTS"
20 LOADSUB ALL FROM "WAVEFORM.HTS"
30 INTEGER I,J
40 REAL A(0:15),Anew1,Anew2,Theta1,Theta2
50 COMPLEX B(0:7)
60 RAD
70 CALL Waveform(16.0,1.0,0.,0.,3,A(*))
80 CALL Fft(4,A(*),B(*))
90 FOR I=0 TO 1024
100 Anew1=0.
120 Anew2=0.
120 FOR J=0 TO 7
130 Theta1=2.0*PI*I*J/1024.0
140 Theta2=2.0*PI*I*(16-J)/1024.0
150 Anew1=Anew1+REAL(B(J))*COS(Theta1)+ IMAG(B(J))*SIN(Theta1)
160 Anew2=Anew2+REAL(B(J))*COS(Theta2)- IMAG(B(J))*SIN(Theta2)
170 NEXT J
180 PRINT I/1024.0*16.0,Anew1,Anew2
190 NEXT I
200 END
```

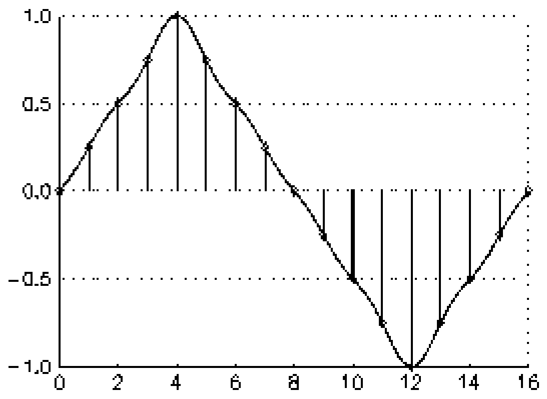


Fig. 3. A waveform having the same sample values as that shown in figure 1.

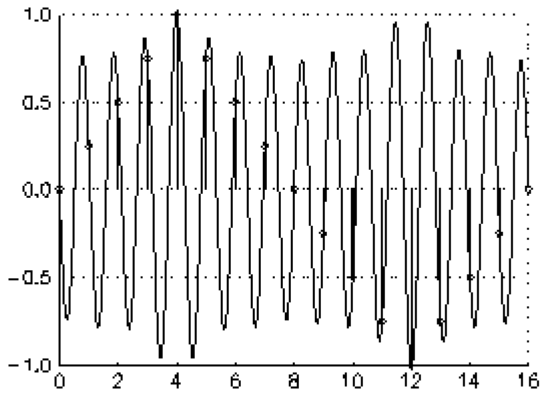


Fig. 4. Another waveform having the same sample values as that shown in figure 1.

Windowing. Consider the function

$$\frac{8}{\pi^2}\sin(\frac{2\pi t}{T}) - \frac{8}{(3\pi)^2}\sin(\frac{6\pi t}{T}) + \frac{8}{(5\pi)^2}\sin(\frac{10\pi t}{T}) - \frac{8}{(7\pi)^2}\mathrm{sir}$$

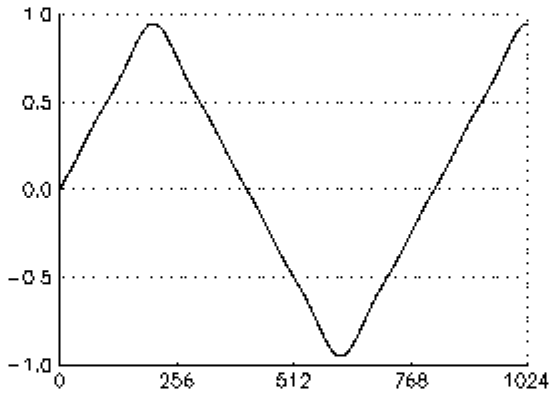


Fig. 5. A test waveform.

Obviously, this function contains no frequency components above 14/*T*, in radians per unit time, distance, or whatever. If the waveform described by this function is sampled every 5*T*/4096 units, the sample frequency is 24096/5*T* or 8192/5*T* radians per unit, which is well above twice the highest frequency component in the waveform, so the Nyquist criterion is satisfied with this sample spacing. A sampling interval of 1024 samples of this waveform is shown in figure 5.

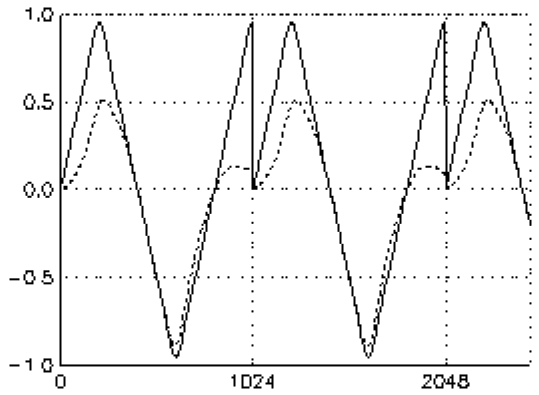


Fig. 6. A larger view of the waveform shown in figure 5 (solid line) and a view of the windowed waveform.

When this waveform is sampled for use in the computer, no information is provided on the behavior before or after the sample interval. When a discrete Fourier transform is done on the test waveform, the discrete Fourier transform algorithm assumes that the data set presented to it represents one period of a periodic waveform. Because the set of samples did not cover exactly one period of the waveform, the discrete Fourier transform connects the last sample in the data set with the first; that is, it computes the transform of a function like that shown by the solid line in figure 6. This introduces high-frequency components in the frequency spectrum. The discrete Fourier transform for this waveform is calculated by the BASIC program below and is plotted in figure 8 on the next page. Figure 7 shows the Fourier sine/cosine transform of the original, continuous waveform for comparison purposes.

```
10 LOADSUB ALL FROM "FFT.CSB"
20 LOADSUB ALL FROM "POLAR.CSB"
30 INTEGER I,J
40 REAL A(0:1023),Amp(0:511),Phase(0:511),Anew,Theta
50 COMPLEX B(0:511)
60 RAD
70 FOR I=0 TO 1023
80 Theta=2.0*PI*I*1.25/1024.0
90 A(I)=(SIN(Theta)-SIN(3.0*Theta)/9+
SIN(5.0*Theta)/25-SIN(7.0*Theta)/49)*8.0/(PI*PI)
100 NEXT I
110 CALL Fft(10,A(*),B(*))
120 CALL Polar(B(*),"D",Amp(*),Phase(*))
130 FOR I=0 TO 511
140 PRINT I,Amp(I),Phase(I)
150 NEXT I
160 END
```

One way to reduce the high-frequency components in the spectrum of the sampled signal is to multiply the signal by a *window function*, such as one of the several provided in the math library. The following BASIC program windows the test function using the Kaiser-Bessel window with parameter 4.0 and calculates the discrete Fourier transform of the windowed waveform. The dashed line in figure 6 shows the windowed function and figure 9 shows the discrete transform of the windowed function.

```
10 LOADSUB ALL FROM "FFT.HTS"
20 LOADSUB ALL FROM "BESMC.HTS"
30 LOADSUB ALL FROM "POLAR.HTS"
40 INTEGER I
50 REAL A(0:1023),Amp(0:511),Phase(0:511)
60 COMPLEX F(0:511)
70 FOR I=0 TO 1023
80 Theta=2.0*PI*I*1.25/1024.0
90 A(I)=(SIN(Theta)-SIN(3.0*Theta)/9+
SIN(5.0*Theta)/25-SIN(7.0*Theta)/49)*8.0/(PI*PI)
100 NEXT I
110 CALL W_kaiser(A(*),4.0,A(*))
120 FOR I=0 TO 1023
130 PRINT I,A(I)
140 NEXT I
150 CALL Fft(10,A(*),F(*))
160 CALL Polar(F(*),"D",Amp(*),Phase(*))
170 FOR I=0 TO 511
180 PRINT I,Amp(I),Phase(I)
190 NEXT I
200 END
```

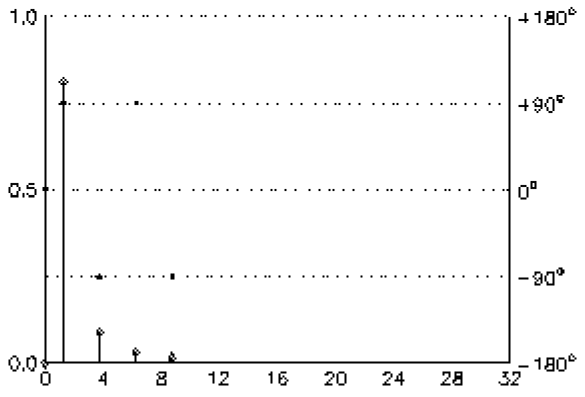


Fig 7. The Fourier sine/cosine transform of the test waveform.

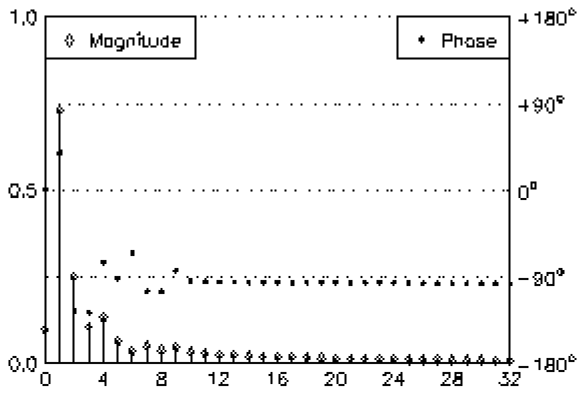


Fig 8. The discrete Fourier transform of the test waveform shown in figure 5.

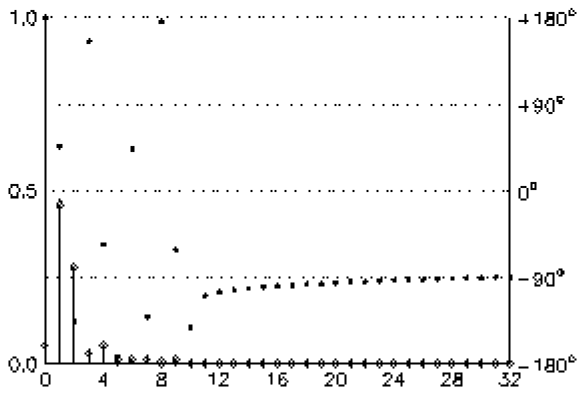


Fig. 9. The discrete Fourier transform of the windowed function.

Note that the magnitudes of the high-frequency components of the spectrum shown in figure 9 are greatly reduced. Note also the wildly-varying phases of the values shown in figure 9. Windowing functions usually reduce the spurious high-frequency components in a sample at the expense of inaccuracies in the phase.

See Also

Cfft, Convolve, Correlate, Filter, Ifft, Power_spectrum, Rfilter

Notes

If the related *Cfft* subroutine is applied to a sequence of real values, it outputs twice as many coefficients as the *Fft* routine. For a real input sequence, the real parts of these coefficients are symmetric about the *N*/2-1st and *N*/2th coefficients (beginning subscripts with the 0th coefficient) and the imaginary parts are antisymmetric around these same coefficients. The coefficients output by *Fft* for the same input sequence are double the first *N*/2 coefficients output by *Cfft* except for the 0th or d. c. coefficient, which have the same value.

Fftz

Dransform of a real sequence lengthened with zeros.

Loading

```
LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Fftz FROM "MATHLIB.HTS"
```

Usage

```
INTEGER Logn
REAL A(*)
COMPLEX F(*)
CALL Fftz(Logn,A(*),F(*))
```

Description

Fftz calculates the discrete Fourier transform of the sequence in the array *A* lengthened with 2*Logn* zeros and stores the result in the array *F*. *Logn* is the base-2 log of the number of points in the sequence in *A*. The arrays *A* and *F* must contain at least 2*Logn* elements. If they have more than the required number of elements, the extra elements are ignored and unmodified. The number of elements denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

Except for the lengthening of the input sequence, the values returned by *Fftz* have the same meaning as those returned by *Fft*; see the entry for *Fft* for an explanation of the meaning of the values returned.

Fftz has been provided as a separate CSUB because lengthened sequences are often used when implementing convolutions and correlations and when implementing multiple-window operations on long streams of data. Such operations often have results that are twice as long in the time or space domain as either of their inputs. This results in their Fourier transforms having twice as many frequency components as the transforms of their inputs, with the extra components halfway between the components in the transforms in the input sequences.

Errors

Fftz causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *F* is smaller than the values described above.

See Also

Convolve, Correlate, Fft, Filter, Rfilter

Filter

Filter a sequence.

Loading

```
LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Filter FROM "MATHLIB.HTS"
```

Usage

```
INTEGER Logn
REAL A(*),C(*)
COMPLEX F(*)
CALL Filter(Logn,A(*),F(*),C(*))
```

Description

Filter calculates the sequence produced by filtering the time-domain (or space-domain) sequence in *A* by the filter whose frequency-domain coefficients are in *F*. It returns the resulting sequence in the array *C*. *Logn* is the base-2 log of the number of points in the sequences in *A* and *F*. The arrays *A* and *F* must contain at least 2*Logn* elements. The array *C* must contain at least 2*Logn*+1 elements. If the arrays have extra elements, the extra elements are ignored and unmodified.

The values in *F* are the amounts by which to scale the corresponding frequency components of *A* to produce the resultant sequence. These values are stored in (*real*,*imaginary*) [rectangular] form. If filter coefficients are to be used that are specified in the more usual (*magnitude*,*angle*) [polar] form, they must be converted to rectangular form when stored in the elements of *F* (the *Polar* routine can do this conversion). If filter coefficients are to be used that have all zero phase, the related *Rfilter* function can be used to save converting the phase data to complex form.

The first element in *F* represents the amount by which the zero-frequency (d. c.) term in *A* is to be scaled, the second the amount by which the 1/*N* frequency component is scaled, the third the amount by which the 2/*N* frequency component is scaled, etc. The meaning of each frequency component is the same for *Filter* as for *Fft* and is explained in the entry for the *Fft* routine.

If the sequence to be used as a filter is specified as an impulse response, the *Convolve* function may be used instead of *Filter* to filter using the impulse response as input.

Errors

Filter causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, if the size of *A* or *C* is smaller than 2*Logn*, or if the size of *F* is smaller than 2*Logn*-1.

See Also

Cfft, Convolve, Rfilter, Fft

Fit

Fit a polynomial curve to a series of data points.

Loading

```
LOADSUB ALL FROM "FIT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Fit FROM "MATHLIB.HTS"
```

Usage

```
INTEGER M,N
REAL X(*),Y(*),C(*)
CALL Fit(M,X(*),Y(*),N,C(*))
```

Description

Fit calculates the coefficients of the polynomial of degree *n* that gives the closest fit to the points whose coordinates are (*xk*,*yk*), where *k* is an index into the *m* values in the arrays *X* and *Y*. The "closest fit" is that which gives the smallest sum of squares of the differences between each point *yk* and the corresponding *p*(*xk*), where *p* is the polynomial generated from the coefficients returned in *C*. The first element in *C* contains the constant or zero-order coefficient, the second element the first-order coefficient, etc. The polynomial described by the coefficients of *C* can be evaluated by *Poly*, described in its own entry.

N must be between zero and 10, inclusive. *M* may be any positive integer. The dimensions of *X* and *Y* must be at least *m*. If either *X* or *Y* has more than *m* data points, the extra points are neither used nor modified by *Fit*. Similarly, *C* must contain at least *n*+1 data points; if *C* has more than *n*+1 data points, the extra points are not modified by *Fit*.

If *n* is zero, *Fit* returns the average value of the elements in *Y*, which is the zero-order polynomial that most closely approximates the points in *X* and *Y*. If *N* is 1, *Fit* returns the coefficients of the linear polynomial that most closely approximates the points in *x* and *y*, and so on for higher values of *n*.

Note that polynomials higher than degree 2 or 3 tend to have extreme values outside the region defined by the smallest and largest *xk*, although they give more accurate approximations of the values of *yk* inside this region. In addition, higher-order polynomials may oscillate between adjacent values of *xk*. If such oscillations occur, a smaller-degree polynomial would probably give a better approximation than a larger-degree one. Because of this, checking higher-order fitting functions with a graph is advisable.

Errors

Fit causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *n* is not between 0 and 10, inclusive, if the size of *X* or *Y* is smaller than *M*, or if the size of *C* is smaller than *N*+1.

Example

The percentage of automobiles in the United States with at least one passenger suffering extreme injury or death in collisions was tabulated in 1970 and categorized by weight of the automobiles involved. The data was distributed as shown in the table below.

Weight of Automobile	Percent Injury or Death
1900 lb.	9.6%
2800	6.4

3400	5.2
3700	4.0
4800	3.1

If *w* is the weight of the automobile and *p* is the percent of injury or death, the following BASIC program finds the coefficients that relate *p* to *w* assuming the relation has the form

$$p = c1w + c0$$

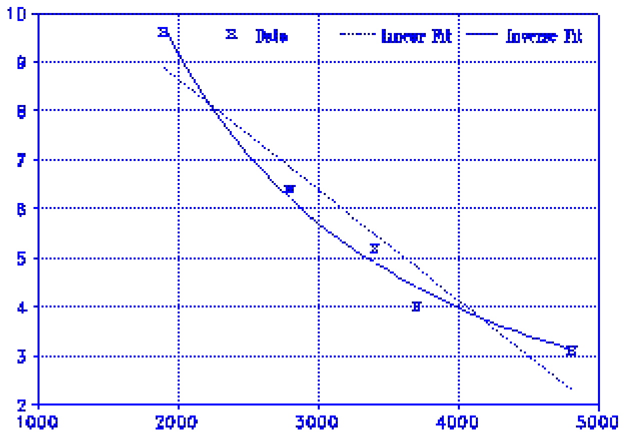
or

$$p = c1/w + c0.$$

```
10 LOADSUB ALL FROM "FIT.CSB"
20 REAL Weight(1:5),Pct(1:5),C(0:1)
30 READ Weight(*),Pct(*)
40 DATA 1900,2800,3400,3700,4800
50 DATA 9.6,6.4,5.2,4.0,3.1
60 CALL Fit(5,Weight(*),Pct(*),1,C(*)) ! linear fit
70 PRINT USING """1. Rate = "",2D.1D,""% -"",1D.6D,
      ""%/pound x Weight""";C(0),-C(1)
80 MAT Weight=(1)/Weight
90 CALL Fit(5,Weight(*),Pct(*),1,C(*)) ! inverse fit
100 PRINT USING """2. Rate = "",6D,
      ""% pound / Weight -"",2D.1D,""%""";C(1),-C(0)
110 END
```

The figure below shows the data from the table and curves drawn from the parameters *c*1 and *c*0 computed by the program.

Percent Injury or Death



Weight of Automobile, lb.

See Also

Ffit, Poly

Froot

Find a root of an equation of the form *f*(*x*) = 0.

Loading

LOADSUB ALL FROM "FROOT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB FNFroot FROM "MATHLIB.HTS"

Usage

REAL A,B,Eps,X
INTEGER N
X=FNFroot(F$,A,B,N,Eps)

Description

FNFroot attempts to find a value of *x* that satisfies the relation *f*(*x*) = 0, where *f* is the HTBasic function named in *F$*. *A* and *b* contain two distinct initial estimates for *x* as near to the root as possible. *N* and *Eps* contain stopping criteria. If the number of iterations within *FNFroot* exceeds *n*, *FNFroot* stops and returns MAXREAL (approximately 1.7×10308) to indicate failure. If *f*(*x*) *Eps* for a value of *x*, *FNFroot* considers the value to be a solution and returns that value of *x*.

F$* should contain the name of an HTBasic subroutine. The subroutine should take two REAL parameters. It should evaluate the function to be integrated at the second parameter and return its value in the first parameter. For example, if *F$ = "Test", then the subroutine *Test* should begin with the definition line

SUB Test(REAL Y,X)

where *X* and *Y* may be replaced by the names of any REAL parameters. The subroutine *Test* would evaluate the desired function at the value *X* and return the value in *Y*.

FNFroot uses the *secant method* to find the root. If it finds a situation where *f*(*a*) and *f*(*b*) have opposite signs, it uses the *bisection method* to find the value of *x* between *a* and *b* that makes *f*(*x*) be 0. These methods are described in most texts on numerical mathematical methods.

Errors

FNFroot causes HTBasic Errors if *a* = *b*, if *N* < 2, or if *Eps* < 0. It also causes an error if the subroutine named in *F$* is undefined. The subroutine named in *F$* may also cause HTBasic Errors when it is evaluated.

Example

The following program finds the roots of the equation *x* - ¼*e*-*x* = 0.

```
10 LOADSUB ALL FROM "FROOT.HTS"
20 X=FNFroot("Func",0,1,100,1.0E-100)
30 CALL Func(Y,X)
40 PRINT "Root 1: (";X;",";Y;")"
50 X=FNFroot("Func",2,3,100,1.0E-100)
60 CALL Func(Y,X)
70 PRINT "Root 2: (";X;",";Y;")"
80 END
90 SUB Func(REAL Y,X)
100 Y=X-EXP(X)*.25
```

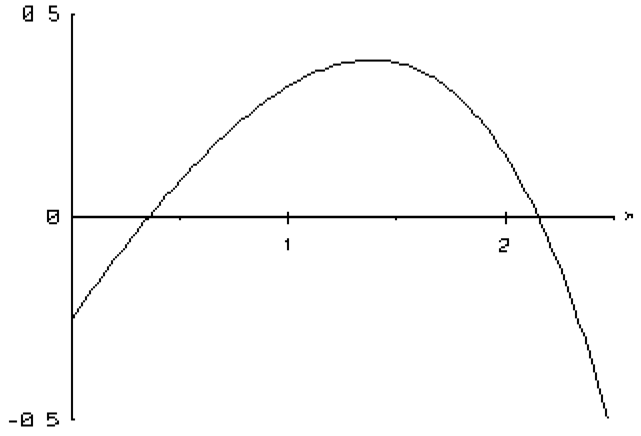
```
110 SUBEND
```

It produces the output

```
Root 1: ( .357401956181 , 0 )
Root 2: ( 2.15329236411 , 4.4408920985E-16 ).
```

The function *x* - ¼*e*-*x* is plotted below.

x - ¼*e*-*x*



F_beta

Probability density for beta distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNF_beta(A,B,X)

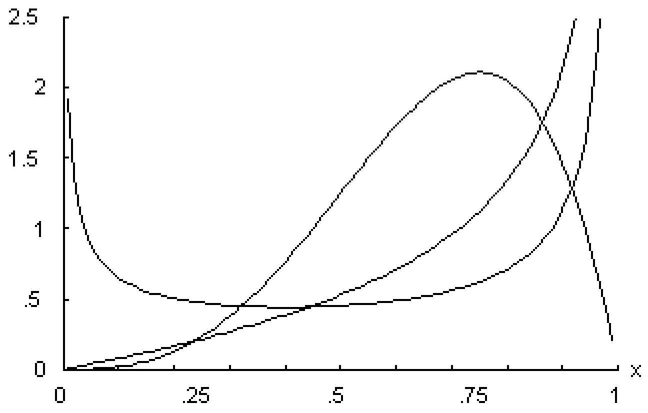
Description

FNF_beta returns the value of the univariate beta probability density function with parameters *a* and *b*, *f*(*x*;*a*,*b*). *F*(*x*;*a*,*b*) is defined only for *a* 0 and *b* 0.

F(*x*;*a*,*b*) is defined by the expression

$$f(x;a,b) = \begin{cases} \dfrac{x^{a-1}(1-x)^{b-1}}{B(a,b)}, & 0 \le x \le 1 \\ 0, & x < 0 \text{ or } x > 1 \end{cases}.$$

f(*x*;*a*,*b*)



Errors

FNF_beta causes a BASIC error if its arguments are not all of type REAL or if *a* or *b* is negative.

See Also

P_beta, Q_beta

F_cauchy

Probability density for Cauchy distribution.

Loading LOADSUB ALL FROM "CAUCHY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNF_cauchy(A,B,X)

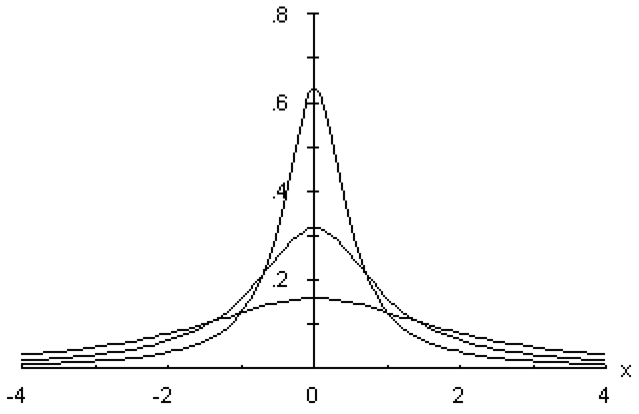
Description

FNF_cauchy returns the value of the probability density function of the Cauchy distribution with parameters *a* and *b* at *x*. This density, *f*(*x*;*a*,*b*), is defined by the expression

$$f(x;a,b) = \frac{1}{\pi b\left[1+\left(\frac{x-a}{b}\right)^2\right]}.$$

B must be greater than zero.

f(*x*;a,*b*)



Errors

FNF_cauchy causes a BASIC error if its arguments are not all of type REAL or if *b* is negative or zero.

See Also

P_cauchy, Q_cauchy

F_chi2

Probability density for chi-squared distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNF_chi2(N,X)
```

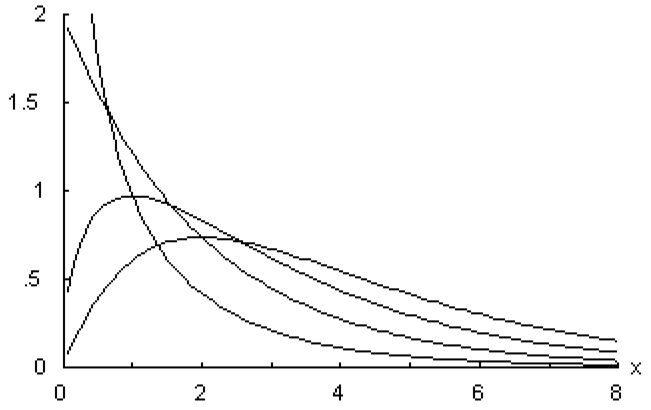
Description

FNF_chi2 returns the value of the probability density function of the univariate chi-squared (χ^2) distribution with parameter *n* at *x*. This density, *f*(*x*;*n*), is defined by the expression

$$f(x;n) = \begin{cases} \dfrac{x^{n/2-1}e^{-x/2}}{2^{n/2}\Gamma(n/2)}, & x > 0 \\ 0, & x \le 0 \end{cases}.$$

X is often written as X²; among other uses, this notation emphasizes the fact that this distribution is only nonzero for values of *x* 0. Although *f* is sometimes defined for *n* < 0, most implementations, including this one, restrict *f* to being defined for *n* 0.

f(*x*;*n*)



Errors

FNF_chi2 causes a BASIC error if its arguments are not of the types listed in the usage section, above, or if *n* is negative.

See Also

P_chi2, Q_chi2

F_exp

Probability density for exponential distribution.

Loading LOADSUB ALL FROM "EXP.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,Y
Y=FNF_exp(A,X)

Description

FNF_exp returns the value of the probability density function of the exponential probability distribution with parameter *a* at *x*, *f*(*x*;*a*). *F*(*x*;*a*) is defined by the expression

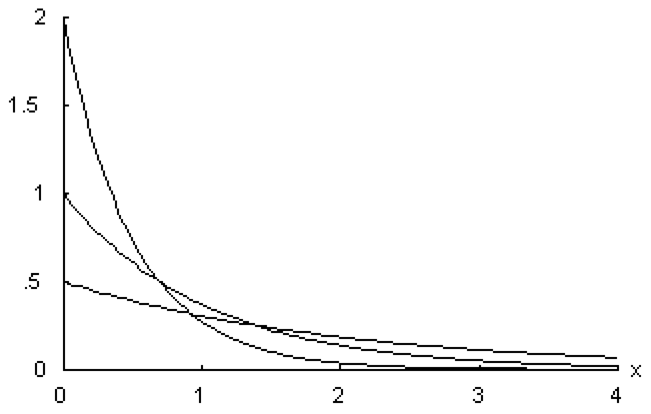
$$f(x;a) = \begin{cases} a e^{-ax}, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

F(*x*;*a*) is defined for positive values of *a*.

Errors

FNF_exp causes a BASIC error if its arguments are not all of type REAL or if *a* is negative or zero.

f(*x*;*a*)



See Also

P_exp, Q_exp

F_f

Probability density for *F* distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

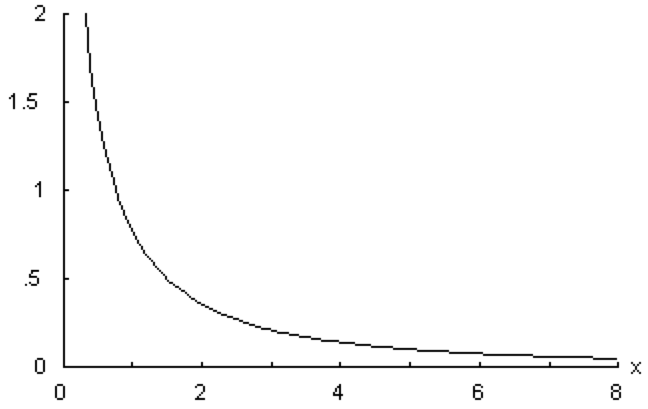
Usage

```
INTEGER M,N
REAL X,Y
Y=FNF_f(M,N,X)
```

Description

FNF_f returns the value of the probability density function of the univariate *F* probability distribution with parameters *m* and *n* at *x*, *f*(*x*;*m*,*n*). This function is defined for *m* and *n* positive or zero.

f(*x*;1,2)



Errors

FNF_f causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if either *m* or *n* is negative.

See Also

P_f, Q_f

F_gauss

Probability density for Gaussian distribution.

Loading LOADSUB ALL FROM "ERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,M,S,Y
Y=FNF_gauss(M,S,X)

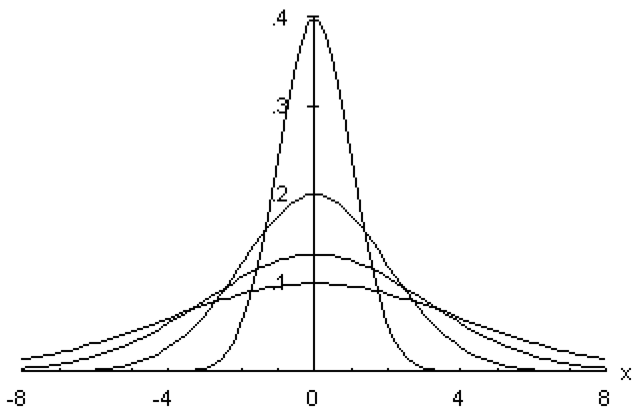
Description

FNF_gauss returns the value of the probability density function of the Gaussian, or normal, probability distribution of mean *m* and standard deviation *s* (represented below by) at *x*, $f(x;m,\sigma)$. $F(x;m,\sigma)$ is defined by the expression

$$f(x;m,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{(x-m)^2}{2\sigma^2}}.$$

F is defined for all values of *x* and *m* and for positive values of .

