TR29124 C++ Special Math Functions

2.0

Generated by Doxygen 1.8.9.1

Tue May 10 2016 12:21:15

Contents

Chapter 1

Mathematical Special Functions

1.1 Introduction and History

The first significant library upgrade on the road to C++2011, TR1, included a set of 23 mathematical functions that significantly extended the standard transcendental functions inherited from C and declared in <cmath>.

Although most components from TR1 were eventually adopted for C++11 these math functions were left behind out of concern for implementability. The math functions were published as a separate international standard `IS 29124 - Extensions to the C++ Library to Support Mathematical Special Functions`.

For C++17 these functions were incorporated into the main standard.

1.2 Contents

The following functions are implemented in namespace `std`:

- assoc_laguerre - Associated Laguerre functions
- assoc_legendre - Associated Legendre functions
- beta - Beta functions
- comp_ellint_1 - Complete elliptic functions of the first kind
- comp_ellint_2 - Complete elliptic functions of the second kind
- comp_ellint_3 - Complete elliptic functions of the third kind
- cyl_bessel_i - Regular modified cylindrical Bessel functions
- cyl_bessel_j - Cylindrical Bessel functions of the first kind
- cyl_bessel_k - Irregular modified cylindrical Bessel functions
- cyl_neumann - Cylindrical Neumann functions or Cylindrical Bessel functions of the second kind
- ellint_1 - Incomplete elliptic functions of the first kind
- ellint_2 - Incomplete elliptic functions of the second kind
- ellint_3 - Incomplete elliptic functions of the third kind

- expint - The exponential integral
- hermite - Hermite polynomials
- laguerre - Laguerre functions
- legendre - Legendre polynomials
- riemann_zeta - The Riemann zeta function
- sph_bessel - Spherical Bessel functions
- sph_legendre - Spherical Legendre functions
- sph_neumann - Spherical Neumann functions

The hypergeometric functions were stricken from the TR29124 and C++17 versions of this math library because of implementation concerns. However, since they were in the TR1 version and since they are popular we kept them as an extension in namespace `__gnu_cxx`:

- conf_hyperg - Confluent hypergeometric functions
- hyperg - Hypergeometric functions

In addition a large number of new functions are added as extensions:

- airy_ai - Airy functions of the first kind
- airy_bi - Airy functions of the second kind
- bincoef - Binomial coefficients
- chebyshev_t - Chebyshev polynomials of the first kind
- chebyshev_u - Chebyshev polynomials of the second kind
- chebyshev_v - Chebyshev polynomials of the third kind
- chebyshev_w - Chebyshev polynomials of the fourth kind
- clausen - Clausen integrals
- clausen_c - Clausen cosine integrals
- clausen_s - Clausen sine integrals
- comp_ellint_d - Incomplete Legendre D elliptic integral
- conf_hyperg_lim - Confluent hypergeometric limit functions
- coshint - Hyperbolic cosine integral
- cosint - Cosine integral
- cyl_hankel_1 - Cylindrical Hankel functions of the first kind
- cyl_hankel_2 - Cylindrical Hankel functions of the second kind
- dawson - Dawson integrals
- dilog - Dilogarithm functions
- dirichlet_beta - Dirichlet beta function

- pochhammer_l - Lower Pochhammer functions
- pochhammer_u - Upper Pochhammer functions
- psi - Psi of digamma function
- radpoly - Radial polynomials
- sinhc - Hyperbolic sinus cardinal function
- sinhc_pi -
- sinc - Sinus cardinal function
- sinc_pi -
- sinhint - Hyperbolic sine integral
- sinint - Sine integral
- sph_bessel_i - Spherical regular modified Bessel functions
- sph_bessel_k - Spherical iregular modified Bessel functions
- sph_hankel_1 - Spherical Hankel functions of the first kind
- sph_hankel_2 - Spherical Hankel functions of the first kind
- sph_harmonic - Spherical
- zernike - Zernike polynomials

1.3 General Features

1.3.1 Argument Promotion

The arguments suppled to the non-suffixed functions will be promoted according to the following rules:

1. If any argument intended to be floating point is given an integral value That integral value is promoted to double.
2. All floating point arguments are promoted up to the largest floating point precision among them.

1.3.2 NaN Arguments

If any of the floating point arguments supplied to these functions is invalid or NaN (std::numeric_limits<Tp>::quiet_NaN), the value NaN is returned.

1.4 Implementation

We strive to implement the underlying math with type generic algorithms to the greatest extent possible. In practice, the functions are thin wrappers that dispatch to function templates. Type dependence is controlled with std::numeric_limits and functions thereof.

We don't promote `float` to `double` or `double` to `long double` reflexively. The goal is for `float` functions to operate more quickly, at the cost of `float` accuracy and possibly a smaller domain of validity. Similaryly, `long double` should give you more dynamic range and slightly more pecision than `double` on many systems.

1.5 Testing

These functions have been tested against equivalent implementations from the Gnu Scientific Library, GSL and <a href="http://www.boost.org/doc/libs/1_60_0/libs/math/doc/html/index.html>Boost and the ratio

$$\frac{|f - f_{test}|}{|f_{test}|}$$

is generally found to be within 10^-15 for 64-bit double on linux-x86_64 systems over most of the ranges of validity.

Todo Provide accuracy comparisons on a per-function basis for a small number of targets.

1.6 General Bibliography

See also

Abramowitz and Stegun: Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables Edited by Milton Abramowitz and Irene A. Stegun, National Bureau of Standards Applied Mathematics Series - 55 Issued June 1964, Tenth Printing, December 1972, with corrections Electronic versions of A&S abound including both pdf and navigable html.
for example http://people.math.sfu.ca/~cbm/aands/
The old A&S has been redone as the NIST Digital Library of Mathematical Functions: http://dlmf.nist.gov/ This version is far more navigable and includes more recent work.
An Atlas of Functions: with Equator, the Atlas Function Calculator 2nd Edition, by Oldham, Keith B., Myland, Jan, Spanier, Jerome
Asymptotics and Special Functions by Frank W. J. Olver, Academic Press, 1974
Numerical Recipes in C, The Art of Scientific Computing, by William H. Press, Second Ed., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, Cambridge University Press, 1992
The Special Functions and Their Approximations: Volumes 1 and 2, by Yudell L. Luke, Academic Press, 1969

Chapter 2

Todo List

page Mathematical Special Functions

Provide accuracy comparisons on a per-function basis for a small number of targets.

Member std::__detail::__dawson_cont_frac (_Tp __x)

this needs some compile-time construction!

Member std::__detail::__expint_E1 (_Tp __x)

Find a good asymptotic switch point in $E_1(x)$.

Member std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)

Find a principled starting number for the $E_n(x)$ downward recursion.

Member std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)

This __hurwitz_zeta prefactor is prone to overflow. positive integer orders s?

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

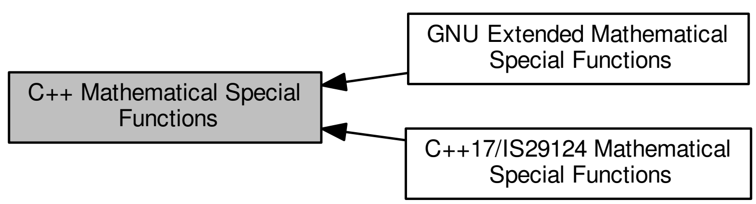
Here is a list of all files with brief descriptions:

Chapter 7

Module Documentation

7.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

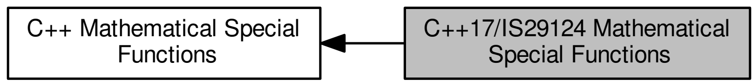
- C++17/IS29124 Mathematical Special Functions
- GNU Extended Mathematical Special Functions

7.1.1 Detailed Description

A collection of advanced mathematical special functions.

7.2 C++17/IS29124 Mathematical Special Functions

Collaboration diagram for C++17/IS29124 Mathematical Special Functions:



Functions

- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)
- float std::assoc_laguerref (unsigned int __n, unsigned int __m, float __x)
- long double std::assoc_laguerrel (unsigned int __n, unsigned int __m, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::assoc_legendre (unsigned int __l, unsigned int __m, _Tp __x)
- float std::assoc_legendref (unsigned int __l, unsigned int __m, float __x)
- long double std::assoc_legendrel (unsigned int __l, unsigned int __m, long double __x)
- template<typename _Tpa , typename _Tpb >
  __gnu_cxx::__promote_2< _Tpa, _Tpb >::__type std::beta (_Tpa __a, _Tpb __b)
- float std::betaf (float __a, float __b)
- long double std::betal (long double __a, long double __b)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1 (_Tp __k)
- float std::comp_ellint_1f (float __k)
- long double std::comp_ellint_1l (long double __k)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2 (_Tp __k)
- float std::comp_ellint_2f (float __k)
- long double std::comp_ellint_2l (long double __k)
- template<typename _Tp , typename _Tpn >
  __gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3 (_Tp __k, _Tpn __nu)
- float std::comp_ellint_3f (float __k, float __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `float` modulus k.
- long double std::comp_ellint_3l (long double __k, long double __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `long double` modulus k.
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_if (float __nu, float __x)
- long double std::cyl_bessel_il (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_jf (float __nu, float __x)
- long double std::cyl_bessel_jl (long double __nu, long double __x)

- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_k (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_kf (float __nu, float __x)
- long double std::cyl_bessel_kl (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_neumann (_Tpnu __nu, _Tp __x)
- float std::cyl_neumannf (float __nu, float __x)
- long double std::cyl_neumannl (long double __nu, long double __x)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_1 (_Tp __k, _Tpp __phi)
- float std::ellint_1f (float __k, float __phi)
- long double std::ellint_1l (long double __k, long double __phi)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_2 (_Tp __k, _Tpp __phi)
- float std::ellint_2f (float __k, float __phi)

  Return the incomplete elliptic integral of the second kind $E(k,\phi)$ for `float` argument.
- long double std::ellint_2l (long double __k, long double __phi)

  Return the incomplete elliptic integral of the second kind $E(k,\phi)$.
- template<typename _Tp , typename _Tpn , typename _Tpp >
  __gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type std::ellint_3 (_Tp __k, _Tpn __nu, _Tpp __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$.
- float std::ellint_3f (float __k, float __nu, float __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$ for `float` argument.
- long double std::ellint_3l (long double __k, long double __nu, long double __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$.
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::expint (_Tp __x)
- float std::expintf (float __x)
- long double std::expintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)
- float std::hermitef (unsigned int __n, float __x)
- long double std::hermitel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::laguerre (unsigned int __n, _Tp __x)
- float std::laguerref (unsigned int __n, float __x)
- long double std::laguerrel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::legendre (unsigned int __l, _Tp __x)
- float std::legendref (unsigned int __l, float __x)
- long double std::legendrel (unsigned int __l, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::riemann_zeta (_Tp __s)
- float std::riemann_zetaf (float __s)
- long double std::riemann_zetal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_bessel (unsigned int __n, _Tp __x)
- float std::sph_besself (unsigned int __n, float __x)
- long double std::sph_bessell (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)

- float std::sph_legendref (unsigned int __l, unsigned int __m, float __theta)
- long double std::sph_legendrel (unsigned int __l, unsigned int __m, long double __theta)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_neumann (unsigned int __n, _Tp __x)
- float std::sph_neumannf (unsigned int __n, float __x)
- long double std::sph_neumannl (unsigned int __n, long double __x)

7.2.1 Detailed Description

A collection of advanced mathematical special functions for C++17 and IS29124.

7.2.2 Function Documentation

7.2.2.1 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::assoc_laguerre (unsigned int *__n*, unsigned int *__m*, _Tp *__x*) [inline]

Return the associated Laguerre polynomial $L_n^m(x)$ of nonnegative order n, nonnegative degree m and real argument x.

The associated Laguerre function of real degree α, $L_n^\alpha(x)$, 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 degree $\alpha = m$ by:

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

where the Laguerre polynomial is defined by:

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

and $x >= 0$.

