TR29124 C++ Special Math Functions

2.0

Generated by Doxygen 1.8.9.1

Thu Mar 31 2016 16:04:02

Contents

Chapter 1

Todo List

globalScope> Member _GLIBCXX_VISIBILITY (default)

this needs some compile-time construction!

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

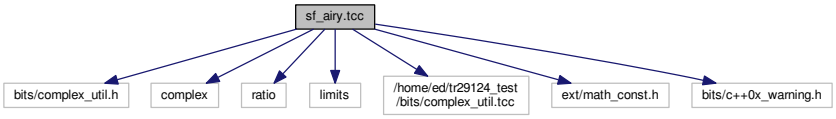
Chapter 3

File Documentation

3.1 sf_airy.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf_airy.tcc:



Macros

- #define _GLIBCXX_BITS_SF_AIRY_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.1.1 Macro Definition Documentation

3.1.1.1 #define _GLIBCXX_BITS_SF_AIRY_TCC 1

3.1.2 Function Documentation

3.1.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This function evaluates Ai(z) and Ai'(z) from their asymptotic expansions for $|\arg(z)| < 2*/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from abs(z).

Note that for the sake of speed and the fact that this function is to be called by another, checks for valid arguments are not made.

See also

Digital Library of Mathematical Finctions Sec. 9.7 Asymptotic Expansions http://dlmf.nist.↩gov/9.7

Parameters

in	*z*	Complex input variable set equal to the value at which Ai(z) and Bi(z) and their derivative are evaluated. This function assumes abs(z) $>$ 15 and $\|$arg(z)$\|$ $<$ 2/3.
in,out	*Ai*	The value computed for Ai(z).
in,out	*Aip*	The value computed for Ai'(z).
in	*sign*	The sign of the series terms amd exponent. The default (-1) gives the Airy Ai functions for $\|$arg(z)$\|$ $<$. The value +1 gives the Airy Bi functions for $\|$arg(z)$\|$ $<$ /3.

This function evaluates Ai(z) and Ai'(z) from their asymptotic expansions for abs(arg(-z)) $<$ pi/3. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from abs(z).

Note that for the sake of speed and the fact that this function is to be called by another, checks for valid arguments are not made. Hence, an error return is not needed.

Parameters

in	*z*	The value at which the Airy function and its derivative are evaluated. This function assumes abs(z) $>$ 15 and abs(arg(-z)) $<$ pi/3.
out	*Ai*	The computed value of the Airy function Ai(z).
out	*Aip*	The computed value of the Airy function derivative Ai'(z).

Compute the modified Bessel functions of the first kind of orders +-1/3 and +-2/3 needed to compute the Airy functions and their derivatives from their representation in terms of the modified Bessel functions. This function is only used for z less than two in modulus and in the closed right half plane. This stems from the fact that the values of the modified Bessel functions occuring in the representations of the Airy functions and their derivatives are almost equal for z large in the right half plane. This means that loss of significance occurs if these representations are used for z to large in magnitude. This algorithm is also not used for z too small, since a low order rational approximation can be used instead.

This routine is an implementation of a modified version of Miller's backward recurrence algorithm for computation by from the recurrence relation

$$I_{\nu-1} = (2\nu/z)I_\nu + I_{\nu+1}$$

satisfied by the modified Bessel functions of the first kind. the normalization relationship used is

$$\frac{z/2)^\nu e^z}{\Gamma(\nu+1)} = I_\nu(z) + 2\sum_{k=1}^{\infty} \frac{(k+\nu)\Gamma(2\nu+k)}{k!\Gamma(1+2\nu)} I_{\nu+k}(z).$$

This modification of the algorithm is given in part in

Olver, F. W. J. and Sookne, D. J., Note on Backward Recurrence Algorithms, Math. of Comp., Vol. 26, no. 120, Oct. 1972.

And further elaborated for the Bessel functions in

Sookne, D. J., Bessel Functions I and J of Complex Argument and Integer Order, J. Res. NBS - Series B, Vol 77B, Nos. 3 & 4, July-December, 1973.

Insight was also gained from

Cody, W. J., Preliminary Report on Software for the Modified Bessel Functions of the First Kind, Argonne National Laboratory, Applied Mathematics Division, Tech. Memo. No. 357, August, 1980.

Cody implements the algorithm of Sookne for fractional order and nonnegative real argument. Like Cody, we do not change the convergence testing mechanism in any substantial way. However, we do trim the overhead by making the additional assumption that performing the convergence test for the functions of order 2/3 will suffice for order 1/3 as well. This assumption has not been established by rigourous analysis at this time. For the sake of speed the convergence tests are performed in the 1-norm instead of the usual Euclidean norm used in the complex plane using the inequality

$$|x| + |y| <= \sqrt(2)\sqrt(x^2 + y^2)$$

Parameters

in	*z*	The argument of the modified Bessel functions.
in	*eps*	The maximum relative error required in the results.
out	*ip1d3*	The value of I_(+1/3)(z).
out	*im1d3*	The value of I_(-1/3)(z).
out	*ip2d3*	The value of I_(+2/3)(z).
out	*im2d3*	The value of I_(-2/3)(z).

Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.

This routine computes

$$E_\nu(z) = \exp z\sqrt{2z/\pi}K_\nu(z), for \nu = 1/3 and \nu = 2/3$$

using a rational approximation given in

Luke, Y. L., Mathematical functions and their approximations, Academic Press, pp 366-367, 1975.

Though the approximation converges in abs(arg(z)) <= pi, The convergence weakens as abs(arg(z)) increases. Also, in the case of real order between 0 and 1, convergence weakens as the order approaches 1. For these reasons, we only use this code for |

- (z)| <= 3pi/4 and the convergence test is performed only for order 2/3.

The coding of this function is also influenced by the fact that it will only be used for about 2 <= |z| <= 15. Hence, certain considerations of overflow, underflow, and loss of significance are unimportant for our purpose.

Parameters

in	*z*	The value for which the quantity E_nu is to be computed. it is recommended that abs(z) not be too small and that | • (z)| <= 3pi/4.
in	*eps*	The maximum relative error allowable in the computed results. The relative error test is based on the comparison of successive iterates.
out	*kp1d3*	The value computed for E_{+1/3}(z).
out	*kp2d3*	The value computed for E_{+2/3}(z).

Note

In the worst case, say, z=2 and

- (z) = 3pi/4, 20 iterations should give 7 or 8 decimals of accuracy.

This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders = +-1/3 and +- 2/3. That is, A(z)/B(z), Where A(z) and B(z) are cubic polynomials with real coefficients, approximates

$$\frac{\Gamma(\nu+1)}{(z/2)^n u} I_\nu(z) =_0 F_1(;\nu+1;z^2/4),$$

where the function on the right is a confluent hypergeometric limit function. For |z| <= 1/4 and |arg(z)| <= pi/2, the approximations are accurate to about 16 decimals.

For further details including the four term recurrence relation satisfied by the numerator and denominator polynomials in the higher order approximants, see

Luke, Y.L., Mathematical Functions and their Approximations, Academic Press, pp 361-363, 1975.

An asymptotic expression for the error is given as well as other useful expressions in the event one wants to extend this function to incorporate higher order approximants.

Note also that for the sake of speed and the fact that this function will be driven by another, checks that are not absolutely necessary are not made.

Parameters

in	*z*	The argument at which the hypergeometric given above is to be evaluated. Since the approximation is of fixed order, abs(z) must be small to ensure sufficient accuracy of the computed results.
out	*Fp1d3*	The approximate value of the confluent hypergeometric limit function related to the modified Bessel function of order +1/3.
out	*Fm1d3*	The approximate value of the confluent hypergeometric limit function related to the modified Bessel function of order -1/3.
out	*Fp2d3*	The approximate value of the confluent hypergeometric limit function related to the modified Bessel function of order +2/3.
out	*Fm2d3*	The approximate value of the confluent hypergeometric limit function related to the modified Bessel function of order -2/3.

This function computes the Airy function Ai(z) and its first derivative in the complex z-plane.

The algorithm used exploits numerous representations of the Airy function and its derivative. The representations are recorded here for reference:

$$(1a)Ai(z) = \frac{\sqrt{z}}{3}(I_{-1/3}(\zeta) - I_{1/3}(\zeta))$$

$$(1b)Bi(z) = \sqrt{\frac{z}{3}}(I_{-1/3}(\zeta) + I_{1/3}(\zeta))$$

$$(2)Ai(z) = \frac{\sqrt{z/3}}{\pi}K_{1/3}(\zeta) = \frac{2^{2/3}3^{-5/6}}{\sqrt{(}\pi)}z\exp(-\zeta)U(5/6;5/3;2\zeta)$$

$$(3a)Ai(-z) = \frac{\sqrt{z}}{3}(J_{-1/3}(\zeta) + J_{1/3}(\zeta))$$

$$(3b)Bi(-z) = \sqrt{\frac{z}{3}}(J_{-1/3}(\zeta) - J_{1/3}(\zeta))$$

$$(4a)Ai'(z) = \frac{z}{3}(I_{2/3}(\zeta) - I_{-2/3}(\zeta))$$

$$(4b)Bi'(z) = \frac{z}{\sqrt{3}}(I_{-2/3}(\zeta) + I_{2/3}(\zeta))$$

$$(5a)Ai'(z) = -\frac{z}{\pi\sqrt{(}3)}K_{(}2/3)(zeta) = -\frac{4^{2/3}3^{-7/6}}{\sqrt{(}\pi)}z^2\exp(-\zeta)U(7/6;7/3;2\zeta)$$

$$(6a)Ai'(-z) = \frac{z}{3}(J_{2/3}(\zeta) - J_{-2/3}(\zeta)),$$

$$(6b)Bi'(-z) = \frac{z}{\sqrt{3}}(J_{-2/3}(\zeta) + J_{2/3}(\zeta)),$$

Where = - {2}{3}z^{3/2} and U(a;b;z) is the confluent hypergeometric function defined in

See also

Stegun, I. A. and Abramowitz, M., Handbook of Mathematical Functions, Natl. Bureau of Standards, AMS 55, pp 504-515, 1964.