$f(x;0,\sigma)$



Errors

FNF_gauss causes a BASIC error if its arguments are not all of type REAL or if the value of *S* is negative or zero.

See Also

P_gauss, Q_gauss

F_laplace

Probability density for LaPlace distribution.

Loading LOADSUB ALL FROM "LAPLACE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNF_laplace(A,B,X)

Description

FNF_laplace returns the value of the probability density function of the Laplace distribution of parameters *a* and *b* at *x*, *f*(*x*;*a*,*b*). *F*(*x*;*a*,*b*) is defined by the expression

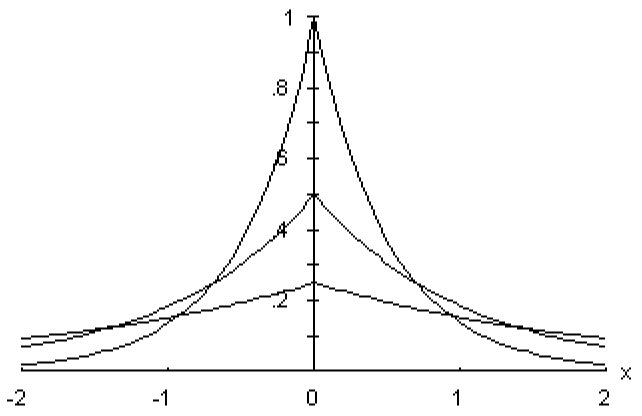
$$f(x;a,b) = \frac{1}{2b} e^{-\left|\frac{x-a}{b}\right|}.$$

F is defined for all positive values of *b*.

Errors

FNF_laplace causes a BASIC error if its arguments are not all of type REAL or if the value of *b* is negative or zero.

f(*x*;0,*b*)



See Also

P_laplace, Q_laplace

F_pareto

Probability density for Pareto distribution.

Loading LOADSUB ALL FROM "PARETO.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,X0,T,Y
Y=FNF_pareto(X0,T,X)

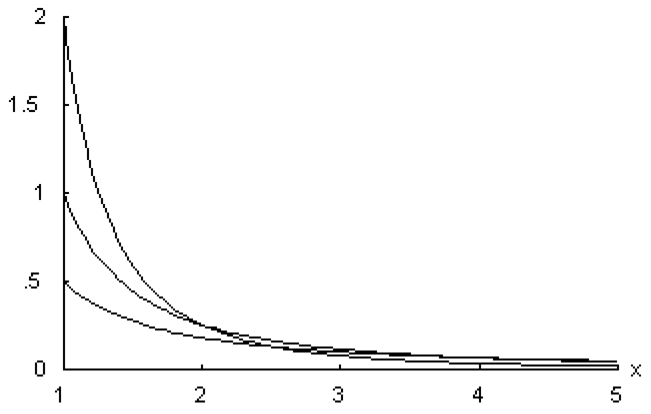
Description

FNF_pareto returns the value of the probability density function of the Pareto distribution of parameters *X0* (here written as *x*0) and *t* at *x*, *f*(*x*;*x*0,*t*). *F*(*x*;*x*0,*t*) is defined by the expression

$$f(x;x_0,t) = \begin{cases} \frac{t}{x_0}\left(\frac{x_0}{x}\right)^{t+1}, & x \geq x_0 \\ 0, & x < x_0 \end{cases}.$$

F is defined for positive values of *x*0 and *t*.

f(*x*;1,*t*)



Errors

FNF_pareto causes a BASIC error if its arguments are not all of type REAL or if the value of *x*0 or *t* is negative or zero.

See Also

P_pareto, Q_pareto

F_rayleigh

Probability density for Rayleigh distribution.

Loading LOADSUB ALL FROM "RAYLEIGH.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL Beta,X,Y
Y=FNF_rayleigh(Beta,X)
```

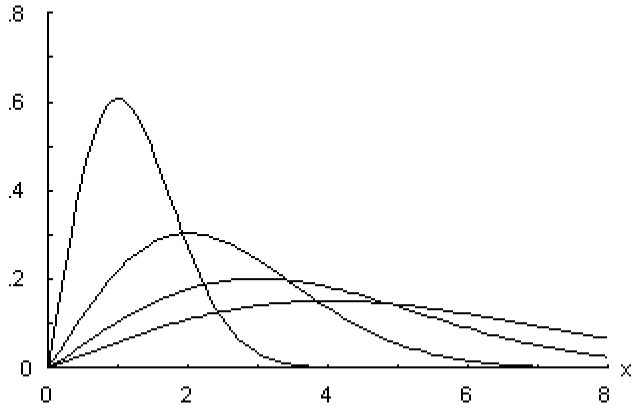
Description

FNF_rayleigh returns the value of the probability density function of the Rayleigh distribution of parameter *Beta* (here written as) at *x*, *f*(*x*;). *F*(*x*;) is defined by the expression

$$f(x;\beta) = \begin{cases} \frac{1}{\beta^2} x e^{\frac{-x^2}{\beta^2}}, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

F is defined for all positive values of .

$f(x;\beta)$



Errors

FNF_rayleigh causes a BASIC error if its arguments are not all of type REAL or if the value of is negative or zero.

See Also

P_rayleigh, Q_rayleigh

F_student

Probability density function for Student's *t* distribution.

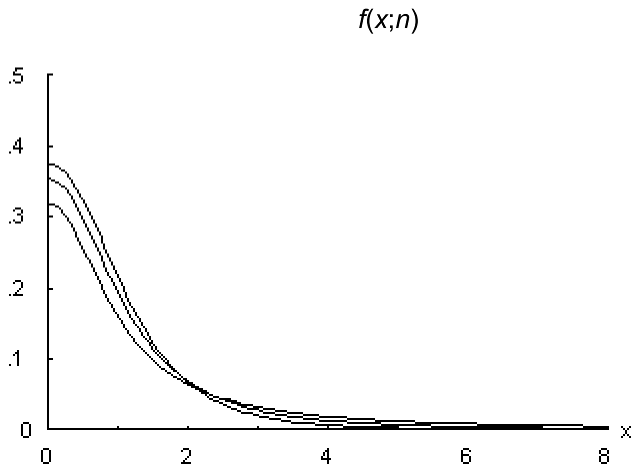
Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNF_student(N,X)
```

Description

FNF_student returns the value of the probability density of the Student's *t* distribution of parameter at *x*, *f*(*x*;*n*). *F* is defined for all positive values of *n*.



Errors

FNF_student causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the value of *n* is negative or zero.

See Also

P_student, Q_student

F_variance

Variance of an array.

Loading LOADSUB ALL FROM "MEAN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A(*),Y
CALL F_Variance(Y,A(*))

Description

FNVariance returns the variance of the elements in the array *A*. The *variance* is the average value of the square of the differences between the elements in the array and the mean value of the elements. This version of the variance uses the number of points in the array *A*, *n*, as the divisor in the averaging calculation, instead of the value *n* - 1 used in some formulas for variance.

Errors

FNVariance causes a BASIC error if its argument is not a REAL array.

See Also

Mean, Std

Gamic

Complementary incomplete gamma function.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A,X,Y
Y=FNGamic(A,X)

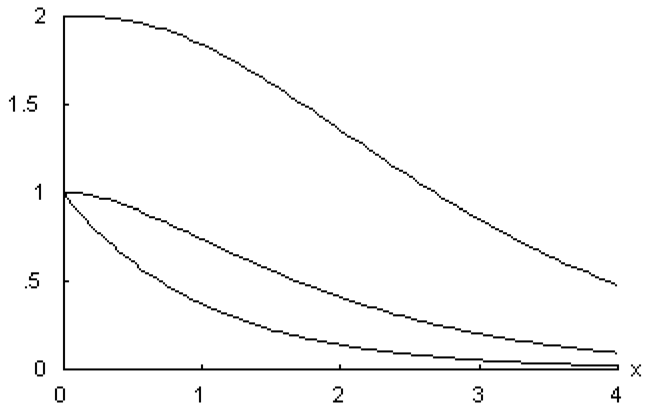
Description

FNGamic returns the value of the complementary incomplete gamma function of *a* and *x*, (*a*,*x*). Although the complementary incomplete gamma function is defined for all values of *x*, this subroutine only works with values of *x* > 0 or *x* = 0 and *a* > 0.

(*a*,*x*) is defined by the expression

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

(*a*,*x*)



Errors

FNGamic causes a BASIC error if its arguments are not all of type REAL, if *x* < 0, or if *x* = 0 and *a* 0.

See Also

Gamit, Gamma, Igamma

Note

The notation (*a*,*x*) causes this function to be confused with the gamma function, (*x*).

Gamit

Tricomi's form of the incomplete gamma function.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A,X,Y
Y=FNGamit(A,X)

Description

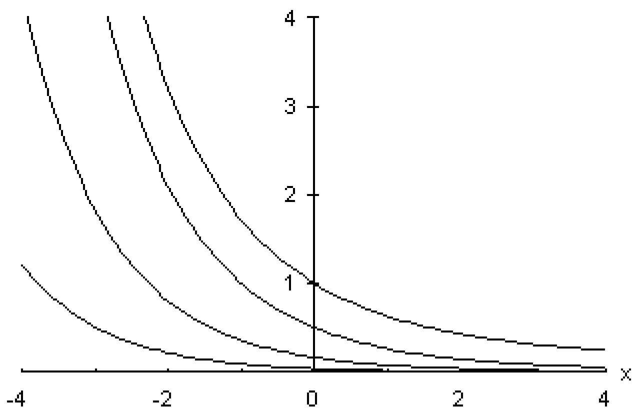
FNGamit returns the value of Tricomi's form of the incomplete gamma function of *a* and *x*, *(*a*,*x*).

*(*a*,*x*) is defined by the expression

$$\gamma^*(a,x) = \frac{x^{-a}}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt.$$

This function is defined for all values of *a* and *x*.

*(*a*,*x*)



Errors

FNGamit causes a BASIC error if its arguments are not all of type REAL.

See Also

Gamic, Gamma, Igamma

Gamma

Gamma function of a real argument.

Loading LOADSUB ALL FROM "GAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNGamma(X)

Description

FNGamma returns the value of the gamma function at *x*, (*x*). This function is defined for all real values of *x* except for *x* zero or a negative integer, at which points (*x*) becomes infinite.

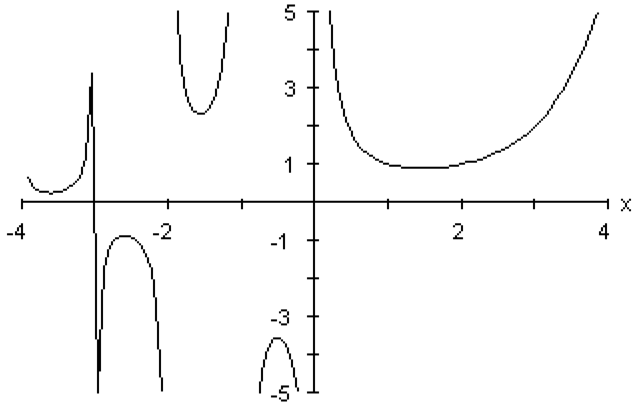
If *x* is equal to a positive integer, *n*, the gamma function is related to the factorial by the relation

$$n! = \Gamma(n+1).$$

This relationship is often used to define a factorial for any real number, by using the definition

$$x! = \Gamma(x+1).$$

(*x*)



Errors

FNGamma causes a BASIC error if its argument is not of type REAL or if *x* 0 and *x* is an integer.

See Also

Fact, Cgamma

H10

Hankel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

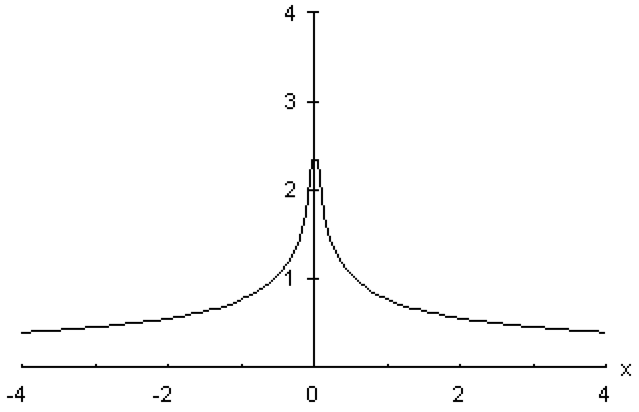
Usage

```
REAL X
COMPLEX C
C=FNH10(X)
```

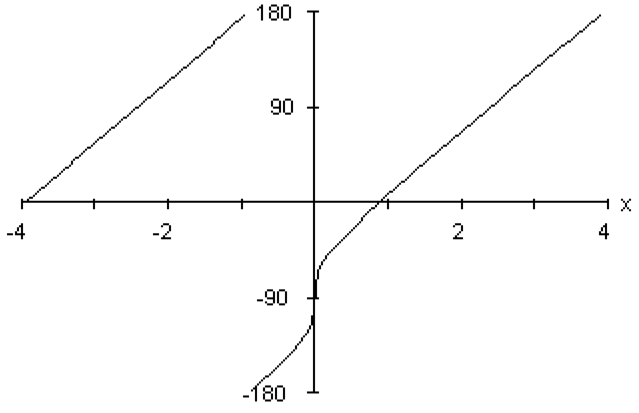
Description

FNH10 returns the value of the Hankel function of the first kind and order zero of *x*, *H*0(1)(*x*). For positive values of *x*, the real component of the value returned contains *J*0(*x*) and the imaginary component contains *Y*0(*x*). For negative values of *x*, the real component contains -*J*0(*x*) and the imaginary component contains *Y*0(*x*).

*H*0(1)(*x*)



Arg[*H*0(1)(*x*)], degrees



Errors

FNH10 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is zero, since the imaginary component of *H*0(1)(0) is -.

See Also

H11, H20, J0, Y0

H11

Hankel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNH11(X)
```

Description

H11 returns the value of the Hankel function of the first kind and order one of *x*, *H*1(1)(*x*). For positive values of *x*, the real component of the value returned contains *J*1(*x*) and the imaginary component contains *Y*1(*x*). For negative values of *x*, the real component contains *J*1(*x*) and the imaginary component contains -*Y*1(*x*).

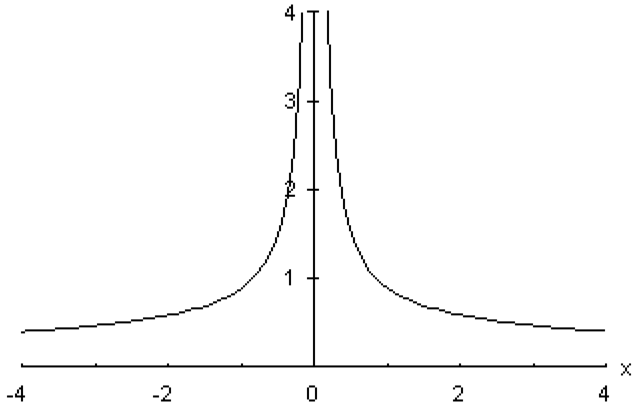
Errors

H11 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *H*1(1)(0) is -.

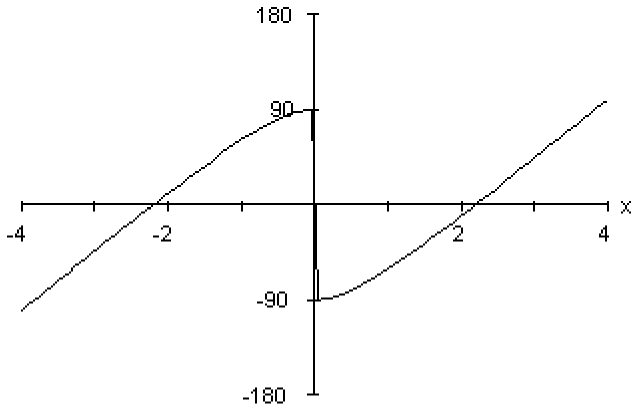
See Also

H10, H21, J1, Y1

*H*1(1)(*x*)



Arg[*H*1(1)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

H1n

Hankel function of the first kind, order *n*.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNH1n(N,X)
```

Description

FNH1n returns the value of the Hankel function of the first kind and order *n* of *x*, *Hn*(1)(*x*). For positive values of *x*, the real component of the value returned contains *Jn*(*x*) and the imaginary component contains *Yn*(*x*). For negative values of *x*, the real component contains (-1)*n*+1*Jn*(*x*) and the imaginary component contains (-1)*nYn*(*x*).

Errors

FNH1n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *Hn*(1)(0) is -.

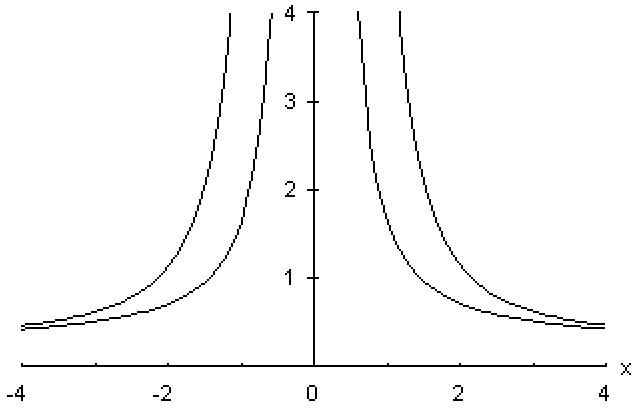
See Also

H10, H11, H2n, J0, J1, Y0, Y1

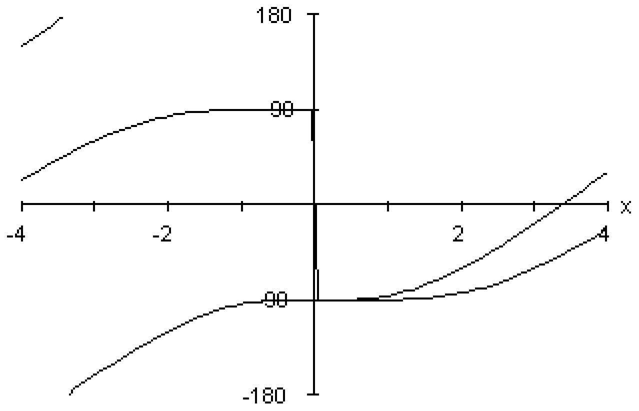
Note

The algorithm used computes the value of *Hn*(1) using a recursion from the values of *H*0(1) and *H*1(1). The computation time increases with *n* and the computation accuracy decreases with *n*.

Hn(1)(*x*)



Arg[*Hn*(1)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

H20

Hankel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

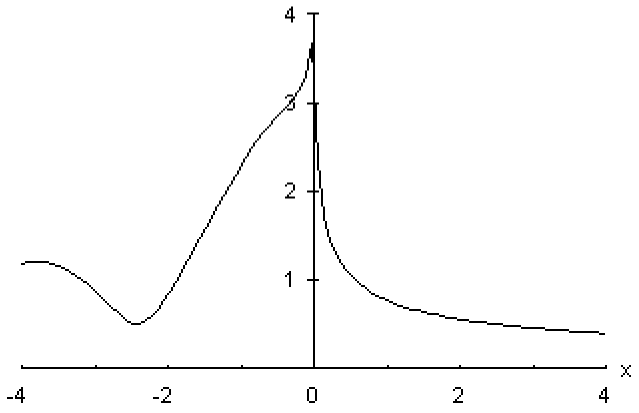
Usage

```
REAL X
COMPLEX C
C=FNH20(X)
```

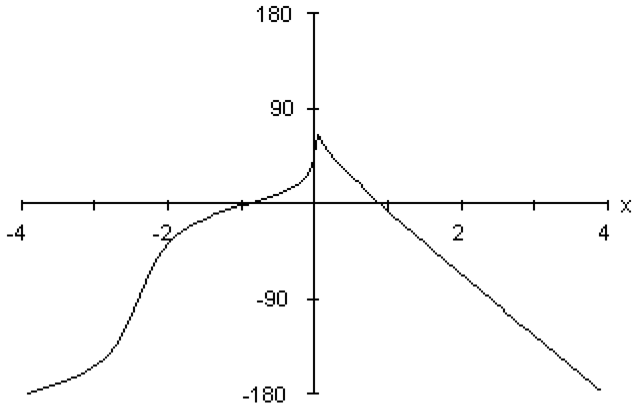
Description

FNH20 returns the value of the Hankel function of the second kind and order zero of *x*, *H*0(2)(*x*). For positive values of *x*, the real component of the value returned contains *J*0(*x*) and the imaginary component contains *Y*0(*x*). For negative values of *x*, the real component contains 3*J*0(*x*) and the imaginary component contains -*Y*0(*x*).

*H*0(2)(*x*)



Arg[*H*0(2)(*x*)], degrees



Errors

FNH20 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *H*0(2)(0) is -.

See Also

H10, H21, J0, Y0

H21

Hankel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

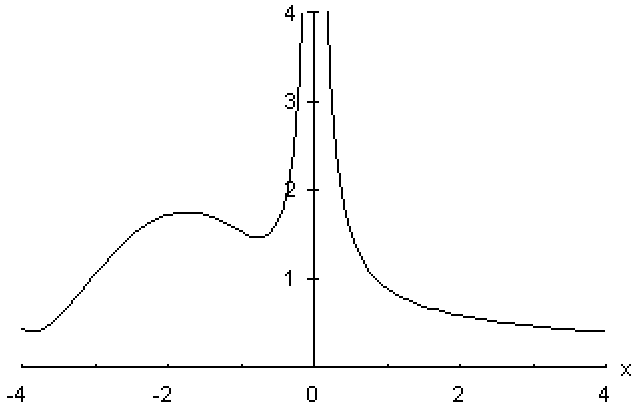
Usage

```
REAL X
COMPLEX C
C=FNH21(X)
```

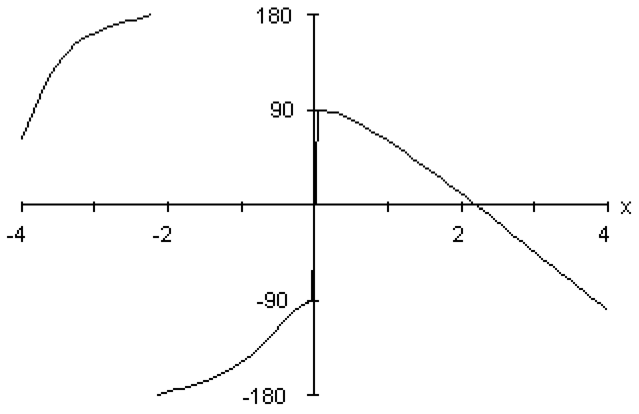
Description

FNH21 returns the value of the Hankel function of the second kind and order one of *x*, *H*1(2)(*x*). For positive values of *x*, the real component of the value returned contains *J*1(*x*) and the imaginary component contains *Y*1(*x*). For negative values of *x*, the real component contains -3*J*1(*x*) and the imaginary component contains *Y*1(*x*).

*H*1(2)(*x*)



Arg[*H*1(2)(*x*)], degrees



Errors

FNH21 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *H*1(2)(0) is -.

See Also

H11, H20, J1, Y1

H2n

Hankel function of the second kind, order *n*.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNH2n(N,X)
```

Description

FNH2n returns the value of the Hankel function of the second kind and order *n* of *x*, *Hn*(2)(*x*). For positive values of *x*, the real component of the value returned contains *Jn*(*x*) and the imaginary component contains *Yn*(*x*). For negative values of *x*, the real component contains (-1)*n*+13*Jn*(*x*) and the imaginary component of *C* contains (-1)*n*+1*Yn*(*x*).

Errors

FNH2n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *Hn*(2)(0) is infinite.

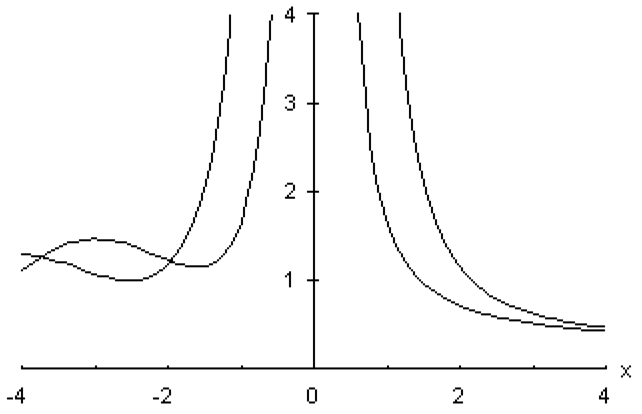
See Also

H1n, H20, H21, J0, J1, Y0, Y1

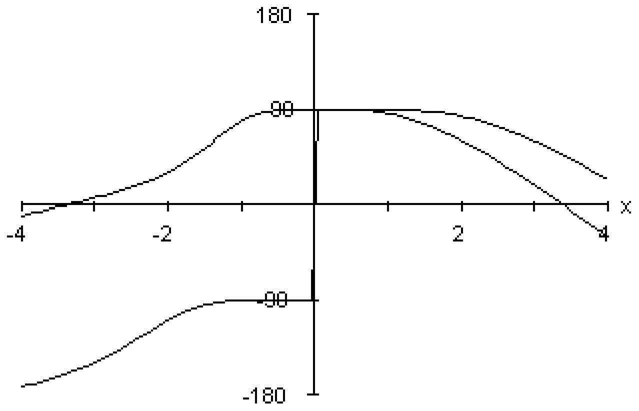
Note

The algorithm used computes the value of *Hn*(2) using a recursion from the values of *H*0(2) and *H*1(2). The computation time increases with *n* and the computation accuracy decreases with *n*.

Hn(2)(*x*)



Arg[*Hn*(2)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

Hh1n

Hankel function of the first kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNHh1n(N,X)
```

Description

FNHh1n returns the value of the cylindrical Hankel function of the first kind and order *n*+½ of *x*, *Hn*+½(1)(*x*). *Hn*+½(1)(*x*) is defined for all values of *n* and for all positive values of *x*.

Hn+½(1)(*x*) is sometimes also called the *cylindrical Bessel function of the third kind, order* n+½.

Errors

FNHh1n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

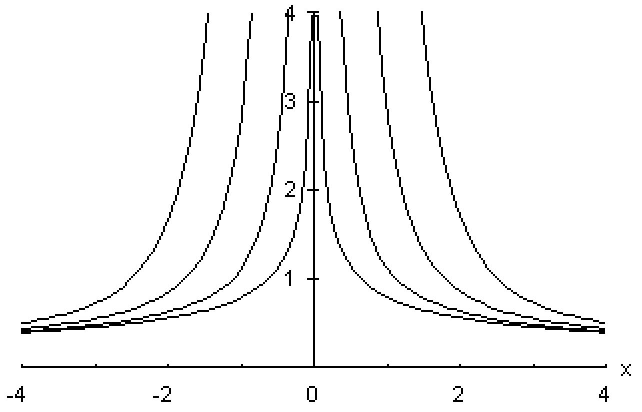
See Also

Hh2n

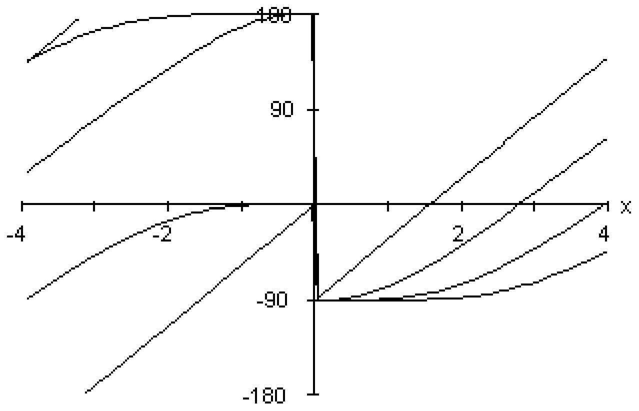
Note

The algorithm used computes the value of *Hn*+½(1) using a recursion from the values of *H*½(1) and *H*1½(1). The computation time increases with *n* and the computation accuracy decreases with *n*.

Hn+½(1)(*x*)



Arg[*Hn*+½(1)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

Hh2n

Hankel function of the second kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNHh2n(N,X)
```

Description

Hh2n returns the value of the cylindrical Hankel function of the second kind and order *n*+½ of *x*, $H_{n+½}^{(2)}(x)$. $H_{n+½}^{(2)}(x)$ is defined for all values of *n* and for all positive values of *x*.

Errors

Hh2n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

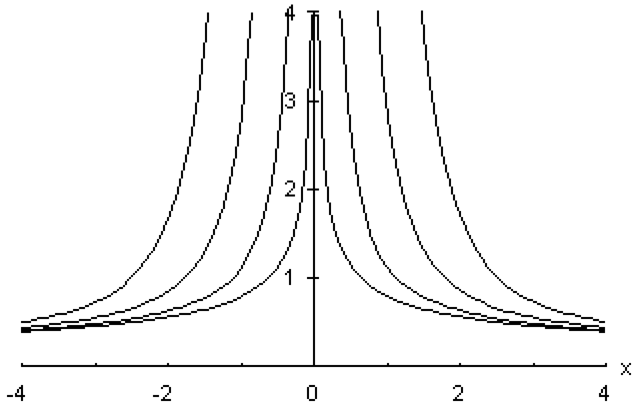
See Also

Hh1n

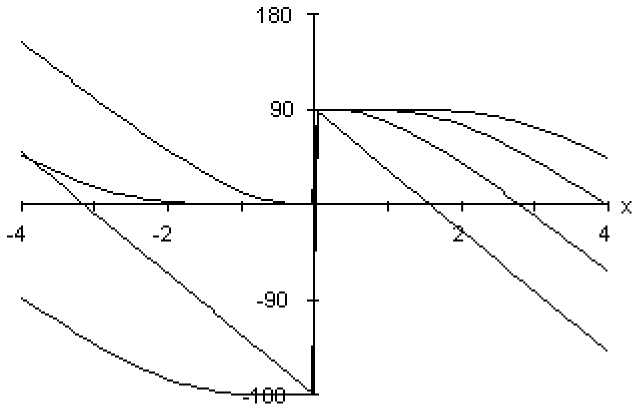
Note

The algorithm used computes the value of $H_{n+½}^{(2)}$ using a recursion from the values of $H_{½}^{(2)}$ and $H_{1½}^{(2)}$. The computation time increases with *n* and the computation accuracy decreases with *n*.

$H_{n+½}^{(2)}(x)$



Arg[$H_{n+½}^{(2)}(x)$], degrees


180
90
-90
-180
-4
-2
2
4
x

Histogram

Histogram of a real array.

Loading

```
LOADSUB ALL FROM "HIST.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Histogram FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N,Hist(*)
REAL Xmin,Xmax,A(*)
CALL Histogram(A(*),Xmin,Xmax,N,Hist(*))
```

Description

Histogram divides the region of values between *xmin* and *xmax* into *n* equal intervals and counts the number of elements in the array *A* whose values lie in each interval. If a value in *A* lies below *xmin* or at or above *xmax*, it is not counted. If a value is exactly the value that separates two intervals, it is counted in the higher of the two intervals.

Histogram counts in the following manner: Let *s* be the width of an interval in which values are counted. *S* is defined by the expression

$$s = \frac{x_{max} - x_{min}}{n}.$$

If a value in *A* falls between *xmin*, inclusive, and *xmin*+*s*, exclusive, the count in the first element of *Hist* is increased. If the value falls between *xmin*+*s*, inclusive, and *xmin*+2*s*, exclusive, the count in the second element of *Hist* is increased, etc.

Hist must contain at least *n* elements.

Errors

Histogram causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *xmin xmax*, if *n* is negative or zero, or if *Hist* contains fewer than *n* elements.

Hn

Hermite polynomial.

Loading LOADSUB ALL FROM "HERMITE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNHn(N,X)
```

Description

FNHn returns the value of the Hermite polynomial of order *n* of *x*, *Hn*(*x*). *N* must be positive or zero.