See also

laguerre for details of the Laguerre function of degree `n`

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__n	The order of the Laguerre function, `__n >= 0`.
__m	The degree of the Laguerre function, `__m >= 0`.
__x	The argument of the Laguerre function, `__x >= 0`.

Exceptions

std::domain_error	if `__x < 0`.

Definition at line 371 of file specfun.h.

7.2.2.2 float std::assoc_laguerref (unsigned int *__n*, unsigned int *__m*, float *__x*) `[inline]`

Return the associated Laguerre polynomial $L_n^m(x)$ of order n, degree m, and `float` argument x.

See also

assoc_laguerre for more details.

Definition at line 323 of file specfun.h.

7.2.2.3 long double std::assoc_laguerrel (unsigned int *__n*, unsigned int *__m*, long double *__x*) `[inline]`

Return the associated Laguerre polynomial $L_n^m(x)$ of order n, degree m and `long double` argument x.

See also

assoc_laguerre for more details.

Definition at line 334 of file specfun.h.

7.2.2.4 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::assoc_legendre (unsigned int *__l*, unsigned int *__m*, _Tp *__x*) `[inline]`

Return the associated Legendre function $P_l^m(x)$ of degree l, order m, and real argument x.

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

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

See also

legendre for details of the Legendre function of degree `l`

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__l	The degree `__l >= 0`.
__m	The order `__m <= l`.
__x	The argument, `abs(__x) <= 1`.

Exceptions

std::domain_error	if `abs(__x) > 1`.

Definition at line 419 of file specfun.h.

7.2.2.5 float std::assoc_legendref (unsigned int __*l*, unsigned int __*m*, float __*x*) `[inline]`

Return the associated Legendre function $P_l^m(x)$ of degree l, order m, and `float` argument x.

See also

assoc_legendre for more details.

Definition at line 386 of file specfun.h.

7.2.2.6 long double std::assoc_legendrel (unsigned int __*l*, unsigned int __*m*, long double __*x*) `[inline]`

Return the associated Legendre function $P_l^m(x)$ of degree l, order m, and `long double` argument x.

See also

assoc_legendre for more details.

Definition at line 397 of file specfun.h.

7.2.2.7 template<typename _Tpa , typename _Tpb > __gnu_cxx::__promote_2<_Tpa, _Tpb>::__type std::beta (_Tpa __*a*, _Tpb __*b*) `[inline]`

Return the beta function, $B(a,b)$, for real parameters `a`, `b`.

The beta function is defined by

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

where $a > 0$ and $b > 0$

Template Parameters

_Tpa	The floating-point type of the parameter `__a`.
_Tpb	The floating-point type of the parameter `__b`.

Parameters

__a	The first argument of the beta function, `__a > 0` .
__b	The second argument of the beta function, `__b > 0` .

Exceptions

std::domain_error	if `__a < 0` or `__b < 0` .

Definition at line 464 of file specfun.h.

7.2.2.8 float std::betaf (float __*a*, float __*b*) `[inline]`

Return the beta function, $B(a,b)$, for `float` parameters `a`, `b`.

See also

beta for more details.

Definition at line 433 of file specfun.h.

7.2.2.9 long double std::betal (long double __a, long double __b) [inline]

Return the beta function, $B(a,b)$, for long double parameters a, b.

See also

beta for more details.

Definition at line 443 of file specfun.h.

7.2.2.10 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::comp_ellint_1 (_Tp __k) [inline]

Return the complete elliptic integral of the first kind $K(k)$ for real modulus k.

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 and the modulus $|k| <= 1$.

See also

ellint_1 for details of the incomplete elliptic function of the first kind.

Template Parameters

_Tp	The floating-point type of the modulus __k.

Parameters

__k	The modulus, abs(__k) <= 1

Exceptions

std::domain_error	if abs(__k) > 1 .

Definition at line 512 of file specfun.h.

7.2.2.11 float std::comp_ellint_1f (float __k) [inline]

Return the complete elliptic integral of the first kind $E(k)$ for float modulus k.

See also

comp_ellint_1 for details.

Definition at line 479 of file specfun.h.

7.2.2.12 long double std::comp_ellint_1l (long double __k) [inline]

Return the complete elliptic integral of the first kind $E(k)$ for long double modulus k.

See also

comp_ellint_1 for details.

Definition at line 489 of file specfun.h.

7.2.2.13 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::comp_ellint_2 (_Tp __k) [inline]

Return the complete elliptic integral of the second kind $E(k)$ for real modulus k.

The complete elliptic integral of the second kind is defined as

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

where $E(k, \phi)$ is the incomplete elliptic integral of the second kind and the modulus $|k| <= 1$.

See also

ellint_2 for details of the incomplete elliptic function of the second kind.

Template Parameters

_Tp	The floating-point type of the modulus `__k`.

Parameters

__k	The modulus, `abs(__k)` <= 1

Exceptions

std::domain_error	if `abs(__k)` > 1.

Definition at line 559 of file specfun.h.

7.2.2.14 float std::comp_ellint_2f (float __k) [inline]

Return the complete elliptic integral of the second kind $E(k)$ for `float` modulus k.

See also

comp_ellint_2 for details.

Definition at line 527 of file specfun.h.

7.2.2.15 long double std::comp_ellint_2l (long double __k) [inline]

Return the complete elliptic integral of the second kind $E(k)$ for `long double` modulus k.

See also

comp_ellint_2 for details.

Definition at line 537 of file specfun.h.

7.2.2.16 template<typename _Tp , typename _Tpn > __gnu_cxx::__promote_2<_Tp, _Tpn>::__type std::comp_ellint_3 (_Tp *__k*, _Tpn *__nu*) [inline]

Return the complete elliptic integral of the third kind $\Pi(k,\nu) = \Pi(k,\nu,\pi/2)$ for real modulus k.

The complete elliptic integral of the third kind is defined as

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

where $\Pi(k,\nu,\phi)$ is the incomplete elliptic integral of the second kind and the modulus $|k| <= 1$.

See also

ellint_3 for details of the incomplete elliptic function of the third kind.

Template Parameters

_Tp	The floating-point type of the modulus `__k`.
_Tpn	The floating-point type of the argument `__nu`.

Parameters

__k	The modulus, `abs(__k) <=` 1
__nu	The argument

Exceptions

std::domain_error	if `abs(__k)` > 1.

Definition at line 610 of file specfun.h.

7.2.2.17 float std::comp_ellint_3f (float *__k*, float *__nu*) [inline]

Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `float` modulus k.

See also

comp_ellint_3 for details.

Definition at line 574 of file specfun.h.

7.2.2.18 long double std::comp_ellint_3l (long double *__k*, long double *__nu*) [inline]

Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `long double` modulus `k`.

See also

comp_ellint_3 for details.

Definition at line 584 of file specfun.h.

7.2.2.19 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_2<_Tpnu, _Tp>::__type std::cyl_bessel_i (_Tpnu __nu, _Tp __x) [inline]

Return the regular modified Bessel function $I_{\nu}(x)$ for real order ν and argument $x >= 0$.

The regular modified cylindrical Bessel function is:

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

Template Parameters

_Tpnu	The floating-point type of the order `__nu`.
_Tp	The floating-point type of the argument `__x`.

Parameters

__nu	The order
__x	The argument, `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 656 of file specfun.h.

7.2.2.20 float std::cyl_bessel_if (float __nu, float __x) [inline]

Return the regular modified Bessel function $I_{\nu}(x)$ for `float` order ν and argument $x >= 0$.

See also

cyl_bessel_i for setails.

Definition at line 625 of file specfun.h.

7.2.2.21 long double std::cyl_bessel_il (long double __nu, long double __x) [inline]

Return the regular modified Bessel function $I_{\nu}(x)$ for `long double` order ν and argument $x >= 0$.

See also

cyl_bessel_i for setails.

Definition at line 635 of file specfun.h.

7.2.2.22 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_2<_Tpnu, _Tp>::__type std::cyl_bessel_j (_Tpnu __nu, _Tp __x) [inline]

Return the Bessel function $J_{\nu}(x)$ of real order ν and argument $x >= 0$.

The cylindrical Bessel function is:

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

Template Parameters

_Tpnu	The floating-point type of the order `__nu`.
_Tp	The floating-point type of the argument `__x`.

Parameters

__nu	The order
__x	The argument, `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 702 of file specfun.h.

7.2.2.23 float std::cyl_bessel_jf (float *__nu*, float *__x*) `[inline]`

Return the Bessel function of the first kind $J_\nu(x)$ for `float` order ν and argument $x >= 0$.

See also

cyl_bessel_j for setails.

Definition at line 671 of file specfun.h.

7.2.2.24 long double std::cyl_bessel_jl (long double *__nu*, long double *__x*) `[inline]`

Return the Bessel function of the first kind $J_\nu(x)$ for `long double` order ν and argument $x >= 0$.

See also

cyl_bessel_j for setails.

Definition at line 681 of file specfun.h.

7.2.2.25 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_2<_Tpnu, _Tp>::__type std::cyl_bessel_k (_Tpnu *__nu*, _Tp *__x*) `[inline]`

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

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:

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

Template Parameters

_Tpnu	The floating-point type of the order `__nu`.
_Tp	The floating-point type of the argument `__x`.

Parameters

__nu	The order
__x	The argument, `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 754 of file specfun.h.

7.2.2.26 float std::cyl_bessel_kf (float *__nu,* float *__x*) `[inline]`

Return the irregular modified Bessel function $K_{\nu}(x)$ for `float` order ν and argument $x >= 0$.

See also

cyl_bessel_k for setails.

Definition at line 717 of file specfun.h.

7.2.2.27 long double std::cyl_bessel_kl (long double *__nu,* long double *__x*) `[inline]`

Return the irregular modified Bessel function $K_{\nu}(x)$ for `long double` order ν and argument $x >= 0$.

See also

cyl_bessel_k for setails.

Definition at line 727 of file specfun.h.

7.2.2.28 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_2<_Tpnu, _Tp>::__type std::cyl_neumann (_Tpnu *__nu,* _Tp *__x*) `[inline]`

Return the Neumann function $N_{\nu}(x)$ of real order ν and argument $x >= 0$.

The Neumann function is defined by:

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

where $x >= 0$ and for integral order $\nu = n$ a limit is taken: $lim_{\nu \to n}$.

Template Parameters

_Tpnu	The floating-point type of the order `__nu`.
_Tp	The floating-point type of the argument `__x`.

Parameters

__nu	The order
__x	The argument, `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 802 of file specfun.h.

7.2.2.29 float std::cyl_neumannf (float *__nu*, float *__x*) `[inline]`

Return the Neumann function $N_\nu(x)$ of `float` order ν and argument x.

See also

cyl_neumann for setails.

Definition at line 769 of file specfun.h.

7.2.2.30 long double std::cyl_neumannl (long double *__nu*, long double *__x*) `[inline]`

Return the Neumann function $N_\nu(x)$ of `long double` order ν and argument x.

See also

cyl_neumann for setails.

Definition at line 779 of file specfun.h.

7.2.2.31 template<typename _Tp , typename _Tpp > __gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_1 (_Tp *__k*, _Tpp *__phi*) `[inline]`

Return the incomplete elliptic integral of the first kind $F(k,\phi)$ for `real` modulus k and angle ϕ.

The incomplete elliptic integral of the first kind is defined as

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

For $\phi = \pi/2$ this becomes the complete elliptic integral of the first kind, $K(k)$.

See also

comp_ellint_1.

Template Parameters

_Tp	The floating-point type of the modulus `__k`.

_Tpp	The floating-point type of the angle `__phi`.

Parameters

__k	The modulus, `abs(__k) <= 1`
__phi	The integral limit argument in radians

Exceptions

std::domain_error	if `abs(__k) > 1` .

Definition at line 850 of file specfun.h.

7.2.2.32 float std::ellint_1f (float *__k*, float *__phi*) [inline]

Return the incomplete elliptic integral of the first kind $E(k, \phi)$ for `float` modulus k and angle ϕ.