The asymptotic expansions derivable from these representations and Hankel's asymptotic expansions for the Bessel functions are used for large modulus of z. The implementation has taken advantage of the error bounds given in

See also

Olver, F. W. J., Error Bounds for Asymptotic Expansions, with an Application to Cylinder Functions of Large Argument, in Asymptotic Solutions of Differential Equations (Wilcox, Ed.), Wiley and Sons, pp 163-183, 1964

and

See also

Olver, F. W. J., Asymptotics and Special Functions, Academic Press, pp 266-268, 1974.

For small modulus of z, a rational approximation is used. This approximant is derived from

Luke, Y. L., Mathematical Functions and their Approximations, Academic Press, pp 361-363, 1975.

The identities given below are for Bessel functions of the first kind in terms of modified Bessel functions of the first kind are also used with the rational approximant.

For moderate modulus of z, three techniques are used. Two use a backward recursion algorithm with (1), (3), (4), and (6). The third uses the confluent hypergeometric representations given by (2) and (5). The backward recursion algorithm generates values of the modified Bessel functions of the first kind of orders + or - 1/3 and + or - 2/3 for z in the right half plane. Values for the corresponding Bessel functions of the first kind are recovered via the identities

$$J_\nu(z) = exp(\nu\pi i/2) I_\nu(z exp(-\pi i/2)), 0 <= arg(z) <= \pi/2$$

and

$$J_\nu(z) = exp(-\nu\pi i/2) I_\nu(z exp(\pi i/2)), -\pi/2 <= arg(z) < 0.$$

The particular backward recursion algorithm used is discussed in

See also

Olver, F. W. J, Numerical solution of second-order linear difference equations, NBS J. Res., Series B, VOL 71B, pp 111-129, 1967.
Olver, F. W. J. and Sookne, D. J., Note on backward recurrence algorithms, Math. Comp. Vol 26, No. 120, pp 941-947, Oct. 1972
Sookne, D. J., Bessel Functions I and J of Complex Argument and Integer Order, NBS J. Res., Series B, Vol 77B, Nos 3& 4, pp 111-113, July-December, 1973.

The following paper was also useful

See also

Cody, W. J., Preliminary report on software for the modified Bessel functions of the first kind, Applied Mathematics Division, Argonne National Laboratory, Tech. Memo. no. 357.

A backward recursion algorithm is also used to compute the confluent hypergeometric function. The recursion relations and a convergence theorem are given in

See also

Wimp, J., On the computation of Tricomi's psi function, Computing, Vol 13, pp 195-203, 1974.

Parameters

in	*z*	The argument at which the Airy function and its derivative are computed.
in	*eps*	Relative error required. Currently, eps is used only in the backward recursion algorithms.

out	*Ai*	The value computed for Ai(z).
out	*Aip*	The value computed for Ai'(z).

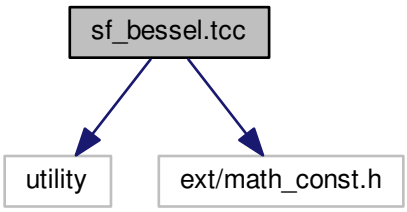
Return the complex Airy Ai function.

Return the complex Airy Bi function.