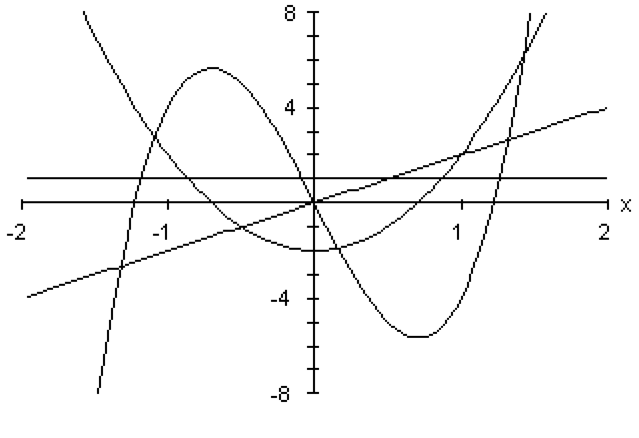
Errors

FNHn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *n* is negative, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

Note

For *n* > 12, the algorithm used computes the value of *Hn* using a recursion from the values of *H*11 and *H*12. The computation time increases with *n*-11 and the computation accuracy decreases with *n*-11.

Hn(*x*)



I0

Modified Bessel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNI0(X)

Description

FNI0 returns the value of the modified cylindrical Bessel function of the first kind and order zero of *x*, *I*0(*x*). *I*0(*x*) is defined for all values of *x*, but large absolute values of *x* may cause the result to be larger than MAXREAL, the largest value representable.

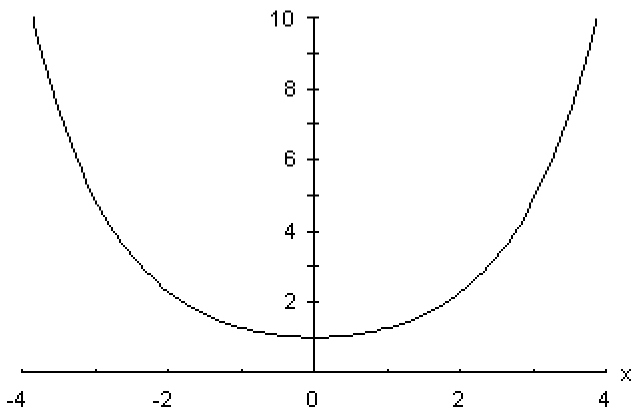
Errors

FNI0 causes a BASIC error if its argument is not of type REAL or if the result would be larger than MAXREAL.

See Also

I1, In, K0

*I*0(*x*)



I0e

Scaled modified Bessel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNI0e(X)

Description

FNI0e returns the value of the modified cylindrical Bessel function of the first kind and order zero of *x* scaled by $e^{-|x|}$, $e^{-|x|}I0(x)$. The scaling is done so that the value of $I0(x)$ can be evaluated for arguments of large absolute value, where the value of $I0(x)$ may be larger than MAXREAL, the largest value representable. The value of $e^{-|x|}I0(x)$ is moderate for arguments of large absolute value.

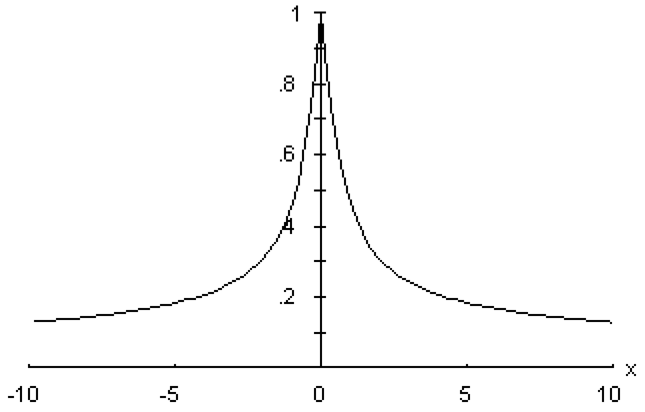
Errors

FNI0e causes a BASIC error if its argument is not of type REAL.

See Also

I0, I1e, In

$e^{-|x|}I0(x)$



I1

Modified Bessel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNI1(X)

Description

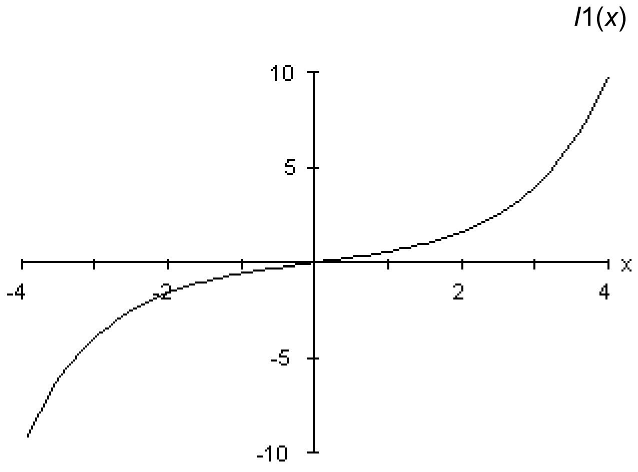
FNI1 returns the value of the modified cylindrical Bessel function of the first kind and order one of *x*, *I*1(*x*). *I*1(*x*) is defined for all values of *x*, but large absolute values of *x* may cause the magnitude of the result to be larger than MAXREAL, the largest value representable.

Errors

FNI1 causes a BASIC error if its argument is not of type REAL or if the magnitude of the result would be larger than MAXREAL.

See Also

I0, I1e, In, K0



I1e

Scaled modified Bessel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNI1e(X)

Description

FNI1e returns the value of the modified cylindrical Bessel function of the first kind and order one of *x* scaled by *e*-|x|, *e*-|x|*I*1(*x*). The scaling is done so that the value of *I*1(*x*) can be evaluated for arguments of large absolute value, where the absolute value of *I*1(*x*) may be larger than MAXREAL, the largest value representable. The absolute value of *e*-|x|*I*1(*x*) is moderate for arguments of large absolute value.

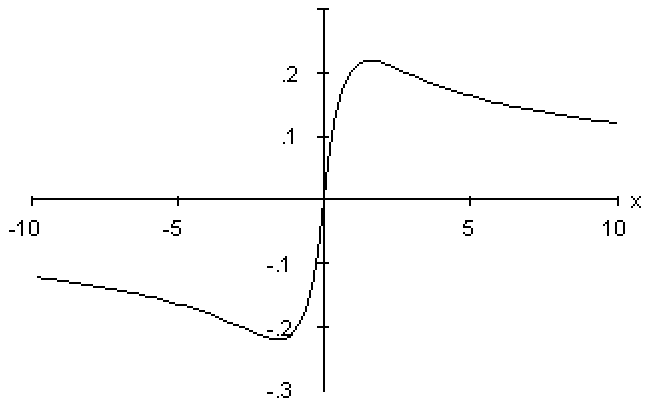
Errors

FNI1e causes a BASIC error if its argument is not of type REAL.

See Also

I0e, I1, In

e-|x|*I*1(*x*)



Ibeta

Incomplete beta function.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A,B,X,Y
Y=FNIbeta(A,B,X)

Description

Ibeta returns the value of the incomplete beta function of *a*, *b*, and *x*, *Bx*(*a*,*b*). This function is only defined for values of *a* 0, *b* 0, and 0 *x* 1.

Bx(*a*,*b*) is defined by the expression

$$B_x(a,b) = \int_0^x t^{a-1}(1-t)^{b-1}dt.$$

Another form of the incomplete beta function, *Ix*(*a*,*b*), is defined by the expression

$$I_x(a,b) = \frac{1}{B(a,b)}\int_0^x t^{a-1}(1-t)^{b-1}dt.$$

This function is evaluated by the routine *P_beta*.

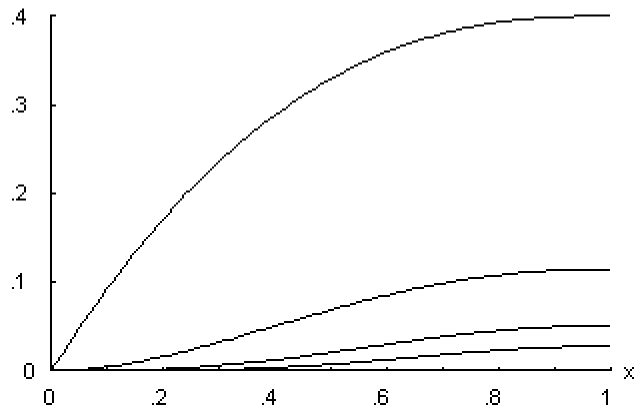
Errors

Ibeta causes a BASIC error if its arguments are not all of type REAL, if *x* < 0, *x* > 1, *a* 0, or *b* 0.

See Also

P_beta

Bx(*a*,2.5)



Icfft

Complex discrete inverse Fourier transform.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Icfft FROM "MATHLIB.HTS"

Usage

```
INTEGER Logn
COMPLEX A(*),F(*)
CALL Icfft(Logn,F(*),A(*))
```

Description

Icfft calculates the discrete inverse Fourier transform of the sequence in the array F and stores the result in the array *A*. *Logn* is the base-2 log of the number of points in the sequences. The arrays *A* and *F* must contain at least 2*Logn* elements; if they have more than this number of elements, the extra elements are ignored and unmodified. The number of elements denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

The meaning of the values input to *Icfft* is the same as that for the values output by the *Cfft* routine; see the entry for the *Cfft* routine for a detailed explanation of the meaning of the values input to *Icfft*.

Errors

Icfft causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *C* is smaller than 2*Logn*.

See Also

Cfft, Ifft

Ifft

Discrete inverse Fourier transform.

Loading

```
LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Ifft FROM "MATHLIB.HTS"
```

Usage

```
INTEGER Logn
REAL A(*)
COMPLEX F(*)
CALL Ifft(Logn,F(*),A(*))
```

Description

Ifft calculates the discrete inverse Fourier transform of the sequence in the array *F* and stores the result in the array *A*. *Logn* is the base-2 log of the number of points in the sequences. The array *F* must contain at least 2*Logn*-1 elements; the array *A* must contain at least 2*Logn* elements. If the arrays have more than the required number of elements, the extra elements are ignored and unmodified. The number of elements in *A* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

The values input to *Ifft* are in the same format as those output by the *Fft* routine. See the entry for *Fft* for a detailed explanation of the meaning of the values returned by *Ifft*.

Errors

Ifft causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *F* is smaller than the values explained above.

See Also

Fft, Icfft

Igamma

Incomplete gamma function.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL A,X,Y
Y=FNIgamma(A,X)
```

Description

FNIgamma returns the value of the incomplete gamma function of *a* and *x*, $\gamma(a,x)$. Although the incomplete gamma function is defined for all values of *x*, this subroutine only works with values of $x > 0$ or $x = 0$ and $a > 0$.

$\gamma(a,x)$ is defined by the expression

$$\gamma(a,x) = \int_0^x e^{-t} t^{a-1} dt .$$

In the above expression, *x* must be positive or zero and *a* must be positive.

When $x \geq 0$ and $a > 0$, the other forms of the incomplete gamma function present in this subroutine library, the complementary form, $\Gamma(a,x)$ and Tricomi's form, $\gamma^*(a,x)$, are related to the incomplete gamma function by the following expressions:

$$\gamma^*(a,x) = \frac{x^{-a}}{\Gamma(a)}\gamma(a,x)$$

$$\Gamma(a,x) = \Gamma(a) - \gamma(a,x) .$$

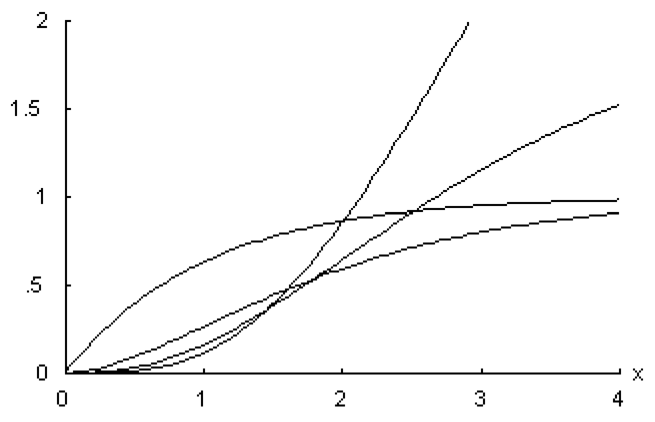
Errors

FNIgamma causes a BASIC error if its arguments are not both of type REAL, if $x < 0$, or if $x = 0$ and $a \leq 0$.

See Also

Gamma, Gamic, Gamit

(*a*,*x*)



Ihn

Modified Bessel function of the first kind, order *n*+½.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNIhn(N,X)
```

Description

FNIhn returns the value of the modified cylindrical Bessel function of the first kind and order *n*+½ of *x*, $I_{n+½}(x)$. $I_{n+½}(x)$ is defined for all values of *n* and all positive values of *x*. If *n* is positive or zero, $I_{n+½}(x)$ is also defined for *x* = 0.

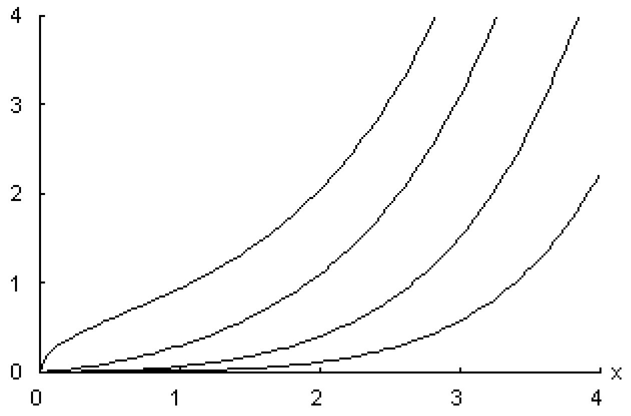
Errors

FNIhn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is out of the range of definition explained above.

See Also

In, Khn

$I_{n+½}(x)$



Note The algorithm used computes the value of $I_{n+½}$ using a recursion from the values of $I_{½}$ and $I_{1½}$. The computation time increases with *n* and the computation accuracy decreases with *n*.

In

Modified Bessel function of the first kind, order *n*.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNIn(N,X)
```

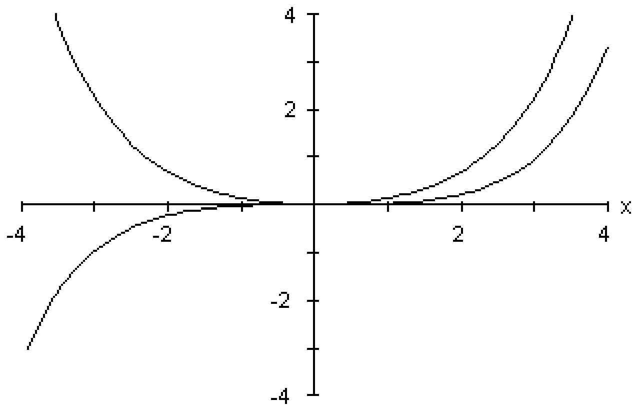
Description

FNIn returns the value of the modified cylindrical Bessel function of the first kind and order *n* of *x*, *In*(*x*). *In*(*x*) is defined for all values of *n* and all values of *x*, but large absolute values of *x* may cause the absolute value of the result to be larger than MAXREAL, the largest value representable.

Errors

FNIn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the result would be larger than MAXREAL.

In(*x*)



See Also I0, I1e, In, K0

Note

The algorithm used computes the value of *In* using a recursion from the values of *I*0 and *I*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Ine

Scaled modified Bessel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNIne(N,X)
```

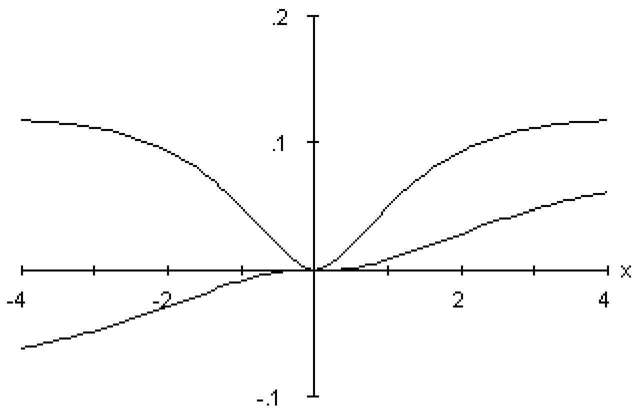
Description

FNIne returns the value of the modified cylindrical Bessel function of the first kind and order *n* of *x* scaled by *e*|x|, *e*-|x| *In*(*x*). The scaling is done so that the value of *In*(*x*) can be evaluated for arguments of large absolute value, where the absolute value of *In*(*x*) may be larger than MAXREAL, the largest value representable. The absolute value of *e*-|x| *In*(*x*) is moderate for arguments of large absolute value.

Errors

FNIne causes a BASIC error if its arguments are not of the types listed in the USAGE section, above.

e-|x| *In*(*x*)



See Also I0, I0e, In, K0

Note The algorithm used computes the value of *In* using a recursion from the values of *I*0 and *I*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Invgamma

Inverse of the gamma function.

Loading LOADSUB ALL FROM "GAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNInvgamma(X)
```

Description

FNInvgamma returns the value of $1/(x)$. Unlike (x), $1/(x)$ is defined at negative integral values of x and at $x = 0$. This function is provided for use in expressions that involve dividing by (x), where the regular gamma function, *FNGamma*, would cause Errors when evaluated at values of x equal to negative integers or zero.

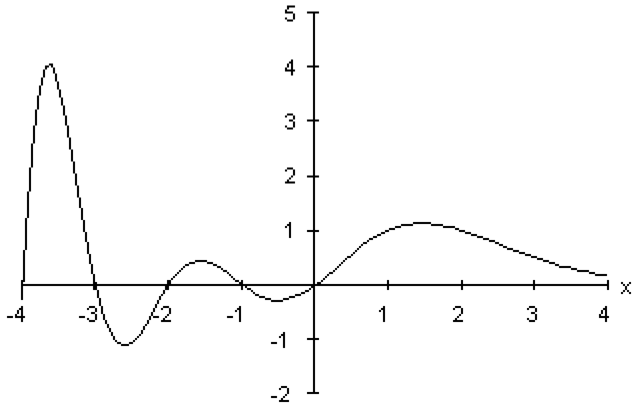
Errors

FNInvgamma causes a BASIC error if its argument is not of type REAL.

See Also

Gamma

$1/(x)$



J0

Bessel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNJ0(X)

Description

FNJ0 returns the value of the cylindrical Bessel function of the first kind and order zero of *x*, *J*0(*x*). *J*0(*x*) is defined for all values of *x*.

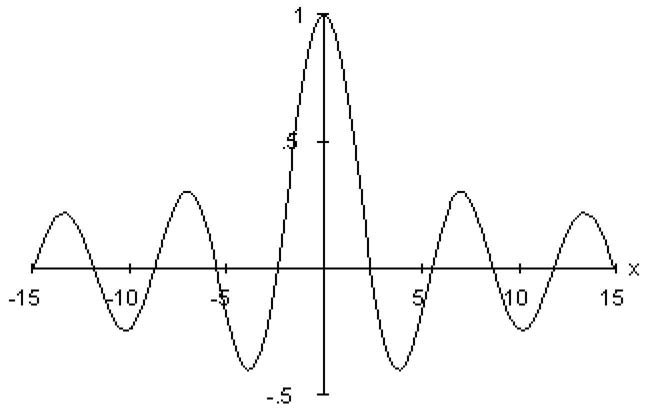
Errors

FNJ0 causes a BASIC error if its argument is not of type REAL.

See Also

J1, Jn, Y0

*J*0(*x*)



J1

Bessel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNJ1(X)

Description

FNJ1 returns the value of the cylindrical Bessel function of the first kind and order one of *x*, *J*1(*x*). *J*1(*x*) is defined for all values of *x*.

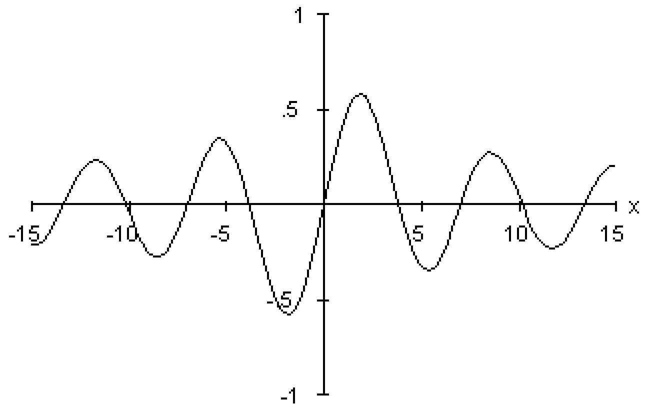
Errors

FNJ1 causes a BASIC error if its argument is not of type REAL.

See Also

J0, Jn, Y0

*J*1(*x*)



Jhn

Bessel function of the first kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNJhn(N,X)
```

Description

FNJhn returns the value of the cylindrical Bessel function of the first kind and order *n*+½ of *x*, *Jn*+½(*x*). *Jn*+½(*x*) is defined for all values of *n* and for all positive values of *x*. If *n* is positive or zero, *Jn*+½(*x*) is also defined for *x* = 0.

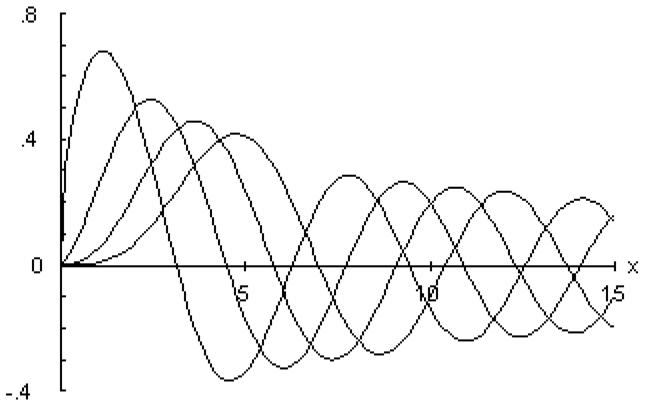
Errors

FNJhn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is out of the range of definition explained above.

See Also

Hh1n, Hh2n, Jn, Yhn

Jn+½(*x*)



The algorithm used computes the value of *Jn*+½ using a recursion from the values of *J*½ and *J*1½. The computation time increases with *n* and the computation accuracy decreases with *n*.

Jn

Bessel function of the first kind, order *n*.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNJn(N,X)
```

Description

FNJn returns the value of the cylindrical Bessel function of the first kind and order *n* of *x*, *Jn*(*x*). *Jn*(*x*) is defined for all values of *n* and *x*.

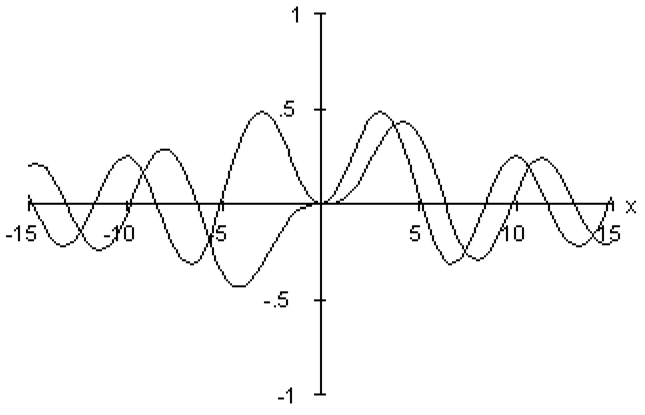
Errors

FNJn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above.

Note

The algorithm used computes the value of *Jn* using a recursion from the values of *J*0 and *J*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Jn(*x*)



K0

Modified Bessel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNK0(X)
```

Description

FNK0 returns the value of the modified cylindrical Bessel function of the second kind and order zero of *x*, *K*0(*x*). *K*0(*x*) is defined for all positive values of *x*, but values of *x* near zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable.

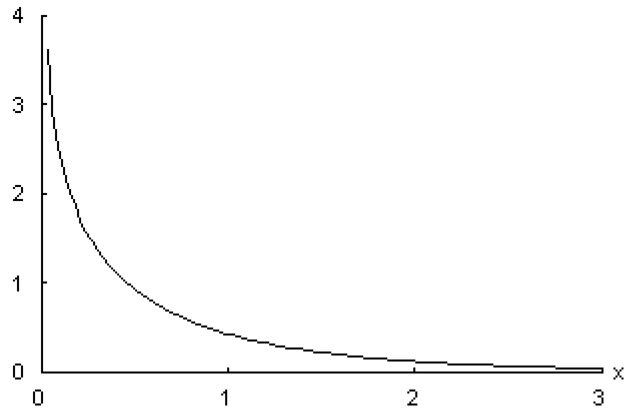
Errors

FNK0 causes a BASIC error if its arguments is not of type REAL, if *x* is not positive, or if the result would be larger than MAXREAL.

See Also

I0, K0e, K1, Kn

*K*0(*x*)



K0e

Scaled modified Bessel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNK0e(X)

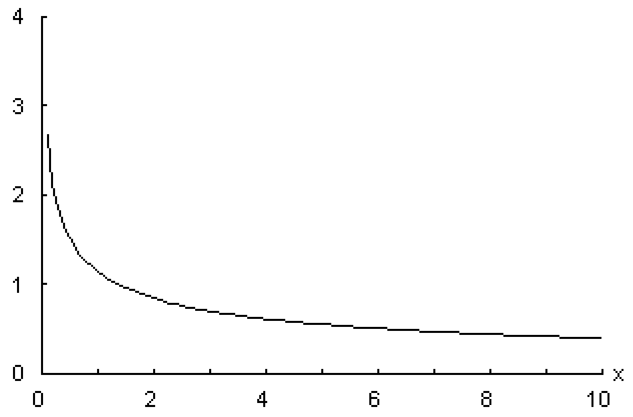
Description

FNK0e returns the value of the modified cylindrical Bessel function of the second kind and order zero of *x* scaled by *ex*, *exK*0(*x*). Although *exK*0(*x*) is defined for all positive values of *x*, values of *x* near zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable. The scaling is done so that the value of *K*0(*x*) can be evaluated for large arguments, where the value of *K*0(*x*) may be near zero.

Errors

FNK0e causes a BASIC error if its argument is not of type REAL, if *x* is not positive, or if the result would be larger than MAXREAL.

*exK*0(*x*)



See Also K0, K1e, Kn

K1

Modified Bessel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNK1(X)

Description

FNK1 returns the value of the modified cylindrical Bessel function of the second kind and order one of *x*, *K*1(*x*). *K*1(*x*) is defined for all positive values of *x*, but values of *x* near zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable.

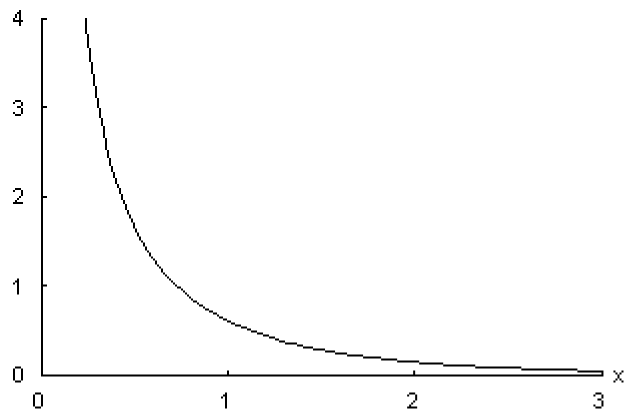
Errors

FNK1 causes a BASIC error if its argument is not of type REAL, if *x* is not positive, or if the result would be larger than MAXREAL.

See Also

I1, K0, K1e, Kn

*K*1(*x*)



K1e

Scaled modified Bessel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNK1e(X)
```

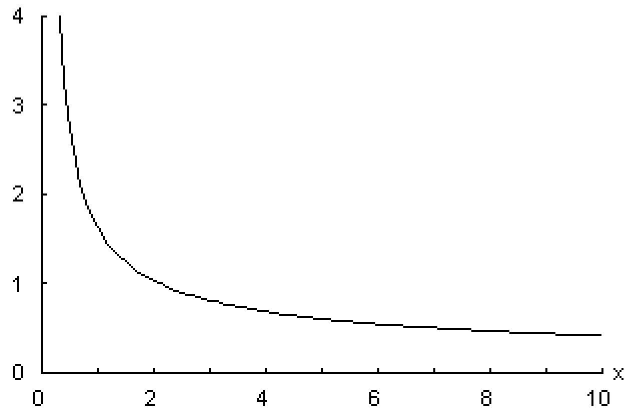
Description

K1e returns the value of the modified cylindrical Bessel function of the second kind and order one of *x* scaled by *ex*, *exK*1(*x*). Although *exK*1(*x*) is defined for all positive values of *x*, values of *x* near zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable. The scaling is done so that the value of *K*1(*x*) can be evaluated for large arguments, where the absolute value of *K*1(*x*) may be near zero.

Errors

K1e causes a BASIC error if its argument is not of type REAL, if *x* is not positive, or if the result would be larger than MAXREAL.

*exK*1(*x*)



See Also K0e, K1, Kn

Ke

Complex Kelvin function of the second kind of a real argument.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNKe(X)
```

Description

FNKe returns the values of the real and imaginary Kelvin functions of the second kind of the value *x*. The real part of the value returned is the value of ker(*x*) and the imaginary part is the value of kei(*x*). Although ker(*x*) and kei(*x*) are defined for all values of *x* except zero, values of *x* near zero may produce results greater than MAXREAL, the largest value representable.