See also

ellint_1 for details.

Definition at line 817 of file specfun.h.

7.2.2.33 long double std::ellint_1l (long double *__k*, long double *__phi*) [inline]

Return the incomplete elliptic integral of the first kind $E(k, \phi)$ for `long double` modulus k and angle ϕ.

See also

ellint_1 for details.

Definition at line 827 of file specfun.h.

7.2.2.34 template<typename _Tp , typename _Tpp > __gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_2 (_Tp *__k*, _Tpp *__phi*) [inline]

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

The incomplete elliptic integral of the second kind is defined as

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

For $\phi = \pi/2$ this becomes the complete elliptic integral of the second kind, $E(k)$.

See also

comp_ellint_2.

Template Parameters

_Tp	The floating-point type of the modulus `__k`.
_Tpp	The floating-point type of the angle `__phi`.

Parameters

__k	The modulus, `abs(__k) <= 1`
__phi	The integral limit argument in radians

Returns

The elliptic function of the second kind.

Exceptions

std::domain_error	if `abs(__k) > 1` .

Definition at line 898 of file specfun.h.

7.2.2.35 float std::ellint_2f (float *__k*, float *__phi*) `[inline]`

Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for `float` argument.

See also

ellint_2 for details.

Definition at line 865 of file specfun.h.

7.2.2.36 long double std::ellint_2l (long double *__k*, long double *__phi*) `[inline]`

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

See also

ellint_2 for details.

Definition at line 875 of file specfun.h.

7.2.2.37 template<typename _Tp , typename _Tpn , typename _Tpp > __gnu_cxx::__promote_3<_Tp, _Tpn, _Tpp>::__type std::ellint_3 (_Tp *__k*, _Tpn *__nu*, _Tpp *__phi*) `[inline]`

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

The incomplete elliptic integral of the third kind is defined by:

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

For $\phi = \pi/2$ this becomes the complete elliptic integral of the third kind, $\Pi(k, \nu)$.

See also

comp_ellint_3.

Template Parameters

_Tp	The floating-point type of the modulus `__k`.
_Tpn	The floating-point type of the argument `__nu`.
_Tpp	The floating-point type of the angle `__phi`.

Parameters

__k	The modulus, `abs(__k) <= 1`
__nu	The second argument
__phi	The integral limit argument in radians

Returns

The elliptic function of the third kind.

Exceptions

std::domain_error	if `abs(__k) > 1` .

Definition at line 951 of file specfun.h.

7.2.2.38 float std::ellint_3f (float *__k,* float *__nu,* float *__phi*) [inline]

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for `float` argument.

See also

ellint_3 for details.

Definition at line 913 of file specfun.h.

7.2.2.39 long double std::ellint_3l (long double *__k,* long double *__nu,* long double *__phi*) [inline]

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

See also

ellint_3 for details.

Definition at line 923 of file specfun.h.

7.2.2.40 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::expint (_Tp *__x*) [inline]

Return the exponential integral $Ei(x)$ for `real` argument `x`.

The exponential integral is given by

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

Template Parameters

_Tp	The floating-point type of the argument ___x.

Parameters

__x	The argument of the exponential integral function.

Definition at line 991 of file specfun.h.

7.2.2.41 float std::expintf (float __x) [inline]

Return the exponential integral $Ei(x)$ for `float` argument `x`.

See also

expint for details.

Definition at line 965 of file specfun.h.

7.2.2.42 long double std::expintl (long double __x) [inline]

Return the exponential integral $Ei(x)$ for `long double` argument `x`.

See also

expint for details.

Definition at line 975 of file specfun.h.

7.2.2.43 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::hermite (unsigned int __n, _Tp __x) [inline]

Return the Hermite polynomial $H_n(x)$ of order n and `real` argument `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:

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

Template Parameters

_Tp	The floating-point type of the argument ___x.

Parameters

__n	The order

__x	The argument

Definition at line 1039 of file specfun.h.

7.2.2.44 float std::hermitef (unsigned int *__n*, float *__x*) [inline]

Return the Hermite polynomial $H_n(x)$ of nonnegative order n and float argument `x`.

See also

hermite for details.

Definition at line 1006 of file specfun.h.

7.2.2.45 long double std::hermitel (unsigned int *__n*, long double *__x*) [inline]

Return the Hermite polynomial $H_n(x)$ of nonnegative order n and `long double` argument `x`.

See also

hermite for details.

Definition at line 1016 of file specfun.h.

7.2.2.46 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::laguerre (unsigned int *__n*, _Tp *__x*) [inline]

Returns the Laguerre polynomial $L_n(x)$ of nonnegative degree `n` and real argument $x >= 0$.

The Laguerre polynomial is defined by:

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

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__n	The nonnegative order
__x	The argument `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 1083 of file specfun.h.

7.2.2.47 float std::laguerref (unsigned int *__n*, float *__x*) [inline]

Returns the Laguerre polynomial $L_n(x)$ of nonnegative degree `n` and `float` argument $x >= 0$.

See also

laguerre for more details.

Definition at line 1054 of file specfun.h.

7.2.2.48 long double std::laguerrel (unsigned int __n, long double __x) [inline]

Returns the Laguerre polynomial $L_n(x)$ of nonnegative degree `n` and `long double` argument $x >= 0$.

See also

laguerre for more details.

Definition at line 1064 of file specfun.h.

7.2.2.49 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::legendre (unsigned int __l, _Tp __x) [inline]

Return the Legendre polynomial $P_l(x)$ of nonnegative degree l and real argument $|x| <= 0$.

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

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

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__l	The degree $l >= 0$
__x	The argument `abs(__x)` <= 1

Exceptions

std::domain_error	if `abs(__x)` $>$ 1

Definition at line 1128 of file specfun.h.

7.2.2.50 float std::legendref (unsigned int __l, float __x) [inline]

Return the Legendre polynomial $P_l(x)$ of nonnegative degree l and `float` argument $|x| <= 0$.

See also

legendre for more details.

Definition at line 1098 of file specfun.h.

7.2.2.51 long double std::legendrel (unsigned int __l, long double __x) [inline]

Return the Legendre polynomial $P_l(x)$ of nonnegative degree l and `long double` argument $|x| <= 0$.

See also

legendre for more details.

Definition at line 1108 of file specfun.h.

7.2.2.52 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::riemann_zeta (_Tp __s) [inline]

Return the Riemann zeta function $\zeta(s)$ for real argument s.

The Riemann zeta function is defined by:

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

and

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{k=1}^{\infty} (-1)^{k-1} k^{-s} \text{ for } 0 <= s <= 1$$

For s $<$ 1 use the reflection formula:

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

Template Parameters

_Tp	The floating-point type of the argument `__s`.

Parameters

__s	The argument `s != 1`

Definition at line 1179 of file specfun.h.

7.2.2.53 float std::riemann_zetaf (float __s) [inline]

Return the Riemann zeta function $\zeta(s)$ for `float` argument s.

See also

riemann_zeta for more details.

Definition at line 1143 of file specfun.h.

7.2.2.54 long double std::riemann_zetal (long double __s) [inline]

Return the Riemann zeta function $\zeta(s)$ for `long double` argument s.

See also

riemann_zeta for more details.

Definition at line 1153 of file specfun.h.

7.2.2.55 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::sph_bessel (unsigned int __n, _Tp __x) [inline]

Return the spherical Bessel function $j_n(x)$ of nonnegative order n and real argument $x >= 0$.

The spherical Bessel function is defined by:

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

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__n	The integral order `n >= 0`
__x	The real argument `x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 1223 of file specfun.h.

7.2.2.56 float std::sph_besself (unsigned int __n, float __x) [inline]

Return the spherical Bessel function $j_n(x)$ of nonnegative order n and `float` argument $x >= 0$.

See also

sph_bessel for more details.

Definition at line 1194 of file specfun.h.

7.2.2.57 long double std::sph_bessell (unsigned int __n, long double __x) [inline]

Return the spherical Bessel function $j_n(x)$ of nonnegative order n and `long double` argument $x >= 0$.

See also

sph_bessel for more details.

Definition at line 1204 of file specfun.h.

7.2.2.58 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta) [inline]

Return the spherical Legendre function of nonnegative integral degree `l` and order `m` and real angle θ in radians.

The spherical Legendre function is defined by

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

Template Parameters

_Tp	The floating-point type of the angle `__theta`.

Parameters

__l	The order `__l >= 0`
__m	The degree `__m >= 0` and `__m <= __l`
__theta	The radian polar angle argument

Definition at line 1270 of file specfun.h.

7.2.2.59 float std::sph_legendref (unsigned int *__l*, unsigned int *__m*, float *__theta*) [inline]

Return the spherical Legendre function of nonnegative integral degree `l` and order `m` and float angle θ in radians.

See also

sph_legendre for details.

Definition at line 1238 of file specfun.h.

7.2.2.60 long double std::sph_legendrel (unsigned int *__l*, unsigned int *__m*, long double *__theta*) [inline]

Return the spherical Legendre function of nonnegative integral degree `l` and order `m` and `long double` angle θ in radians.

See also

sph_legendre for details.

Definition at line 1249 of file specfun.h.

7.2.2.61 template<typename _Tp > __gnu_cxx::__promote<_Tp>::__type std::sph_neumann (unsigned int *__n*, _Tp *__x*) [inline]

Return the spherical Neumann function of integral order $n >= 0$ and real argument $x >= 0$.

The spherical Neumann function is defined by

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

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__n	The integral order `n >= 0`

__x	The real argument `__x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 1314 of file specfun.h.

7.2.2.62 float std::sph_neumannf (unsigned int *__n*, float *__x*) [inline]

Return the spherical Neumann function of integral order $n >= 0$ and `float` argument $x >= 0$.

See also

sph_neumann for details.

Definition at line 1285 of file specfun.h.

7.2.2.63 long double std::sph_neumannl (unsigned int *__n*, long double *__x*) [inline]

Return the spherical Neumann function of integral order $n >= 0$ and `long double` $x >= 0$.

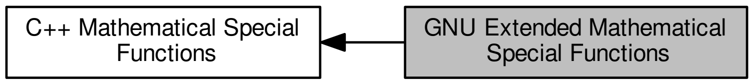
See also

sph_neumann for details.

Definition at line 1295 of file specfun.h.