3.2 sf_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
```

Include dependency graph for sf_bessel.tcc:



Macros

- #define _GLIBCXX_BITS_SF_BESSEL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.2.1 Macro Definition Documentation

3.2.1.1 #define _GLIBCXX_BITS_SF_BESSEL_TCC 1

3.2.2 Function Documentation

3.2.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This routine computes the asymptotic cylindrical Bessel and Neumann functions of order nu: $J_\nu(x)$, $N_\nu(x)$. Use this for $x >> nu^2 + 1$.

References: (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Section 9 p. 364, Equations 9.2.5-9.2.10

Parameters

__nu	The order of the Bessel functions.
__x	The argument of the Bessel functions.
_Jnu	The output Bessel function of the first kind.
_Nnu	The output Neumann function (Bessel function of the second kind).

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu}[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)}]$$

and

$$\Gamma_2 = \frac{1}{2}[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)}]$$

where $-1/2 <= \mu <= 1/2$ is $\mu = \nu - N$ and N. is the nearest integer to ν. The values of $\Gamma(1+\mu)$ and $\Gamma(1-\mu)$ are returned as well.

The accuracy requirements on this are exquisite.

Parameters

__mu	The input parameter of the gamma functions.
__gam1	The output function $\Gamma_1(\mu)$
__gam2	The output function $\Gamma_2(\mu)$
__gampl	The output function $\Gamma(1+\mu)$
__gammi	The output function $\Gamma(1-\mu)$

Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

__nu	The order of the Bessel functions.
__x	The argument of the Bessel functions.
_Jnu	The output Bessel function of the first kind.
_Nnu	The output Neumann function (Bessel function of the second kind).
_Jpnu	The output derivative of the Bessel function of the first kind.
_Npnu	The output derivative of the Neumann function.

This routine returns the cylindrical Bessel functions of order ν: J_ν or I_ν by series expansion.

The modified cylindrical Bessel function is:

$$Z_\nu(x) = \sum_{k=0}^{\infty} \frac{\sigma^k (x/2)^{\nu+2k}}{k!\Gamma(\nu+k+1)}$$

where $\sigma = +1$ or -1 for $Z = I$ or J respectively.

See Abramowitz & Stegun, 9.1.10 Abramowitz & Stegun, 9.6.7 (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Equation 9.1.10 p. 360 and Equation 9.6.10 p. 375

Parameters

__nu	The order of the Bessel function.
__x	The argument of the Bessel function.
__sgn	The sign of the alternate terms -1 for the Bessel function of the first kind. +1 for the modified Bessel function of the first kind.

Returns

The output Bessel function.

Return the cylindrical Bessel functions and their derivatives of order ν by various means.

Return the Bessel function of order ν: $J_\nu(x)$.

The cylindrical Bessel function is:

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k!\Gamma(\nu+k+1)}$$

Parameters

__nu	The order of the Bessel function.
__x	The argument of the Bessel function.

Returns

The output Bessel function.

Return the Neumann function of order ν: $N_\nu(x)$.

The Neumann function is defined by:

$$N_\nu(x) = \frac{J_\nu(x)\cos\nu\pi - J_{-\nu}(x)}{\sin\nu\pi}$$

where for integral $\nu = n$ a limit is taken: $lim_{\nu\to n}$.

Parameters

__nu	The order of the Neumann function.
__x	The argument of the Neumann function.

Returns

The output Neumann function.

Return the cylindrical Hankel function of the first kind $h_n^{(1)}(x)$.

The cylindrical Hankel function of the first kind is defined by:

$$H_n^{(1)}(x) = J_n(x) + iN_n(x)$$

Parameters

__n	The order of the spherical Neumann function.
__x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Return the cylindrical Hankel function of the second kind $h_n^{(1)}(x)$.

The cylindrical Hankel function of the second kind is defined by:

$$H_n^{(2)}(x) = J_n(x) - iN_n(x)$$

Parameters

__n	The order of the spherical Neumann function.
__x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j_n'(x)$ and $n_n'(x)$ respectively.

Parameters

__n	The order of the spherical Bessel function.
__x	The argument of the spherical Bessel function.
__j_n	The output spherical Bessel function.
__n_n	The output spherical Neumann function.
__jp_n	The output derivative of the spherical Bessel function.
__np_n	The output derivative of the spherical Neumann function.

Return the spherical Bessel function $j_n(x)$ of order n.

The spherical Bessel function is defined by:

$$j_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} J_{n+1/2}(x)$$

Parameters

__n	The order of the spherical Bessel function.
__x	The argument of the spherical Bessel function.

Returns

The output spherical Bessel function.

Return the spherical Neumann function $n_n(x)$.

The spherical Neumann function is defined by:

$$n_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} N_{n+1/2}(x)$$

Parameters

__n	The order of the spherical Neumann function.
__x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = j_n(x) + in_n(x)$$

Parameters

__n	The order of the spherical Neumann function.
__x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Return the spherical Hankel function of the second kind $h_n^{(1)}(x)$.

The spherical Hankel function of the second kind is defined by:

$$h_n^{(2)}(x) = j_n(x) - in_n(x)$$

Parameters

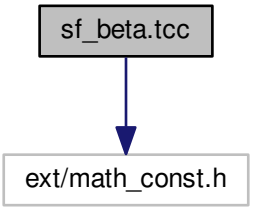
__n	The order of the spherical Neumann function.
__x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

3.3 sf_beta.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_beta.tcc:



Macros

- #define _GLIBCXX_BITS_SF_BETA_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.3.1 Macro Definition Documentation

3.3.1.1 #define _GLIBCXX_BITS_SF_BETA_TCC 1

3.3.2 Function Documentation

3.3.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Return the beta function: $B(x, y)$.

The beta function is defined by

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x + y)}$$

Parameters

__x	The first argument of the beta function.
__y	The second argument of the beta function.

Returns

The beta function.

Return the beta function $B(x, y)$ using the log gamma functions.

The beta function is defined by

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x + y)}$$

Parameters

__x	The first argument of the beta function.
__y	The second argument of the beta function.

Returns

The beta function.

Return the beta function $B(x, y)$ using the product form.

The beta function is defined by

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x + y)}$$

Parameters

__x	The first argument of the beta function.
__y	The second argument of the beta function.

Returns

The beta function.

Return the beta function $B(x, y)$.

The beta function is defined by

$$B(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x + y)}$$

Parameters

__x	The first argument of the beta function.
__y	The second argument of the beta function.

Returns

The beta function.

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}(\frac{\nu}{2}, \frac{1}{2})A(t|\nu) =$$

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}(\frac{\nu}{2}, \frac{1}{2}) = 1 - A(t|\nu)$$

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value $f[^2 $f].

The f-distribution propability function is related to the incomplete beta function:

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2+\nu_1 F}}(\frac{\nu_2}{2}, \frac{\nu_1}{2})$$

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value $f[^2 $f].

The f-distribution propability function is related to the incomplete beta function:

$$P(F|\nu_1, \nu_2) = 1 - I_{\frac{\nu_2}{\nu_2+\nu_1 F}}(\frac{\nu_2}{2}, \frac{\nu_1}{2}) = 1 - Q(F|\nu_1, \nu_2)$$

Return the binomial cumulative distribution function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$P(p|n, k) = I_p(k, n - k + 1)$$

Return the complementary binomial cumulative distribution function.

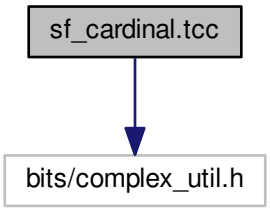
The binomial cumulative distribution function is related to the incomplete beta function:

$$Q(p|n, k) = I_{1-p}(n - k + 1, k)$$

3.4 sf_cardinal.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf_cardinal.tcc:



Macros

- #define _GLIBCXX_BITS_SF_CARDINAL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.4.1 Macro Definition Documentation

3.4.1.1 #define _GLIBCXX_BITS_SF_CARDINAL_TCC 1

3.4.2 Function Documentation

3.4.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Return the generalized sinus cardinal function

$$sinc_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

Return the unnormalized sinus cardinal function

$$sinc_\pi(x) = \frac{\sin(x)}{x}$$

Return the normalized sinus cardinal function

$$sinc(x) = \frac{\sin(\pi x)}{\pi x}$$

Return the generalized hyperbolic sinus cardinal function

$$sinhc_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

Return the unnormalized hyperbolic sinus cardinal function

$$sinhc_\pi(x) = \frac{\sinh(x)}{x}$$

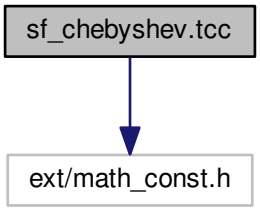
Return the normalized hyperbolic sinus cardinal function

$$sinhc(x) = \frac{\sinh(\pi x)}{\pi x}$$

3.5 sf_chebyshev.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_chebyshev.tcc:



Macros

- #define _GLIBCXX_SF_CHEBYSHEV_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.5.1 Macro Definition Documentation

3.5.1.1 #define _GLIBCXX_SF_CHEBYSHEV_TCC 1

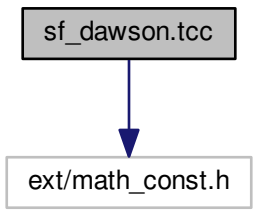
3.5.2 Function Documentation

3.5.2.1 namespace std _GLIBCXX_VISIBILITY (default)

3.6 sf_dawson.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_dawson.tcc:



Macros

- #define _GLIBCXX_SF_DAWSON_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.6.1 Macro Definition Documentation

3.6.1.1 #define _GLIBCXX_SF_DAWSON_TCC 1

3.6.2 Function Documentation

3.6.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Compute the Dawson integral using the series expansion.

Compute the Dawson integral using a sampling theorem representation.

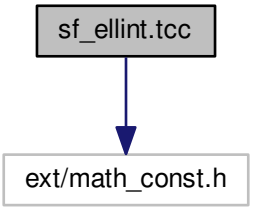
Todo this needs some compile-time construction!

Compute the Dawson integral.

3.7 sf_ellint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_ellint.tcc:



Macros

- #define _GLIBCXX_BITS_SF_ELLINT_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.7.1 Macro Definition Documentation

3.7.1.1 #define _GLIBCXX_BITS_SF_ELLINT_TCC 1

3.7.2 Function Documentation

3.7.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.

The Carlson elliptic function is defined by:

$$R_C(x, y) = \frac{1}{2}\int_0^\infty \frac{dt}{(t + x)^{1/2}(t + y)}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

__x	The first argument.
__y	The second argument.

Returns

The Carlson elliptic function.

Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.

The Carlson elliptic function of the second kind is defined by:

$$R_D(x, y, z) = \frac{3}{2}\int_0^\infty \frac{dt}{(t + x)^{1/2}(t + y)^{1/2}(t + z)^{3/2}}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

__x	The first of two symmetric arguments.
__y	The second of two symmetric arguments.
__z	The third argument.

Returns

The Carlson elliptic function of the second kind.

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.

The Carlson elliptic function of the first kind is defined by:

$$R_F(x, y, z) = \frac{1}{2}\int_0^\infty \frac{dt}{(t + x)^{1/2}(t + y)^{1/2}(t + z)^{1/2}}$$

Parameters

__x	The first of three symmetric arguments.
__y	The second of three symmetric arguments.
__z	The third of three symmetric arguments.

Returns

The Carlson elliptic function of the first kind.

Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.

The Carlson symmetric elliptic function of the second kind is defined by:

$$R_G(x, y, z) = \frac{1}{4}\int_0^\infty dtt[(t + x)(t + y)(t + z)]^{-1/2}(\frac{x}{t + x} + \frac{y}{t + y} + \frac{z}{t + z})$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

__x	The first of three symmetric arguments.
__y	The second of three symmetric arguments.
__z	The third of three symmetric arguments.

Returns

The Carlson symmetric elliptic function of the second kind.

Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.

The Carlson elliptic function of the third kind is defined by:

$$R_J(x, y, z, p) = \frac{3}{2}\int_0^\infty \frac{dt}{(t + x)^{1/2}(t + y)^{1/2}(t + z)^{1/2}(t + p)}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

__x	The first of three symmetric arguments.
__y	The second of three symmetric arguments.
__z	The third of three symmetric arguments.
__p	The fourth argument.

Returns

The Carlson elliptic function of the fourth kind.

Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.

The complete elliptic integral of the first kind is defined as

$$K(k) = F(k, \pi/2) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - k^2 sin^2\theta}}$$

where $F(k, \phi)$ is the incomplete elliptic integral of the first kind.

Parameters

__k	The argument of the complete elliptic function.

Returns

The complete elliptic function of the first kind.

Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the first kind is defined as

$$F(k, \phi) = \int_0^\phi \frac{d\theta}{\sqrt{1 - k^2 sin^2\theta}}$$

Parameters

__k	The argument of the elliptic function.
__phi	The integral limit argument of the elliptic function.

Returns

The elliptic function of the first kind.

Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.

The complete elliptic integral of the second kind is defined as

$$E(k, \pi/2) = \int_0^{\pi/2} \sqrt{1 - k^2 sin^2\theta}$$

Parameters

__k	The argument of the complete elliptic function.

Returns

The complete elliptic function of the second kind.

Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the second kind is defined as

$$E(k, \phi) = \int_0^\phi \sqrt{1 - k^2 sin^2\theta}$$

Parameters

__k	The argument of the elliptic function.
__phi	The integral limit argument of the elliptic function.

Returns

The elliptic function of the second kind.

Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.

The complete elliptic integral of the third kind is defined as

$$\Pi(k, \nu) = \int_0^{\pi/2} \frac{d\theta}{(1 - \nu \sin^2\theta)\sqrt{1 - k^2 \sin^2\theta}}$$

Parameters

__k	The argument of the elliptic function.
__nu	The second argument of the elliptic function.

Returns

The complete elliptic function of the third kind.

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the third kind is defined as

$$\Pi(k, \nu, \phi) = \int_0^\phi \frac{d\theta}{(1 - \nu \sin^2\theta)\sqrt{1 - k^2 \sin^2\theta}}$$

Parameters

__k	The argument of the elliptic function.
__nu	The second argument of the elliptic function.
__phi	The integral limit argument of the elliptic function.

Returns

The elliptic function of the third kind.

Return the Legendre elliptic integral D.

Return the complete Legendre elliptic integral D.

Return the Bulirsch elliptic integrals of the first kind.

Return the Bulirsch elliptic integrals of the second kind.

Return the Bulirsch elliptic integrals of the third kind.

Return the Bulirsch complete elliptic integrals.

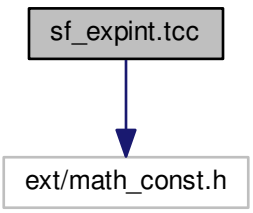
Return the Heuman lambda function.

Return the Jacobi zeta function.

3.8 sf_expint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_expint.tcc:



Macros

- #define _GLIBCXX_BITS_SF_EXPINT_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.8.1 Macro Definition Documentation

3.8.1.1 #define _GLIBCXX_BITS_SF_EXPINT_TCC 1

3.8.2 Function Documentation

3.8.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

The exponential integral is given by

$$E_1(x) = \int_1^\infty \frac{e^{-xt}}{t} dt$$

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_1(x)$ by asymptotic expansion.

The exponential integral is given by

$$E_1(x) = \int_1^\infty \frac{e^{-xt}}{t} dt$$

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$ by series summation.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$ by continued fractions.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $Ei(x)$ by series summation.

The exponential integral is given by

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

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $Ei(x)$ by asymptotic expansion.

The exponential integral is given by

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

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $Ei(x)$.

The exponential integral is given by

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

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_1(x)$.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$ for large argument.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$ for large order.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $E_n(x)$.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

Parameters

__n	The order of the exponential integral function.
__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the exponential integral $Ei(x)$.

The exponential integral is given by

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

Parameters

__x	The argument of the exponential integral function.

Returns

The exponential integral.

Return the logarithmic integral $li(x)$.

The logarithmic integral is given by

$$li(x) = Ei(\log(x))$$

Parameters

__x	The argument of the logarithmic integral function.

Returns

The logarithmic integral.

Return the hyperbolic cosine integral $li(x)$.

The hyperbolic cosine integral is given by

$$Chi(x) = (Ei(x) - E_1(x))/2$$

Parameters

__x	The argument of the hyperbolic cosine integral function.

Returns

The hyperbolic cosine integral.

Return the hyperbolic sine integral $li(x)$.

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) - E_1(x))/2$$

Parameters

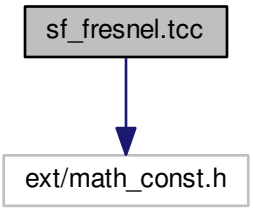
__x	The argument of the hyperbolic sine integral function.

Returns

The hyperbolic sine integral.

3.9 sf_fresnel.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_fresnel.tcc:



Macros

- #define _GLIBCXX_SF_FRESNEL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.9.1 Macro Definition Documentation

3.9.1.1 #define _GLIBCXX_SF_FRESNEL_TCC 1

3.9.2 Function Documentation

3.9.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

This function returns the Fresnel cosine and sine integrals as a complex number $f[C(x) + iS(x) $f].

The Fresnel cosine integral is defined by:

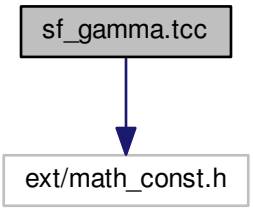
$$C(x) = \int_0^x \cos(\frac{\pi}{2}t^2)dt$$

The Fresnel sine integral is defined by:

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

3.10 sf_gamma.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_gamma.tcc:



Macros

- #define _GLIBCXX_BITS_SF_GAMMA_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.10.1 Macro Definition Documentation

3.10.1.1 #define _GLIBCXX_BITS_SF_GAMMA_TCC 1

3.10.2 Function Documentation

3.10.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This returns Bernoulli numbers from a table or by summation for larger values.

Upward recursion is unstable.

Parameters

__n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

This returns Bernoulli number B_n.

Parameters

__n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

This returns Bernoulli number B_n.

Parameters

__n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

Return $log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

Parameters

__x	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

See also

Spouge, J.L., Computation of the gamma, digamma, and trigamma functions. SIAM Journal on Numerical Analysis 31, 3 (1994), pp. 931-944

Parameters

__z	The argument of the gamma function.

Returns

The the gamma function.

Return $log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

Parameters

__x	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Return $log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use *__log_gamma_sign*.

Parameters

__x	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

Parameters

__x	The argument of the gamma function.

Returns

The sign of the gamma function.

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

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

Parameters

__n	The first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Return the binomial coefficient. The binomial coefficient is given by:

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

Parameters

__n	The first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Return $\Gamma(x)$.

Parameters

__x	The argument of the gamma function.

Returns

The gamma function.

Return the regularized lower incomplete gamma function. The regularized lower incomplete gamma function is defined by

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

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

is the lower incomplete gamma function.

Return the regularized upper incomplete gamma function. The regularized upper incomplete gamma function is defined by

$$Q(a,x) = \frac{\Gamma(a,x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\Gamma(a,x) = \int_x^\infty e^{-t}t^{a-1}dt(a>0)$$

is the upper incomplete gamma function.

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

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

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a,x) = \int_x^\infty e^{-t}t^{a-1}dt(a>0)$$

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1}(a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Thus this function returns

$$ln[(a)_n] = \Gamma(a+n) - \Gamma(n), ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{array}{c} a \\ n \end{array} \right]$$

, and others.

Return the (upper) Pochhammer function or the rising factorial function. The Pochammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1}(a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{array}{c} a \\ n \end{array} \right]$$

, and others.

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1}(a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $f[(n)_n = n! $f]. Thus this function returns

$$ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), ln[(a)_0] = 0$$

Many notations exist:

$$a^{\underline{n}}$$

,

$$\left\{ \begin{array}{c} a \\ n \end{array} \right\}$$

, and others.

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1}(a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $f[(n)_n = n! $f].

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

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

The series is given by:

$$\psi(x) = -\gamma_E - \frac{1}{x}\sum_{k=1}^{\infty}\frac{x-1}{(k+1)(x+k)}$$

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

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

The asymptotic series is given by:

$$\psi(x) = \ln(x) - \frac{1}{2x} - \sum_{n=1}^{\infty}\frac{B_{2n}}{2nx^{2n}}$$

Return the digamma function. The digamma or $\psi(x)$ function is defined by

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

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1 - x) - \pi \cot(\pi x)$$

Return the polygamma function $\psi^{(n)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(n)}(x) = (-1)^{n+1} m! \zeta(m + 1, x)$$

Return the factorial of the integer n.

The factorial is:

$$n! = 12...(n - 1)n, 0! = 1$$

Return the logarithm of the factorial of the integer n.

The factorial is:

$$n! = 12...(n - 1)n, 0! = 1$$

Return the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135...(n - 2)n, noddn!! = 246...(n - 2)n, neven - 1!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m - 1)!! = 1/(1(-1)(-3)...(-2m + 1)(-2m - 1)) = \frac{(-1)^m}{(2m - 1)!!}$$

for $f[n = -2m - 1 $f].

Return the logarithm of the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135...(n - 2)n, noddn!! = 246...(n - 2)n, neven - 1!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m - 1)!! = 1/(1(-1)(-3)...(-2m + 1)(-2m - 1)) = \frac{(-1)^m}{(2m - 1)!!}$$

for $f[n = -2m - 1 $f].

Return the chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is less than the value χ^2.

The chi-squared propability function is related to the normalized lower incomplete gamma function:

$$P(\chi^2|\nu) = \Gamma_P(\frac{\nu}{2}, \frac{\chi^2}{2})$$

Return the complementary chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is greater than the value χ^2.

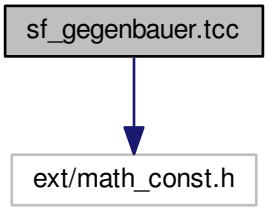
The complementary chi-squared propability function is related to the normalized upper incomplete gamma function:

$$Q(\chi^2|\nu) = \Gamma_Q(\frac{\nu}{2}, \frac{\chi^2}{2})$$

3.11 sf_gegenbauer.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_gegenbauer.tcc:



Macros

- #define _GLIBCXX_SF_GEGENBAUER_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.11.1 Macro Definition Documentation

3.11.1.1 #define _GLIBCXX_SF_GEGENBAUER_TCC 1

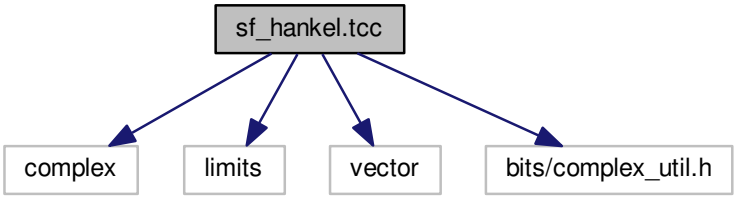
3.11.2 Function Documentation

3.11.2.1 namespace std _GLIBCXX_VISIBILITY (default)

3.12 sf_hankel.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include <bits/complex_util.h>
```