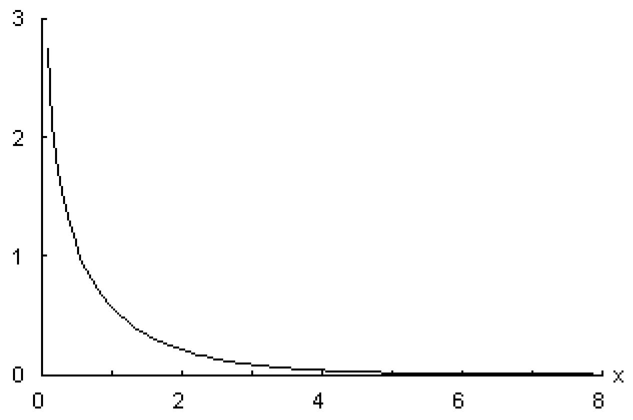
Errors

FNKe causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if *x* is zero or the value of either the real or imaginary component of the value returned would be larger than MAXREAL.

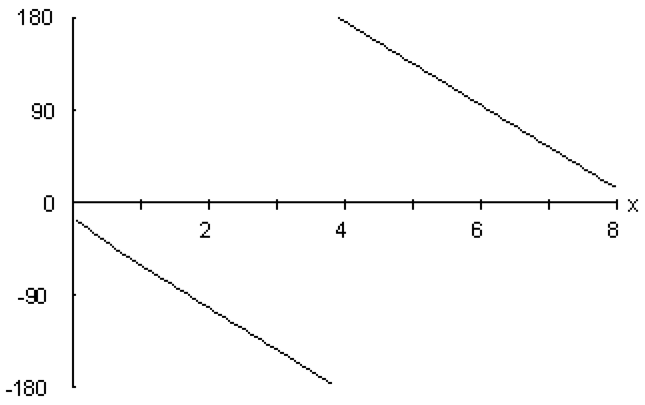
See Also

Ker, Kei, Be

ker(*x*) + *i*kei(*x*)



Arg[ker(*x*) + *i*kei(*x*)], degrees


180
90
0
-90
-180
2
4
6
8
x

Kei

Imaginary Kelvin function of the second kind of a real argument.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNKei(X)

Description

FNKei returns the value of the imaginary Kelvin function of the second kind of the value *x*. Although kei(*x*) is defined for all values of *x* except zero, values of *x* near zero may produce results greater than MAXREAL, the largest value representable. Note that sometimes Kelvin functions are written with an order, as in kei3(*x*). In this notation, the function *FNKei* returns the value of kei0(*x*).

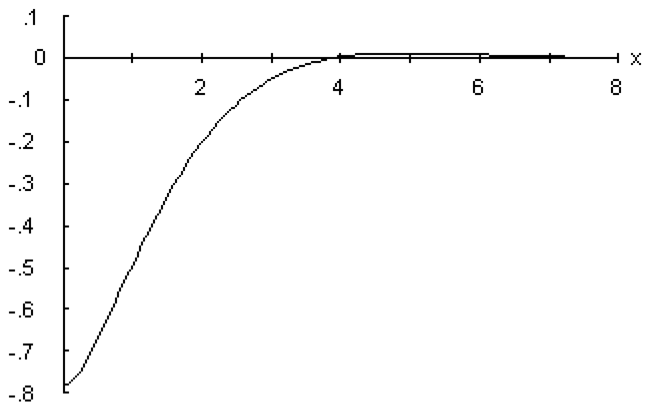
Errors

FNKei causes a BASIC error if its argument is not of type REAL, if *x* is negative or zero, or if the value returned would be larger than MAXREAL.

See Also

Ke, Ker

kei(*x*)



Ker

Real Kelvin function of the second kind.

Loading LOADSUB ALL FROM "KELVIN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNKer(X)

Description

FNKer returns the value of the real Kelvin function of the second kind of the value *x*. Although ker(*x*) is defined for all values of *x* except zero, values of *x* near zero may produce results greater than MAXREAL, the largest value representable. Note that sometimes Kelvin functions are written with an order, as in ker3(*x*). In this notation, the function *FNKer* returns the value of ker0(*x*).

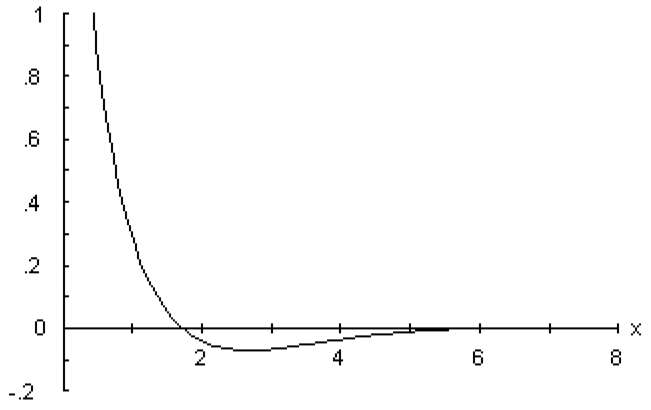
Errors

FNKer causes a BASIC error if its argument is not of type REAL, if *x* is negative or zero, or if the value returned would be larger than MAXREAL.

See Also

Ke, Ker

ker(*x*)



Khn

Modified Bessel function of the second kind, order *n*+½.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNKhn(N,X)
```

Description

FNKhn returns the value of the modified cylindrical Bessel function of the second kind and order *n*+½ of *x*, *Kn*+½(*x*). *Kn*+½(*x*) is defined for all values of *n* and for all positive values of *x*.

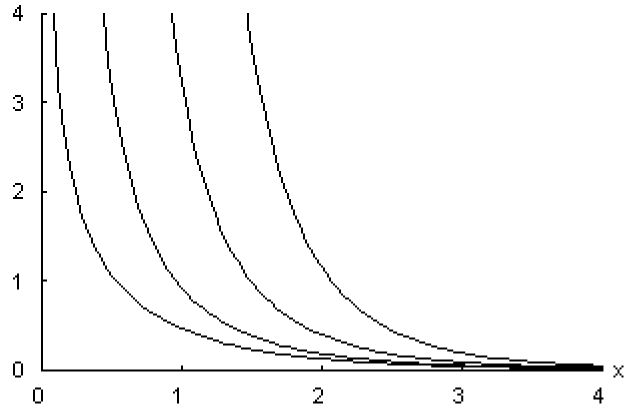
Errors

FNKhn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

See Also

Ihn, Kn

Kn+½(*x*)



Note The algorithm used computes the value of *Kn*+½ using a recursion from the values of *K*½ and *K*1½. The computation time increases with *n* and the computation accuracy decreases with *n*.

Kn

Modified Bessel function of the second kind, order *n*.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNKn(N,X)
```

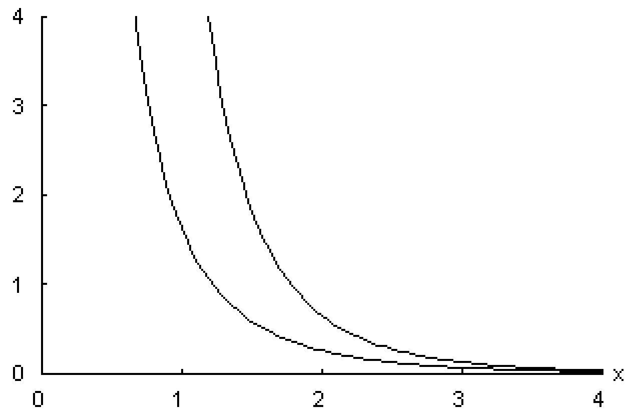
Description

FNKn returns the value of the modified cylindrical Bessel function of the second kind and order *n* of *x*, *Kn*(*x*). *Kn*(*x*) is defined for all values of *n* and all positive values of *x*, but values of *x* close to zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable.

Errors

FNKn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *x* is not positive, or if the result would be larger than MAXREAL.

Kn(*x*)



See Also In, K0, K1, Kn, Kne

Note

The algorithm used computes the value of *K*n using a recursion from the values of *K*0 and *K*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Kne

Scaled modified Bessel function of the second kind, order *n*.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNKne(N,X)
```

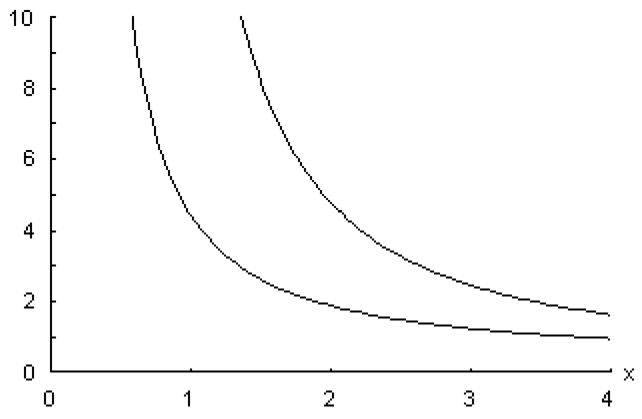
Description

FNKne returns the value of the modified cylindrical Bessel function of the second kind and order *n* of *x* scaled by *ex*, *exKn*(*x*). Although *exKn*(*x*) is defined for all positive values of *x*, values of *x* near zero may cause the absolute value of the result to be larger than MAXREAL, the largest value representable. The scaling is done so that the value of *Kn*(*x*) can be evaluated for arguments of large absolute value, where the value of *Kn*(*x*) may be near zero.

Errors

FNKne causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *x* is not positive, or if the result would be larger than MAXREAL.

exKn(*x*)



See Also I0e, In, K0e, K1e, Kn

Note

The algorithm used computes the value of *Kn* using a recursion from the values of *K*0 and *K*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Li

Log integral.

Loading LOADSUB ALL FROM "LI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNLi(X)

Description

FNLi returns the value of the log integral of *x*, Li(*x*). Li(*x*) is defined by the relation

$$\mathrm{Li}(x) = -\int_0^x \frac{1}{\log(t)}\,dt.$$

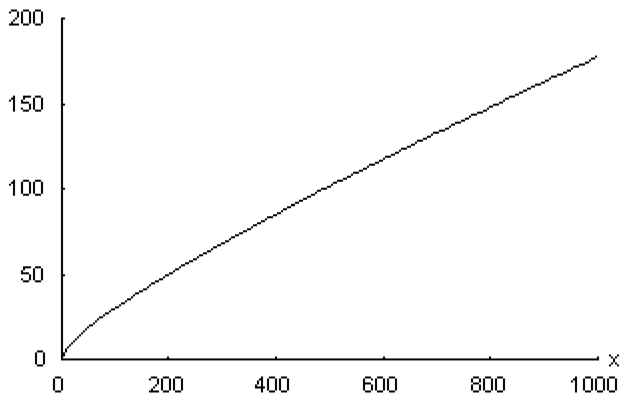
The path of integration must not include the points *x* = 0 and *x* = 1 and must not cross the real axis.

The real version of Li(*x*) is only defined for *x* positive or zero.

Errors

FNLi causes a BASIC error if its argument is not of type REAL or if it is negative.

Li(*x*)



See Also Ci, Ei, Si

Logbeta

Logarithm of the beta function of a real argument.

Loading LOADSUB ALL FROM "GAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A,B,Y
Y=FNLogbeta(A,B)

Description

FNLogbeta returns the value of the logarithm of the beta function of *a* and *b*, log[*B*(*a*,*b*)]. *B*(*a*,*b*) is defined as (*a*)(*b*)/(*a*+*b*) (see *Gamma*). *B*(*a*,*b*) is only defined for *a* > 0 and *b* > 0.

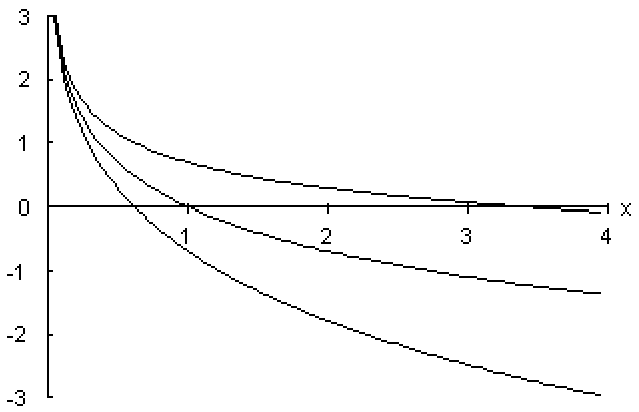
Errors

FNLogbeta causes a BASIC error if its arguments are not both of type REAL or if either *a* or *b* is negative or zero.

See Also

Beta, Clogbeta, Gamma

log[*B*(*a*,*b*)]



Loggamma

Logarithm of the gamma function of a real argument.

Loading LOADSUB ALL FROM "GAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNLoggamma(X)

Description

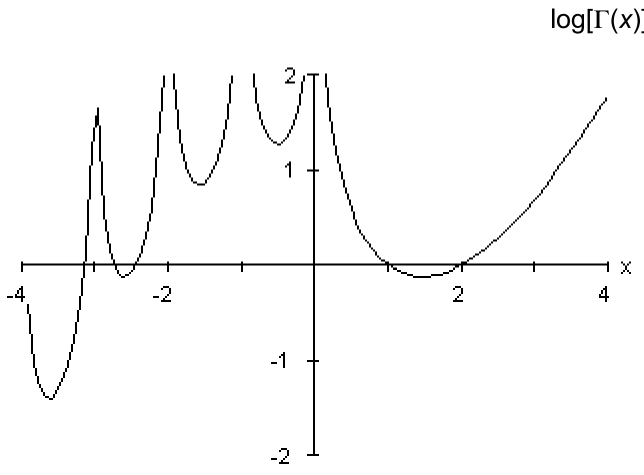
FNLoggamma returns the value of the logarithm of the gamma function of x, $\log[\Gamma(x)]$. The value of $\log[\Gamma(x)]$ approaches ± as x approaches a negative integer or zero.

Errors

FNLoggamma causes a BASIC error if its argument is not of type REAL or if the magnitude of $\log[\Gamma(x)]$ exceeds MAXREAL, the largest number representable.

See Also

Cgamma, Gamma, Loggamma



Mean

Mean of values in an array.

Loading

```
LOADSUB ALL FROM "MEAN.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),Y
Y=FNMean(A(*))
```

Description

FNMean returns the mean value of the elements in the array *A*. The *mean* is the average value of the elements in the array, which is computed by adding the values of the elements in the array and dividing the sum by the number of elements.

Errors

FNMean causes a BASIC error if its argument is not a REAL array.

See Also

Median, Std, F_variance

Median

Median of values in an array.

Loading

LOADSUB ALL FROM "MEAN.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB FNMedian FROM "MATHLIB.HTS"

Usage

REAL A(*),X
X=FNMedian(A(*))

Description

FNMedian returns the median value or values of the elements in the array *A*. The *median* is the value which half the points in *A* are less than and half the points in *A* are greater than.

FNMedian computes the median by sorting the elements in the array *A.* If *A* has an even number of points, there are two central values in the return value in A. In this case, *FNMedian* returns the average of the two central values.

When it executes, *FNMedian* creates a temporary integer array to hold values used in sorting. This requires 2 bytes of memory per point in the array *A*. *FNMedian* causes an error if this amount is memory is not available.

Errors

FNMedian causes a BASIC error if its arguments is not a REAL array.

See Also

Mean, MAT SORT (in the *HTBasic Reference Manual*)

Norm

Euclidean or *F*-norm of a vector.

Loading LOADSUB ALL FROM "NORM.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL Y,A(*)
Y=FNNorm(A(*))
```

Description

FNNorm returns the value of the Euclidean norm of the vector represented in *A*. This norm is computed by summing the squares of the elements and taking the square root of the sum. This norm is also called the *L*-2 norm or the *F*-norm.

Errors

FNNorm causes an HTBasic error if its argument is not a REAL array.

See Also

Norm1, Norminf

Norm1

***L*-1 norm of a vector.**

Loading

LOADSUB ALL FROM "NORM.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

REAL Y,A(*)
Y=FNNorm1(A(*))

Description

FNNorm1 returns the value of the *L*-1 norm of the vector represented in *A*. This norm is computed by summing the absolute values of the elements in *A*.

Errors

FNNorm1 causes an HTBasic error if its argument is not a REAL array.

See Also

Norm, Norminf

Norminf

***L*- norm of a vector.**

Loading

LOADSUB ALL FROM "NORM.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL Y,A(*)
Y=FNNorminf(A(*))
```

Description

FNNorminf returns the value of the *L*- (*L*-infinity) norm of the vector represented in *A*. This norm is computed by finding the absolute value of the element of *A* with the largest absolute value.

Errors

FNNorminf causes an HTBasic error if its argument is not a REAL array.

See Also

Norm, Norm1

P1n

LeGendre function of the first kind, degree one.

Loading LOADSUB ALL FROM "LEGENDRE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNP1n(N,X)
```

Description

FNP1n returns the value of the Legendre function of the first kind, degree one, and order *n* of *x*, *Pn*1(*x*), into the real variable *Y*. *N* must be positive or zero.

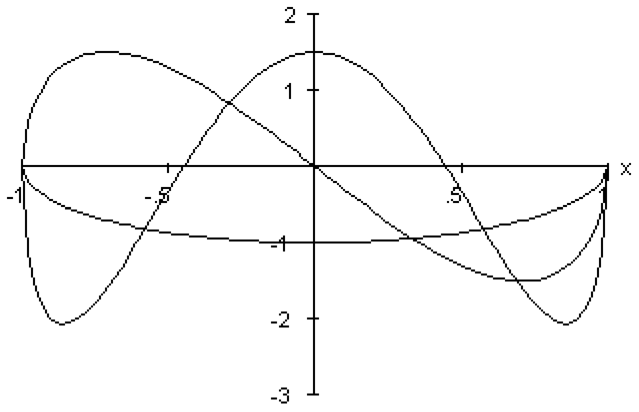
Errors

FNP1n causes a BASIC error if its arguments are not of the types listed in the **USAGE** section, above, if *n* < 0, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

See Also

Pn, Q1n

*Pn*1(*x*)



Note For *n* > 12, the algorithm used computes the value of *Pn*1 using a recursion from the values of *P*11 and *P*21. The computation time increases with *n* - 1 and the computation accuracy decreases with *n* - 1.

Paderiv

Antiderivative of a polynomial.

Loading

```
LOADSUB ALL FROM "PADERIV.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Paderiv FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL P(*),A(*)
CALL Paderiv(N,P(*),A(*))
```

Description

Paderiv calculates the polynomial that is the antiderivative of that described in *n* and *P*. It returns the coefficients of the antiderivative in *A*.

The first element in the arrays *P* and *A* represents the constant term in the polynomial. In the case of *A*, this element is set to zero, as the constant term in an antiderivative may take any value. The second element in *P* and *A* represents the linear term; the third the quadratic term, etc. The interpretation of the elements in *P* and *A* is without regard to the OPTION BASE in effect or any lower bound specified when *P* and *A* were declared. *N* is the degree of the polynomial whose coefficients are in *P*.

Errors

Paderiv causes an HTBasic error if $n > 10$, if *P* contains fewer than $n + 1$ elements, or if *A* contains fewer than $n + 2$ elements.

See Also

Pderiv, Pinteg

Pderiv

Derivative of a polynomial.

Loading

```
LOADSUB ALL FROM "PDERIV.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Pderiv FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL P(*),D(*)
CALL Pderiv(N,P(*),D(*))
```

Description

Pderiv calculates the polynomial that is the derivative of that described in *P*. It returns the coefficients of the derivative in *D*.

The first element in the arrays *P* and *D* represents the constant term in the polynomial. The second element in *P* and *D* represents the linear term; the third the quadratic term, etc. The interpretation of the elements in *P* and *D* is without regard to the OPTION BASE in effect or any lower bound specified when *P* and *D* were declared. *N* is the degree of the polynomial whose coefficients are in *P*.

Errors

Pderiv causes an HTBasic error if $n > 10$, if *P* contains fewer than $n + 1$ elements, or if *D* contains fewer than *n* elements.

See Also

Paderiv

Pinteg

Integral of a polynomial.

Loading

```
LOADSUB ALL FROM "PDERIV.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL A,B,P(*),Y
Y=FNPinteg(N,P(*),A,B)
```

Description

FNPinteg calculates the definite integral of the real polynomial function *p*(*x*) over the interval *x* (*a*,*b*). It does this by using the polynomial antiderivative function used by the *Paderiv* subroutine and returning the difference of the antiderivatives at the points *b* and *a*.

The first element in the array *P* represents the constant term in the polynomial; the second element represents the linear term; the third the quadratic term, etc. The interpretation of the elements in *P* is without regard to the OPTION BASE in effect or any lower bound specified when *P* was declared.

Errors

Pinteg causes an HTBasic error if *n* > 10 or if *P* contains fewer than *n* + 1 elements.

See Also

Paderiv

Pn

LeGendre function of the first kind.

Loading LOADSUB ALL FROM "LEGENDRE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNPn(N,X)
```

Description

FNPn returns the value of the Legendre function of the first kind, degree zero, and order *n* of *x*, *Pn*(*x*), into the real variable *Y*. *N* must be positive or zero.

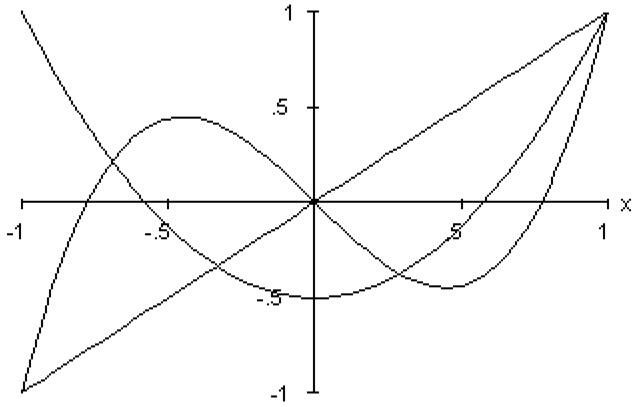
Errors

FNPn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if $n < 0$, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

See Also

P1n, Qn

Pn(*x*)



Note For $n > 12$, the algorithm used computes the value of *Pn* using a recursion from the values of *P*1 and *P*2. The computation time increases with *n* - 11 and the computation accuracy decreases with *n* - 11.

Polar

Polar form of a complex array.

Loading

```
LOADSUB ALL FROM "POLAR.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Polar FROM "MATHLIB.HTS"
```

Usage

```
COMPLEX C(*)
REAL A(*),P(*)
CALL Polar(C(*),T$,A(*),P(*))
```

Description

Polar changes the elements of the array *C* into polar form. The magnitude of each element in *C* is placed into the corresponding element of array *A* and the phase of each element is placed into the corresponding element of array *P*. The elements of *C* remain unchanged. If the first character in *T$* is "D" or "d", the angles in *P* are output in degrees; if *T$* is null or begins with any character other than "d" or "D", the angles in *P* are output in radians. *A* and *P* must contain at least as many elements as *C*. If *A* or *P* contains more elements than *C*, the extra elements are unchanged or ignored.

Errors

Polar causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *A* or *P* contain fewer elements than *C*.

See Also

Rect

Poly

Evaluate a polynomial.

Loading

```
LOADSUB ALL FROM "POLY.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL X,Y,C(*)
Y=FNPoly(N,C(*),X)
```

Description

FNPoly evaluates a polynomial of degree *n* whose coefficients are given in the elements of *C* at argument *x*. The first element in *C* is the constant term in the polynomial, the second element is the first-degree term (the multiplier of *x*), the third element is the second-degree term (the multiplier of *x*2), etc. *C* must contain at least *n*+1 elements; if it contains more than *n*+1 elements, the extra elements are ignored.

Errors

FNPoly causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the array *C* has fewer than *n*+1 elements.

See Also

Cpoly, Paderiv, Pderiv, Pinteg, Proots

Power_spectrum

Calculate power spectral density.

Loading LOADSUB ALL FROM "FFT.HTS"
or LOADSUB Power_spectrum FROM "MATHLIB.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER Logn
REAL A(*),B(*)
CALL Power_spectrum(Logn,A(*),B(*))
```

Description

Power_spectrum calculates the power spectral density of the data in the array *A* and returns the information in the array *B*. *Logn* is the base-2 logarithm of the number of points in the sequence to be correlated. The array *A* must have at least 2*Logn* elements; if it has more than this number of elements, the extra elements are ignored. The array *B* must have at least 2*Logn*+1 elements; if it has more than this number of elements, the extra elements are unmodified. The number of elements in *A* denoted by each permitted value of *Logn* is shown in the table below:

Logn	No. Elements (2*Logn*)
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384

If *N* = 2*Logn*, after *Power_spectrum* has run, the first *N/2* elements in the array *B* contain the power spectral density of *A*. The second *N/2* elements in *B* contain zeros.

If the values in *A* are taken to be values of a continuous complex signal, *a*(*t*), sampled at constant intervals of *T* (time, distance, or whatever units apply), and if the signal sampled contained no terms at or above the frequency 1/2*T*, then the first *N/2* elements in the array *B* are proportional to the power at the frequencies *k*/2*NT*, where *k* is the position in *B*, beginning with *k* = 0.

The power spectral density of a set of data is the Fourier transform of the autocorrelation of that set of data.

Errors

Power_spectrum causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *B* is smaller than the values described above.

See Also

Autocorrelate, Power_spectrum

Proots

Find the roots of a polynomial.

Loading

LOADSUB ALL FROM "PROOTS.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Proots FROM "MATHLIB.HTS"

Usage

INTEGER M,N
REAL P(*)
COMPLEX Z(*)
CALL Proots(N,P(*),M,Z(*))

Description

Proots attempts to finds all values of *z* that satisfy the relation *p*(*z*) = 0, where *p* is a polynomial of degree *n*, which must be 10 or less. The roots will usually be returned in *Z* in order of increasing modulus. Since the coefficients of *p* are real, complex roots returned in *Z* will occur in conjugate pairs.

If a root is not found after *m* iterations of the algorithm, that root is set to CMPLX(MAXREAL,0.0) to indicate an error and *Proot* returns without attempting to find more roots.

The first element in the array *P* represents the constant term in the polynomial; the second element, the linear term; the third the quadratic term, etc. The interpretation of the elements in *P* is without regard to the OPTION BASE in effect or any lower bound specified when *P* was declared.

The array *Z* must contain enough elements to hold all the solutions to the expression *p*(*z*) = 0, that is, *Z* should contain at least *n* elements.

Proots uses LaGuerre's method to find a real root or a conjugate pair of roots to the equation *p*(*z*) = 0. It then reduces the equation by dividing *p*(*z*) by the term *z* - *zn* for a real root *zn* or $z - 2\Re(zn) + |zn|2$ for a complex root and repeats the procedure for the reduced polynomial. This stops when all the roots are found or when the algorithm fails to find a new root after *m* iterations.

Typical values for *m* might be 50, 100, or 200.

Errors

Proots causes an HTBasic error if *n* < 2, *n* > 10, if *P* contains fewer than *n* + 1 elements, if *Z* contains fewer than *n* elements, or if an evaluation of the polynomial being used results in a value greater in magnitude than MAXREAL.

Example

The following BASIC program calculates the roots of the function

$$x3 - 3x2 + 3x - 2 = 0.$$

```
10 LOADSUB ALL FROM "PROOTS.HTS"
20 INTEGER I
```

```
30 REAL C(0:3)
40 COMPLEX Z(1:3)
50 READ C(*)
60 DATA -2.0,3.0,-3.0,1.0
70 CALL Proots(3,C(*),100,Z(*))
80 FOR I=1 TO 3
90 PRINT USING """(""",MZ.6D,""",""",MZ.6D,""")""";Z(I)
100 NEXT I
110 END
```

When run, it prints

```
( 0.500000,-0.866025)
( 0.500000, 0.866025)
( 2.000000, 0.000000).
```

The roots of the equation can be found by hand. They are
$\frac{1-\sqrt{3}}{2}$, $\frac{1+\sqrt{3}}{2}$, and 2.

See Also

Froot, Paderiv, Pderiv, Pinteg, Poly

Pulse

Generate a pulse waveform.

Loading LOADSUB ALL FROM "PULSE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Pulse FROM "MATHLIB.HTS"

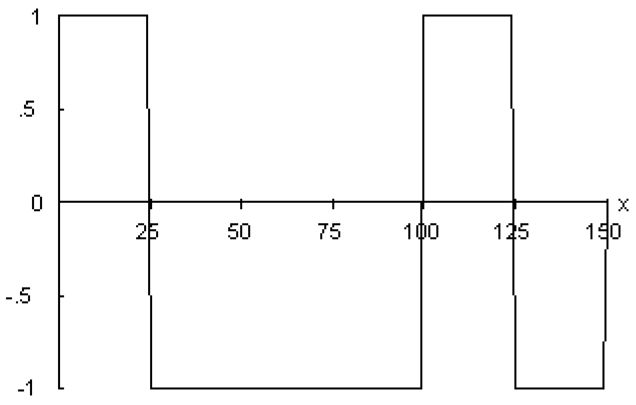
Usage

```
REAL P,A,B,S,C,Y(*)
CALL Pulse(P,A,B,S,C,Y(*))
```

Description

Pulse fills the elements of the array *Y* with a pulse of duty cycle *c*, period *p*, start point *s*, and high and low values *a* and *b*, respectively. *P* and *S* are expressed in units of the number of array elements, although they need not be integers. *P* must be positive and *c* must be between 0 and 1, inclusive.

If *Yk* refers to an element *Yk* for *a*=+1, *b*=-1, *c*=0.25, and *p*=100



of array *Y*, beginning with *k* = 0, the expression for *Yk* is:

$$Y_k = \begin{cases} a, & 0 \le \mathrm{fract}\left(\dfrac{k-s}{p}\right) < c \\ b, & c \le \mathrm{fract}\left(\dfrac{k-s}{p}\right) < 1 \end{cases}.$$

In the above expressions, fract(*x*) is the fractional part of *x*, calculated by finding the difference between *x* and the next lower integer from *x*. Fract(*x*) is between 0, inclusive, and 1, exclusive.

If *p* or *s* is contained in a variable of type INTEGER, be sure to use the BASIC REAL command to change the variable to a REAL value when passing it to the *Pulse* routine.

Errors

Pulse causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *p* is negative or zero, or if *c* is not between 0 and 1, inclusive.

See Also

Waveform

P_beta

Probability integral for beta distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNP_beta(A,B,X)

Description

FNP_beta returns the value of the probability integral of the univariate beta probability density function with parameters *a* and *b* at *x*, *P*(*x*;*a*,*b*). *P*(*x*;*a*,*b*) is defined only for *a* 0 and *b* 0.