7.3 GNU Extended Mathematical Special Functions

Collaboration diagram for GNU Extended Mathematical Special Functions:



Enumerations

- enum { __gnu_cxx::_GLIBCXX_JACOBI_SN, __gnu_cxx::_GLIBCXX_JACOBI_CN, __gnu_cxx::_GLIBCXX_JACOBI_DN }

Functions

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)
- float __gnu_cxx::airy_aif (float __x)
- long double __gnu_cxx::airy_ail (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)
- float __gnu_cxx::airy_bif (float __x)
- long double __gnu_cxx::airy_bil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::bernoulli (unsigned int __n)
- float __gnu_cxx::bernoullif (unsigned int __n)
- long double __gnu_cxx::bernoullil (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::bincoef (unsigned int __n, unsigned int __k)
- float __gnu_cxx::bincoeff (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::bincoefl (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_t (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_tf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_tl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_u (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_uf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_ul (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x)

- float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_wl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __w)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c (unsigned int __m, _Tp __w)
- float __gnu_cxx::clausen_cf (unsigned int __m, float __w)
- long double __gnu_cxx::clausen_cl (unsigned int __m, long double __w)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s (unsigned int __m, _Tp __w)
- float __gnu_cxx::clausen_sf (unsigned int __m, float __w)
- long double __gnu_cxx::clausen_sl (unsigned int __m, long double __w)
- float __gnu_cxx::clausenf (unsigned int __m, float __w)
- std::complex< float > __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __w)
- long double __gnu_cxx::clausenl (unsigned int __m, long double __w)
- std::complex< long double > __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __w)
- template<typename _Tk >
  __gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d (_Tk __k)
- float __gnu_cxx::comp_ellint_df (float __k)
- long double __gnu_cxx::comp_ellint_dl (long double __k)
- float __gnu_cxx::comp_ellint_rf (float __x, float __y)
- long double __gnu_cxx::comp_ellint_rf (long double __x, long double __y)
- template<typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf (_Tx __x, _Ty __y)
- float __gnu_cxx::comp_ellint_rg (float __x, float __y)
- long double __gnu_cxx::comp_ellint_rg (long double __x, long double __y)
- template<typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)
- template<typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)
- template<typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_2< _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg_lim (_Tpc __c, _Tp __x)
- float __gnu_cxx::conf_hyperg_limf (float __c, float __x)
- long double __gnu_cxx::conf_hyperg_liml (long double __c, long double __x)
- float __gnu_cxx::conf_hypergf (float __a, float __c, float __x)
- long double __gnu_cxx::conf_hypergl (long double __a, long double __c, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::coshint (_Tp __x)
- float __gnu_cxx::coshintf (float __x)
- long double __gnu_cxx::coshintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::cosint (_Tp __x)
- float __gnu_cxx::cosintf (float __x)
- long double __gnu_cxx::cosintl (long double __x)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)

- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)
- std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)
- std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)
- std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)
- std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)
- float __gnu_cxx::dawsonf (float __x)
- long double __gnu_cxx::dawsonl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)
- float __gnu_cxx::digammaf (float __z)
- long double __gnu_cxx::digammal (long double __z)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dilog (_Tp __x)
- float __gnu_cxx::dilogf (float __x)
- long double __gnu_cxx::dilogl (long double __x)
- template<typename _Tp >
  _Tp __gnu_cxx::dirichlet_beta (_Tp __s)
- float __gnu_cxx::dirichlet_betaf (float __s)
- long double __gnu_cxx::dirichlet_betal (long double __s)
- template<typename _Tp >
  _Tp __gnu_cxx::dirichlet_eta (_Tp __s)
- float __gnu_cxx::dirichlet_etaf (float __s)
- long double __gnu_cxx::dirichlet_etal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::double_factorial (int __n)
- float __gnu_cxx::double_factorialf (int __n)
- long double __gnu_cxx::double_factoriall (int __n)
- template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tk, _Tp, _Ta, _Tb > __gnu_cxx::ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float __gnu_cxx::ellint_celf (float __k_c, float __p, float __a, float __b)
- long double __gnu_cxx::ellint_cell (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::ellint_d (_Tk __k, _Tphi __phi)
- float __gnu_cxx::ellint_df (float __k, float __phi)
- long double __gnu_cxx::ellint_dl (long double __k, long double __phi)

- template<typename _Tp , typename _Tk >
  __gnu_cxx::__promote_num_t< _Tp, _Tk > __gnu_cxx::ellint_el1 (_Tp __x, _Tk __k_c)
- float __gnu_cxx::ellint_el1f (float __x, float __k_c)
- long double __gnu_cxx::ellint_el1l (long double __x, long double __k_c)
- template<typename _Tp , typename _Tk , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > __gnu_cxx::ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float __gnu_cxx::ellint_el2f (float __x, float __k_c, float __a, float __b)
- long double __gnu_cxx::ellint_el2l (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx , typename _Tk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > __gnu_cxx::ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p)
- float __gnu_cxx::ellint_el3f (float __x, float __k_c, float __p)
- long double __gnu_cxx::ellint_el3l (long double __x, long double __k_c, long double __p)
- template<typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::ellint_rc (_Tp __x, _Up __y)
- float __gnu_cxx::ellint_rcf (float __x, float __y)
- long double __gnu_cxx::ellint_rcl (long double __x, long double __y)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rd (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rdf (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rdl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rf (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rff (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rfl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rg (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rgf (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rgl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > __gnu_cxx::ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float __gnu_cxx::ellint_rjf (float __x, float __y, float __z, float __p)
- long double __gnu_cxx::ellint_rjl (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
  _Tp __gnu_cxx::ellnome (_Tp __k)
- float __gnu_cxx::ellnomef (float __k)
- long double __gnu_cxx::ellnomel (long double __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_e1 (_Tp __x)
- float __gnu_cxx::expint_e1f (float __x)
- long double __gnu_cxx::expint_e1l (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_en (unsigned int __n, _Tp __x)
- float __gnu_cxx::expint_enf (unsigned int __n, float __x)
- long double __gnu_cxx::expint_enl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::factorial (unsigned int __n)
- float __gnu_cxx::factorialf (unsigned int __n)
- long double __gnu_cxx::factoriall (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_c (_Tp __x)

- float __gnu_cxx::fresnel_cf (float __x)
- long double __gnu_cxx::fresnel_cl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_s (_Tp __x)
- float __gnu_cxx::fresnel_sf (float __x)
- long double __gnu_cxx::fresnel_sl (long double __x)
- template<typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_l (_Tn __n, _Tp __x)
- float __gnu_cxx::gamma_lf (float __n, float __x)
- long double __gnu_cxx::gamma_ll (long double __n, long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_p (_Ta __a, _Tp __x)
- float __gnu_cxx::gamma_pf (float __a, float __x)
- long double __gnu_cxx::gamma_pl (long double __a, long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_q (_Ta __a, _Tp __x)
- float __gnu_cxx::gamma_qf (float __a, float __x)
- long double __gnu_cxx::gamma_ql (long double __a, long double __x)
- template<typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_u (_Tn __n, _Tp __x)
- float __gnu_cxx::gamma_uf (float __n, float __x)
- long double __gnu_cxx::gamma_ul (long double __n, long double __x)
- template<typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)
- float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)
- long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi)
- float __gnu_cxx::heuman_lambdaf (float __k, float __phi)
- long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)
- template<typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta (_Tp __s, _Up __a)
- template<typename _Tp , typename _Up >
  std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)
- float __gnu_cxx::hurwitz_zetaf (float __s, float __a)
- long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)
- template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)
- long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)
- float __gnu_cxx::ibetacf (float __a, float __b, float __x)
- long double __gnu_cxx::ibetacl (long double __a, long double __b, long double __x)
- float __gnu_cxx::ibetaf (float __a, float __b, float __x)
- long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)

- template<typename _Talpha , typename _Tbeta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_cnf (float __k, float __u)
- long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_dnf (float __k, float __u)
- long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_snf (float __k, float __u)
- long double __gnu_cxx::jacobi_snl (long double __k, long double __u)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta (_Tk __k, _Tphi __phi)
- float __gnu_cxx::jacobi_zetaf (float __k, float __phi)
- long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)
- float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)
- long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)
- float __gnu_cxx::lbincoeff (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::lbincoefl (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)
- float __gnu_cxx::ldouble_factorialf (int __n)
- long double __gnu_cxx::ldouble_factoriall (int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q (unsigned int __n, _Tp __x)
- float __gnu_cxx::legendre_qf (unsigned int __n, float __x)
- long double __gnu_cxx::legendre_ql (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)
- float __gnu_cxx::lfactorialf (unsigned int __n)
- long double __gnu_cxx::lfactoriall (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint (_Tp __x)
- float __gnu_cxx::logintf (float __x)
- long double __gnu_cxx::logintl (long double __x)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l (_Tp __a, _Tn __n)
- float __gnu_cxx::lpochhammer_lf (float __a, float __n)
- long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_u (_Tp __a, _Tn __n)
- float __gnu_cxx::lpochhammer_uf (float __a, float __n)
- long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n)
- template<typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_num_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)
- float __gnu_cxx::owens_tf (float __h, float __a)

- long double __gnu_cxx::owens_tl (long double __h, long double __a)
- template< typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::pochhammer_l (_Tp __a, _Tn __n)
- float __gnu_cxx::pochhammer_lf (float __a, float __n)
- long double __gnu_cxx::pochhammer_ll (long double __a, long double __n)
- template< typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::pochhammer_u (_Tp __a, _Tn __n)
- float __gnu_cxx::pochhammer_uf (float __a, float __n)
- long double __gnu_cxx::pochhammer_ul (long double __a, long double __n)
- template< typename _Tp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)
- template< typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > __gnu_cxx::polylog (_Tp __s, std::complex< _Tp > __w)
- float __gnu_cxx::polylogf (float __s, float __w)
- std::complex< float > __gnu_cxx::polylogf (float __s, std::complex< float > __w)
- long double __gnu_cxx::polylogl (long double __s, long double __w)
- std::complex< long double > __gnu_cxx::polylogl (long double __s, std::complex< long double > __w)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::psi (_Tp __x)
- float __gnu_cxx::psif (float __x)
- long double __gnu_cxx::psil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::radpoly (unsigned int __n, unsigned int __m, _Tp __rho)
- float __gnu_cxx::radpolyf (unsigned int __n, unsigned int __m, float __rho)
- long double __gnu_cxx::radpolyl (unsigned int __n, unsigned int __m, long double __rho)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc_pi (_Tp __x)
- float __gnu_cxx::sinc_pif (float __x)
- long double __gnu_cxx::sinc_pil (long double __x)
- float __gnu_cxx::sincf (float __x)
- long double __gnu_cxx::sincl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc_pi (_Tp __x)
- float __gnu_cxx::sinhc_pif (float __x)
- long double __gnu_cxx::sinhc_pil (long double __x)
- float __gnu_cxx::sinhcf (float __x)
- long double __gnu_cxx::sinhcl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhint (_Tp __x)
- float __gnu_cxx::sinhintf (float __x)
- long double __gnu_cxx::sinhintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinint (_Tp __x)
- float __gnu_cxx::sinintf (float __x)
- long double __gnu_cxx::sinintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)

- float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)
- long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)
- float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)
- long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std↩
  ::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)
- std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)
- std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std↩
  ::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)
- std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)
- std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)
- template<typename _Ttheta , typename _Tphi >
  std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int ↩
  __l, int __m, _Ttheta __theta, _Tphi __phi)
- std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)
- std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_1f (float __nu, float __x)
- long double __gnu_cxx::theta_1l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_2f (float __nu, float __x)
- long double __gnu_cxx::theta_2l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_3f (float __nu, float __x)
- long double __gnu_cxx::theta_3l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_4f (float __nu, float __x)
- long double __gnu_cxx::theta_4l (long double __nu, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_c (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_cf (float __k, float __x)
- long double __gnu_cxx::theta_cl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_d (_Tpk __k, _Tp __x)

- float __gnu_cxx::theta_df (float __k, float __x)
- long double __gnu_cxx::theta_dl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_n (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_nf (float __k, float __x)
- long double __gnu_cxx::theta_nl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_s (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_sf (float __k, float __x)
- long double __gnu_cxx::theta_sl (long double __k, long double __x)
- template<typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Trho, _Tphi > __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)
- long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)

7.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

7.3.2 Enumeration Type Documentation

7.3.2.1 anonymous enum

Enumerator

_GLIBCXX_JACOBI_SN

_GLIBCXX_JACOBI_CN

_GLIBCXX_JACOBI_DN

Definition at line 1734 of file specfun.h.

7.3.3 Function Documentation

7.3.3.1 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::airy_ai (_Tp __x) [inline]

Return the Airy function $Ai(x)$ of real argument x.

The Airy function is defined by:

$$Ai(x) = \frac{1}{\pi}\int_0^\infty \cos\left(\frac{t^3}{3} + xt\right) dt$$

Template Parameters

_Tp	The real type of the argument

Parameters

__x	The argument

Definition at line 2676 of file specfun.h.

7.3.3.2 float __gnu_cxx::airy_aif (float __x) [inline]

Return the Airy function $Ai(x)$ for `float` argument x.

See also

airy_ai for details.

Definition at line 2641 of file specfun.h.

7.3.3.3 long double __gnu_cxx::airy_ail (long double __x) [inline]

Return the Airy function $Ai(x)$ for `long double` argument x.

See also

airy_ai for details.

Definition at line 2655 of file specfun.h.

7.3.3.4 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::airy_bi (_Tp __x) [inline]

Return the Airy function $Bi(x)$ of real argument x.