Include dependency graph for sf_hankel.tcc:



Macros

- #define _GLIBCXX_BITS_SF_HANKEL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.12.1 Macro Definition Documentation

3.12.1.1 #define _GLIBCXX_BITS_SF_HANKEL_TCC 1

3.12.2 Function Documentation

3.12.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of safe_div.

Parameters

`in`	*num2d3*	nu^(-2/3) - output from hankel_params.
`in`	*zeta*	zeta in the uniform asymptotic expansions - output from hankel_params.
`out`	*argp*	exp(+2∗pi∗i/3) ∗ nu^(2/3) ∗ zeta.
`out`	*argm*	exp(-2∗pi∗i/3) ∗ nu^(2/3) ∗ zeta.

Exceptions

std::runtime_error.	

Compute outer factors and associated functions of `z` and `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error `eps`.

Parameters

out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

Parameters

in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

Parameters

in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.
in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.
in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

Return the complex cylindrical Hankel function of the first kind.

Parameters

in	*nu*	The order for which the cylindrical Hankel function of the first kind is evaluated.
in	*z*	The argument at which the cylindrical Hankel function of the first kind is evaluated.

Returns

The complex cylindrical Hankel function of the first kind.

Return the complex cylindrical Hankel function of the second kind.

Parameters

in	*nu*	The order for which the cylindrical Hankel function of the second kind is evaluated.
in	*z*	The argument at which the cylindrical Hankel function of the second kind is evaluated.

Returns

The complex cylindrical Hankel function of the second kind.

Return the complex cylindrical Bessel function.

Parameters

in	*nu*	The order for which the cylindrical Bessel function is evaluated.
in	*z*	The argument at which the cylindrical Bessel function is evaluated.

Returns

The complex cylindrical Bessel function.

Return the complex cylindrical Neumann function.

Parameters

in	*nu*	The order for which the cylindrical Neumann function is evaluated.
in	*z*	The argument at which the cylindrical Neumann function is evaluated.

Returns

The complex cylindrical Neumann function.

Helper to compute complex spherical Hankel functions and their derivatives.