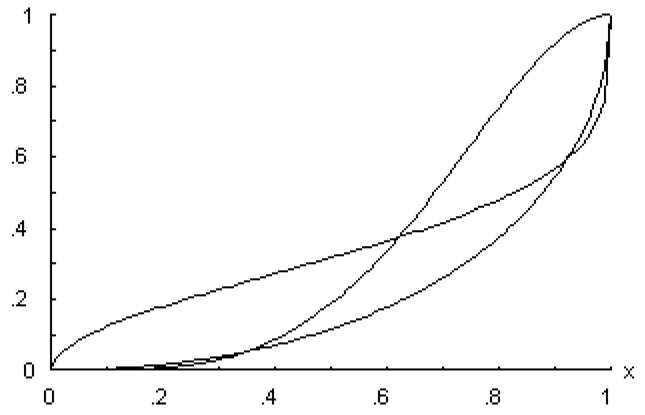
P(*x*;*a*,*b*) is defined by the expression

$$P(x;a,b) = \begin{cases} 0, & x < 0 \\ \dfrac{1}{B(a,b)} \displaystyle\int_0^x t^{a-1}(1-t)^{b-1}\,dt, & 0 \le x \le 1. \\ 1, & x > 1 \end{cases}$$

When 0 *x* 1, *P*(*x*;*a*,*b*) is related to the incomplete beta function returned by *FNIbeta*, *Bx*(*a*,*b*), by the expression

$$P(x;a,b) = \frac{B_x(a,b)}{B(a,b)}$$

P(*x*;*a*,*b*)



The function *P*(*x*;*a*,*b*) is sometimes written *Ix*(*a*,*b*).

Errors

FNP_beta causes a BASIC error if its arguments are not all of type REAL or if *a* or *b* is negative.

See Also

Beta, Ibeta, F_beta, Q_beta

P_cauchy

Probability integral for Cauchy distribution.

Loading LOADSUB ALL FROM "CAUCHY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNP_cauchy(A,B,X)

Description

FNP_cauchy returns the value of the integral of the Cauchy probability density function with parameters *a* and *b* at *x*. This integral, *P*(*x*;*a*,*b*), is defined by the expression

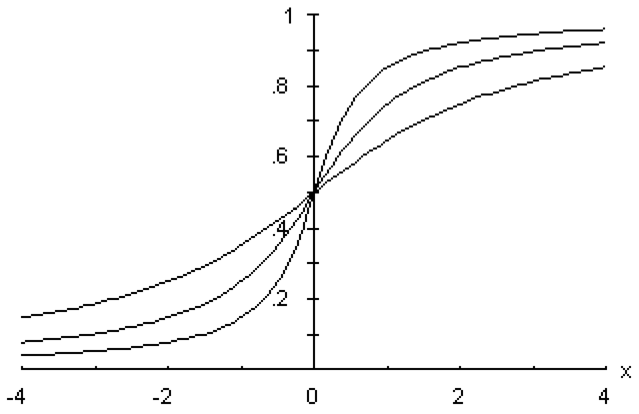
$$P(x;a,b) = \int_{-\infty}^{x} \frac{1}{\pi b\left[1+\left(\frac{t-a}{b}\right)^2\right]}\,dt,$$

which reduces to

$$P(x;a,b) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x-a}{b}\right).$$

B must be greater than zero.

P(*x*;0,*b*)



Errors *FNP_cauchy* causes a BASIC error if its arguments are not all of type REAL or if *b* is negative or zero.

See Also

F_cauchy, Q_cauchy

P_chi2

Probability integral for Chi-squared distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNP_chi2(N,X)
```

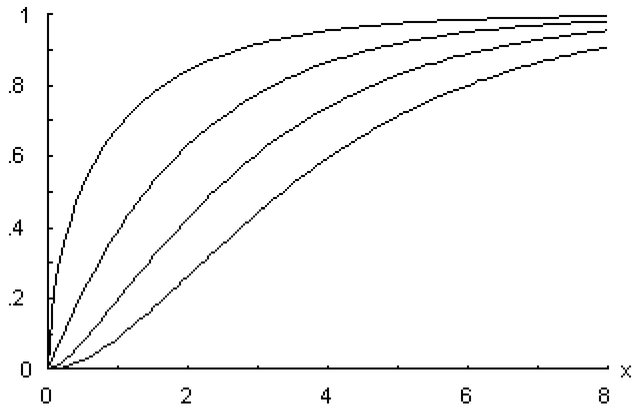
Description

FNP_chi2 returns the value of the probability integral of the univariate chi-squared (χ^2) probability density function with parameter *n*. This integral, *P*(*x*;*n*), is defined by the expression

$$P(x;n) = \begin{cases} \frac{1}{2^{n/2}\Gamma(n/2)} \int_0^x t^{n/2-1} e^{-t/2}\,dt, & x \ge 0 \\ 0, & x < 0 \end{cases}.$$

X is often written as ²; among other uses, this notation emphasizes the fact that this integral is only nonzero for values of *x* 0. Although *P* is sometimes defined for *n* < 0, most implementations, including this one, restrict *P* to being defined for *n* 0.

P(*x*;*n*)



Errors *FNP_chi2* causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *n* is negative.

See Also

F_chi2, Q_chi2

P_exp

Probability integral for exponential distribution.

Loading LOADSUB ALL FROM "EXP.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,Y
Y=FNP_exp(A,X)

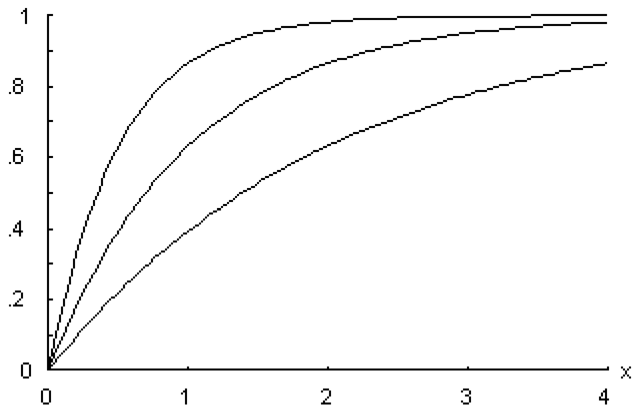
Description

FNP_exp returns the value of the integral of the exponential probability density function with parameter *a* at *x*, *P*(*x*;*a*). *P*(*x*;*a*) is defined by the expression

$$P(x;a) = \begin{cases} 1 - e^{-ax}, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

P(*x*;*a*) is defined for positive values of *a*.

P(*x*;*a*)



Errors *FNP_exp* causes a BASIC error if its arguments are not all of type REAL or if *a* is zero or negative.

See Also

F_exp, Q_exp

P_f

Probability integral for *F* distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage INTEGER M,N
REAL X,Y
Y=FNP_f(M,N,X)

Description

FNP_f returns the value of the probability integral of the univariate *F* probability density function with parameters *m* and *n* at *x*, *P*(*x*;*m*,*n*). This integral is defined for *m* and *n* positive or zero.

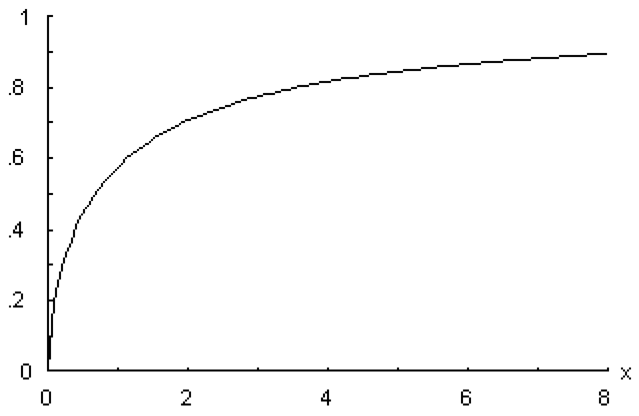
Errors

FNP_f causes a BASIC error if its arguments are not of the types listed in the **USAGE** section, above, or if either *m* or *n* is negative.

See Also

F_f, Q_f

P(*x*;1,2)



P_gauss

Probability integral for Gaussian distribution.

Loading LOADSUB ALL FROM "ERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,M,S,Y
Y=FNP_gauss(M,S,X)

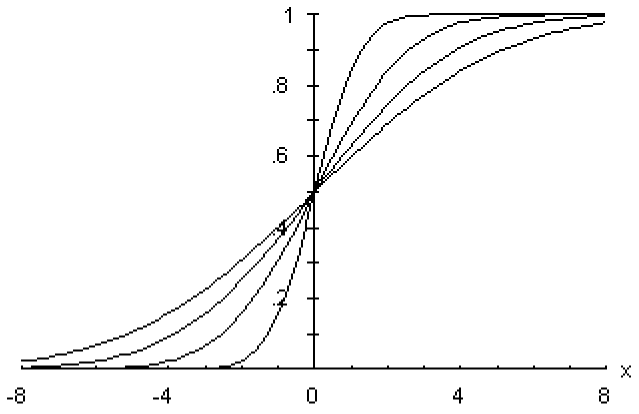
Description

FNP_gauss returns the value of the integral of the Gaussian, or normal, probability density function of mean *m* and standard deviation *s* (represented below by σ) at *x*. The integral, *P*(*x*;*m*,σ), is defined by the expression

$$P(x;m,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x} e^{\frac{(t-m)^2}{2\sigma^2}}\,dt.$$

P is defined for all values of *x* and *m* and for positive values of σ.

P(*x*;0,σ)



Errors *FNP_gauss* causes a BASIC error if its arguments are not all of type REAL or if the value of *S* is negative or zero.

See Also

F_gauss, Q_gauss

P_laplace

Probability integral for LaPlace distribution.

Loading LOADSUB ALL FROM "LAPLACE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNP_laplace(A,B,X)

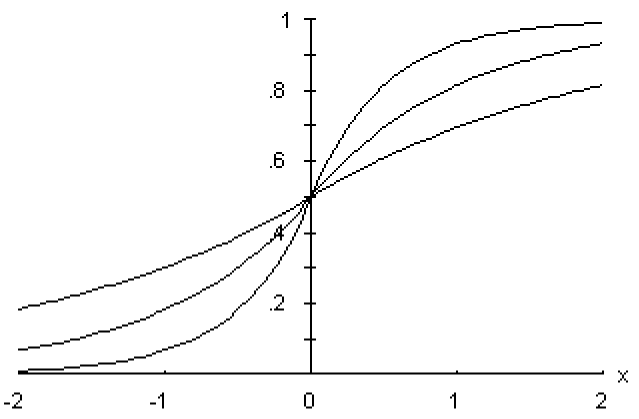
Description

FNP_laplace returns the value of the integral of the LaPlace probability density function of parameters *a* and *b* at *x*, *P*(*x*;*a*,*b*). *P*(*x*;*a*,*b*) is defined by the expression

$$P(x;a,b) = \begin{cases} \frac{1}{2} e^{-\left|\frac{x-a}{b}\right|}, & x \le a \\ 1 - \frac{1}{2} e^{-\left|\frac{x-a}{b}\right|}, & x > a \end{cases}.$$

P is defined for all positive values of *b*.

P(*x*;0,*b*)



Errors *FNP_laplace* causes a BASIC error if its arguments are not all of type REAL or if the value of *b* is negative or zero.

See Also

F_laplace, Q_laplace

P_pareto

Probability integral for Pareto distribution.

Loading LOADSUB ALL FROM "PARETO.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,X0,T,Y
Y=FNP_pareto(X0,T,X)

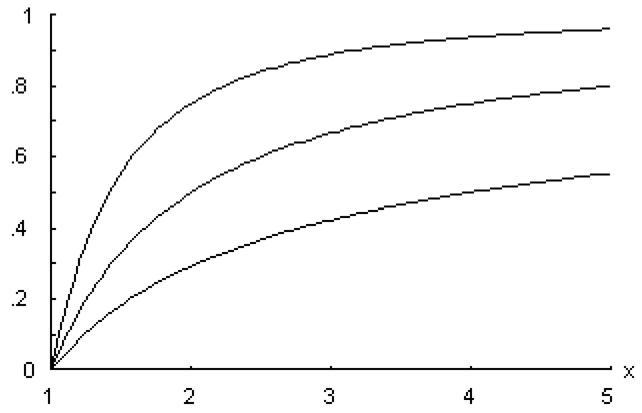
Description

FNP_pareto returns the value of the integral of the Pareto probability density function of parameters *X0* (here written as *x*0) and *t* at *x*, *P*(*x*;*x*0,*t*). *P*(*x*;*x*0,*t*) is defined by the expression

$$P(x;x_0,t) = \begin{cases} 1-\left(\dfrac{x_0}{x}\right)^{t+1}, & x \ge x_0 \\ 0, & x < x_0 \end{cases}.$$

P is defined for positive values of *x*0 and *t*.

P(*x*;1,*t*)



Errors *FNP_pareto* causes a BASIC error if its arguments are not all of type REAL or if the value of *x*0 or *t* is negative or zero.

See Also

F_pareto, Q_pareto

P_rayleigh

Probability integral for Rayleigh distribution.

Loading LOADSUB ALL FROM "RAYLEIGH.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL Beta,X,Y
Y=FNP_rayleigh(Beta,X)

Description

FNP_rayleigh returns the value of the integral of the Rayleigh probability density function of parameter *Beta* (here written as β), at *x*, *P*(*x*;). *P* is defined by the expression

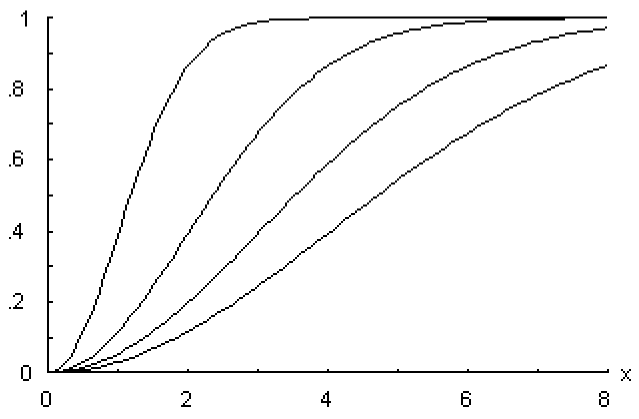
$$P(x;\beta) = \begin{cases} \dfrac{1}{\beta^2}\displaystyle\int_0^x t\, e^{-\frac{t}{2\beta^2}}\, dt, & x \geq 0 \\ 0, & x < 0 \end{cases},$$

which reduces to

$$P(x;\beta) = \begin{cases} 1 - x\, e^{-\frac{x}{2\beta^2}}, & x \geq 0. \\ 0, & x < 0 \end{cases}$$

P is defined for all positive values of .

P(*x*;)



Errors *FNP_rayleigh* causes a BASIC error if its arguments are not all of type REAL or if the value of is negative or zero.

See Also

F_rayleigh, Q_rayleigh

P_student

Probability integral for Student's *t* distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNP_student(N,X)
```

Description

FNP_student returns the value of the integral of the Student's *t* probability density function of parameter *n* at *x*, *P*(*x*;*n*). *P* is defined for all positive or zero values of *x* and all positive values of *n*.

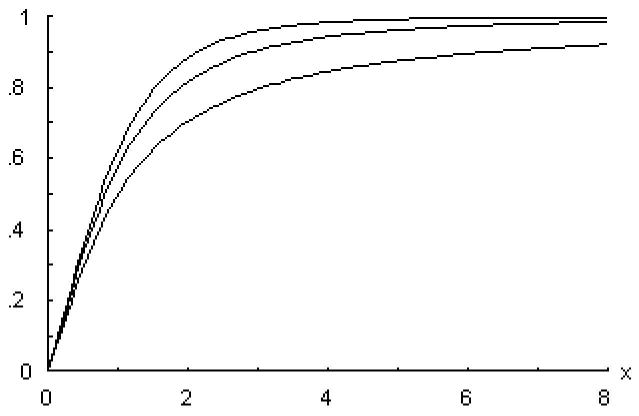
Errors

FNP_student causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the value of *n* is negative or zero.

See Also

F_student, Q_student

P(*x*;*n*)



Q1n

LeGendre function of the second kind, degree one.

Loading LOADSUB ALL FROM "LEGENDRE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNQ1n(N,X)
```

Description

FNQ1n returns the value of the Legendre function of the second kind, degree one, and order *n* of *x*, *Qn*1(*x*). *N* must be positive or zero.

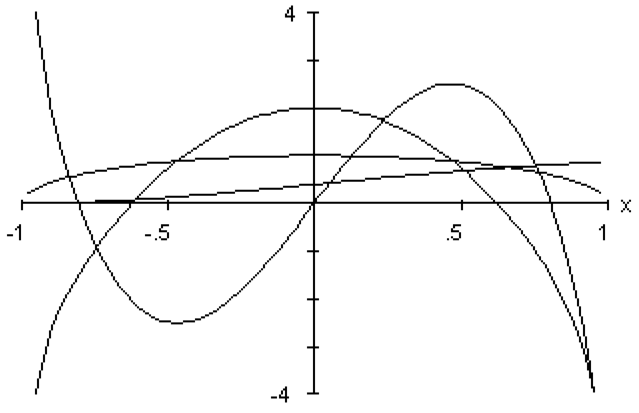
Errors

FNQ1n causes a BASIC error if its arguments are not of the types listed in the usage section, above, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

See Also

P1n, Qn

*Qn*1(*x*)



Note For *n* > 2, the algorithm used computes the value of *Qn*1 using a recursion from the values of *Q*11 and *Q*21. The computation time increases with *n*-1 and the computation accuracy decreases with *n*-1.

Qn

LeGendre function of the second kind.

Loading LOADSUB ALL FROM "LEGENDRE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNQn(N,X)
```

Description

FNQn returns the value of the Legendre function of the second kind, degree zero, and order *n* of *x*, *Qn*(*x*). *N* must be positive or zero.

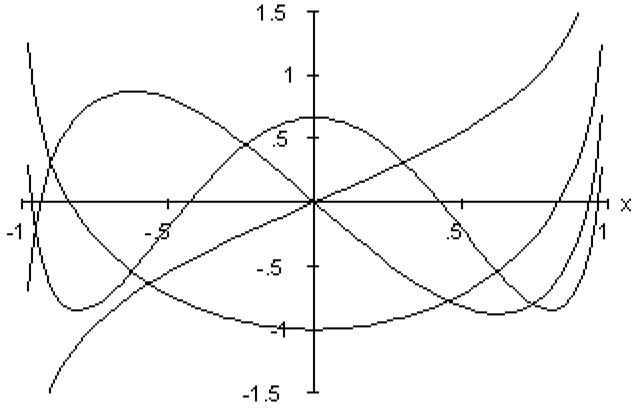
Errors

FNQn causes a BASIC error if its arguments are not of the types listed in the usage section, above, if *n* is negative, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

See Also

Pn, Q1n

Qn(*x*)



Note For *n* > 2, the algorithm used computes the value of *Qn* using a recursion from the values of *Q*1 and *Q*2. The computation time increases with *n*-1 and the computation accuracy decreases with *n*-1.

Q_beta

Complementary probability integral for beta distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,A,B,Y
Y=FNQ_beta(A,B,X)
```

Description

FNQ_beta returns the value of the complementary probability integral of the univariate beta probability density function with parameters *a* and *b*, *Q*(*x*;*a*,*b*). *Q*(*x*;*a*,*b*) is defined only for *a* 0 and *b* 0.

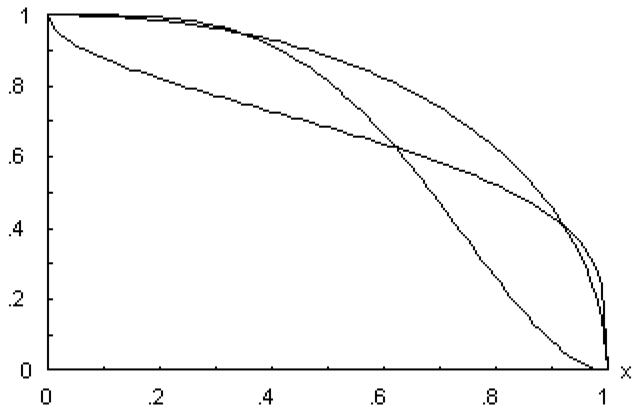
Q(*x*;*a*,*b*) is defined by the expression

$$Q(x;a,b) = \begin{cases} 1, & x < 0 \\ \dfrac{1}{B(a,b)} \displaystyle\int_x^1 t^{a-1}(1-t)^{b-1}dt, & 0 \le x \le 1. \\ 0, & x > 1 \end{cases}$$

When $0 \le x \le 1$, *Q*(*x*;*a*,*b*) is related to the incomplete beta function returned by *FNIbeta*, *Bx*(*a*,*b*) by the expression

$$Q(x;a,b) = \frac{B(a,b) - B_x(a,b)}{B(a,b)}.$$

Q(*x*;*a*,*b*)



Errors *FNQ_beta* causes a BASIC error if its arguments are not all of type REAL or if *a* or *b* is negative.

See Also

Beta, Ibeta, F_beta, P_beta

Q_cauchy

Complementary probability integral for Cauchy distribution.

Loading LOADSUB ALL FROM "CAUCHY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNQ_cauchy(A,B,X)

Description

FNQ_cauchy returns the value of the complementary integral of the Cauchy probability density function with parameters *a* and *b*. This integral, *Q*(*x*;*a*,*b*), is defined by the expression

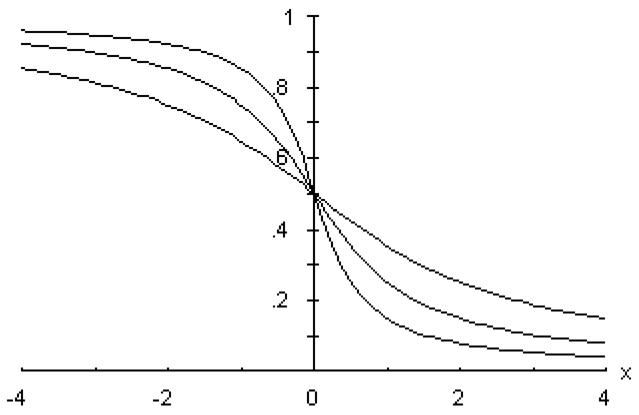
$$Q(x;a,b) = \int_x^\infty \frac{1}{\pi b\left[1+\left(\frac{t-a}{b}\right)^2\right]}\,dt,$$

which reduces to

$$Q(x;a,b) = \frac{1}{2} - \frac{1}{\pi}\arctan\left(\frac{x-a}{b}\right).$$

B must be greater than zero.

Q(*x*;0,*b*)



Errors *FNQ_cauchy* causes a BASIC error if its arguments are not all of type REAL or if *b* is negative or zero.

See Also

F_cauchy, P_cauchy

Q_chi2

Complementary probability integral for Chi-squared distribution.

Loading LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNQ_chi2(N,X)
```

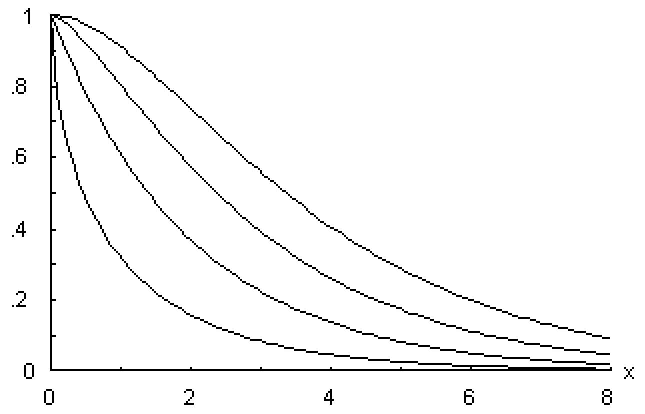
Description

FNQ_chi2 returns the value of the complementary probability integral of the univariate chi-squared (χ^2) probability density function with parameter *n*. This integral, *Q*(*x*;*n*), is defined by the expression

$$Q(x;n) = \begin{cases} \dfrac{1}{2^{n/2}\Gamma(n/2)} \displaystyle\int_x^\infty t^{n/2-1} e^{-t/2}\,dt, & x \ge 0 \\ 1, & x < 0 \end{cases}.$$

X is often written as χ^2; among other uses, this notation emphasizes the fact that this integral is only less than one for values of $x \ge 0$. Although *Q* is sometimes defined for $n < 0$, most implementations, including this one, restrict *Q* to being defined for $n \ge 0$.

Q(*x*;*n*)



Errors *FNQ_chi2* causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *n* is negative.

See Also

F_chi2, P_chi2

Q_exp

Complementary probability integral for exponential distribution.

Loading LOADSUB ALL FROM "EXP.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,Y
Y=FNQ_exp(A,X)

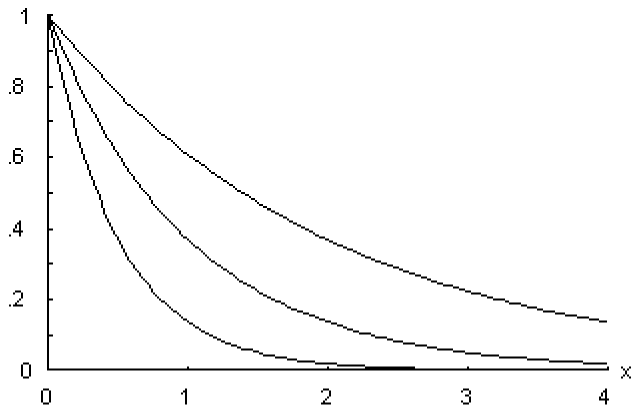
Description

FNQ_exp returns the value of the complementary integral of the exponential probability density function with parameter *a* at *x*, *Q*(*x*;*a*). *Q*(*x*;*a*) is defined by the expression

$$Q(x;a) = \begin{cases} e^{-ax}, & x \geq 0 \\ 1, & x < 0 \end{cases}.$$

Q(*x*;*a*) is defined for positive values of *a*.

Q(*x*;*a*)



Errors *FNQ_exp* causes a BASIC error if its arguments are not all of type REAL or if *a* is negative or zero.

See Also

F_exp, P_exp

Q_f

Complementary probability integral for *F* distribution.

Loading

LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER M,N
REAL X,Y
Y=FNQ_f(M,N,X)
```

Description

FNQ_f returns the value of the complementary probability integral of the univariate *F* probability density function with parameters *m* and *n* at *x*, *Q*(*x*;*m*,*n*). This integral is defined for *m* and *n* positive or zero.

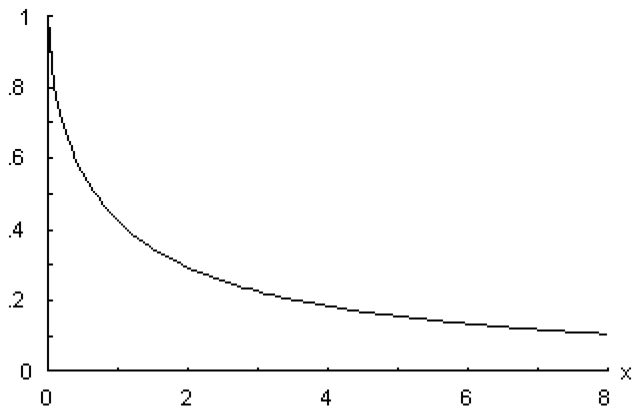
Errors

FNQ_f causes a BASIC error if its arguments are not of the types listed in the usage section, above, or if either *m* or *n* is negative.

See Also

F_f, P_f

Q(*x*;1,2)



Q_gauss

Complementary probability integral for Gaussian distribution.

Loading LOADSUB ALL FROM "ERF.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,M,S,Y
Y=FNQ_gauss(M,S,X)

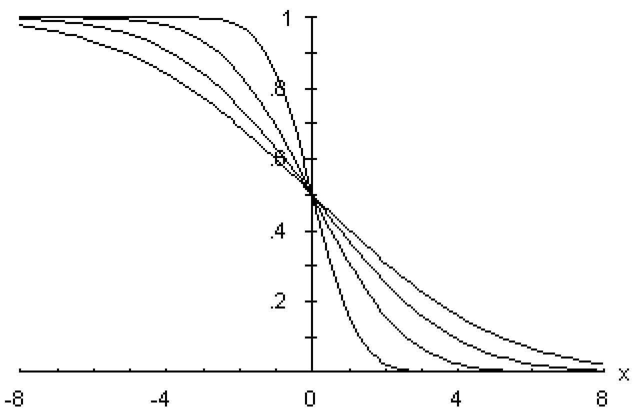
Description

FNQ_gauss returns the value of the complementary integral of the Gaussian, or normal, probability density function of mean *m* and standard deviation *s* (represented below by σ) at *x*. The integral, *Q*(*x*;*m*,σ), is defined by the expression

$$Q(x;m,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_x^\infty e^{\frac{(t-m)^2}{2\sigma^2}}\,dt.$$

Q is defined for all values of *x* and *m* and for positive values of .

Q(*x*;0,σ)



Errors *FNQ_gauss* causes a BASIC error if its arguments are not all of type REAL or if the value of *S* is negative or zero.

See Also

F_gauss, P_gauss

Q_laplace

Complementary probability integral for LaPlace distribution.

Loading LOADSUB ALL FROM "LAPLACE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,A,B,Y
Y=FNQ_laplace(A,B,X)

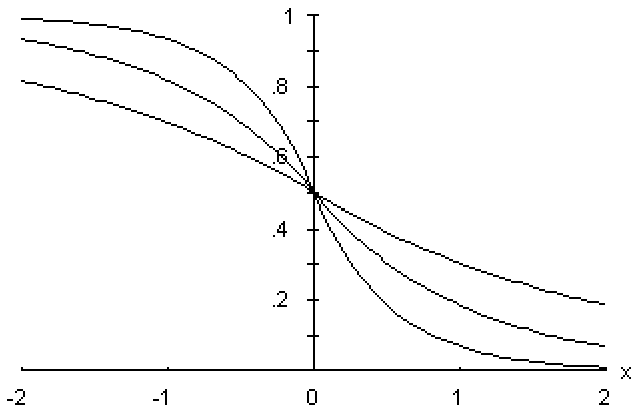
Description

FNQ_laplace returns the value of the complementary integral of the LaPlace probability density function of parameters *a* and *b* at *x*, *Q*(*x*;*a*,*b*). *Q*(*x*;*a*,*b*) is defined by the expression

$$Q(x;a,b) = \begin{cases} 1 - \frac{1}{2} e^{-\left|\frac{x-a}{b}\right|}, & x \le a \\ \frac{1}{2} e^{-\left|\frac{x-a}{b}\right|}, & x > a \end{cases}.$$

Q is defined for all positive values of *b*.

Q(*x*;0,*b*)



Errors *FNQ_laplace* causes a BASIC error if its arguments are not all of type REAL or if the value of *b* is negative or zero.

See Also

F_laplace, P_laplace

Q_pareto

Complementary probability integral for Pareto distribution.

Loading LOADSUB ALL FROM "PARETO.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

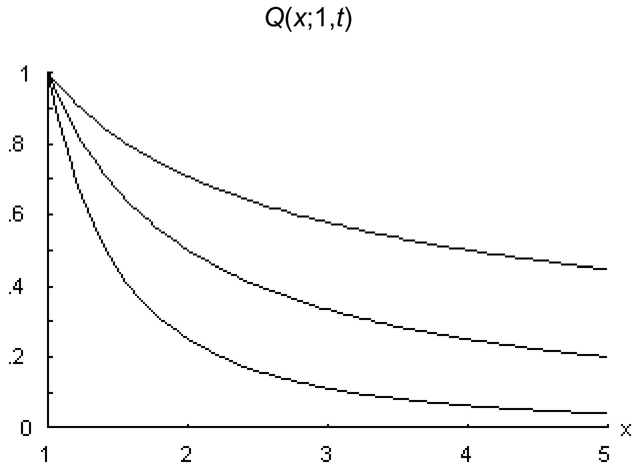
```
REAL X,X0,T,Y
Y=FNQ_pareto(X0,T,X)
```

Description

FNQ_pareto returns the value of the complementary integral of the Pareto probability density function of parameters *X0* (here written as *x*0) and *t* at *x*, *Q*(*x*;*x*0,*t*). *Q*(*x*;*x*0,*t*) is defined by the expression

$$Q(x;x_0,t) = \begin{cases} \left(\frac{x_0}{x}\right)^{t+1}, & x \ge x_0 \\ 1, & x < x_0 \end{cases}.$$

Q is defined for positive values of *x*0 and *t*.



Errors *FNQ_pareto* causes a BASIC error if its arguments are not all of type REAL or if the value of *x*0 or *t* is negative or zero.

See Also

F_pareto, P_pareto

Q_rayleigh

Complementary probability integral for Rayleigh distribution.

Loading LOADSUB ALL FROM "RAYLEIGH.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL Beta,X,Y
Y=FNQ_rayleigh(Beta,X)

Description

FNQ_rayleigh returns the value of the integral of the Rayleigh probability density function of parameter *Beta* (here written as β), at *x*, *Q*(*x*;β). *Q* is defined by the expression

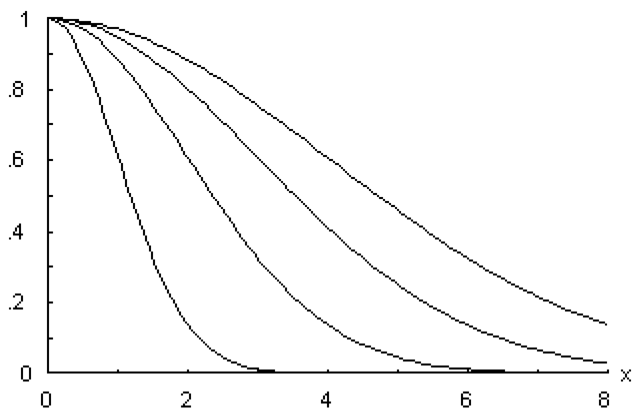
$$Q(x;\beta) = \begin{cases} \dfrac{1}{\beta^2}\displaystyle\int_x^\infty t\,e^{-\frac{t}{2\beta^2}}\,dt, & x \ge 0 \\ 1, & x < 0 \end{cases},$$

which reduces to

$$Q(x;\beta) = \begin{cases} x\,e^{-\frac{x}{2\beta^2}}, & x \ge 0. \\ 1, & x < 0 \end{cases}$$

Q is defined for all positive values of β.

Errors *Q*(*x*;β)



FNQ_rayleigh causes a BASIC error if its arguments are not all of type REAL or if the value of is negative or zero.

See Also

F_rayleigh, P_rayleigh

Q_student

Complementary probability integral for Student's *t* distribution.

Loading