The Airy function is defined by:

$$Bi(x) = \frac{1}{\pi}\int_0^\infty \left[\exp\left(-\frac{t^3}{3} + xt\right) + \sin\left(\frac{t^3}{3} + xt\right)\right] dt$$

Template Parameters

_Tp	The real type of the argument

Parameters

__x	The argument

Definition at line 2728 of file specfun.h.

7.3.3.5 float __gnu_cxx::airy_bif (float __x) [inline]

Return the Airy function $Bi(x)$ for `float` argument x.

See also

airy_bi for details.

Definition at line 2692 of file specfun.h.

7.3.3.6 long double __gnu_cxx::airy_bil (long double __x) [inline]

Return the Airy function $Bi(x)$ for `long double` argument x.

See also

airy_bi for details.

Definition at line 2706 of file specfun.h.

7.3.3.7 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::bernoulli (unsigned int __n) [inline]

Return the Bernoulli number of integer order `n`.

The Bernoulli numbers are defined by

Parameters

__n	The order.

Definition at line 3704 of file specfun.h.

7.3.3.8 float __gnu_cxx::bernoullif (unsigned int __n) [inline]

Return the Bernoulli number of integer order `n` as a `float`.

See also

bernoulli for details.

Definition at line 3679 of file specfun.h.

7.3.3.9 long double __gnu_cxx::bernoullil (unsigned int __n) [inline]

Return the Bernoulli number of integer order `n` as a `long double`.

See also

bernoulli for details.

Definition at line 3689 of file specfun.h.

7.3.3.10 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::bincoef (unsigned int __n, unsigned int __k) [inline]

Definition at line 3644 of file specfun.h.

7.3.3.11 float __gnu_cxx::bincoeff (unsigned int __n, unsigned int __k) [inline]

Definition at line 3632 of file specfun.h.

7.3.3.12 long double __gnu_cxx::bincoefl (unsigned int *__n*, unsigned int *__k*) [inline]

Definition at line 3636 of file specfun.h.

7.3.3.13 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_t (unsigned int *__n*, _Tp *__x*) [inline]

Return the Chebyshev polynomial of the first kind $T_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the first kind is defined by:

$$T_n(x) = \cos(n\theta)$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 1935 of file specfun.h.

7.3.3.14 float __gnu_cxx::chebyshev_tf (unsigned int *__n*, float *__x*) [inline]

Return the Chebyshev polynomials of the first kind $T_n(x)$ of non-negative order n and `float` argument x.

See also

chebyshev_t for details.

Definition at line 1906 of file specfun.h.

7.3.3.15 long double __gnu_cxx::chebyshev_tl (unsigned int *__n*, long double *__x*) [inline]

Return the Chebyshev polynomials of the first kind $T_n(x)$ of non-negative order n and real argument x.

See also

chebyshev_t for details.

Definition at line 1916 of file specfun.h.

7.3.3.16 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_u (unsigned int *__n*, _Tp *__x*) [inline]

Return the Chebyshev polynomial of the second kind $U_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the second kind is defined by:

$$U_n(x) = \frac{\sin\left[(n+1)\theta\right]}{\sin(\theta)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 1979 of file specfun.h.

7.3.3.17 float __gnu_cxx::chebyshev_uf (unsigned int __n, float __x) [inline]

Return the Chebyshev polynomials of the second kind $U_n(x)$ of non-negative order n and `float` argument x.

See also

chebyshev_u for details.

Definition at line 1950 of file specfun.h.

7.3.3.18 long double __gnu_cxx::chebyshev_ul (unsigned int __n, long double __x) [inline]

Return the Chebyshev polynomials of the second kind $U_n(x)$ of non-negative order n and real argument x.

See also

chebyshev_u for details.

Definition at line 1960 of file specfun.h.

7.3.3.19 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x) [inline]

Return the Chebyshev polynomial of the third kind $V_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the third kind is defined by:

$$V_n(x) = \frac{\cos\left[\left(n + \frac{1}{2}\right)\theta\right]}{\cos\left(\frac{\theta}{2}\right)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 2024 of file specfun.h.

7.3.3.20 float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x) [inline]

Return the Chebyshev polynomials of the third kind $V_n(x)$ of non-negative order n and `float` argument x.

See also

chebyshev_v for details.

Definition at line 1994 of file specfun.h.

7.3.3.21 long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x) [inline]

Return the Chebyshev polynomials of the third kind $V_n(x)$ of non-negative order n and real argument x.

See also

chebyshev_v for details.

Definition at line 2004 of file specfun.h.

7.3.3.22 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x) [inline]

Return the Chebyshev polynomial of the fourth kind $W_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the fourth kind is defined by:

$$W_n(x) = \frac{\sin\left[\left(n + \frac{1}{2}\right)\theta\right]}{\sin\left(\frac{\theta}{2}\right)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 2069 of file specfun.h.

7.3.3.23 float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x) [inline]

Return the Chebyshev polynomials of the fourth kind $W_n(x)$ of non-negative order n and `float` argument x.

See also

chebyshev_w for details.

Definition at line 2039 of file specfun.h.

7.3.3.24 long double __gnu_cxx::chebyshev_wl (unsigned int __*n*, long double __*x*) [inline]

Return the Chebyshev polynomials of the fourth kind $W_n(x)$ of non-negative order n and real argument x.

See also

chebyshev_w for details.

Definition at line 2049 of file specfun.h.

7.3.3.25 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::clausen (unsigned int __*m*, _Tp __*w*) [inline]

Return the Clausen function of integer order m and complex argument w.

The Clausen function is defined by

Parameters

__*m*	
__*w*	The complex argument

Definition at line 4647 of file specfun.h.

7.3.3.26 template<typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __gnu_cxx::clausen (unsigned int __*m*, std::complex< _Tp > __*w*) [inline]

Definition at line 4668 of file specfun.h.

7.3.3.27 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::clausen_c (unsigned int __*m*, _Tp __*w*) [inline]

Return the Clausen cosine function of order m and real argument x.

The Clausen cosine function is defined by

Parameters

__*m*	
__*w*	

Definition at line 4608 of file specfun.h.

7.3.3.28 float __gnu_cxx::clausen_cf (unsigned int __*m*, float __*w*) [inline]

Return the Clausen cosine function of order m and real argument x.

See also

clausen_c for details.

Definition at line 4583 of file specfun.h.

7.3.3.29 long double __gnu_cxx::clausen_cl (unsigned int __m, long double __w) [inline]

Return the Clausen cosine function of order m and real argument x.

See also

clausen_c for details.

Definition at line 4592 of file specfun.h.

7.3.3.30 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::clausen_s (unsigned int __m, _Tp __w) [inline]

Return the Clausen sine function of order m and real argument x.

The Clausen sine function is defined by

Parameters

__m	
__w	

Definition at line 4569 of file specfun.h.

7.3.3.31 float __gnu_cxx::clausen_sf (unsigned int __m, float __w) [inline]

Return the Clausen sine function of order m and real argument x.

See also

clausen_s for details.

Definition at line 4544 of file specfun.h.

7.3.3.32 long double __gnu_cxx::clausen_sl (unsigned int __m, long double __w) [inline]

Return the Clausen sine function of order m and real argument x.

See also

clausen_s for details.

Definition at line 4553 of file specfun.h.

7.3.3.33 float __gnu_cxx::clausenf (unsigned int __m, float __w) [inline]

Return the Clausen function of integer order m and complex argument w.

See also

clausen for details.

Definition at line 4622 of file specfun.h.

7.3.3.34 std::complex<float> __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __w) [inline]

Definition at line 4656 of file specfun.h.

7.3.3.35 long double __gnu_cxx::clausenl (unsigned int __m, long double __w) [inline]

Return the Clausen function of integer order m and complex argument w.

See also

clausen for details.

Definition at line 4631 of file specfun.h.

7.3.3.36 std::complex<long double> __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __w) [inline]

Definition at line 4660 of file specfun.h.

7.3.3.37 template<typename _Tk > __gnu_cxx::__promote_num_t<_Tk> __gnu_cxx::comp_ellint_d (_Tk __k) [inline]

Return the complete Legendre elliptic integral $D(k)$ of real modulus k.

The complete Legendre elliptic integral D is defined by

$$D(k) = \int_0^{\pi/2} \frac{\sin^2\theta d\theta}{\sqrt{1 - k^2 sin2\theta}}$$

Template Parameters

_Tk	The type of the modulus `k`

Parameters

__k	The modulus `-1 <= __k <= +1`

Definition at line 3890 of file specfun.h.

7.3.3.38 float __gnu_cxx::comp_ellint_df (float __k) [inline]

Return the complete Legendre elliptic integral $D(k)$ of `float` modulus k.

See also

comp_ellint_d for details.

Definition at line 3863 of file specfun.h.

7.3.3.39 long double __gnu_cxx::comp_ellint_dl (long double __k) [inline]

Return the complete Legendre elliptic integral $D(k)$ of `long double` modulus k.

See also

comp_ellint_d for details.

Definition at line 3873 of file specfun.h.

7.3.3.40 float __gnu_cxx::comp_ellint_rf (float *__x*, float *__y*) [inline]

Return the complete Carlson elliptic function $R_F(x, y, z)$ for `float` arguments.

See also

comp_ellint_rf for details.

Definition at line 2849 of file specfun.h.

7.3.3.41 long double __gnu_cxx::comp_ellint_rf (long double *__x*, long double *__y*) [inline]

Return the complete Carlson elliptic function $R_F(x, y)$ for `long double` arguments.

See also

comp_ellint_rf for details.

Definition at line 2859 of file specfun.h.

7.3.3.42 template<typename _Tx , typename _Ty > __gnu_cxx::__promote_num_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rf (_Tx *__x*, _Ty *__y*) [inline]

Return the complete Carlson elliptic function $R_F(x, y)$ for real arguments.

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

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

Parameters

__x	The first argument.
__y	The second argument.

Definition at line 2877 of file specfun.h.

7.3.3.43 float __gnu_cxx::comp_ellint_rg (float *__x*, float *__y*) [inline]

Return the Carlson complementary elliptic function $R_G(x, y)$.

See also

comp_ellint_rg for details.

Definition at line 3082 of file specfun.h.

7.3.3.44 long double __gnu_cxx::comp_ellint_rg (long double __x, long double __y) [inline]

Return the Carlson complementary elliptic function $R_G(x, y)$.

See also

comp_ellint_rg for details.

Definition at line 3091 of file specfun.h.

7.3.3.45 template<typename _Tx , typename _Ty > __gnu_cxx::__promote_num_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y) [inline]

Return the complete Carlson elliptic function $R_G(x, y)$ for real arguments.

The complete Carlson elliptic function is defined by:

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

Parameters

__x	The first argument.
__y	The second argument.

Definition at line 3110 of file specfun.h.

7.3.3.46 template<typename _Tpa , typename _Tpc , typename _Tp > __gnu_cxx::__promote_3<_Tpa, _Tpc, _Tp>::__type __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x) [inline]

Return the confluent hypergeometric function ${}_1F_1(a; c; x)$ of real numeratorial parameter a, denominatorial parameter c, and argument x.

The confluent hypergeometric function is defined by

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

where the Pochhammer symbol is $(x)_k = (x)(x + 1)...(x + k - 1)$, $(x)_0 = 1$

Parameters

__a	The numeratorial parameter
__c	The denominatorial parameter
__x	The argument

Definition at line 1377 of file specfun.h.

7.3.3.47 template<typename _Tpc , typename _Tp > __gnu_cxx::__promote_2<_Tpc, _Tp>::__type __gnu_cxx::conf_hyperg_lim (_Tpc __c, _Tp __x) [inline]

Return the confluent hypergeometric limit function ${}_0F_1(; c; x)$ of real numeratorial parameter c and argument x.

The confluent hypergeometric limit function is defined by

$${}_0F_1(; c; x) = \sum_{n=0}^{\infty} \frac{x^n}{(c)_n n!}$$

where the Pochhammer symbol is $(x)_k = (x)(x+1)...(x+k-1)$, $(x)_0 = 1$

Parameters

__c	The denominatorial parameter
__x	The argument

Definition at line 1473 of file specfun.h.