Parameters

in	*n*	The order for which the spherical Hankel functions are evaluated.
in	*z*	The argument at which the spherical Hankel functions are evaluated.
out	*H1*	The spherical Hankel function of the first kind.
out	*H1p*	The derivative of the spherical Hankel function of the first kind.
out	*H2*	The spherical Hankel function of the second kind.
out	*H2p*	The derivative of the spherical Hankel function of the second kind.

Return the complex spherical Hankel function of the first kind.

Parameters

in	*n*	The order for which the spherical Hankel function of the first kind is evaluated.
in	*z*	The argument at which the spherical Hankel function of the first kind is evaluated.

Returns

The complex spherical Hankel function of the first kind.

Return the complex spherical Hankel function of the second kind.

Parameters

in	*n*	The order for which the spherical Hankel function of the second kind is evaluated.
in	*z*	The argument at which the spherical Hankel function of the second kind is evaluated.

Returns

The complex spherical Hankel function of the second kind.

Return the complex spherical Bessel function.

Parameters

in	*n*	The order for which the spherical Bessel function is evaluated.
in	*z*	The argument at which the spherical Bessel function is evaluated.

Returns

The complex spherical Bessel function.

Return the complex spherical Neumann function.

Parameters

in	*n*	The order for which the spherical Neumann function is evaluated.
in	*z*	The argument at which the spherical Neumann function is evaluated.

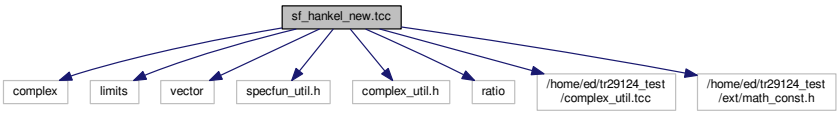
Returns

The complex spherical Neumann function.

3.13 sf_hankel_new.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include "specfun_util.h"
#include "complex_util.h"
```

Include dependency graph for sf_hankel_new.tcc:



Macros

- #define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.13.1 Macro Definition Documentation

3.13.1.1 #define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1

3.13.2 Function Documentation

3.13.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of safe_div.

Parameters

in	*num2d3*	nu^(-2/3) - output from hankel_params.
in	*zeta*	zeta in the uniform asymptotic expansions - output from hankel_params.
out	*argp*	exp(+2∗pi∗i/3) ∗ nu^(2/3) ∗ zeta.
out	*argm*	exp(-2∗pi∗i/3) ∗ nu^(2/3) ∗ zeta.

Exceptions

std::runtime_error.	

Compute outer factors and associated functions of `z` and `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error `eps`.

Parameters

out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

Parameters

in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

Parameters

in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.

out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.
in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.
in	*nu*	The order for which the Hankel functions are evaluated.
in	*z*	The argument at which the Hankel functions are evaluated.
out	*H1*	The Hankel function of the first kind.
out	*H1p*	The derivative of the Hankel function of the first kind.
out	*H2*	The Hankel function of the second kind.
out	*H2p*	The derivative of the Hankel function of the second kind.

Return the complex cylindrical Hankel function of the first kind.

Parameters

in	*nu*	The order for which the cylindrical Hankel function of the first kind is evaluated.
in	*z*	The argument at which the cylindrical Hankel function of the first kind is evaluated.

Returns

The complex cylindrical Hankel function of the first kind.

Return the complex cylindrical Hankel function of the second kind.

Parameters

in	*nu*	The order for which the cylindrical Hankel function of the second kind is evaluated.
in	*z*	The argument at which the cylindrical Hankel function of the second kind is evaluated.

Returns

The complex cylindrical Hankel function of the second kind.

Return the complex cylindrical Bessel function.

Parameters

in	*nu*	The order for which the cylindrical Bessel function is evaluated.
in	*z*	The argument at which the cylindrical Bessel function is evaluated.

Returns

The complex cylindrical Bessel function.

Return the complex cylindrical Neumann function.

Parameters

in	*nu*	The order for which the cylindrical Neumann function is evaluated.
in	*z*	The argument at which the cylindrical Neumann function is evaluated.

Returns

The complex cylindrical Neumann function.

Helper to compute complex spherical Hankel functions and their derivatives.

Parameters

in	*n*	The order for which the spherical Hankel functions are evaluated.
in	*z*	The argument at which the spherical Hankel functions are evaluated.
out	*H1*	The spherical Hankel function of the first kind.
out	*H1p*	The derivative of the spherical Hankel function of the first kind.
out	*H2*	The spherical Hankel function of the second kind.
out	*H2p*	The derivative of the spherical Hankel function of the second kind.

Return the complex spherical Hankel function of the first kind.

Parameters

in	*n*	The order for which the spherical Hankel function of the first kind is evaluated.
in	*z*	The argument at which the spherical Hankel function of the first kind is evaluated.

Returns

The complex spherical Hankel function of the first kind.

Return the complex spherical Hankel function of the second kind.

Parameters

in	*n*	The order for which the spherical Hankel function of the second kind is evaluated.
in	*z*	The argument at which the spherical Hankel function of the second kind is evaluated.

Returns

The complex spherical Hankel function of the second kind.

Return the complex spherical Bessel function.

Parameters

in	*n*	The order for which the spherical Bessel function is evaluated.
in	*z*	The argument at which the spherical Bessel function is evaluated.

Returns

The complex spherical Bessel function.

Return the complex spherical Neumann function.

Parameters

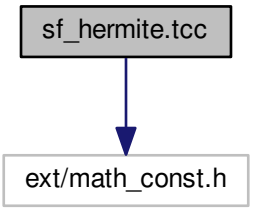
in	*n*	The order for which the spherical Neumann function is evaluated.
in	*z*	The argument at which the spherical Neumann function is evaluated.

Returns

The complex spherical Neumann function.

3.14 sf_hermite.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_hermite.tcc:



Macros

- #define _GLIBCXX_BITS_SF_HERMITE_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.14.1 Macro Definition Documentation

3.14.1.1 #define _GLIBCXX_BITS_SF_HERMITE_TCC 1

3.14.2 Function Documentation

3.14.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This routine returns the Hermite polynomial of order n: $H_n(x)$ by recursion on n.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

Parameters

__n	The order of the Hermite polynomial.
__x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

This routine returns the Hermite polynomial of large order n: $H_n(x)$. We assume here that x >= 0.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

See also

"Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, arXiv:math/0601078v1 [math.CA] 4 Jan 2006

Parameters

__n	The order of the Hermite polynomial.
__x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

This routine returns the Hermite polynomial of order n: $H_n(x)$.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

The Hermite polynomial obeys a reflection formula:

$$H_n(-x) = (-1)^n H_n(x)$$

Parameters

__n	The order of the Hermite polynomial.
__x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

3.15 sf_hydrogen.tcc File Reference

Macros

- #define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.15.1 Macro Definition Documentation

3.15.1.1 #define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1

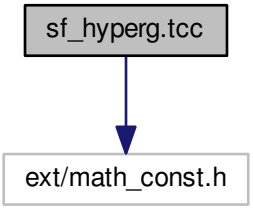
3.15.2 Function Documentation

3.15.2.1 namespace std _GLIBCXX_VISIBILITY (default)

3.16 sf_hyperg.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_hyperg.tcc:



Macros

- #define _GLIBCXX_BITS_SF_HYPERG_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.16.1 Macro Definition Documentation

3.16.1.1 #define _GLIBCXX_BITS_SF_HYPERG_TCC 1

3.16.2 Function Documentation

3.16.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This routine returns the confluent hypergeometric limit function by series expansion.

$$ {}_0F_1(-;c;x) = \Gamma(c) \sum_{n=0}^{\infty} \frac{1}{\Gamma(c+n)} \frac{x^n}{n!} $$

If a and b are integers and a $<$ 0 and either b $>$ 0 or b $<$ a then the series is a polynomial with a finite number of terms.

Parameters

__c	The "denominator" parameter.
__x	The argument of the confluent hypergeometric limit function.

Returns

The confluent hypergeometric limit function.

Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.

Parameters

__c	The *denominator* parameter.
__x	The argument of the confluent hypergeometric limit function.

Returns

The confluent limit hypergeometric function.

This routine returns the confluent hypergeometric function by series expansion.

$$ {}_1F_1(a; c; x) = \frac{\Gamma(c)}{\Gamma(a)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)}{\Gamma(c+n)} \frac{x^n}{n!} $$

Parameters

__a	The "numerator" parameter.
__c	The "denominator" parameter.
__x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Like the case of the 2F1 rational approximations, these are probably guaranteed to converge for x $<$ 0, barring gross numerical instability in the pre-asymptotic regime.

Return the confluent hypergeometric function ${}_1F_1(a; c; x)$.

Parameters

__a	The *numerator* parameter.
__c	The *denominator* parameter.
__x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.

The hypergeometric function is defined by

$$ {}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!} $$

This works and it's pretty fast.

Parameters

__a	The first *numerator* parameter.
__a	The second *numerator* parameter.
__c	The *denominator* parameter.
__x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.↩ 3.6 for d = c - a - b not integral and formula 15.3.11 for d = c - a - b integral. This assumes a, b, c != negative integer.

The hypergeometric function is defined by

$$ {}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!} $$

The reflection formula for nonintegral $d = c - a - b$ is:

$$ {}_2F_1(a, b; c; x) = \frac{\Gamma(c)\Gamma(d)}{\Gamma(c-a)\Gamma(c-b)} {}_2F_1(a, b; 1-d; 1-x) + \frac{\Gamma(c)\Gamma(-d)}{\Gamma(a)\Gamma(b)} {}_2F_1(c-a, c-b; 1+d; 1-x) $$

The reflection formula for integral $m = c - a - b$ is:

$$ {}_2F_1(a, b; a+b+m; x) = \frac{\Gamma(m)\Gamma(a+b+m)}{\Gamma(a+m)\Gamma(b+m)} \sum_{k=0}^{m-1} \frac{(m+a)_k (m+b)_k}{k!(1-m)_k} - $$

Return the hypergeometric function ${}_2F_1(a, b; c; x)$.

The hypergeometric function is defined by

$$ {}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!} $$

Parameters

__a	The first *numerator* parameter.
__a	The second *numerator* parameter.
__c	The *denominator* parameter.
__x	The argument of the confluent hypergeometric function.

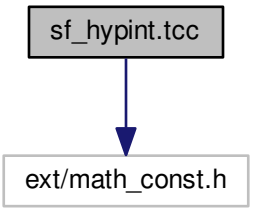
Returns

The confluent hypergeometric function.

3.17 sf_hypint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_hypint.tcc:



Macros

- #define _GLIBCXX_SF_HYPINT_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.17.1 Macro Definition Documentation

3.17.1.1 #define _GLIBCXX_SF_HYPINT_TCC 1

3.17.2 Function Documentation

3.17.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

This function returns the hyperbolic cosine $Ci(x)$ and hyperbolic sine $Si(x)$ integrals as a pair.

The hyperbolic cosine integral is defined by:

$$Chi(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cosh(t) - 1}{t}$$

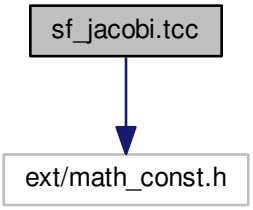
The hyperbolic sine integral is defined by:

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

3.18 sf_jacobi.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_jacobi.tcc:



Macros

- #define _GLIBCXX_SF_JACOBI_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.18.1 Macro Definition Documentation

3.18.1.1 #define _GLIBCXX_SF_JACOBI_TCC 1

3.18.2 Function Documentation

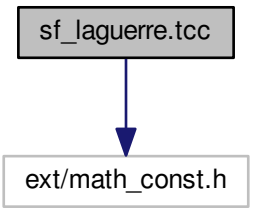
3.18.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Compute the Jacobi polynomial by recursion on x:

$$2k(\alpha+\beta+k)(\alpha+\beta+2k-2)P_k^{(\alpha,\beta)}(x) = (\alpha+\beta+2k-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2k-2)(\alpha+\beta+2k))P_{k-1}^{(\alpha,\beta)}(x)-2(\alpha+k-1)(\beta+k-1)($$

3.19 sf_laguerre.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_laguerre.tcc:



Macros

- #define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.19.1 Macro Definition Documentation

3.19.1.1 #define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1

3.19.2 Function Documentation

3.19.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This routine returns the associated Laguerre polynomial of order n, degree α for large n. Abramowitz & Stegun, 13.5.21

Parameters

__n	The order of the Laguerre function.
__alpha	The degree of the Laguerre function.
__x	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n, degree α, and argument x.

This is from the GNU Scientific Library.

Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!}\,{}_1F_1(-n;\alpha+1;x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a;c;x)$ is the confluent hypergeometric function.

This function assumes x != 0.

This is from the GNU Scientific Library.

This routine returns the associated Laguerre polynomial of order n, degree α: $L_n^\alpha(x)$ by recursion.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!}\,{}_1F_1(-n;\alpha+1;x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a;c;x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!}\frac{d^n}{dx^n}(x^n e^{-x})$$

Parameters

__n	The order of the Laguerre function.
__alpha	The degree of the Laguerre function.
__x	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n, degree α, and argument x.

This routine returns the associated Laguerre polynomial of order n, degree α: $L_n^alpha(x)$.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!}\,{}_1F_1(-n;\alpha+1;x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a;c;x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!}\frac{d^n}{dx^n}(x^n e^{-x})$$

Parameters

__n	The order of the Laguerre function.
__alpha	The degree of the Laguerre function.
__x	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n, degree α, and argument x.

This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n}(x^n e^{-x})$$

Parameters

__n	The order of the Laguerre polynomial.
__m	The degree of the Laguerre polynomial.
__x	The argument of the Laguerre polynomial.

Returns

The value of the associated Laguerre polynomial of order n, degree m, and argument x.

This routine returns the Laguerre polynomial of order n: $L_n(x)$.

The Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n}(x^n e^{-x})$$

Parameters

__n	The order of the Laguerre polynomial.
__x	The argument of the Laguerre polynomial.

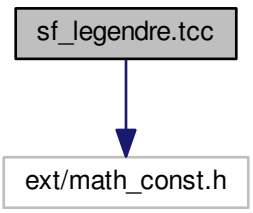
Returns

The value of the Laguerre polynomial of order n and argument x.

3.20 sf_legendre.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_legendre.tcc:



Macros

- #define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.20.1 Macro Definition Documentation

3.20.1.1 #define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1

3.20.2 Function Documentation

3.20.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Return the Legendre polynomial by upward recursion on order l.

The Legendre function of order l and argument x, $P_l(x)$, is defined by:

$$P_l(x) = \frac{1}{2^l l!}\frac{d^l}{dx^l}(x^2 - 1)^l$$

Parameters

l	The order of the Legendre polynomial. $l >= 0$.
x	The argument of the Legendre polynomial. $|x| <= 1$.

Return the Legendre function of the second kind by upward recursion on order l.

The Legendre function of order l and argument x, $Q_l(x)$, is defined by:

$$Q_l(x) = \frac{1}{2^l l!}\frac{d^l}{dx^l}(x^2 - 1)^l$$

Parameters

l	The order of the Legendre polynomial. $l >= 0$.
x	The argument of the Legendre polynomial. $|x| <= 1$.

Return the associated Legendre function by recursion on l and downward recursion on m.

The associated Legendre function is derived from the Legendre function $P_l(x)$ by the Rodrigues formula:

$$P_l^m(x) = (1 - x^2)^{m/2}\frac{d^m}{dx^m}P_l(x)$$

Parameters

l	The order of the associated Legendre function. $l >= 0$.
m	The order of the associated Legendre function. $m <= l$.
x	The argument of the associated Legendre function. $|x| <= 1$.

Return the spherical associated Legendre function.

The spherical associated Legendre function of l, m, and θ is defined as $Y_l^m(\theta, 0)$ where

$$Y_l^m(\theta, \phi) = (-1)^m[\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}]P_l^m(\cos\theta)\exp^{im\phi}$$

is the spherical harmonic function and $P_l^m(x)$ is the associated Legendre function.

This function differs from the associated Legendre function by argument ($x = \cos(\theta)$) and by a normalization factor but this factor is rather large for large l and m and so this function is stable for larger differences of l and m.

Parameters

l	The order of the spherical associated Legendre function. $l >= 0$.
m	The order of the spherical associated Legendre function. $m <= l$.
theta	The radian polar angle argument of the spherical associated Legendre function.

Return the spherical harmonic function.

The spherical harmonic function of l, m, and θ, ϕ is defined by:

$$Y_l^m(\theta,\phi) = (-1)^m[\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}]P_l^{|m|}(\cos\theta)\exp^{im\phi}$$

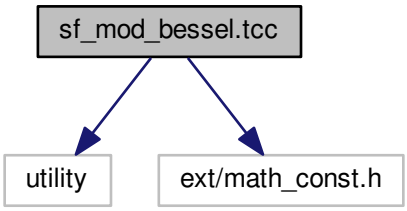
Parameters

l	The order of the spherical harmonic function. $l >= 0$.
m	The order of the spherical harmonic function. $m <= l$.
theta	The radian polar angle argument of the spherical harmonic function.
phi	The radian azimuthal angle argument of the spherical harmonic function.

3.21 sf_mod_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
```