```
LOADSUB ALL FROM "IGAMMA.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
INTEGER N
REAL X,Y
Y=FNQ_student(N,X)
```

Description

FNQ_student returns the value of the complementary integral of the Student's *t* probability density function of parameter *n*, *Q*(*x*;*n*). *Q* is defined for all positive values of *n*.

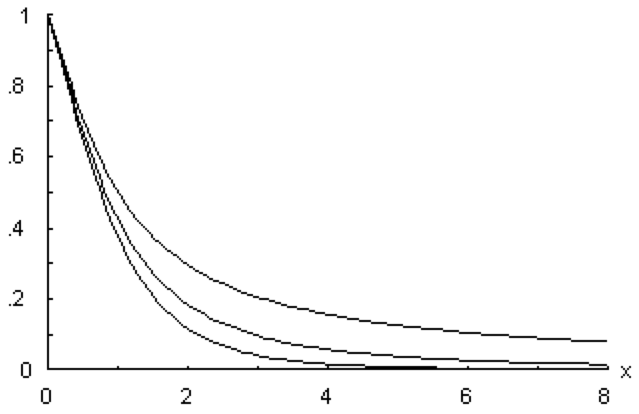
Errors

FNQ_student causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the value of *n* is negative or zero.

See Also

F_student, P_student

Q(*x*;*n*)



Rect

Rectangular form of a complex array.

Loading LOADSUB ALL FROM "RECT.HTS"
or LOADSUB Rect FROM "MATHLIB.HTS"

Usage

```
REAL A(*),P(*)
COMPLEX C(*)
CALL Rect(A(*),P(*),T$,C(*))
```

Description

Rect changes the elements of the arrays *A* and *P* into rectangular form. The magnitude of each element is in *A* and the phase of each element is in the corresponding position in *P*. The rectangular form of the data in *A* and *P* is placed into the corresponding element of array *C*. The elements of *A* and *P* remain unchanged. If the first character in *T$* is "D" or "d", the angles in *P* are taken to be in degrees; if *T$* is null or begins with any character other than "d" or "D", the angles in *P* are taken to be in radians. *A* and *P* must contain the same number of elements; *C* must contain at least as many element as *A* and *P*. If *C* contains more than this number of elements, the extra elements are unchanged or ignored.

Errors

Rect causes a BASIC error if its arguments are not of the types listed in the USAGE section, if *A* and *P* contain different numbers of elements or if *C* contains fewer elements than *A* and *P*.

See Also

Polar

Rfilter

Filter a real sequence.

Loading

```
LOADSUB ALL FROM "FFT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Rfilter FROM "MATHLIB.HTS"
```

Usage

```
INTEGER Logn
REAL A(*),B(*),F(*)
CALL Rfilter(Logn,A(*),F(*),B(*))
```

Description

Rfilter calculates the sequence produced by filtering the time-domain (or space-domain) sequence in *A* by the filter whose frequency-domain coefficients are in *F*. It returns the resulting sequence in the array *B*. *Logn* is the base-2 log of the number of points in the sequences in *A* and *B*. The arrays *A* and *B* must contain at least 2*Logn* elements. The array *F* must contain at least 2*Logn*-1 elements. If the arrays have extra elements, the extra elements are ignored and unmodified.

The values in *F* are the amounts by which to scale the corresponding frequency components in *A* to produce the resultant sequence. These values are magnitudes; *Rfilter* assumes that all filter coefficients have zero phase; use *Filter* to use a filter function having both magnitude and phase.

The first element in *F* represents the amount by which the d. c. term in *A* is to be scaled, the second the amount by which the 1/*N* frequency component is scaled, the third the amount by which the 2/*N* frequency component is scaled, etc. The meaning of each frequency component is the same for *Rfilter* as for *Fft* and is explained in the entry for *Fft*.

If the sequence to be used as a filter is specified as an impulse response, the *Convolve* function may be used to filter using the impulse response as input.

Errors

Rfilter causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, if *Logn* is not between 2 and 15, inclusive, or if the size of *A* or *B* is smaller than 2*Logn*, or if *F* is smaller than 2*Logn*-1.

See Also

Cfft, Convolve, Filter, Fft

Romberg

Integrate a function using Romberg's method.

Loading

```
LOADSUB ALL FROM "ROMBERG.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB FNRomberg FROM "MATHLIB.HTS"
```

Usage

```
REAL A,B,X,Eps
INTEGER Nmin,Nmax
X=FNRomberg(F$,A,B,Eps,Nmin,Nmax)
```

Description

FNRomberg estimates the integral of the function named in *F$* between the points *a* and *b*. *Eps*, *Nmin*, and *Nmax* control when the estimating process stops. *FNRomberg* evaluates the integral using successively more points in the function, doing at least *Nmin* evaluations of the integral. When the difference between successive values of the integral is less than *Eps*, integration stops. If *FNRomberg* does more than *Nmax* evaluations of the integral without successive evaluations differing by less that *Eps*, *FNRomberg* stops evaluating the integral and returns MAXREAL to indicate failure to evaluate the integral.

F$* should contain the name of an HTBasic subroutine. The subroutine should take two REAL parameters. It should evaluate the function to be integrated at the second parameter and return its value in the first parameter. For example, if *F$ = "Test", then the subroutine *Test* should begin with the definition line

```
SUB Test(REAL Y,X)
```

where *X* and *Y* may be replaced by the names of any REAL parameters. The subroutine *Test* would evaluate the desired function at the value *X* and return the value in *Y*.

Errors

FNRomberg causes an HTBasic error if the subroutine named in *F$* is undefined. The subroutine named in *F$* may also cause HTBasic Errors when it is evaluated.

S

Fresnel sine integral of a real argument.

Loading LOADSUB ALL FROM "FRESNEL.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNS(X)

Description

FNS returns the value of the Fresnel sine integral of *x*. *S*(*x*) is defined by the relation

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

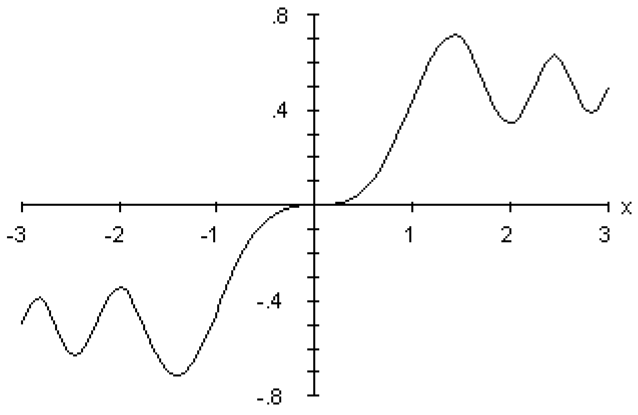
Errors

FNS causes a BASIC error if its argument is not of type REAL.

See Also

C

S(*x*)



Shi

Hyperbolic sine integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNShi(X)

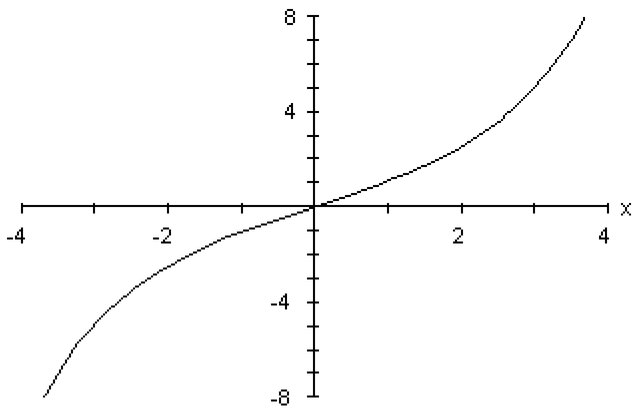
Description

FNShi returns the value of the hyperbolic sine integral of *x*, Shi(*x*). Shi(*x*) is defined by the relation

$$\mathrm{Shi}(x) = \int_0^x \frac{\sinh(t) - 1}{t}\,dt.$$

Large absolute values of *x* may produce results greater in magnitude than MAXREAL.

Shi(*x*)



Errors *FNShi* causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if Shi(*x*) would be greater than MAXREAL.

See Also

Chi, Ei, Si

Si

Sine integral.

Loading LOADSUB ALL FROM "EI.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNSi(X)

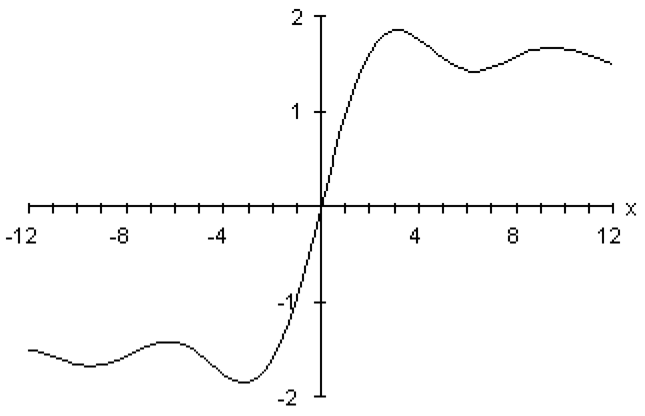
Description

FNSi returns the value of the sine integral of *x*, Si(*x*). Si(*x*) is defined by the relation

$$\mathrm{Si}(x) = \int_0^x \frac{\sin(t) - 1}{t}\,dt.$$

Si(*x*) is defined for all values of *x*.

Si(*x*)



Errors *Si* causes a BASIC error if its argument is not of type REAL.

See Also

Ei, Ci, Shi

Simpson

Integrate a function using Simpson's rule.

Loading

```
LOADSUB ALL FROM "SIMPSON.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB FNSimpson FROM "MATHLIB.HTS"
```

Usage

```
REAL A,B,X
INTEGER N
X=FNSimpson(F$,A,B,N)
```

Description

FNSimpson estimates the integral of the function named in *F$* between the points *a* and *b* where N is the number of divisions between a and b. The estimate is exact for functions that can be represented as polynomials of degree 2 or less.

F$* should contain the name of an HTBasic subroutine. The subroutine should take two REAL parameters. It should evaluate the function to be integrated at the second parameter and return its value in the first parameter. For example, if *F$ = "Test", then the subroutine *Test* should begin with the definition line

```
SUB Test(REAL Y,X)
```

where *X* and *Y* may be replaced by the names of any REAL parameters. The subroutine *Test* would evaluate the desired function at the value *X* and return the value in *Y*.

Errors

FNSimpson causes an HTBasic error if the subroutine named in *F$* is undefined. The subroutine named in *F$* may also cause HTBasic Errors when it is evaluated.

Solve

Solve a system of linear equations.

Loading

```
LOADSUB ALL FROM "SOLVE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Solve FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*)
CALL Solve(A(*),B(*))
```

Description

Solve finds the solution to the system of linear equations represented by *A* and *B* and returns the solution in *B*. *A* must be square, that is, it must have the same number of rows as columns. *B* must have the same number of rows as *A* and usually is a one-dimensional array (a vector). If A represents the matrix whose entries are stored in *A* and b represents the vector whose entries are stored in *B*, *Solve* finds the solution vector, x, for the matrix equation

$$Ax = b$$

and returns the solution in *B*, replacing the former contents of *B*. The contents of the array *A* are also destroyed by *Solve*.

The array *B* may be two-dimensional. In this case, after *solve* is called, each column in *B* contains the solution vector for the case when the input values in that column were used as b in the above equation.

Solve is equivalent to the BASIC lines

```
MAT Temp=INV(A)
MAT X=Temp*B
MAT B=X
```

except that the arrays *Temp* and *X* are not needed; the intermediate results overwrite some of the elements of *A*. *Solve* is faster than the above BASIC fragment, because the matrix inversion is not needed.

Errors

Solve causes a BASIC error if its arguments are both REAL arrays, if *A* is not square, if *B* doesn't have the same number of rows as *A*, or if *A* is singular.

See Also

Csolve

Std

Standard deviation of an array.

Loading LOADSUB ALL FROM "MEAN.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL A(*),Y
Y=FNStd(A(*))
```

Description

FNStd returns the standard deviation of the elements in the array *A*. The *standard deviation* is the square root of the average value of the squares of the differences between the elements in the array and the mean value of the elements. This version of the standard deviation uses the number of points in the array *A*, *n*, as the divisor in the averaging calculation, instead of the value *n* - 1 used in some formulas for standard deviation.

The *F_variance* subroutine returns the square of the standard deviation.

Errors

FNStd causes a BASIC error if its arguments are not of the types listed in the USAGE section, above.

See Also

Mean, F_variance

S_h10

Spherical Hankel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNS_h10(X)
```

Description

FNS_h10 returns the value of the spherical Hankel function of the first kind and order zero of *x*, *h*0(1)(*x*). The real component returned contains *j*0(*x*) and the imaginary component returned contains *y*0(*x*).

*H*0(1)(*x*) is sometime also called the *spherical Bessel function of the third kind, order 0*.

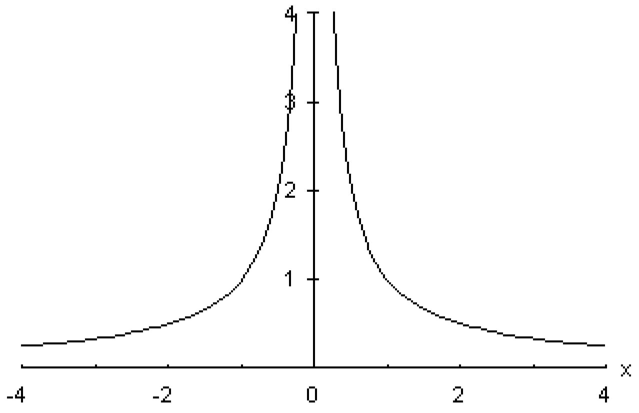
Errors

FNS_h10 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is zero, since the imaginary component of *h*0(1)(0) is infinite.

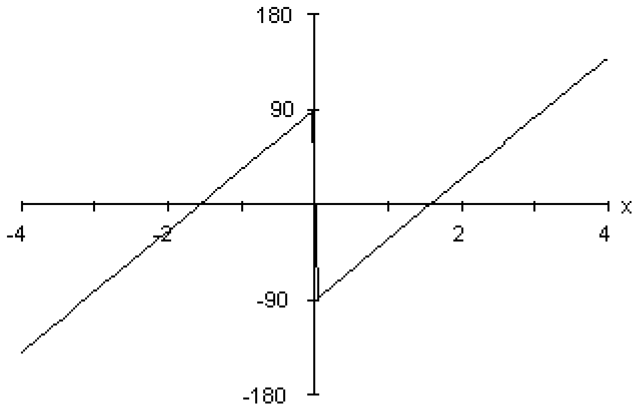
See Also

S_h11, S_h20, S_j0, S_y0

*h*0(1)(*x*)



Arg[*h*0(1)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

S_h11

Spherical Hankel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNS_h11(X)
```

Description

FNS_h11 returns the value of the spherical Hankel function of the first kind and order one of *x*, *h*1(1)(*x*). The real component returned contains *j*1(*x*) and the imaginary component returned contains *y*1(*x*).

*H*1(1)(*x*) is sometimes also called the *spherical Bessel function of the third kind, order 1*.

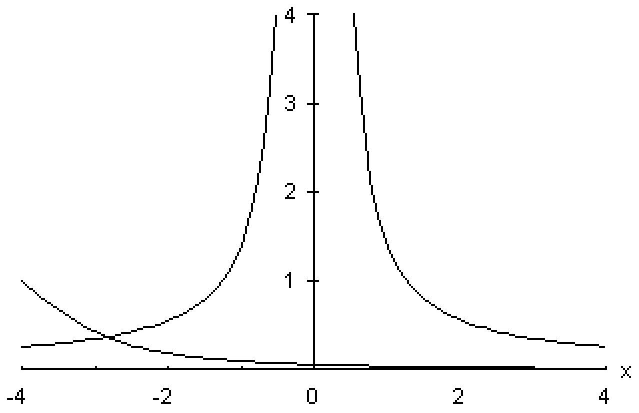
Errors

FNS_h11 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *h*1(1)(0) is -.

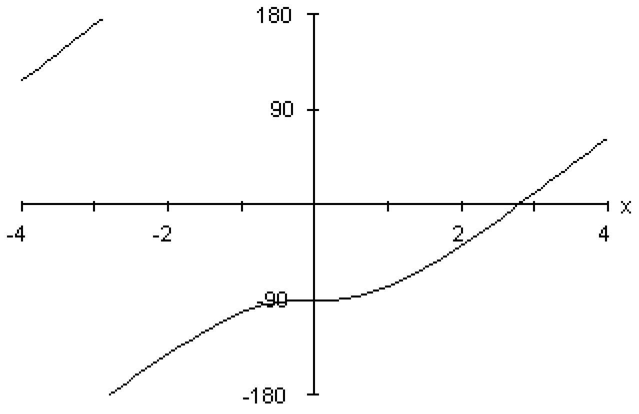
See Also

S_h10, S_h21, S_j1, S_y1

*h*1(1)(*x*)



Arg[*h*1(1)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

S_h1n

Spherical Hankel function of the first kind, order *n*.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNS_h1n(N,X)
```

Description

FNS_h1n returns the value of the spherical Hankel function of the first kind and order *n* of *x*, *hn*(1)(*x*). The real component returned contains *jn*(*x*) and the imaginary component returned contains *yn*(*x*).

Hn(1)(*x*) is sometime also called the *spherical Bessel function of the third kind, order* n.

Errors

FNS_h1n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *hn*(1)(0) is infinite.

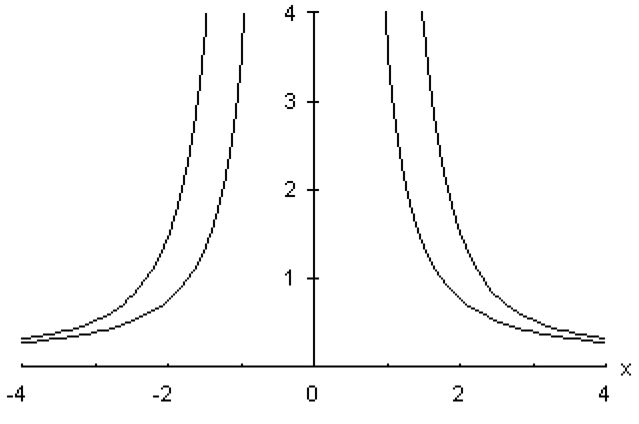
See Also

S_h10, S_h11, S_h2n, S_j0, S_j1, S_y0, S_y1

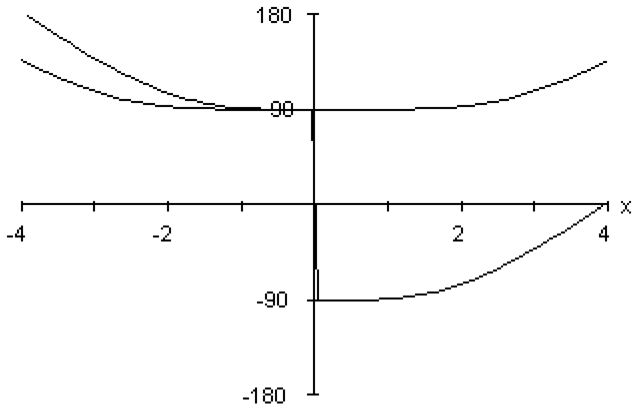
Note

The algorithm used computes the value of *hn*(1) using a recursion from the values of *h*0(1) and *h*1(1). The computation time increases with *n* and the computation accuracy decreases with *n*.

hn(1)(*x*)



Arg[*hn*(1)(*x*)], degrees


180
90
-90
-180
x
-4
-2
2
4

S_h20

Spherical Hankel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNS_h20(X)
```

Description

FNS_h20 returns the value of the spherical Hankel function of the second kind and order zero of *x*, *h*0(2)(*x*). The real component returned contains *j*0(*x*) and the imaginary component returned contains -*y*0(*x*).

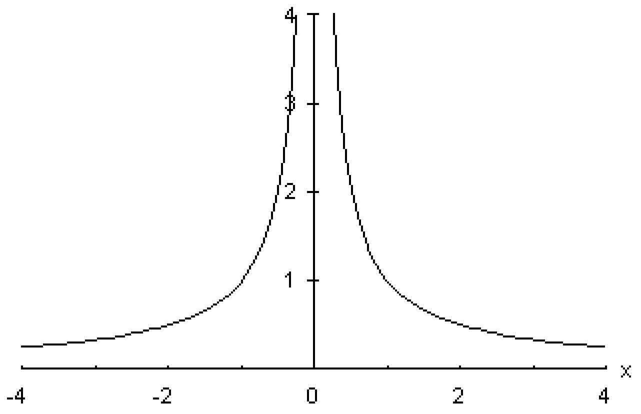
Errors

FNS_h20 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *h*0(2)(0) is -.

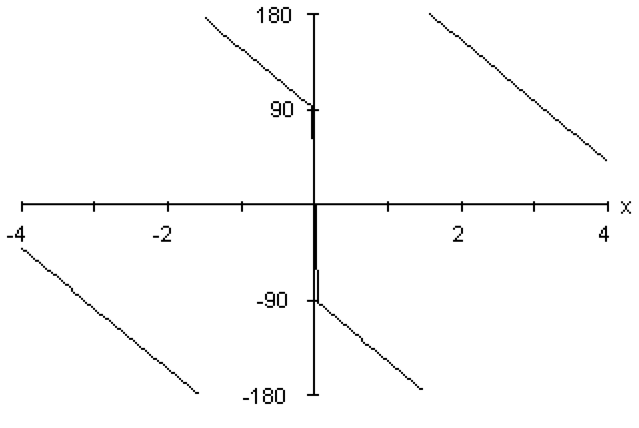
See Also

S_h10, S_h21, S_j0, S_y0

*h*0(2)(*x*)



Arg[*h*0(2)(*x*)], degrees


180
90
-90
-180
x
-4
-2
2
4

S_h21

Spherical Hankel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X
COMPLEX C
C=FNS_h21(X)
```

Description

FNS_h21 returns the value of the Hankel function of the second kind and order one of *x*, *h*1(2)(*x*). The real component returned contains *j*1(*x*) and the imaginary component returned contains -*y*1(*x*).

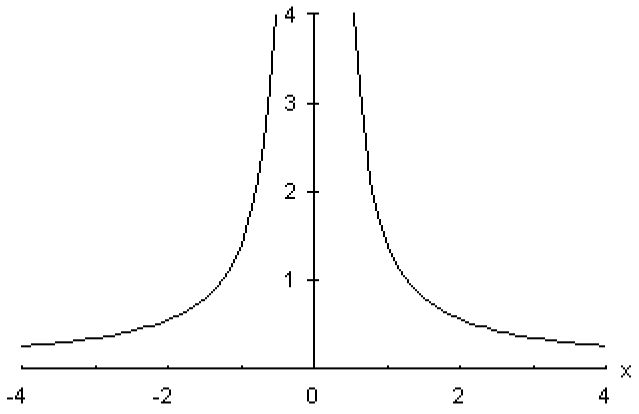
Errors

FNS_h21 causes a BASIC error if its argument is not of type REAL. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *h*1(2)(0) is infinite.

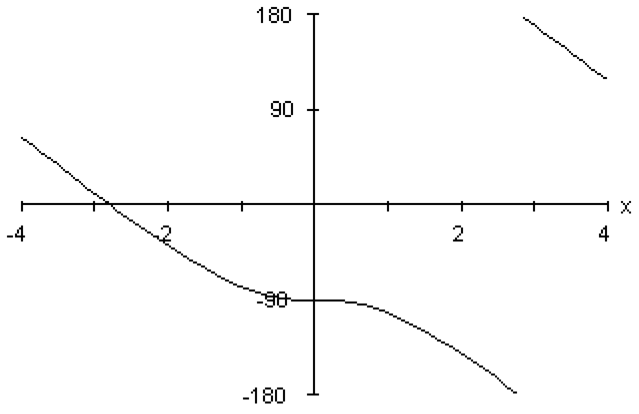
See Also

S_h11, S_h20, S_j1, S_y1

*h*1(2)(*x*)



Arg[*h*1(2)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

S_h2n

Spherical Hankel function of the second kind, order *n*.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNS_h2n(N,X)
```

Description

FNS_h2n returns the value of the spherical Hankel function of the second kind and order *n* of *x*, *hn*(2)(*x*). The real component returned contains *jn*(*x*) and the imaginary component returned contains -*yn*(*x*).

Errors

FNS_h2n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above. It also causes a BASIC error if the value of *x* is near zero, since the imaginary component of *hn*(2)(0) is infinite.