7.3.3.48 float __gnu_cxx::conf_hyperg_limf (float *__c*, float *__x*) [inline]

Return the confluent hypergeometric limit function ${}_0F_1(;c;x)$ of `float` numeratorial parameter `c` and argument `x`.

See also

conf_hyperg_lim for details.

Definition at line 1444 of file specfun.h.

7.3.3.49 long double __gnu_cxx::conf_hyperg_liml (long double *__c*, long double *__x*) [inline]

Return the confluent hypergeometric limit function ${}_0F_1(;c;x)$ of `long double` numeratorial parameter `c` and argument `x`.

See also

conf_hyperg_lim for details.

Definition at line 1454 of file specfun.h.

7.3.3.50 float __gnu_cxx::conf_hypergf (float *__a*, float *__c*, float *__x*) [inline]

Return the confluent hypergeometric function ${}_1F_1(a;c;x)$ of `float` numeratorial parameter `a`, denominatorial parameter `c`, and argument `x`.

See also

conf_hyperg for details.

Definition at line 1345 of file specfun.h.

7.3.3.51 long double __gnu_cxx::conf_hypergl (long double *__a*, long double *__c*, long double *__x*) [inline]

Return the confluent hypergeometric function ${}_1F_1(a;c;x)$ of `long double` numeratorial parameter `a`, denominatorial parameter `c`, and argument `x`.

See also

conf_hyperg for details.

Definition at line 1356 of file specfun.h.

7.3.3.52 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::coshint (_Tp __x) [inline]

Return the hyperbolic cosine integral $Chi(x)$ of real argument x.

The hyperbolic cosine integral is defined by

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

Template Parameters

_Tp	The type of the real argument

Parameters

__x	The real argument

Definition at line 1727 of file specfun.h.

7.3.3.53 float __gnu_cxx::coshintf (float __x) [inline]

Return the hyperbolic cosine integral of `float` argument x.

See also

coshint for details.

Definition at line 1699 of file specfun.h.

7.3.3.54 long double __gnu_cxx::coshintl (long double __x) [inline]

Return the hyperbolic cosine integral $Chi(x)$ of `long double` argument x.

See also

coshint for details.

Definition at line 1709 of file specfun.h.

7.3.3.55 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::cosint (_Tp __x) [inline]

Return the cosine integral $Ci(x)$ of real argument x.

The cosine integral is defined by

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

Parameters

__x	The real upper integration limit

Definition at line 1644 of file specfun.h.

7.3.3.56 float __gnu_cxx::cosintf (float *__x*) [inline]

Return the cosine integral $Ci(x)$ of `float` argument x.

See also

cosint for details.

Definition at line 1618 of file specfun.h.

7.3.3.57 long double __gnu_cxx::cosintl (long double *__x*) [inline]

Return the cosine integral $Ci(x)$ of `long double` argument x.

See also

cosint for details.

Definition at line 1628 of file specfun.h.

7.3.3.58 template<typename _Tpnu , typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_1 (_Tpnu *__nu,* _Tp *__z*) [inline]

Return the cylindrical Hankel function of the first kind $H^{(1)}_n(x)$ of real order ν and argument $x >= 0$.

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

$$H^{(1)}_\nu(x) = \left(\frac{\pi}{2x}\right)^{1/2}\left[J_{n+1/2}(x) + iN_{n+1/2}(x)\right]$$

where $J_\nu(x)$ and $N_\nu(x)$ are the cylindrical Bessel and Neumann functions respectively (

See also

cyl_bessel and cyl_neumann).

Template Parameters

_Tp	The real type of the argument

Parameters

__nu	The real order
__z	The real argument

Definition at line 2378 of file specfun.h.

7.3.3.59 template<typename _Tpnu , typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x) [inline]

Return the complex cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$ of complex order ν and argument x.

The cylindrical Hankel function of the first kind is defined by

$$H^{(1)}_\nu(x) = J_\nu(x) + iN_\nu(x)$$

Template Parameters

_Tpnu	The complex type of the order
_Tp	The complex type of the argument

Parameters

__nu	The complex order
__x	The complex argument

Definition at line 4167 of file specfun.h.

7.3.3.60 std::complex<float> __gnu_cxx::cyl_hankel_1f (float __nu, float __z) [inline]

Return the cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$ of `float` order ν and argument $x >= 0$.

See also

cyl_hankel_1 for details.

Definition at line 2345 of file specfun.h.

7.3.3.61 std::complex<float> __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x) [inline]

Return the complex cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$ of `std::complex<float>` order ν and argument x.

See also

cyl_hankel_1 for more details.

Definition at line 4136 of file specfun.h.

7.3.3.62 std::complex<long double> __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z) [inline]

Return the cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$ of `long double` order ν and argument $x >= 0$.

See also

cyl_hankel_1 for details.

Definition at line 2356 of file specfun.h.

7.3.3.63 std::complex<long double> __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x) [inline]

Return the complex cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$ of `std::complex<long double>` order ν and argument x.

See also

cyl_hankel_1 for more details.

Definition at line 4147 of file specfun.h.

7.3.3.64 template<typename _Tpnu , typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z) [inline]

Return the cylindrical Hankel function of the second kind $H^{(2)}_n(x)$ of real order ν and argument $x >= 0$.

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

$$H^{(2)}_\nu(x) = \left(\frac{\pi}{2x}\right)^{1/2}\left[J_{n+1/2}(x) - iN_{n+1/2}(x)\right]$$

where $J_\nu(x)$ and $N_\nu(x)$ are the cylindrical Bessel and Neumann functions respectively (

See also

cyl_bessel and cyl_neumann).

Template Parameters

_Tp	The real type of the argument

Parameters

__nu	The real order
__z	The real argument

Definition at line 2427 of file specfun.h.

7.3.3.65 template<typename _Tpnu , typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x) [inline]

Return the complex cylindrical Hankel function of the second kind $H^{(2)}_\nu(x)$ of complex order ν and argument x.

The cylindrical Hankel function of the second kind is defined by

$$H^{(2)}_\nu(x) = J_\nu(x) - iN_\nu(x)$$

Template Parameters

_Tpnu	The complex type of the order

_Tp	The complex type of the argument

Parameters

__nu	The complex order
__x	The complex argument

Definition at line 4214 of file specfun.h.

7.3.3.66 std::complex<float> __gnu_cxx::cyl_hankel_2f (float __nu, float __z) [inline]

Return the cylindrical Hankel function of the second kind $H^{(2)}_\nu(x)$ of `float` order ν and argument $x >= 0$.

See also

cyl_hankel_2 for details.

Definition at line 2394 of file specfun.h.

7.3.3.67 std::complex<float> __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x) [inline]

Return the complex cylindrical Hankel function of the second kind $H^{(2)}_\nu(x)$ of `std::complex<float>` order ν and argument x.

See also

cyl_hankel_2 for more details.

Definition at line 4183 of file specfun.h.

7.3.3.68 std::complex<long double> __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z) [inline]

Return the cylindrical Hankel function of the second kind $H^{(2)}_\nu(x)$ of `long double` order ν and argument $x >= 0$.

See also

cyl_hankel_2 for details.

Definition at line 2405 of file specfun.h.

7.3.3.69 std::complex<long double> __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x) [inline]

Return the complex cylindrical Hankel function of the second kind $H^{(2)}_\nu(x)$ of `std::complex<long double>` order ν and argument x.

See also

cyl_hankel_2 for more details.

Definition at line 4194 of file specfun.h.

7.3.3.70 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::dawson (_Tp __x) [inline]

Return the Dawson integral, $F(x)$, for real argument `x`.

The Dawson integral is defined by:

$$F(x) = e^{-x^2} \int_0^x e^{y^2} dy$$

and it's derivative is:

$$F'(x) = 1 - 2xF(x)$$

Parameters

__x	The argument $-inf < x < inf$.

Definition at line 3420 of file specfun.h.

7.3.3.71 float __gnu_cxx::dawsonf (float __x) [inline]

Return the Dawson integral, $F(x)$, for `float` argument `x`.

See also

dawson for details.

Definition at line 3392 of file specfun.h.

7.3.3.72 long double __gnu_cxx::dawsonl (long double __x) [inline]

Return the Dawson integral, $F(x)$, for `long double` argument `x`.

See also

dawson for details.

Definition at line 3401 of file specfun.h.

7.3.3.73 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::digamma (_Tp __z) [inline]

Definition at line 2793 of file specfun.h.

7.3.3.74 float __gnu_cxx::digammaf (float __z) [inline]

Definition at line 2781 of file specfun.h.

7.3.3.75 long double __gnu_cxx::digammal (long double __z) [inline]

Definition at line 2785 of file specfun.h.

7.3.3.76 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::dilog (_Tp __x) [inline]

Return the dilogarithm function $\psi(z)$ for real argument.

The dilogarithm is defined by:

$$Li_2(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^2}$$

Parameters

__x	The argument.

Definition at line 2834 of file specfun.h.

7.3.3.77 float __gnu_cxx::dilogf (float __x) [inline]

Return the dilogarithm function $\psi(z)$ for `float` argument.

See also

dilog for details.

Definition at line 2808 of file specfun.h.

7.3.3.78 long double __gnu_cxx::dilogl (long double __x) [inline]

Return the dilogarithm function $\psi(z)$ for `long double` argument.

See also

dilog for details.

Definition at line 2818 of file specfun.h.

7.3.3.79 template<typename _Tp > _Tp __gnu_cxx::dirichlet_beta (_Tp __s) [inline]

Return the Dirichlet beta function of real argument s.

The Dirichlet beta function is defined by:

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

An important reflection formula is:

$$\beta(1-s) = \left(\frac{2}{\pi}\right)^s \sin(\frac{\pi s}{2})\Gamma(s)\beta(s)$$

Parameters

__s	

Definition at line 4530 of file specfun.h.

7.3.3.80 float __gnu_cxx::dirichlet_betaf (float __s) [inline]

Return the Dirichlet beta function of real argument s.

See also

dirichlet_beta for details.

Definition at line 4501 of file specfun.h.

7.3.3.81 long double __gnu_cxx::dirichlet_betal (long double __s) [inline]

Return the Dirichlet beta function of real argument s.

See also

dirichlet_beta for details.

Definition at line 4510 of file specfun.h.

7.3.3.82 template<typename _Tp > _Tp __gnu_cxx::dirichlet_eta (_Tp __s) [inline]

Return the Dirichlet eta function of real argument s.

The Dirichlet eta function is defined by

$$\eta(s) = \sum_{k=1}^{\infty} \frac{(-1)^k}{k^s} = \left(1 - 2^{1-s}\right) \zeta(s)$$

An important reflection formula is:

$$\eta(-s) = 2 \frac{1 - 2^{-s-1}}{1 - 2^{-s}} \pi^{-s-1} s \sin(\frac{\pi s}{2}) \Gamma(s) \eta(s+1)$$

Parameters

__s	

Definition at line 4487 of file specfun.h.

7.3.3.83 float __gnu_cxx::dirichlet_etaf (float __s) [inline]

Return the Dirichlet eta function of real argument s.

See also

dirichlet_eta for details.

Definition at line 4457 of file specfun.h.

7.3.3.84 long double __gnu_cxx::dirichlet_etal (long double __s) [inline]

Return the Dirichlet eta function of real argument s.

See also

dirichlet_eta for details.

Definition at line 4466 of file specfun.h.

7.3.3.85 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::double_factorial (int __n) [inline]

Definition at line 3581 of file specfun.h.

7.3.3.86 float __gnu_cxx::double_factorialf (int __n) [inline]

Definition at line 3569 of file specfun.h.

7.3.3.87 long double __gnu_cxx::double_factoriall (int __n) [inline]

Definition at line 3573 of file specfun.h.