Include dependency graph for sf_mod_bessel.tcc:



Macros

- #define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.21.1 Macro Definition Documentation

3.21.1.1 #define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1

3.21.2 Function Documentation

3.21.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This routine computes the asymptotic modified cylindrical Bessel and functions of order nu: $I_\nu(x)$, $N_\nu(x)$. Use this for $x >> nu^2 + 1$.

References: (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Section 9 p. 364, Equations 9.2.5-9.2.10

Parameters

__nu	The order of the Bessel functions.
__x	The argument of the Bessel functions.
_Jnu	The output Bessel function of the first kind.
_Knu	The output Neumann function (Bessel function of the second kind).

Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

__nu	The order of the Bessel functions.
__x	The argument of the Bessel functions.
_Inu	The output regular modified Bessel function.
_Knu	The output irregular modified Bessel function.
_Ipnu	The output derivative of the regular modified Bessel function.
_Kpnu	The output derivative of the irregular modified Bessel function.

Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.

Return the regular modified Bessel function of order ν: $I_\nu(x)$.

The regular modified cylindrical Bessel function is:

$$I_\nu(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{k!\Gamma(\nu+k+1)}$$

Parameters

__nu	The order of the regular modified Bessel function.
__x	The argument of the regular modified Bessel function.

Returns

The output regular modified Bessel function.

Return the irregular modified Bessel function $K_\nu(x)$ of order ν.

The irregular modified Bessel function is defined by:

$$K_\nu(x) = \frac{\pi}{2}\frac{I_{-\nu}(x) - I_\nu(x)}{\sin\nu\pi}$$

where for integral $\nu = n$ a limit is taken: $lim_{\nu\to n}$. For negative argument we have simply:

$$K_{-\nu}(x) = K_\nu(x)$$

Parameters

__nu	The order of the irregular modified Bessel function.

__x	The argument of the irregular modified Bessel function.

Returns

The output irregular modified Bessel function.

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

Parameters

__n	The order of the modified spherical Bessel function.
__x	The argument of the modified spherical Bessel function.
__i_n	The output regular modified spherical Bessel function.
__k_n	The output irregular modified spherical Bessel function.
__ip_n	The output derivative of the regular modified spherical Bessel function.
__kp_n	The output derivative of the irregular modified spherical Bessel function.

Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi(x)$ respectively.

Parameters

__x	The argument of the Airy functions.
_Ai	The output Airy function of the first kind.
_Bi	The output Airy function of the second kind.
_Aip	The output derivative of the Airy function of the first kind.
_Bip	The output derivative of the Airy function of the second kind.

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

Parameters

__x	The argument of the Airy functions.
__w1	The output Fock-type Airy function of the first kind.
__w2	The output Fock-type Airy function of the second kind.
__w1p	The output derivative of the Fock-type Airy function of the first kind.
__w2p	The output derivative of the Fock-type Airy function of the second kind.

3.22 sf_owens_t.tcc File Reference

Macros

- #define _GLIBCXX_BITS_SF_OWENS_T_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.22.1 Macro Definition Documentation

3.22.1.1 #define _GLIBCXX_BITS_SF_OWENS_T_TCC 1

3.22.2 Function Documentation

3.22.2.1 namespace std _GLIBCXX_VISIBILITY (default)

The CDF of the normal distribution. i.e. the integrated lower tail of the normal PDF.

Return the Owens T function:

$$T(h,a) = \frac{1}{2\pi}\int_0^a \frac{\exp[-\frac{1}{2}h^2(1+x^2)]}{1+x^2}dx$$

This implementation is a translation of the Fortran implementation in

See also

Patefield, M. and Tandy, D. "Fast and accurate Calculation of Owen's T-Function", Journal of Statistical Software, 5 (5), 1 - 25 (2000)

Parameters

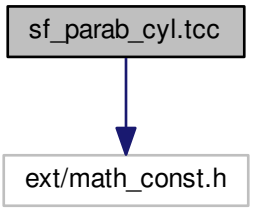
in	*h*	The scale parameter.
in	*a*	The integration limit.

Returns

The owens T function.

3.23 sf_parab_cyl.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_parab_cyl.tcc:



Macros

- #define _GLIBCXX_BITS_SF_PARAB_CYL_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.23.1 Macro Definition Documentation

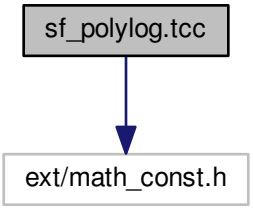
3.23.1.1 #define _GLIBCXX_BITS_SF_PARAB_CYL_TCC 1

3.23.2 Function Documentation

3.23.2.1 namespace std _GLIBCXX_VISIBILITY (default)

3.24 sf_polylog.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_polylog.tcc:



Macros

- #define _GLIBCXX_BITS_SF_POLYLOG_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.24.1 Macro Definition Documentation

3.24.1.1 #define _GLIBCXX_BITS_SF_POLYLOG_TCC 1

3.24.2 Function Documentation

3.24.2.1 namespace std _GLIBCXX_VISIBILITY (default)

A function to reliably compare two floating point numbers.

Parameters

a	the left hand side.
b	the right hand side

Returns

returns true if a and b are equal to zero or differ only by max(a,b)∗ 5∗ eps

A function to reliably test a complex number for realness.

Parameters

w	The complex argument.

Returns

true if Im(w) is zero within 5 $*$ epsilon, false otherwize.

A function to reliably test a complex number for imaginaryness [?].

Parameters

w	The complex argument.

Returns

true if Re(w) is zero within 5 $*$ epsilon, false otherwize.

A function to calculate the values of zeta at even positive integers. For values smaller than thirty a table is used.

Parameters

k	an integer at which we evaluate the Riemann zeta function.

Returns

zeta(k)

This function treats the cases of positive integer index s.

$$Li_s(e^w) = \sum_{k=0,k!=s-1} \zeta(s-k)w^k/k! + (H_{s-1} - log(-w))w^(s-1)/(s-1)!$$

The radius of convergence is |w| $<$ 2 pi. Note that this series involves a log(-x). gcc and Mathematica differ in their implementation of (e^(i)): gcc: (e^(+- i $*$)) = +- i whereas Mathematica doesn't preserve the sign in this case: (e^(+- i)) = +i

Parameters

s	the index s.
w	the argument w.

Returns

the value of the polylogarithm.

This function treats the cases of positive integer index s for real w.

This specialization is worthwhile to catch the differing behaviour of log(x).

$$Li_s(e^w) = \sum_{k=0,k!=s-1} \zeta(s-k)w^k/k! + (H_{s-1} - log(-w))w^(s-1)/(s-1)!$$

The radius of convergence is |w| $<$ 2 pi. Note that this series involves a log(-x). The use of evenzeta yields a speedup of about 2.5. gcc and Mathematica differ in their implementation of (e^(i)): gcc: (e^(+- i)) = +- i whereas Mathematica doesn't preserve the sign in this case: (e^(+- i $*$)) = +i

Parameters

s	the index s.
w	the argument w

Returns

the value of the Polylogarithm

This function treats the cases of negative real index s. Theoretical convergence is present for $|w| < 2*pi$. We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^(s-1) + (2\pi)^(-s)/piA_p(w)$$

$$A_p(w) = \sum_k \Gamma(1+k-s)/k!\sin(\pi/2*(s-k))(w/2/\pi)^k\zeta(1+k-s)$$

Parameters

s	The index s.
w	The Argument w.

Returns

The value of the polylogarithm.

This function treats the cases of negative integer index s which are multiples of two.

In that case the sine occuring in the expansion occasionally takes on the value zero. We use that to provide an optimized series for p = 2n:

In the template parameter sigma we transport whether p = 4k (sigma = 1) or p = 4k + 2 (sigma = -1)

$$Li_p(e^w) = Gamma(1-p)(-w)^{p-1} - A_p(w) - sigma*B_p(w)$$

with

$$A_p(w) = 2(2\pi)^(p-1)(-p)!/(2\pi)^(-p/2)(1+w^2/(4pi^2))^{-1/2+p/2}cos((1-p)ArcTan(2pi/w))$$

and

$$B_p(w) = -2(2pi)^(p-1)\sum_{k=0}^{\infty}\Gamma(2+2k-p)/(2k+1)!(-1)^k(w/2\pi)^(2k+1)(Zeta(2+2k-p) -_T p1)$$

This is suitable for $|w| < 2$ pi The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = Gamma(1-p)(-w)^{p-1} - sigma(2\pi)^p/pi\sum_{k=0}^{\infty}\Gamma(2+2k-p)/(2k+1)!(-1)^k(w/2\pi)^(2k+1)\zeta(2+2k-p)$$

Parameters

n	the index n = 4k.
w	The Argument w

Returns

the value of the Polylogarithm.

This function treats the cases of negative integer index s which are odd.

In that case the sine occuring in the expansion occasionally vanishes. We use that to provide an optimized series for p = 1 + 2k: In the template parameter sigma we transport whether p = 1 + 4k (sigma = 1) or p = 3 + 4k (sigma = -1)

$$Li_p(e^w) = Gamma(1-p) * (-w)^{p-1} + sigma * A_p(w) - sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^(p-1) * Gamma(1-p)(1+w^2/(4pi^2))^{-1/2+p/2} cos((1-p)ArcTan(2pi/w))$$

and

$$B_p(w) = 2(2pi)^(p-1) * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)!(-w^2/4/\pi^2)^k (Zeta(1+2k-p) -_T p1)$$

This is suitable for |w| $<$ 2 pi . The use of evenzeta gives a speedup of about 50 The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = Gamma(1-p)*(-w)^{p-1} - sigma*2*(2pi)^(p-1)*\sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)!(-1)^k (w/2/\pi)^2k) Zeta(1+2k-p)$$

Parameters

n	the index n = 4k.
w	The Argument w.

Returns

The value of the Polylogarithm.

This function treats the cases of negative integer index s and branches accordingly

Parameters

s	the integer index s.
w	The Argument w

Returns

The value of the Polylogarithm evaluated by a suitable function.

This function treats the cases of positive real index s.

The defining series is

$$Li_s(e^w) = A_s(w) + B_s(w) + \Gamma(1-s)(-w)^(s-1)$$

with

$$A_s(w) = \sum_{k=0}^{m} \zeta(s-k) w^k/k!$$

$$B_s(w) = \sum_{k=m+1}^{\infty} \sin(\pi/2(s-k))\Gamma(1-s+k)\zeta(1-s+k)(w/2/\pi)^k/k!$$

Parameters

s	the positive real index s.
w	The argument w.

Returns

the value of the polylogarithm.

This function implements the asymptotic series for the polylog. It is given by 2 {k=0}^ (2k) w^{s-2k}/Gamma(s-2k+1) -i w^(s-1)/Gamma(s) for Re(w) $>>$ 1

Don't check this against Mathematica 8. For real u the imaginary part of the polylog is given by Im(Li_s(e^u)) = -u^{s-1}/Gamma(s) Check this relation for any benchmark that you use. The use of evenzeta leads to a speedup of about 1000.

Parameters

s	the index s.
w	the large argument w.

Returns

the value of the polylogarithm.

Theoretical convergence for Re(w) $<$ 0.

Seems to beat the other expansions for Re(w) $<$ -pi/2 - pi/5. Note that this is an implementation of the basic series:

$$Li_s(e^z) = \sum_{k=1} e^(k * z) * k^(-s)$$

Parameters

s	is an arbitrary type, Integer or float.
w	something with a negative real part.

Returns

the value of the polylogarithm.

Here s is a positive integer and the function descends into the different kernels depending on w.

Parameters

s	a positive integer.
w	an arbitrary complex number.

Returns

The value of the polylogarithm.

Here s is a positive integer and the function descends into the different kernels depending on w.