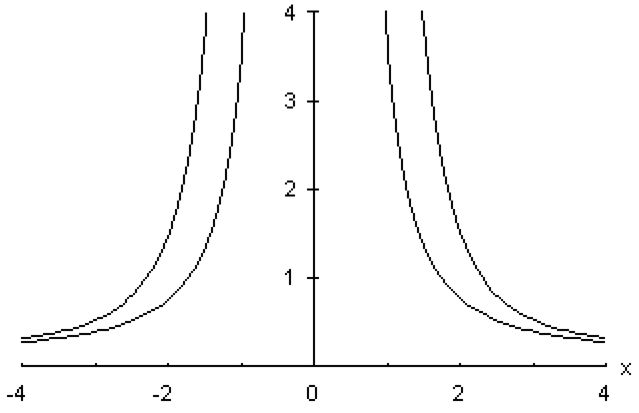
See Also

S_h1n, S_h20, S_h21, S_j0, S_j1, S_y0, S_y1

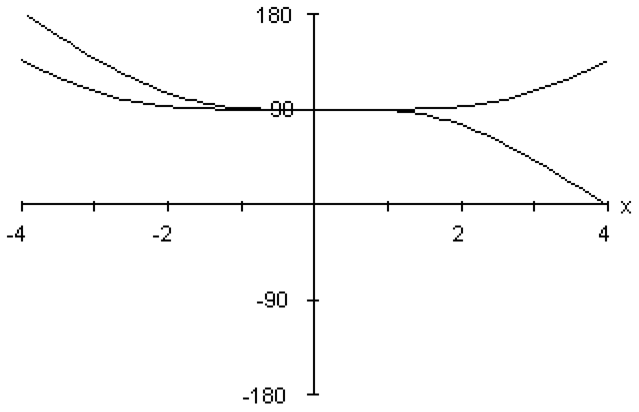
Note

The algorithm used computes the value of *hn*(2) using a recursion from the values of *h*0(2) and *h*1(2). The computation time increases with *n* and the computation accuracy decreases with *n*.

hn(2)(*x*)



Arg[*hn*(2)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

S_hh1n

Spherical Hankel function of the first kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X
COMPLEX C
C=FNS_hh1n(N,X)
```

Description

FNS_hh1n returns the value of the spherical Hankel function of the first kind and order *n*+½ of *x*, *hn*+½(1)(*x*). *Hn*+½(1)(*x*) is defined for all values of *n* and for all positive values of *x*.

Hn+½(1)(*x*) is sometimes also called the *spherical Bessel function of the third kind, order* n+½.

Errors

FNS_hh1n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

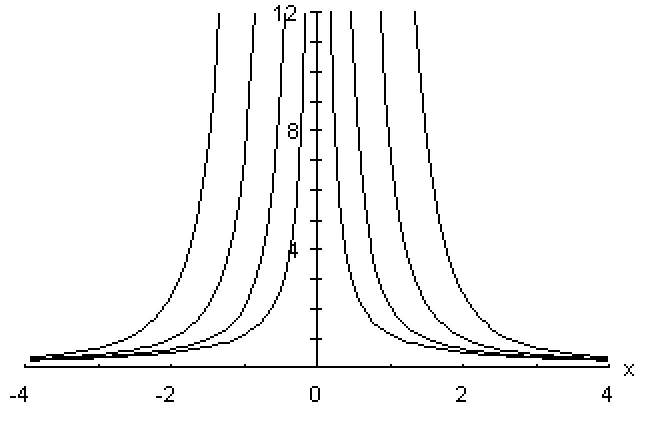
See Also

S_hh2n

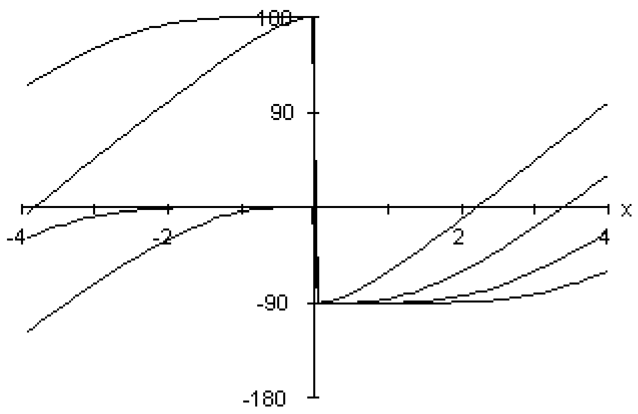
Note

The algorithm used computes the value of *hn*+½(1) using a recursion from the values of *h*½(1) and *h*1½(1). The computation time increases with *n* and the computation accuracy decreases with *n*.

hn+½(1)(*x*)



Arg[*hn*+½(1)(*x*)], degrees



180
90
-90
-180
-4
-2
2
4
x

S_hh2n

Spherical Hankel function of the second kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage
```
INTEGER N
REAL X
COMPLEX C
C=FNS_hh2n(N,X)
```

Description

FNS_hh2n returns the value of the spherical Hankel function of the second kind and order *n*+½ of *x*, *hn*+½(2)(*x*). *Hn*+½(2)(*x*) is defined for all values of *n* and for all positive values of *x*.

Errors

FNS_hh2n causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

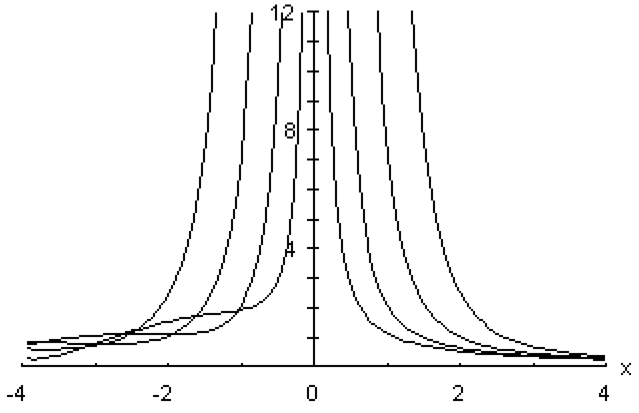
See Also

S_hh1n

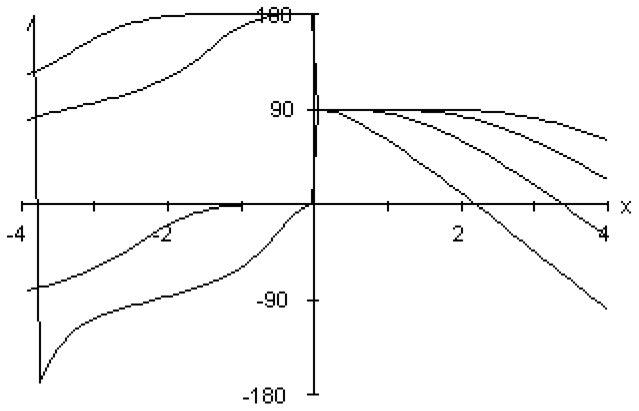
Note

The algorithm used computes the value of *hn*+½(2) using a recursion from the values of *h*½(2) and *h*1½(2). The computation time increases with *n* and the computation accuracy decreases with *n*.

hn+½(2)(*x*)



Arg[*hn*+½(2)(*x*)], degrees


180
90
-90
-180
-4
-2
2
4
x

S_i0

Modified spherical Bessel function of the first kind and order zero.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

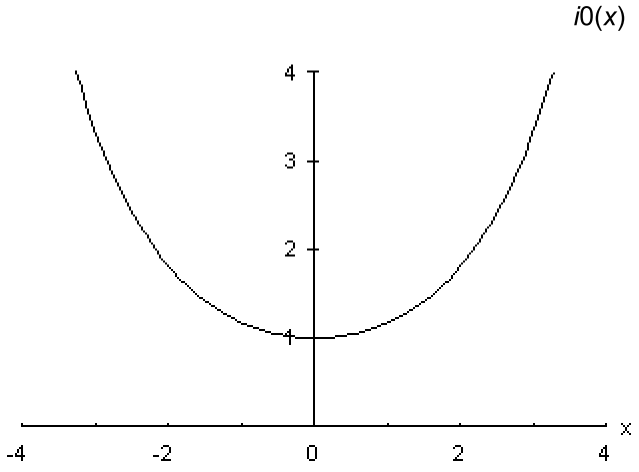
Usage REAL X,Y
Y=FNS_i0(X)

Description

FNS_i0 returns the value of the modified spherical Bessel function of the first kind and order zero of *x*, *i*0(*x*). *I*0(*x*) is defined for all values of *x*, but large positive values of *x* may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_i0 causes a BASIC error if its argument is not of type REAL or if the result would be larger than MAXREAL.



See Also S_i1, S_im0, S_in, S_k0

S_i1

Modified spherical Bessel function of the first kind and order one.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNS_i1(X)
```

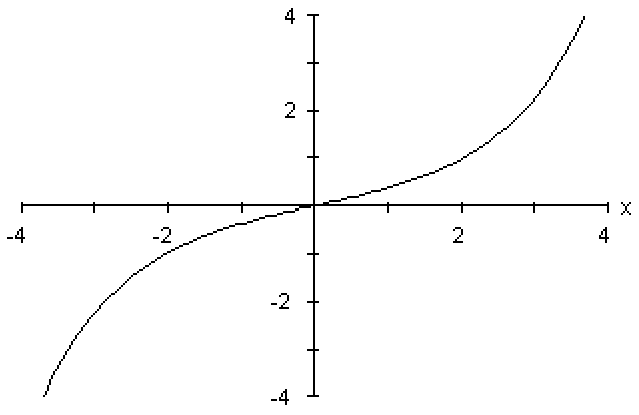
Description

FNS_i1 returns the value of the modified spherical Bessel function of the first kind and order one of *x*, *i*1(*x*). *I*1(*x*) is defined for all values of *x*, but large absolute values of *x* may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_i1 causes a BASIC error if its argument is not of type REAL or if the result would be larger than MAXREAL.

*i*1(*x*)



See Also S_i0, S_im1, S_in, S_k1

S_ihn

Modified spherical Bessel function of the first and second kinds, order *n*+½.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_ihn(N,X)
```

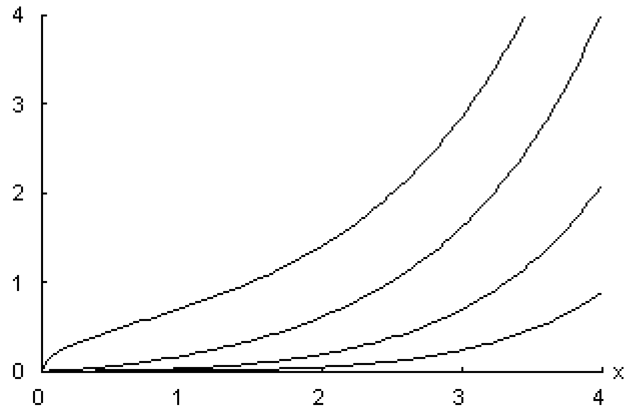
Description

FNS_ihn returns the value of the modified spherical Bessel function of the first or second kind and order *n*+½ of *x*, $i_{n+½}(x)$. The function is called a function of the first kind if *n* is positive and a function of the second kind if *n* is negative. $I_{n+½}(x)$ is defined for all values of *n* and for all positive values of *x*. If *n* is positive or zero, $i_{n+½}(x)$ is also defined for *x* = 0.

Errors

FNS_ihn causes a BASIC error if its arguments are not of the types shown in the **USAGE** section, above, or if *x* is out of the range of definition explained above.

$i_{n+½}(x)$



See Also S_in, S_khn

Note

The algorithm used computes the value of $i_{n+½}$ using a recursion from the values of $i_{½}$ and $i_{1½}$. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_in

Modified spherical Bessel function of the first and second kinds and order *n*.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_in(N,X)
```

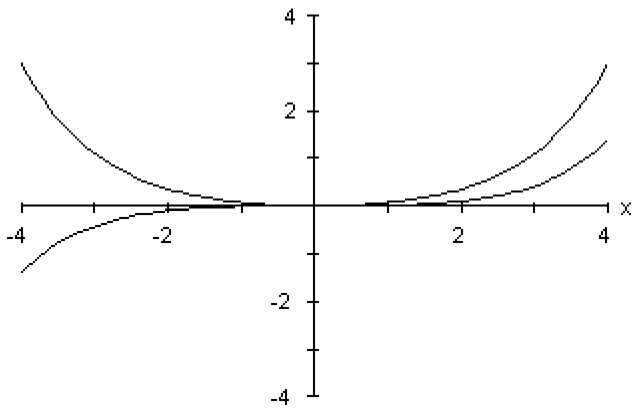
Description

FNS_in returns the value of the modified spherical Bessel function of order *n* of *x*, *in*(*x*). The function is called a function of the first kind if *n* is positive and a function of the second kind if *n* is negative. *In*(*x*) is defined for all values of *n* and for all values of *x*, but large absolute values of *x* may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_in causes a BASIC error if its arguments are not of the types listed in the **USAGE** section, above, or if the result would be larger than MAXREAL.

in(*x*)



See Also S_i0, S_i1, S_ihn, S_kn

Note

The algorithm used computes the value of *in* using a recursion from the values of *i*0 and *i*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_j0

Spherical Bessel function of the first kind, order zero.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNS_j0(X)

Description

FNS_j0 returns the value of the spherical Bessel function of the first kind and order zero of *x*, *j*0(*x*). *J*0(*x*) is defined for all values of *x*.

*J*0(*x*) is defined by the expression

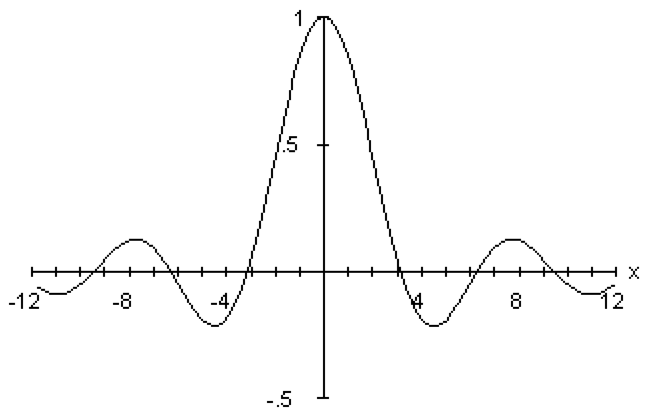
$$J_0(x) = \begin{cases} \dfrac{\sin(x)}{x}, & x \neq 0 \\ 1, & x = 0 \end{cases}.$$

This function is also often called the *sinc* function.

Errors

FNS_j0 causes a BASIC error if its argument is not of type REAL.

See Also *j*0(*x*)



See Also S_j1, S_jn, S_y0

S_j1

Spherical Bessel function of the first kind, order one.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNS_j1(X)

Description

FNS_j1 returns the value of the spherical Bessel function of the first kind and order one of *x*, *j*1(*x*). *J*1(*x*) is defined for all values of *x*.

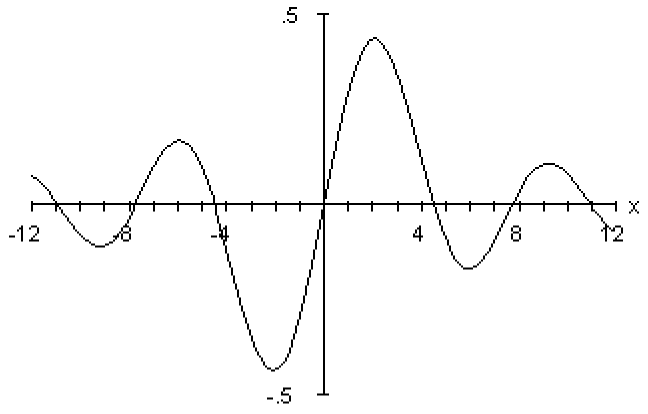
Errors

FNS_j1 causes a BASIC error if its argument is not of type REAL.

See Also

S_j0, S_jn, S_y1

*j*1(*x*)



S_jhn

Spherical Bessel function of the first kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_jhn(N,X)
```

Description

FNS_jhn returns the value of the spherical Bessel function of the first kind and order $n+½$ of x, $j_{n+½}(x)$. $J_{n+½}(x)$ is defined for all values of n and for all positive values of x. If n is positive or zero, $j_{n+½}(x)$ is also defined for $x = 0$.

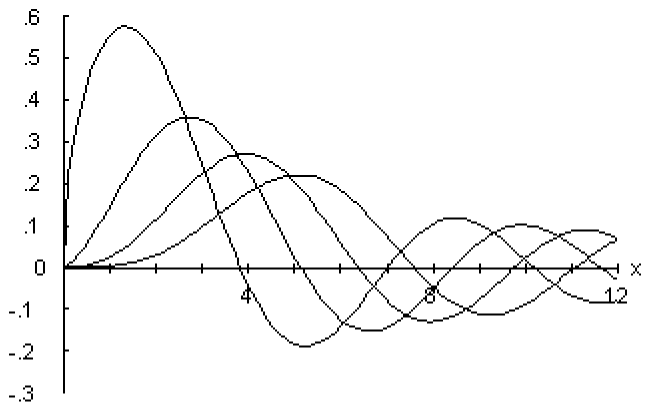
Errors

FNS_jhn causes a BASIC error if its arguments are not of the types shown in the **USAGE** section, above, or if x is out of the range of definition explained above.

See Also

S_jn, S_yh1n, S_hh1n, S_hh2n, S_yhn

$j_{n+½}(x)$



Note The algorithm used computes the value of $j_{n+½}$ using a recursion from the values of $j_{½}$ and $j_{1½}$. The computation time increases with n and the computation accuracy decreases with n.

S_jn

Spherical Bessel function of the first kind, order *n*.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_jn(N,X)
```

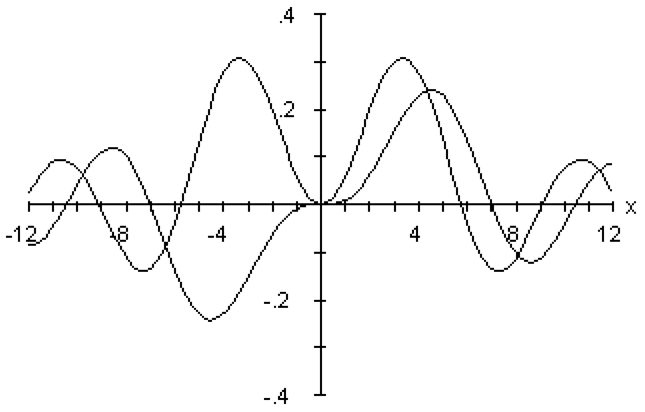
Description

FNS_jn returns the value of the spherical Bessel function of the first kind and order *n* of *x*, *jn*(*x*). *Jn*(*x*) is defined for all values of *x*. For *n* < 0, large negative values of *x* or values of *x* near zero may cause *jn*(*x*) to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_jn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the result would be larger than MAXREAL.

jn(*x*)



See Also S_j0, S_j1, S_yn

Note The algorithm used computes the value of *jn* using a recursion from the values of *j*0 and *j*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_k0

Modified spherical Bessel function of the third kind and order zero.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB S_k0 FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNS_k0(X)

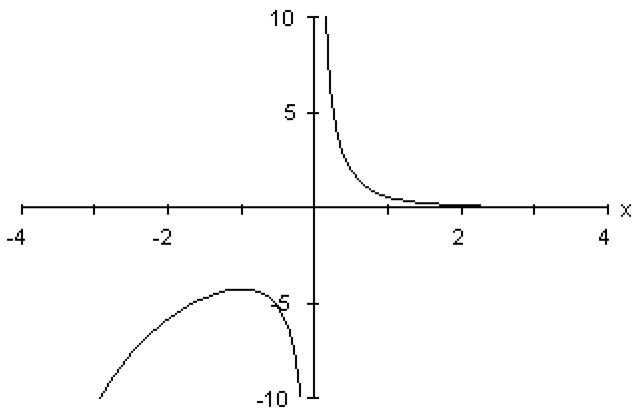
Description

FNS_k0 returns the value of the modified spherical Bessel function of the third kind and order zero of *x*, *k*0(*x*). *K*0(*x*) is defined for all values of *x* except zero, but large negative values of *x* or values of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_k0 causes a BASIC error if its argument is not of type REAL, if *x* is zero, or if the result would be larger than MAXREAL.

*k*0(*x*)



See Also S_i0, S_im0, S_k1, S_kn

S_k1

Modified spherical Bessel function of the third kind and order one.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNS_k1(X)
```

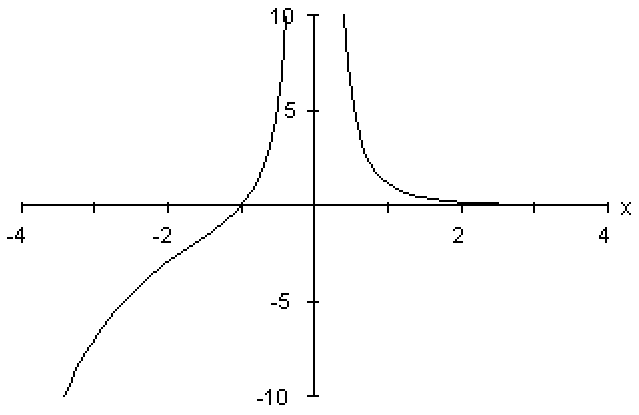
Description

FNS_k1 returns the value of the modified spherical Bessel function of the third kind and order one of *x*, *k*1(*x*). *K*1(*x*) is defined for all values of *x* except zero, but large negative values of *x* or values of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_k1 causes a BASIC error if its argument is not of type REAL, if *x* is zero, or if the result would be larger than MAXREAL.

*k*1(*x*)



See Also S_i0, S_im0, S_k0, S_kn

S_khn

Modified spherical Bessel function of the third kind, order *n*+½.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_khn(N,X)
```

Description

FNS_khn returns the value of the modified spherical Bessel function of the third kind and order *n*+½ of *x*, *kn*+½(*x*). *Kn*+½(*x*) is defined for all values of *n* and for all positive values of *x*.

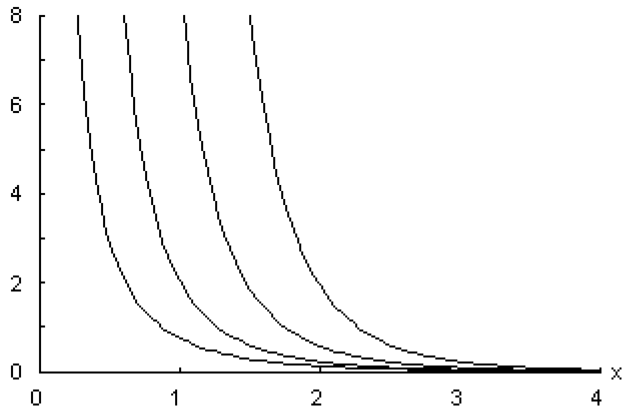
Errors

FNS_khn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

See Also

S_ihn, S_kn

kn+½(*x*)



Note The algorithm used computes the value of *kn*+½ using a recursion from the values of *k*½ and *k*1½. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_kn

Modified spherical Bessel function of the third kind, order *n*.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage INTEGER N
REAL X,Y
Y=FNS_kn(N,X)

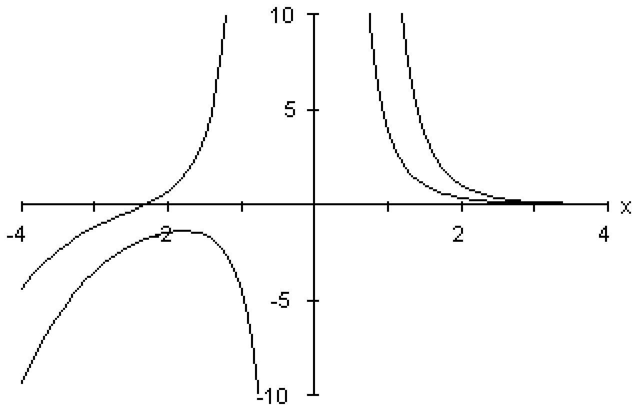
Description

FNS_kn returns the value of the modified spherical Bessel function of the third kind and order *n* of *x*, *kn*(*x*). *Kn*(*x*) is defined for all values of *x* and *n* except *x* = 0, but large negative values of *x* or values of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_kn causes a BASIC error if its arguments are not of the types listed in the usage section, above, if *x* is zero, or if the result would be larger than MAXREAL.

kn(*x*)



See Also S_in, S_imn, S_k0, S_k1

Note

The algorithm used computes the value of *kn* using a recursion from the values of *k*0 and *k*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_y0

Spherical Bessel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL X,Y
Y=FNS_y0(X)
```

Description

FNS_y0 returns the value of the spherical Bessel function of the second kind and order zero of *x*, *y*0(*x*). *Y*0(*x*) is defined for all values of *x* except zero, but values of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

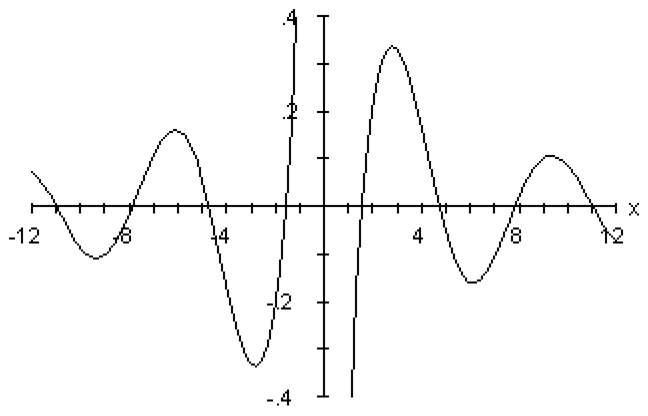
Errors

FNS_y0 causes a BASIC error if its argument is not of type REAL, if *x* is zero, or if the result would be larger in magnitude than MAXREAL.

See Also

S_y1, S_yn

*y*0(*x*)



S_y1

Spherical Bessel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESMS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNS_y1(X)

Description

FNS_y1 returns the value of the spherical Bessel function of the second kind and order one of *x*, *y*1(*x*). *Y*1(*x*) is defined for all values of *x* except zero, but values of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

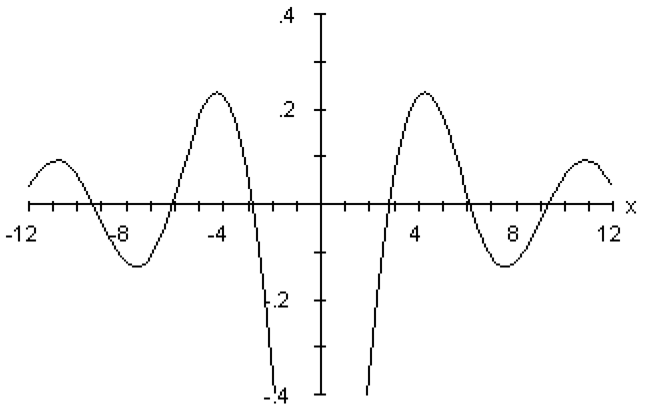
Errors

FNS_y1 causes a BASIC error if its argument is not of type REAL, if *x* is zero, or if the result would be larger in magnitude than MAXREAL.

See Also

S_y0, S_yn

*y*1(*x*)



S_yhn

Spherical Bessel function of the second kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_yh1(N,X)
```

Description

FNS_yhn returns the value of the spherical Bessel function of the second kind and order *n*+½ of *x*, *yn*+½(*x*). *Yn*+½(*x*) is defined for all values of *n* and for all positive values of *x*.

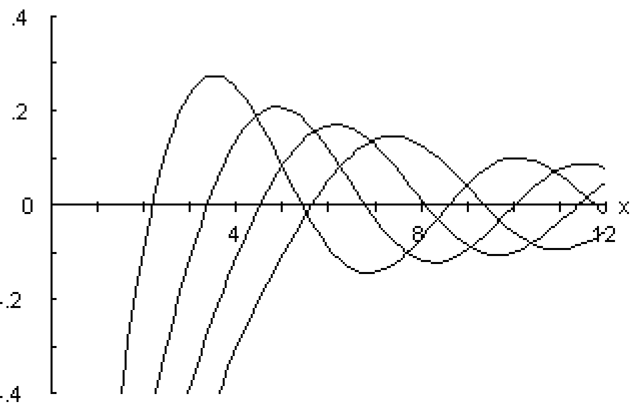
Errors

FNS_yhn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

See Also

S_jhn, S_yh1n, S_hh1n, S_hh2n, S_yhn, S_yn

yn+½(*x*)



Note

The algorithm used computes the value of *yn*+½ using a recursion from the values of *y*½ and *y*1½. The computation time increases with *n* and the computation accuracy decreases with *n*.

S_yn

Spherical Bessel function of the second kind and order *n*.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNS_yn(N,X)
```

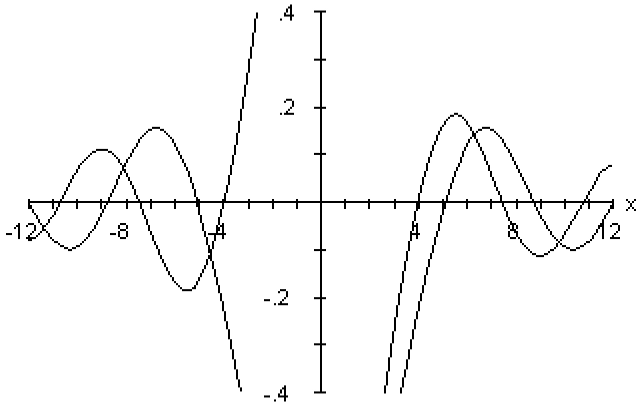
Description

FNS_yn returns the value of the spherical Bessel function of the second kind and order *n* of *x*, *yn*(*x*). For *n* < 0, *yn*(*x*) is defined for all values of *x*. For *n* 0, *yn*(*x*) is defined for all values of *x* except zero, but large negative values of *x* or values *n* of *x* near zero may cause the result to be larger in magnitude than MAXREAL, the largest value representable.

Errors

FNS_yn causes a BASIC error if its arguments are not of the types listed in the **USAGE** section, above, if *x* is zero, if *n* is not in the range described above, or if the result would be larger than MAXREAL.

yn(*x*)



See Also S_jn, S_y0, S_y1

Note

The algorithm used computes the value of *yn* using a recursion from the values of *y*0 and *y*1. The computation time increases with *n* and the computation accuracy decreases with *n*.

Tolinear

Conversion from log to linear representation.

Loading

```
LOADSUB ALL FROM "TOLINEAR.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Tolinear FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*),X
CALL Tolinear(A(*),X,B(*))
```

Description

Tolinear converts the data in the array *A* into linear representation and returns the results in the array *B*. The factor *x* is used to scale the data in the array *A* before it is converted; each point in *A* is divided by *x* before conversion is done. *Tolinear* is usually used to convert data in decibel representation into linear form. *Tolinear* is equivalent to the following BASIC lines

```
INTEGER I
FOR I=1 TO size(A)
B(I)=10^(A(I)/X)
NEXT I
```

where *size(A)* is the number of elements in the array *A*. In most cases, *x* is either 10 or 20.

Errors

Tolinear causes a BASIC error if its arguments are not all of type REAL, if *x* is zero, or if *A* and *B* do not have the same number of elements.

Tolog

Conversion from linear to logarithmic representation.

Loading

```
LOADSUB ALL FROM "TOLOG.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB Tolog FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*),X
CALL Tolog(A(*),X,B(*))
```

Description

Tolog converts the data in the array *A* into logarithmic representation and returns the results in the array *B*. The factor *x* is used to scale the data in the array *A* after it is converted; each point in *A* is multiplied by *x* after conversion is done. *Tolog* is usually used to convert data to decibel representation. *Tolog* is equivalent to the following BASIC lines

```
INTEGER I
FOR I=1 TO size(A)
B(I)=LOG(A(I))*X
NEXT I
```

where *size(A)* is the number of elements in the array *A*. In most cases, *x* is either 10 or 20.

Errors

Tolog causes a BASIC error if its arguments are not all of type REAL or if *A* and *B* do not have the same number of elements.

Tn

Chebyshev polynomial of the first kind.

Loading LOADSUB ALL FROM "CHEBY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNTn(N,X)
```

Description

FNTn returns the value of the Chebyshev polynomial of the first kind and order *n* of *x*, *Tn*(*x*). *N* must be positive or zero. *Tn*(*x*) is defined for all values of *x*, although it is most commonly used only with values of *x* between -1 and +1.

There are other, less used, types of Chebyshev polynomials defined that are not included in the Math Library. These can easily calculated from *Tn*(*x*) and *Un*(*x*) by using the following formulas:

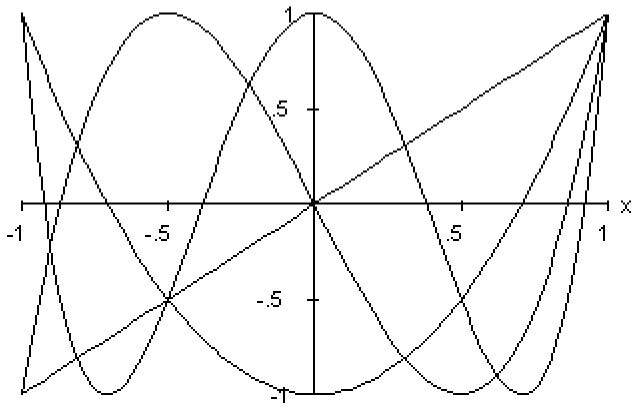
$$C_n(x) = 2T_n\left(\frac{x}{2}\right)$$

$$S_n(x) = U_n\left(\frac{x}{2}\right)$$

$$T*_n(x) = T_n(2x-1)$$

$$U*_n(x) = U_n(2x-1)$$

Tn(*x*)



Errors *FNTn* causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

See Also

Un

Note

For *n* > 12, the algorithm used computes the value of *Tn* using a recursion from the values of *T*11 and *T*12. The computation time increases with *n*-11 and the computation accuracy decreases with *n*-11.