7.3.3.88 template<typename _Tk , typename _Tp , typename _Ta , typename _Tb > __gnu_cxx::__promote_num_t<_Tk, _Tp, _Ta, _Tb> __gnu_cxx::ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b) [inline]

Return the Bulirsch complete elliptic integral $cel(k_c, p, a, b)$ of real complementary modulus k_c, and parameters p, a, and b.

The Bulirsch complete elliptic integral is defined by

$$cel(k_c, p, a, b) = \int_0^{\pi/2} \frac{a\cos^2\theta + b\sin^2\theta}{cos^2\theta + p\sin^2\theta} \frac{d\theta}{\sqrt{cos^2\theta + k_c^2\sin^2\theta}}$$

Parameters

__k_c	The complementary modulus $k_c = \sqrt{1-k^2}$
__p	The parameter
__a	The parameter
__b	The parameter

Definition at line 4120 of file specfun.h.

7.3.3.89 float __gnu_cxx::ellint_celf (float __k_c, float __p, float __a, float __b) [inline]

Return the Bulirsch complete elliptic integral $cel(k_c, p, a, b)$ of real complementary modulus k_c, and parameters p, a, and b.

See also

ellint_cel for details.

Definition at line 4088 of file specfun.h.

7.3.3.90 long double __gnu_cxx::ellint_cell (long double *__k_c*, long double *__p*, long double *__a*, long double *__b*) [inline]

Return the Bulirsch complete elliptic integral $cel(k_c, p, a, b)$.

See also

ellint_cel for details.

Definition at line 4097 of file specfun.h.

7.3.3.91 template<typename _Tk , typename _Tphi > __gnu_cxx::__promote_num_t<_Tk, _Tphi> __gnu_cxx::ellint_d (_Tk *__k*, _Tphi *__phi*) [inline]

Return the incomplete Legendre elliptic integral $D(k, \phi)$ of real modulus k and angular limit ϕ.

The Legendre elliptic integral D is defined by

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

Parameters

__k	The modulus `-1 <= __k <= +1`
__phi	The angle

Definition at line 3933 of file specfun.h.

7.3.3.92 float __gnu_cxx::ellint_df (float *__k*, float *__phi*) [inline]

Return the incomplete Legendre elliptic integral $D(k, \phi)$ of `float` modulus k and angular limit ϕ.

See also

ellint_d for details.

Definition at line 3905 of file specfun.h.

7.3.3.93 long double __gnu_cxx::ellint_dl (long double *__k*, long double *__phi*) [inline]

Return the incomplete Legendre elliptic integral $D(k, \phi)$ of `long double` modulus k and angular limit ϕ.

See also

ellint_d for details.

Definition at line 3915 of file specfun.h.

7.3.3.94 template<typename _Tp , typename _Tk > __gnu_cxx::__promote_num_t<_Tp, _Tk> __gnu_cxx::ellint_el1 (_Tp *__x*, _Tk *__k_c*) [inline]

Return the Bulirsch elliptic integral $el1(x, k_c)$ of the first kind of real tangent limit x and complementary modulus k_c.

The Bulirsch elliptic integral of the first kind is defined by

$$el1(x, k_c) = el2(x, k_c, 1, 1) = \int_0^{\arctan x} \frac{1 + 1\tan^2\theta}{\sqrt{(1+\tan^2\theta)(1+k_c^2\tan^2\theta)}} d\theta$$

Parameters

__x	The tangent of the angular integration limit
__k_c	The complementary modulus $k_c = \sqrt{1-k^2}$

Definition at line 3979 of file specfun.h.

7.3.3.95 float __gnu_cxx::ellint_el1f (float __x, float __k_c) [inline]

Return the Bulirsch elliptic integral $el1(x, k_c)$ of the first kind of `float` tangent limit x and complementary modulus k_c.

See also

ellint_el1 for details.

Definition at line 3949 of file specfun.h.

7.3.3.96 long double __gnu_cxx::ellint_el1l (long double __x, long double __k_c) [inline]

Return the Bulirsch elliptic integral $el1(x, k_c)$ of the first kind of real tangent limit x and complementary modulus k_c.

See also

ellint_el1 for details.

Definition at line 3960 of file specfun.h.

7.3.3.97 template<typename _Tp , typename _Tk , typename _Ta , typename _Tb > __gnu_cxx::__promote_num_t<_Tp, _Tk, _Ta, _Tb> __gnu_cxx::ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b) [inline]

Return the Bulirsch elliptic integral of the second kind $el2(x, k_c, a, b)$.

The Bulirsch elliptic integral of the second kind is defined by

$$el2(x, k_c, a, b) = \int_0^{\arctan x} \frac{a + b\tan^2\theta}{\sqrt{(1+\tan^2\theta)(1+k_c^2\tan^2\theta)}} d\theta$$

Parameters

__x	The tangent of the angular integration limit
__k_c	The complementary modulus $k_c = \sqrt{1-k^2}$
__a	The parameter
__b	The parameter

Definition at line 4025 of file specfun.h.

7.3.3.98 float __gnu_cxx::ellint_el2f (float __x, float __k_c, float __a, float __b) [inline]

Return the Bulirsch elliptic integral of the second kind $el2(x, k_c, a, b)$.

See also

ellint_el2 for details.

Definition at line 3994 of file specfun.h.

7.3.3.99 long double __gnu_cxx::ellint_el2l (long double __x, long double __k_c, long double __a, long double __b) [inline]

Return the Bulirsch elliptic integral of the second kind $el2(x, k_c, a, b)$.

See also

ellint_el2 for details.

Definition at line 4004 of file specfun.h.

7.3.3.100 template<typename _Tx , typename _Tk , typename _Tp > __gnu_cxx::__promote_num_t<_Tx, _Tk, _Tp> __gnu_cxx::ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p) [inline]

Return the Bulirsch elliptic integral of the third kind $el3(x, k_c, p)$ of real tangent limit x, complementary modulus k_c, and parameter p.

The Bulirsch elliptic integral of the third kind is defined by

$$el3(x, k_c, p) = \int_0^{\arctan x} \frac{d\theta}{(cos^2\theta + p\sin^2\theta)\sqrt{cos^2\theta + k_c^2\sin^2\theta}}$$

Parameters

__x	The tangent of the angular integration limit
__k_c	The complementary modulus $k_c = \sqrt{1 - k^2}$
__p	The paramenter

Definition at line 4072 of file specfun.h.

7.3.3.101 float __gnu_cxx::ellint_el3f (float __x, float __k_c, float __p) [inline]

Return the Bulirsch elliptic integral of the third kind $el3(x, k_c, p)$ of `float` tangent limit x, complementary modulus k_c, and parameter p.

See also

ellint_el3 for details.

Definition at line 4041 of file specfun.h.

7.3.3.102 long double __gnu_cxx::ellint_el3l (long double __x, long double __k_c, long double __p) [inline]

Return the Bulirsch elliptic integral of the third kind $el3(x, k_c, p)$ of `long double` tangent limit x, complementary modulus k_c, and parameter p.

See also

ellint_el3 for details.

Definition at line 4052 of file specfun.h.

7.3.3.103 template<typename _Tp , typename _Up > __gnu_cxx::__promote_num_t<_Tp, _Up> __gnu_cxx::ellint_rc (_Tp __x, _Up __y) [inline]

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.

Definition at line 2969 of file specfun.h.

7.3.3.104 float __gnu_cxx::ellint_rcf (float __x, float __y) [inline]

Return the Carlson elliptic function $R_C(x, y)$.

See also

ellint_rc for details.

Definition at line 2935 of file specfun.h.

7.3.3.105 long double __gnu_cxx::ellint_rcl (long double __x, long double __y) [inline]

Return the Carlson elliptic function $R_C(x, y)$.

See also

ellint_rc for details.

Definition at line 2944 of file specfun.h.

7.3.3.106 template<typename _Tp , typename _Up , typename _Vp > __gnu_cxx::__promote_num_t<_Tp, _Up, _Vp> __gnu_cxx::ellint_rd (_Tp __x, _Up __y, _Vp __z) [inline]

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.

Definition at line 3068 of file specfun.h.

7.3.3.107 float __gnu_cxx::ellint_rdf (float __x, float __y, float __z) [inline]

Return the Carlson elliptic function $R_D(x,y,z)$.

See also

ellint_rd for details.

Definition at line 3032 of file specfun.h.

7.3.3.108 long double __gnu_cxx::ellint_rdl (long double __x, long double __y, long double __z) [inline]

Return the Carlson elliptic function $R_D(x,y,z)$.

See also

ellint_rd for details.

Definition at line 3041 of file specfun.h.

7.3.3.109 template<typename _Tp , typename _Up , typename _Vp > __gnu_cxx::__promote_num_t<_Tp, _Up, _Vp> __gnu_cxx::ellint_rf (_Tp __x, _Up __y, _Vp __z) [inline]

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

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.

Definition at line 2921 of file specfun.h.

7.3.3.110 float __gnu_cxx::ellint_rff (float __x, float __y, float __z) [inline]

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

See also

ellint_rf for details.

Definition at line 2892 of file specfun.h.

7.3.3.111 long double __gnu_cxx::ellint_rfl (long double __x, long double __y, long double __z) [inline]

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind for `long double` arguments.

See also

ellint_rf for details.

Definition at line 2902 of file specfun.h.

7.3.3.112 template<typename _Tp , typename _Up , typename _Vp > __gnu_cxx::__promote_num_t<_Tp, _Up, _Vp> __gnu_cxx::ellint_rg (_Tp __x, _Up __y, _Vp __z) [inline]

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.

Definition at line 3159 of file specfun.h.

7.3.3.113 float __gnu_cxx::ellint_rgf (float __x, float __y, float __z) [inline]

Return the Carlson elliptic function $R_G(x, y)$.

See also

ellint_rg for details.

Definition at line 3124 of file specfun.h.

7.3.3.114 long double __gnu_cxx::ellint_rgl (long double __x, long double __y, long double __z) [inline]

Return the Carlson elliptic function $R_G(x, y)$.

See also

ellint_rg for details.

Definition at line 3133 of file specfun.h.

7.3.3.115 template<typename _Tp , typename _Up , typename _Vp , typename _Wp > __gnu_cxx::__promote_num_t<_Tp, _Up, _Vp, _Wp> __gnu_cxx::ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p) [inline]

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.

Definition at line 3018 of file specfun.h.

7.3.3.116 float __gnu_cxx::ellint_rjf (float __x, float __y, float __z, float __p) [inline]

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

See also

ellint_rj for details.

Definition at line 2983 of file specfun.h.

7.3.3.117 long double __gnu_cxx::ellint_rjl (long double __x, long double __y, long double __z, long double __p) [inline]

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

See also

ellint_rj for details.

Definition at line 2992 of file specfun.h.

7.3.3.118 template<typename _Tp > _Tp __gnu_cxx::ellnome (_Tp __k) [inline]

Return the elliptic nome function $q(k)$ of modulus k.

The elliptic nome function is defined by

$$q(k) =$$

Parameters

__k	The modulus $-1 <= k <= +1$

Definition at line 4880 of file specfun.h.

7.3.3.119 float __gnu_cxx::ellnomef (float __k) [inline]

Return the elliptic nome function $q(k)$ of modulus k.

See also

ellnome for details.

Definition at line 4855 of file specfun.h.

7.3.3.120 long double __gnu_cxx::ellnomel (long double __k) [inline]

Return the elliptic nome function $q(k)$ of `long double` modulus k.

See also

ellnome for details.

Definition at line 4865 of file specfun.h.

7.3.3.121 template< typename _Tp > __gnu_cxx::__promote_num_t< _Tp> __gnu_cxx::expint_e1 (_Tp __x) [inline]

Definition at line 3436 of file specfun.h.

7.3.3.122 float __gnu_cxx::expint_e1f (float __x) [inline]

Definition at line 3427 of file specfun.h.

7.3.3.123 long double __gnu_cxx::expint_e1l (long double __x) [inline]

Definition at line 3431 of file specfun.h.