Parameters

s	a positive integer

w	an arbitrary real argument w

Returns

the value of the polylogarithm.

This treats the case where s is a negative integer.

Parameters

s	a negative integer.
w	an arbitrary complex number

Returns

the value of the polylogarith,.

This treats the case where s is a negative integer and w is a real.

Parameters

s	a negative integer.
w	the argument.

Returns

the value of the polylogarithm.

Return the polylog where s is a positive real value and for complex argument.

Parameters

s	A positive real number.
w	the complex argument.

Returns

The value of the polylogarithm.

Return the polylog where s is a positive real value and the argument is real.

Parameters

s	A positive real number tht does not reduce to an integer.
w	The real argument w.

Returns

The value of the polylogarithm.

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

Parameters

s	A negative real value that does not reduce to a negative integer.
w	The complex argument.

Returns

The value of the polylogarithm.

Return the polylog where s is a negative real value and for real argument. Now we branch depending on the properties of w in the specific functions.

Parameters

s	A negative real value.
w	A real argument.

Returns

The value of the polylogarithm.

This is the frontend function which calculates Li_s(e^w) First we branch into different parts depending on the properties of s. This function is the same irrespective of a real or complex w, hence the template parameter Arg↩ Type.

Note

: I *really* wish we could return a variant<Tp, std::complex<Tp>>.

Parameters

s	The real order.
w	The real or complex argument.

Returns

The real or complex value of Li_s(e^w).

Return the polylog Li_s(x) for two real arguments.

Parameters

s	The real index.
x	The real argument.

Returns

The complex value of the polylogarithm.

Return the polylog in those cases where we can calculate it.

Parameters

s	The real index.
w	The complex argument.

Returns

The complex value of the polylogarithm.

Return the Dirichlet eta function. Currently, w must be real (complex type but negligible imaginary part.) Otherwise std::domain_error is thrown.

Parameters

w	The complex (but on-real-axis) argument.

Returns

The complex Dirichlet eta function.

Exceptions

std::domain_error	if the argument has a significant imaginary part.

Return the Dirichlet eta function for real argument.

Parameters

w	The real argument.

Returns

The Dirichlet eta function.

Return the Dirichlet beta function. Currently, w must be real (complex type but negligible imaginary part.) Otherwise std::domain_error is thrown.

Parameters

w	The complex (but on-real-axis) argument.

Returns

The Dirichlet Beta function of real argument.

Exceptions

std::domain_error	if the argument has a significant imaginary part.

Return the Dirichlet beta function for real argument.

Parameters

w	The real argument.

Returns

The Dirichlet Beta function of real argument.

Return Clausen's function of integer order m and complex argument w. The notation and connection to polylog is from Wikipedia

Parameters

w	The complex argument.

Returns

The complex Clausen function.

Return Clausen's function of integer order m and real argument w. The notation and connection to polylog is from Wikipedia

Parameters

m	The integer order m >= 1.
w	The real argument.

Returns

The Clausen function.

Return Clausen's sine sum Sl_m for positive integer order m and complex argument w.

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

m	The integer order m >= 1.
w	The complex argument.

Returns

The Clausen sine sum Sl_m(w),

Return Clausen's sine sum Sl_m for positive integer order m and real argument w.

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

m	The integer order m >= 1.
w	The complex argument.

Returns

The Clausen sine sum Sl_m(w),

Return Clausen's cosine sum Cl_m for positive integer order m and complex argument w.

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

m	The integer order m >= 1.
w	The real argument.

Returns

The Clausen cosine sum Cl_m(w),

Return Clausen's cosine sum Cl_m for positive integer order m and real argument w.

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

m	The integer order m >= 1.
w	The real argument.

Returns

The real Clausen cosine sum Cl_m(w),

Return the Fermi-Dirac integral of real order s and real argument x.

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12#iii

Parameters

s	The order s >= 0.
x	The real argument.

Returns

The real Fermi-Dirac cosine sum F_s(x),

Return the Bose-Einstein integral of real order s and real argument x.

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12#iii

Parameters

s	The order s >= 0.
x	The real argument.

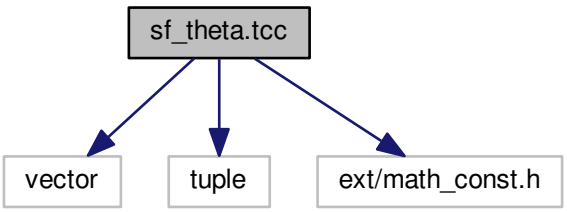
Returns

The real Fermi-Dirac cosine sum G_s(x),

3.25 sf_theta.tcc File Reference

```
#include <vector>
#include <tuple>
#include <ext/math_const.h>
```

Include dependency graph for sf_theta.tcc:



Macros

- #define _GLIBCXX_SF_THETA_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.25.1 Macro Definition Documentation

3.25.1.1 #define _GLIBCXX_SF_THETA_TCC 1

3.25.2 Function Documentation

3.25.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Compute and return the function by series expansion.

Compute and return the function by series expansion.

Compute and return the function by series expansion.

Compute and return the function by asymptotic series expansion.

Return the function

Return the function

Return the function

Return the function

Use MacLaurin series to calculate the elliptic nome given the , k.

Use the arithmetic-geometric mean to calculate the elliptic nome given the , k.

Return the elliptic nome given the , k.

Return the Neville function

Return the Neville function

Return the Neville function

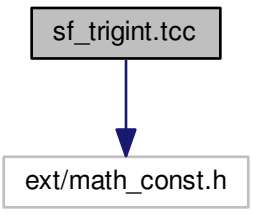
Return the Neville function

Return a tuple of the three primary Jacobi elliptic functions: sn(k, u), cn(k, u), dn(k, u).

3.26 sf_trigint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_trigint.tcc:



Macros

- #define _GLIBCXX_SF_TRIGINT_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.26.1 Macro Definition Documentation

3.26.1.1 #define _GLIBCXX_SF_TRIGINT_TCC 1

3.26.2 Function Documentation

3.26.2.1 namespace std _GLIBCXX_VISIBILITY (default)

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for x $>$ 50.

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.

The sine integral is defined by:

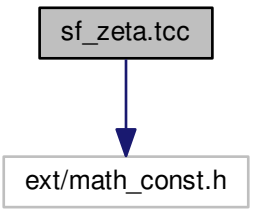
$$Si(x) = \int_0^x dt \frac{\sin(t)}{t}$$

The cosine integral is defined by:

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

3.27 sf_zeta.tcc File Reference

`#include <ext/math_const.h>`
Include dependency graph for sf_zeta.tcc:



Macros

- #define _GLIBCXX_BITS_SF_ZETA_TCC 1

Functions

- namespace std _GLIBCXX_VISIBILITY (default)

3.27.1 Macro Definition Documentation

3.27.1.1 #define _GLIBCXX_BITS_SF_ZETA_TCC 1

3.27.2 Function Documentation

3.27.2.1 namespace std _GLIBCXX_VISIBILITY (default)

Coefficients for Euler-Maclaurin summation of zeta functions.

$$B_{2j}/(2j)!$$

where B_k are the Bernoulli numbers.

Compute the dilogarithm function $Li_2(x)$ by summation for x <= 1.

The Riemann zeta function is defined by:

$$Li_2(x) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

For |x| near 1 use the reflection formulae:

$$Li_2(-x) + Li_2(1-x) = \frac{\pi^2}{6} - \ln(x)\ln(1-x)$$

$$Li_2(-x) - Li_2(1-x) - \frac{1}{2}Li_2(1-x^2) = -\frac{\pi^2}{12} - \ln(x)\ln(1-x)$$

For x $<$ 1 use the reflection formula:

$$Li_2(1-x) - Li_2(1-\frac{1}{1-x}) - \frac{1}{2}(\ln(x))^2$$

Compute the Riemann zeta function $\zeta(s)$ by summation for s $>$ 1.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

For s $<$ 1 use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s)\zeta(1-s)$$

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s $>$ 0.

This is a specialization of the code for the Hurwitz zeta function.

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s $>$ 0.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

For s $<$ 1 use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s)\zeta(1-s)$$

Evaluate the Riemann zeta function by series for all s != 1. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.

The series is:

$$\zeta(s) = \frac{1}{1-2^{1-s}} \sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \sum_{k=0}^{n} (-1)^k \frac{n!}{(n-k)!k!} (k+1)^{-s}$$

Havil 2003, p. 206.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

For s < 1 use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

$$\zeta(s) = \Pi_{i=1}^{\infty} \frac{1}{1-p_i^{-s}}$$

where p_i are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

For s < 1 use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Return the Riemann zeta function $\zeta(s)$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} for s > 1 \frac{(2\pi)^s}{pi} sin(\frac{\pi s}{2}) \Gamma(1-s) \zeta(1-s) for s < 1$$

For s < 1 use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Return the Hurwitz zeta function $\zeta(s, a)$ for all s != 1 and x $>$ -1.

See also

An efficient algorithm for accelerating the convergence of oscillatory series, useful for computing the polylogarithm and Hurwitz zeta functions, Linas Vep

Table of zeta(n) - 1 from 2 - 32. MPFR - 128 bits.

Return the Riemann zeta function $\zeta(s) - 1$ by summation for s > 1. This is a small remainder for large s.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} for s > 1$$

Return the Riemann zeta function $\zeta(s) - 1$

Return the Hurwitz zeta function $\zeta(s, a)$ for all s != 1 and a $>$ -1.

The Hurwitz zeta function is defined by:

$$\zeta(s, a) = \sum_{n=0}^{\infty} \frac{1}{(n + a)^s}$$

The Riemann zeta function is a special case:

$$\zeta(s) = \zeta(s, 1)$$

Index