Trapezoid

Integration using the Trapezoid Rule.

Loading LOADSUB ALL FROM "TRAPEZOID.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
REAL A(*),S,Y
Y=FNTrapezoid(A(*),S)
```

Description

FNTrapezoid approximates the integral of the function whose samples are in the array *A*. The elements of *A* are assumed to be equally-spaced. The parameter *s* contains the value of the distance between adjacent elements of *A*.

The integral is calculated by summing half the value of the first and last points in *A* and the values of the interior points in *A*. This sum is multiplied by *s*. This method is often called the *trapezoid rule*, and is described in most texts on numerical mathematical methods.

Errors

FNTrapezoid causes an HTBasic error if the array *A* contains fewer than 2 points.

See Also

Simpson

Un

Chebyshev polynomial of the second kind.

Loading LOADSUB ALL FROM "CHEBY.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNUn(N,X)
```

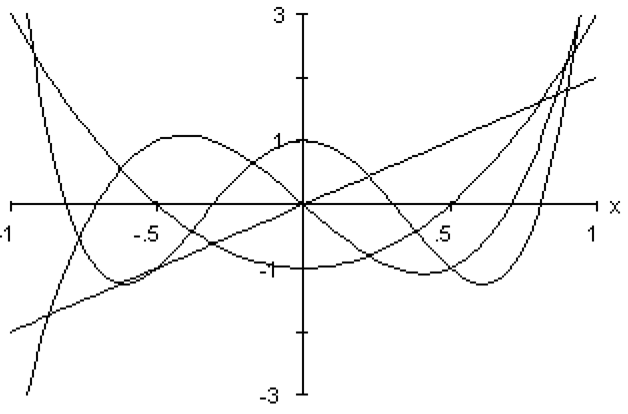
Description

FNUn returns the value of the Chebyshev polynomial of the second kind and order *n* of *x*, *Un*(*x*). *N* must be positive or zero. *Un*(*x*) is defined for all values of *x*, although it is most commonly used only with values of *x* between -1 and +1.

Errors

FNUn causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if the polynomial's absolute value would be larger than MAXREAL, the largest value representable.

Un(*x*)



See Also Tn

Note

For *n* > 12, the algorithm used computes the value of *Un* using a recursion from the values of *U*11 and *U*12. The computation time increases with *n*-11 and the computation accuracy decreases with *n*-11.

V_cdot

Scalar, or dot, product of two complex vectors.

Loading

```
LOADSUB ALL FROM "VDOT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
COMPLEX A(*),B(*),Z
Z=FNV_cdot(A(*),B(*))
```

Description

FNV_cdot computes the scalar, or dot, product of the vectors *a* and *b*. This is done by multiplying each element of *a* by the complex conjugate of the corresponding element *b* and summing the products.

Errors

FNV_cdot causes a BASIC error if *A* and *B* are not both of type COMPLEX or if they do not have the same number of elements.

See Also

V_dot

V_cosine

Cosine of angle between two vectors.

Loading LOADSUB ALL FROM "VCOSINE.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL A(*),B(*),Y
Y=FNV_cosine(A(*),B(*))

Description

FNV_cosine computes the cosine of the angle between the vectors *a* and *b*. This is done by evaluating

$$\frac{a \cdot b}{|a||b|}.$$

where "" denotes the dot, or scalar, product and *a* and *b* denote the *L*-2 norm of the vectors *a* and *b*.

Errors

V_cosine causes a BASIC error if *A* and *B* are not both of type REAL, if they do not have the same number of elements, or either *A* or *B* contain all zeros.

See Also

Norm, V_dot, V_proj

V_dot

Scalar, or dot, product of two real vectors.

Loading

```
LOADSUB ALL FROM "VDOT.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*),Y
Y=FNV_dot(A(*),B(*))
```

Description

FNV_dot computes the scalar, or dot, product of the vectors *a* and *b*. This is done by multiplying each element of *a* by the corresponding element *b* and summing the products.

Errors

FNV_dot causes a BASIC error if *A* and *B* are not both of type REAL or if they do not have the same number of elements.

See Also

V_cosine, V_cdot, V_prod

V_prod

Vector or cross product.

Loading

```
LOADSUB ALL FROM "VPROD.HTS"
or LOADSUB V_prod FROM "MATHLIB.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*),P(*)
CALL V_prod(A(*),B(*),P(*))
```

Description

V_prod computes the vector, or cross, product of the real vectors *a* and *b* and returns the result in *c*. *A*, *b*, and *c* must have exactly three elements each.

Errors

V_prod causes a BASIC error if *A*, *B*, or *C* are not all of type REAL or do not all have exactly three elements.

See Also

V_dot, V_proj

V_proj

Projection of one vector on another.

Loading

```
LOADSUB ALL FROM "VPROJ.HTS"
or LOADSUB FROM "MATHLIB.HTS"
```

Usage

```
REAL A(*),B(*),Y
Y=FNV_proj(A(*),B(*))
```

Description

FNV_proj computes the length of the projection of vector *a* onto the direction of vector *b*. This is done by evaluating

$$\frac{a \cdot b}{|b|}.$$

where "•" denotes the dot, or scalar, product and *b* denotes the *L*-2 norm of the vector *b*.

Errors

V_proj causes a BASIC error if *A* and *B* are not both of type REAL, if they do not have the same number of elements, or if *B* contains all zeros.

See Also

V_cosine, V_dot, V_prod

Waveform

Fill an array with a periodic waveform.

Loading LOADSUB ALL FROM "WAVEFORM.HTS"
or LOADSUB Waveform FROM "MATHLIB.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL P,A,B,S,Y(*)
CALL Waveform(P,A,B,S,N,Y(*))
```

Description

Waveform fills the array *Y* with a periodic waveform of type *n* having the period *p*, amplitude *a*, bias *b*, and starting point *s*. If *Yk* refers to an element of array *Y*, beginning with *k* = 0, the table below shows the expression for *Yk* for each value of *n*.

n	Type	Expression
1	sine	$Y_k = a\sin\left(\frac{2\pi[k-s]}{p}\right) + b$
2	square	$Y_k = \begin{cases} a+b, & 0 \le \text{fract}\left(\frac{k-s}{p}\right) < \frac{1}{2} \\ -a+b, & \frac{1}{2} \le \text{fract}\left(\frac{k-s}{p}\right) < 1 \end{cases}$
3	triangle	$Y_k = \begin{cases} 4a\,\text{fract}\left(\frac{k-s}{p}\right) + b, & 0 \le \text{fract}\left(\frac{k-s}{p}\right) < \frac{1}{4} \\ 4a\left[\frac{1}{2} - \text{fract}\left(\frac{k-s}{p}\right)\right] + b, & \frac{1}{4} \le \text{fract}\left(\frac{k-s}{p}\right) < \frac{3}{4} \\ 4a\left[-1 + \text{fract}\left(\frac{k-s}{p}\right)\right] + b, & \frac{3}{4} \le \text{fract}\left(\frac{k-s}{p}\right) < 1 \end{cases}$
4	sawtooth	$Y_k = \begin{cases} 2a\,\text{fract}\left(\frac{k-s}{p}\right) + b, & 0 \le \text{fract}\left(\frac{k-s}{p}\right) < \frac{1}{2} \\ 2a\left[-1 + \text{fract}\left(\frac{k-s}{p}\right)\right] + b, & \frac{1}{2} \le \text{fract}\left(\frac{k-s}{p}\right) < 1 \end{cases}$

In the above expressions, fract(*x*) is the fractional part of *x*, calculated by finding the difference between *x* and the next lower integer from *x*. Fract(*x*) is between 0, inclusive, and 1, exclusive.

All the parameters may take any value except the period, *p*, which must be positive. *P* refers to the number of elements in the array *Y* between repetitions of the waveform. *P* and *s* do not need to be integers. The type, *n*, must be between 1 and 4, inclusive.

If *p* or *s* is contained in a variable of type INTEGER, be sure to use the BASIC REAL command to change the variable to a REAL value when passing it to the *Pulse* routine.

Other periodic waveforms can be produced using these four types. For example, a cosine wave can be produced from the sine waveform by setting *s* to -*p*/4. A falling sawtooth wave can be produced from the sawtooth waveform by using a negative value for *a*.

The four types of waveform are plotted on the following pages for *a* = 1, *b* = 0, *s* = 0, and *p* = 100.

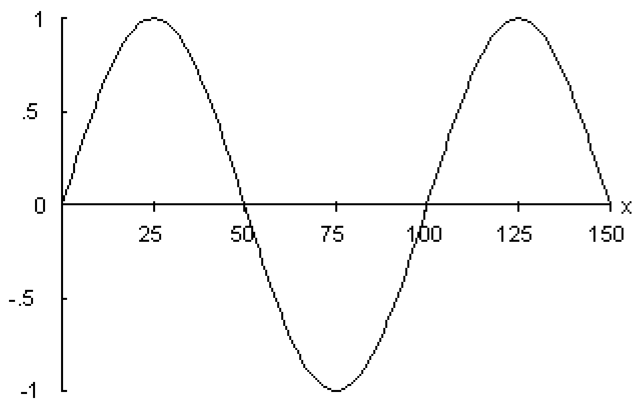
Errors

Waveform causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *p* is not positive, or if *n* is not between 1 and 4, inclusive.

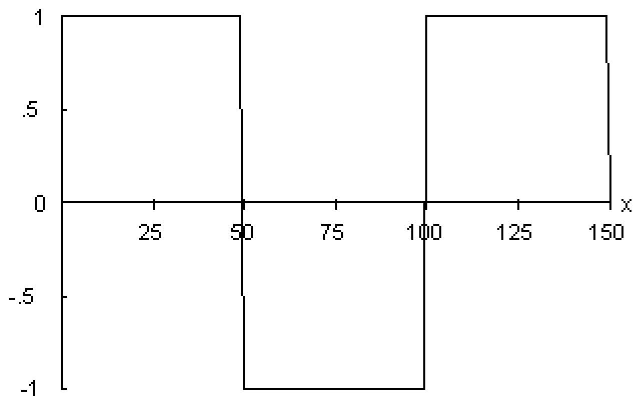
See Also

Pulse

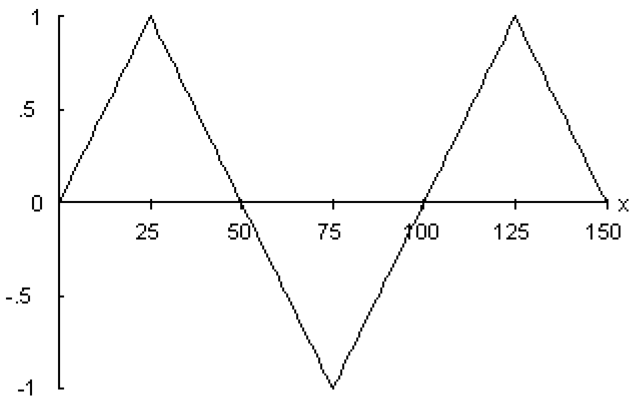
n = 1



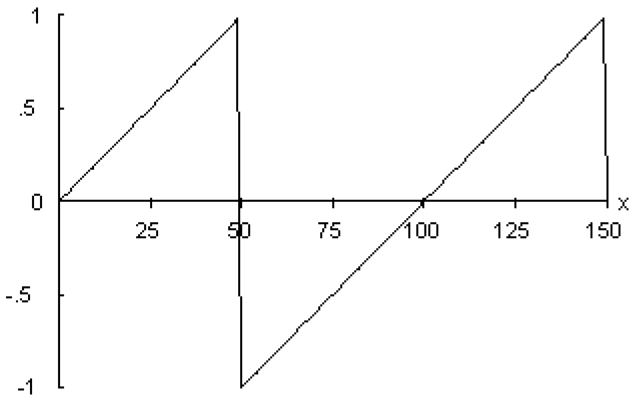
n = 2



n = 3



n = 4



W_bartlett

Bartlett window.

Loading LOADSUB ALL FROM "W_TRAPEZ.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_bartlett FROM "MATHLIB.HTS"

Usage REAL A(*),B(*)
CALL W_bartlett(A(*),B(*))

Description

W_bartlett multiplies the sequence in array *A* by a Bartlett window function and returns the product in array *B*. Array *A* is unmodified. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

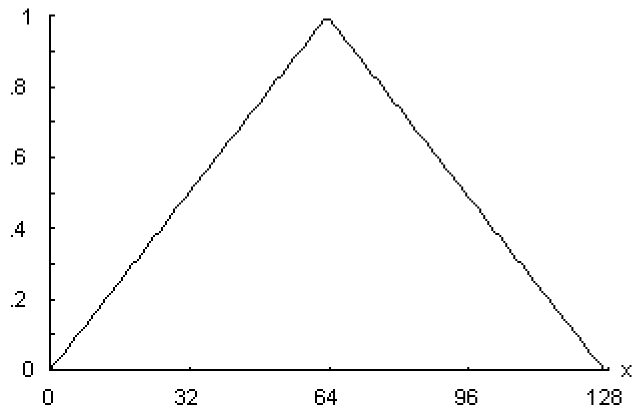
$$w_k = \begin{cases} \frac{2k}{N-1}, & k \le \frac{N-1}{2} \\ 2 - \frac{2k}{N-1}, & k > \frac{N-1}{2} \end{cases}$$

In the above formula, *k* ranges in value from 0 to *N* - 1.

If the array *B* contains more elements than *A*, the extra elements in *B* are unmodified.

The Bartlett window is a special case of the window generated by the *W_trapezoid* routine.

Bartlett window for *N*=128



Errors *W_bartlett* causes a BASIC error if its arguments are not both REAL arrays or if *B* contains fewer elements than *A*.

See Also

Waveform, W_blackman, W_cosine, W_hamming, W_hanning, W_trapezoid

W_blackman

Blackman window.

Loading LOADSUB ALL FROM "BLACKMAN.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_blackman FROM "MATHLIB.HTS"

Usage REAL A(*),B(*)
CALL W_blackman(A(*),B(*))

Description

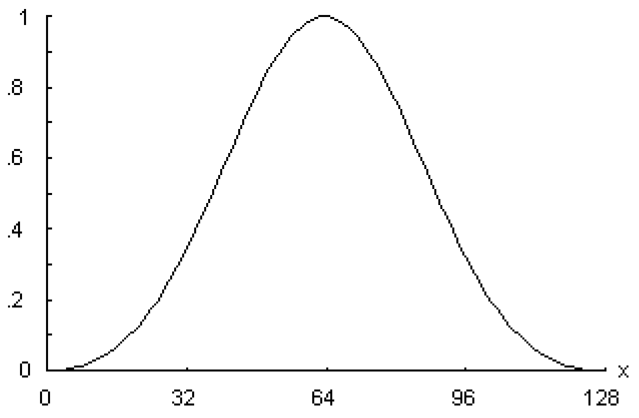
W_blackman multiplies the sequence in array *A* by a Blackman window function and returns the product in array *B*. Array *A* is unmodified. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

$$w_k = 0.42 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) + 0.08\cos\left(\frac{4\pi k}{N-1}\right).$$

In the above formula, *k* ranges in value from 0 to *N* - 1.

If the array *B* contains more elements than *A*, the extra elements in *B* are unmodified.

Blackman window for *N*=128



Errors *W_blackman* causes a BASIC error if its arguments are both REAL arrays or if *B* contains fewer elements than *A*.

See Also

Waveform, W_bartlett, W_cosine, W_hamming, W_hanning

W_cosine

Cosine window.

Loading LOADSUB ALL FROM "W_COSINE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_cosine FROM "MATHLIB.HTS"

Usage

```
REAL A(*),B(*)
REAL R
CALL W_cosine(A(*),R,B(*))
```

Description

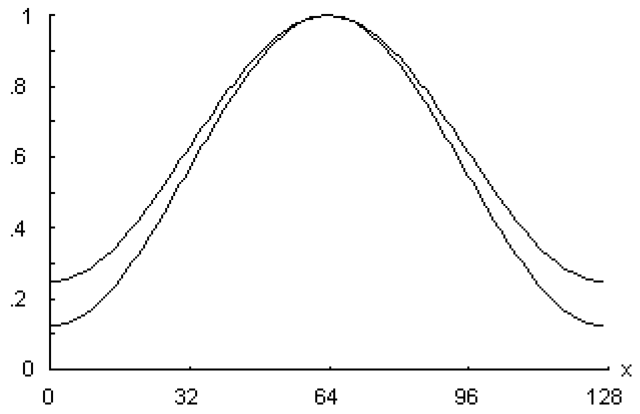
W_cosine multiplies the sequence in array *A* by a cosine window function of parameter *r* and returns the product in array *B*. Array *A* is unmodified. *R* is the peak-to-peak amplitude of the cosine portion of the window; it must be between 0 and 1, inclusive. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

$$w_k = 1 - \frac{r}{2} - \frac{r}{2}\cos\left(\frac{2\pi k}{N-1}\right).$$

In the above formula, *k* ranges in value from 0 to *N* - 1.

If the array *B* contains more elements than *A*, the extra elements in *B* are unmodified.

Cosine windows for *N*=128



Special cases of the cosine window are the Hamming and Hanning windows; these are available as separate subroutines.

Errors

W_cosine causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *r* is not between 0 and 1, inclusive, or if *B* contains fewer elements than *A*.

See Also

Waveform, W_bartlett, W_blackman, W_hamming, W_hanning

W_hamming

Hamming window.

Loading LOADSUB ALL FROM "W_COSINE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_hamming FROM "MATHLIB.HTS"

Usage

```
REAL A(*),B(*)
CALL W_hamming(A(*),B(*))
```

Description

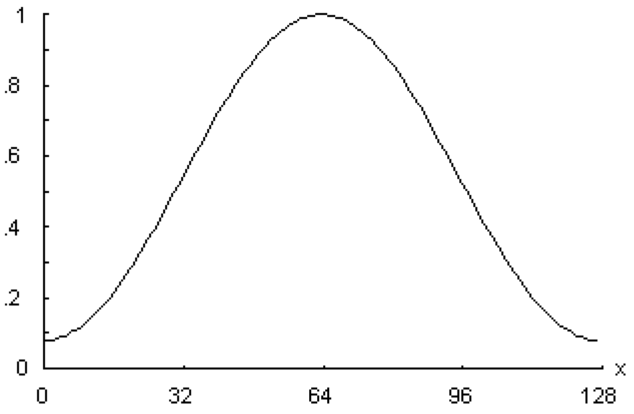
W_hamming multiplies the sequence in array *A* by the Hamming window function and returns the product in array *B*. Array *A* is unmodified. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

$$w_k = 0.54 - 0.46\cos\left(\frac{2\pi k}{N-1}\right).$$

In the above formula, *k* ranges in value from 0 to *N* - 1.

The Hamming window is the same as the cosine window with the parameter *r* set to 0.46.

Hamming window for *N*=128



Errors *W_hamming* causes a BASIC error if its arguments are not both REAL arrays or if *B* contains fewer elements than *A*.

See Also

Waveform, W_bartlett, W_blackman, W_cosine, W_hanning

W_hanning

Hanning window.

Loading

LOADSUB ALL FROM "W_COSINE.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_hanning FROM "MATHLIB.HTS"

Usage

REAL A(*),B(*)
CALL W_hanning(A(*),B(*))

Description

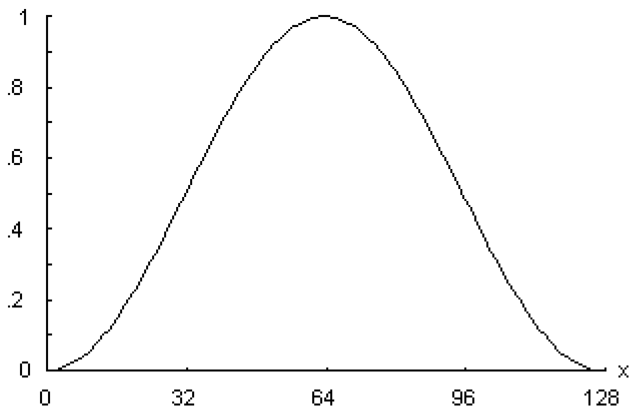
W_hanning multiplies the sequence in array *A* by the Hanning window function and returns the product in array *B*. Array *A* is unmodified. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

$$w_k = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi k}{N-1}\right).$$

In the above formula, *k* ranges in value from 0 to *N* - 1.

The Hanning window is the same as the cosine window with the parameter *r* set to 0.5.

Hanning window for *N*=128



Errors

W_hanning causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *B* contains fewer elements than *A*.

See Also

Waveform, W_bartlett, W_blackman, W_cosine, W_hamming

W_kaiser

Kaiser-Bessel window.

Loading LOADSUB ALL FROM "BESMC.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_kaiser FROM "MATHLIB.HTS"

Usage

```
REAL A(*),B(*)
REAL P
CALL W_kaiser(A(*),P,B(*))
```

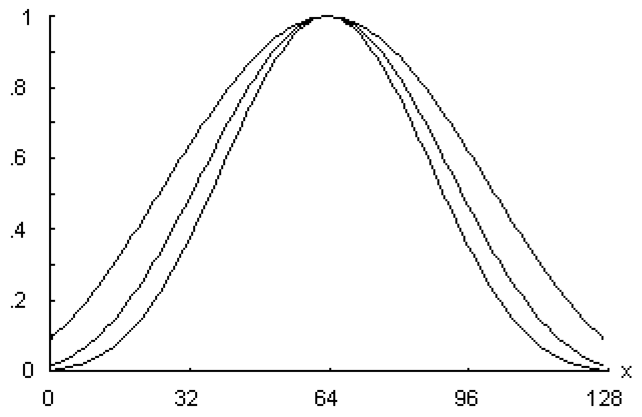
Description

W_kaiser multiplies the sequence in array *A* by a Kaiser-Bessel window function of parameter *p* and returns the product in array *B*. Array *A* is unmodified. *P* controls the width of the central portion of the window; it must be positive, and is usually between 4 and 9. If *N* is the number of elements in the array *A* and *k* is the position in the array *B*, the formula for the window function, *wk*, is

$$w_k = \frac{I_0\left(p\sqrt{\frac{4k}{N-1} - \left(\frac{2k}{N-1}\right)^2}\right)}{I_0(p)}.$$

In the above formula, *k* ranges in value from 0 to *N* - 1. *I0* is the modified cylindrical Bessel function of the first kind (see *I0*).

P is often expressed in Kaiser-Bessel window for *N*=128



terms of a radian frequency parameter, *a*, using the expression

$$p = \omega_a\left(\frac{N-1}{2}\right).$$

Errors

W_kaiser causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, if *p* is negative or zero, or if *B* contains fewer elements than *A*.

Example

The section on the *Fft* subroutine contains an example of windowing using the *W_kaiser* routine.

See Also

I0, W_cosine, W_trapezoid

W_trapezoid

Trapezoid window.

Loading

LOADSUB ALL FROM "W_TRAPEZ.HTS"
or LOADSUB FROM "MATHLIB.HTS"
or LOADSUB W_trapezoid FROM "MATHLIB.HTS"

Usage

```
REAL A(*),B(*)
REAL H,P
CALL W_trapezoid(A(*),H,P,B(*))
```

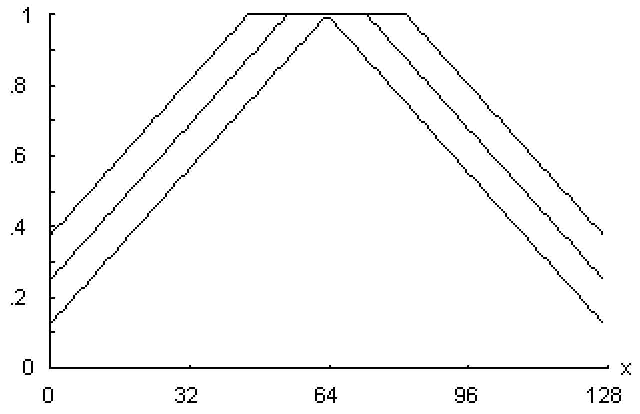
Description

W_trapezoid multiplies the sequence in array *A* by a trapezoid window function of parameters *p* and *h* and returns the product in array *B*. Array *A* is unmodified. *H* is the peak-to-peak amplitude of the untruncated trapezoid portion of the window. *P* is the value of the pedestal portion of the window. *H* and *p* must be between 0 and 1, inclusive.

The window generated by the *W_bartlett* routine is a special case of the trapezoid window, with *p* = 0 and *h* = 1.

Errors

Trapezoid window for *h*=0.875 and *N*=128



W_trapezoid causes a BASIC error if its arguments are not of the types listed in the USAGE section, above, or if *h* and *p* are not in the ranges discussed above.

See Also

Waveform, W_bartlett

Y0

Bessel function of the second kind, order zero.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNY0(X)

Description

FNY0 returns the value of the cylindrical Bessel function of the second kind and order zero of *x*, *Y*0(*x*). *Y*0(*x*) is defined for all positive values of *x*.

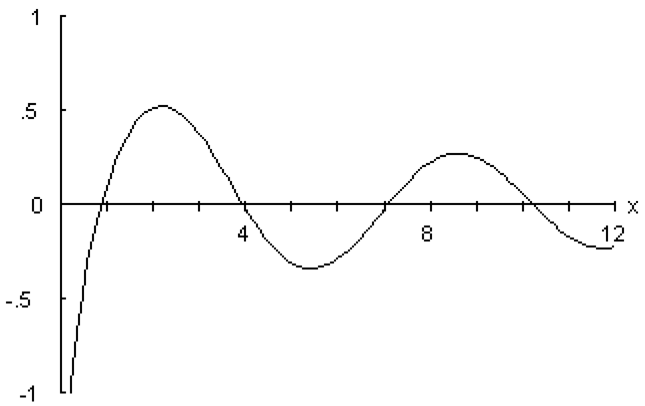
Errors

FNY0 causes a BASIC error if its argument is not of type REAL or if *x* is negative or zero.

See Also

J1, Jn, Y0

*Y*0(*x*)



Y1

Bessel function of the second kind, order one.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage REAL X,Y
Y=FNY1(X)

Description

FNY1 returns the value of the cylindrical Bessel function of the second kind and order one of *x*, *Y*1(*x*). *Y*1(*x*) is defined for all positive values of *x*.

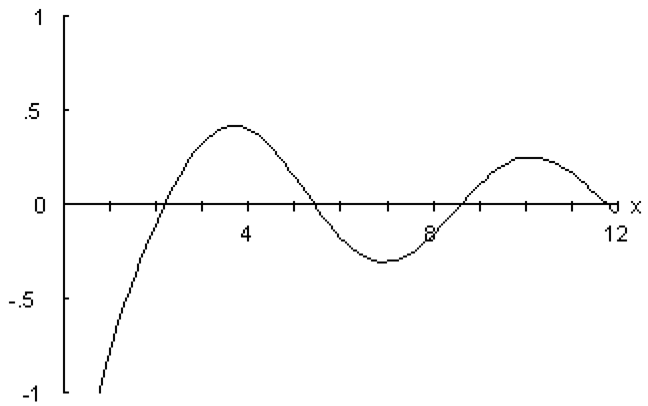
Errors

FNY1 causes a BASIC error if its argument is not of type REAL or if *x* is negative or zero.

See Also

J0, Jn, Y0

*Y*1(*x*)



Yhn

Bessel function of the second kind, order *n*+½.

Loading LOADSUB ALL FROM "BESRS.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNYhn(N,X)
```

Description

FNYhn returns the value of the cylindrical Bessel function of the second kind and order *n*+½ of *x*, *Yn*+½(*x*). *Yn*+½(*x*) is defined for all values of *n* and for all positive values of *x*.

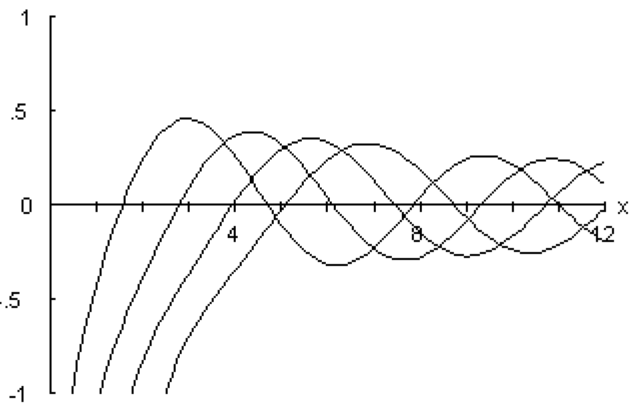
Errors

FNYhn causes a BASIC error if its arguments are not of the types shown in the USAGE section, above, or if *x* is negative or zero.

See Also

Hh1n, Hh2n, Jhn, Yn

Yn+½(*x*)



Note

The algorithm used computes the value of *Yn*+½ using a recursion from the values of *Y*½ and *Y*1½. The computation time increases with *n* and the computation accuracy decreases with *n*.

Yn

Bessel function of the second kind, order *n*.

Loading LOADSUB ALL FROM "BESRC.HTS"
or LOADSUB FROM "MATHLIB.HTS"

Usage

```
INTEGER N
REAL X,Y
Y=FNYn(N,X)
```

Description

FNYn returns the value of the cylindrical Bessel function of the second kind and order *n* of *x*, *Yn*(*x*). *Yn*(*x*) is defined for all positive values of *x*.

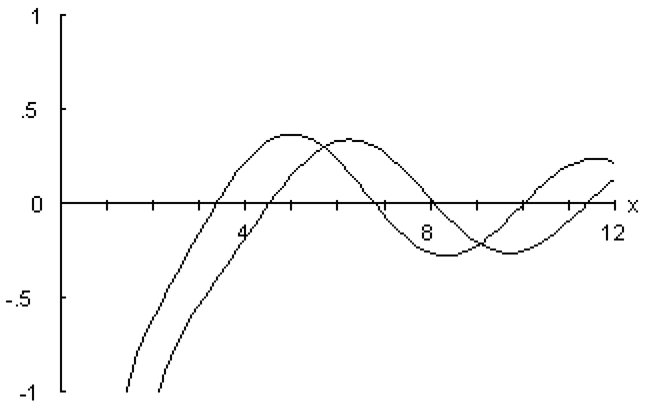
Errors

FNYn causes a BASIC error if its argument is not of type REAL or if *x* is negative or zero.

See Also

Y0, Y1

Yn(*x*)



Note The algorithm used computes the value of *Yn* using a recursion from the values of *Y*0 and *Y*1. The computation time increases with *n* and the computation accuracy decreases with *n* - 1.

{ewl RoboEx32.dll, WinHelp2000, }