7.3.3.124 template< typename _Tp > __gnu_cxx::__promote_num_t< _Tp> __gnu_cxx::expint_en (unsigned int __n, _Tp __x) [inline]

Definition at line 3455 of file specfun.h.

7.3.3.125 float __gnu_cxx::expint_enf (unsigned int __n, float __x) [inline]

Definition at line 3443 of file specfun.h.

7.3.3.126 long double __gnu_cxx::expint_enl (unsigned int __n, long double __x) [inline]

Definition at line 3447 of file specfun.h.

7.3.3.127 template< typename _Tp > __gnu_cxx::__promote_num_t< _Tp> __gnu_cxx::factorial (unsigned int __n) [inline]

Definition at line 3560 of file specfun.h.

7.3.3.128 float __gnu_cxx::factorialf (unsigned int __n) [inline]

Definition at line 3548 of file specfun.h.

7.3.3.129 long double __gnu_cxx::factoriall (unsigned int __n) [inline]

Definition at line 3552 of file specfun.h.

7.3.3.130 template< typename _Tp > __gnu_cxx::__promote_num_t< _Tp> __gnu_cxx::fresnel_c (_Tp __x) [inline]

Return the Fresnel cosine integral of argument x.

The Fresnel cosine integral is defined by

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

Parameters

__x	The argument

Definition at line 3378 of file specfun.h.

7.3.3.131 float __gnu_cxx::fresnel_cf (float __x) [inline]

Definition at line 3359 of file specfun.h.

7.3.3.132 long double __gnu_cxx::fresnel_cl (long double __x) [inline]

Definition at line 3363 of file specfun.h.

7.3.3.133 template< typename _Tp > __gnu_cxx::__promote_num_t< _Tp> __gnu_cxx::fresnel_s (_Tp __x) [inline]

Return the Fresnel sine integral of argument x.

The Fresnel sine integral is defined by

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

Parameters

__x	The argument

Definition at line 3350 of file specfun.h.

7.3.3.134 float __gnu_cxx::fresnel_sf (float __x) [inline]

Definition at line 3331 of file specfun.h.

7.3.3.135 long double __gnu_cxx::fresnel_sl (long double __x) [inline]

Definition at line 3335 of file specfun.h.

7.3.3.136 template< typename _Tn , typename _Tp > __gnu_cxx::__promote_num_t< _Tn, _Tp> __gnu_cxx::gamma_l (_Tn __n, _Tp __x) [inline]

Definition at line 2772 of file specfun.h.

7.3.3.137 float __gnu_cxx::gamma_lf (float __n, float __x) [inline]

Definition at line 2760 of file specfun.h.

7.3.3.138 long double __gnu_cxx::gamma_ll (long double __n, long double __x) [inline]

Definition at line 2764 of file specfun.h.

7.3.3.139 template<typename _Ta , typename _Tp > __gnu_cxx::__promote_num_t<_Ta, _Tp> __gnu_cxx::gamma_p (_Ta __a, _Tp __x) [inline]

Definition at line 3758 of file specfun.h.

7.3.3.140 float __gnu_cxx::gamma_pf (float __a, float __x) [inline]

Definition at line 3746 of file specfun.h.

7.3.3.141 long double __gnu_cxx::gamma_pl (long double __a, long double __x) [inline]

Definition at line 3750 of file specfun.h.

7.3.3.142 template<typename _Ta , typename _Tp > __gnu_cxx::__promote_num_t<_Ta, _Tp> __gnu_cxx::gamma_q (_Ta __a, _Tp __x) [inline]

Definition at line 3779 of file specfun.h.

7.3.3.143 float __gnu_cxx::gamma_qf (float __a, float __x) [inline]

Definition at line 3767 of file specfun.h.

7.3.3.144 long double __gnu_cxx::gamma_ql (long double __a, long double __x) [inline]

Definition at line 3771 of file specfun.h.

7.3.3.145 template<typename _Tn , typename _Tp > __gnu_cxx::__promote_num_t<_Tn, _Tp> __gnu_cxx::gamma_u (_Tn __n, _Tp __x) [inline]

Definition at line 2751 of file specfun.h.

7.3.3.146 float __gnu_cxx::gamma_uf (float __n, float __x) [inline]

Definition at line 2739 of file specfun.h.

7.3.3.147 long double __gnu_cxx::gamma_ul (long double __n, long double __x) [inline]

Definition at line 2743 of file specfun.h.

7.3.3.148 template<typename _Talpha , typename _Tp > __gnu_cxx::__promote_num_t<_Talpha, _Tp> __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x) [inline]

Return the Gegenbauer polynomial $C_n^{\alpha}(x)$ of degree n and real order $\alpha > -1/2, \alpha \neq 0$ and argument x.

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^{\alpha}(x) = \frac{1}{n}\left[2x(n+\alpha-1)C_{n-1}^{\alpha}(x) - (n+2\alpha-2)C_{n-2}^{\alpha}(x)\right]$$

and $C_0^{\alpha}(x) = 1$, $C_1^{\alpha}(x) = 2\alpha x$.

Template Parameters

_Talpha	The real type of the order
_Tp	The real type of the argument

Parameters

__n	The non-negative integral degree
__alpha	The real order
__x	The real argument

Definition at line 2177 of file specfun.h.

7.3.3.149 float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x) [inline]

Return the Gegenbauer polynomial $C_n^{\alpha}(x)$ of degree n and `float` order $\alpha > -1/2, \alpha \neq 0$ and argument x.

See also

gegenbauer for details.

Definition at line 2144 of file specfun.h.

7.3.3.150 long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x) [inline]

Return the Gegenbauer polynomial $C_n^{\alpha}(x)$ of degree n and `long double` order $\alpha > -1/2, \alpha \neq 0$ and argument x.

See also

gegenbauer for details.

Definition at line 2155 of file specfun.h.

7.3.3.151 template<typename _Tk , typename _Tphi > __gnu_cxx::__promote_num_t<_Tk, _Tphi> __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi) [inline]

Return the Heuman lambda function $\Lambda(k,\phi)$ of modulus k and angular limit ϕ.

The complete Heuman lambda function is defined by

$$\Lambda(k,\phi) = \frac{F(1-m,\phi)}{K(1-m)} + \frac{2}{\pi}K(m)Z(1-m,\phi)$$

where $m = k^2$, $K(k)$ is the complete elliptic function of the first kind, and $Z(k,phi)$ is the Jacobi zeta function.

Template Parameters

_Tk	the floating-point type of the modulus
_Tphi	the floating-point type of the angular limit argument

Parameters

__k	The modulus
__phi	The angle

Definition at line 3848 of file specfun.h.

7.3.3.152 float __gnu_cxx::heuman_lambdaf (float *__k*, float *__phi*) [inline]

Definition at line 3822 of file specfun.h.

7.3.3.153 long double __gnu_cxx::heuman_lambdal (long double *__k*, long double *__phi*) [inline]

Definition at line 3826 of file specfun.h.

7.3.3.154 template<typename _Tp , typename _Up > __gnu_cxx::__promote_num_t<_Tp, _Up> __gnu_cxx::hurwitz_zeta (_Tp *__s*, _Up *__a*) [inline]

Return the Hurwitz zeta function of real argument `s`, and parameter `a`.

The the Hurwitz zeta function is defined by

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

Parameters

__s	The argument
__a	The parameter

Definition at line 3200 of file specfun.h.

7.3.3.155 template<typename _Tp , typename _Up > std::complex<_Tp> __gnu_cxx::hurwitz_zeta (_Tp *__s*, std::complex< _Up > *__a*)

Return the Hurwitz zeta function of real argument `s`, and complex parameter `a`.

See also

hurwitz_zeta for details.

Definition at line 3214 of file specfun.h.

7.3.3.156 float __gnu_cxx::hurwitz_zetaf (float *__s*, float *__a*) [inline]

Return the Hurwitz zeta function of `float` argument `s`, and parameter `a`.

See also

hurwitz_zeta for details.

Definition at line 3174 of file specfun.h.

7.3.3.157 long double __gnu_cxx::hurwitz_zetal (long double *__s*, long double *__a*) [inline]

Return the Hurwitz zeta function of `long double` argument `s`, and parameter `a`.

See also

hurwitz_zeta for details.

Definition at line 3184 of file specfun.h.

7.3.3.158 template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp > __gnu_cxx::__promote_4<_Tpa, _Tpb, _Tpc, _Tp>::__type __gnu_cxx::hyperg (_Tpa *__a*, _Tpb *__b*, _Tpc *__c*, _Tp *__x*) [inline]

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ of real numeratorial parameters `a` and `b`, denominatorial parameter `c`, and argument `x`.

The hypergeometric function is defined by

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

where the Pochhammer symbol is $(x)_k = (x)(x+1)...(x+k-1)$, $(x)_0 = 1$

Parameters

__a	The first numeratorial parameter
__b	The second numeratorial parameter
__c	The denominatorial parameter
__x	The argument

Definition at line 1426 of file specfun.h.

7.3.3.159 float __gnu_cxx::hypergf (float *__a*, float *__b*, float *__c*, float *__x*) [inline]

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ of @ float numeratorial parameters `a` and `b`, denominatorial parameter `c`, and argument `x`.

See also

hyperg for details.

Definition at line 1393 of file specfun.h.

7.3.3.160 long double __gnu_cxx::hypergl (long double *__a*, long double *__b*, long double *__c*, long double *__x*) [inline]

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ of `long double` numeratorial parameters `a` and `b`, denominatorial parameter `c`, and argument `x`.

See also

hyperg for details.

Definition at line 1404 of file specfun.h.

7.3.3.161 template<typename _Ta , typename _Tb , typename _Tp > __gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp> __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x) [inline]

Return the regularized incomplete beta function of parameters `a`, `b`, and argument `x`.

The regularized incomplete beta function is defined by

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)}$$

where

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

is the non-regularized beta function and $B(a,b)$ is the usual beta function.

Parameters

__a	The first parameter
__b	The second parameter
__x	The argument

Definition at line 3291 of file specfun.h.

7.3.3.162 template<typename _Ta , typename _Tb , typename _Tp > __gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp> __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x) [inline]

Return the regularized complementary incomplete beta function of parameters `a`, `b`, and argument `x`.

The regularized complementary incomplete beta function is defined by

$$I_x(a,b) = I_x(a,b)$$

Parameters

__a	The parameter
__b	The parameter
__x	The argument

Definition at line 3322 of file specfun.h.

7.3.3.163 float __gnu_cxx::ibetacf (float __a, float __b, float __x) [inline]

Definition at line 3300 of file specfun.h.

References __gnu_cxx::ibetaf().

7.3.3.164 long double __gnu_cxx::ibetacl (long double __a, long double __b, long double __x) [inline]

Definition at line 3304 of file specfun.h.

References __gnu_cxx::ibetal().

7.3.3.165 float __gnu_cxx::ibetaf (float *__a*, float *__b*, float *__x*) [inline]

Return the regularized incomplete beta function of parameters `a`, `b`, and argument `x`.

See ibeta for details.

Definition at line 3257 of file specfun.h.

Referenced by __gnu_cxx::ibetacf().

7.3.3.166 long double __gnu_cxx::ibetal (long double *__a*, long double *__b*, long double *__x*) [inline]

Return the regularized incomplete beta function of parameters `a`, `b`, and argument `x`.

See ibeta for details.

Definition at line 3267 of file specfun.h.

Referenced by __gnu_cxx::ibetacl().

7.3.3.167 template<typename _Talpha , typename _Tbeta , typename _Tp > __gnu_cxx::__promote_num_t<_Talpha, _Tbeta, _Tp> __gnu_cxx::jacobi (unsigned *__n*, _Talpha *__alpha*, _Tbeta *__beta*, _Tp *__x*) [inline]

Return the Jacobi polynomial $P_n^{(\alpha,\beta)}(x)$ of degree `n` and `float` orders $\alpha, \beta > -1$ and argument `x`.

The Jacobi polynomials are generated by a three-term recursion relation:

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

where $P_0^{(\alpha,\beta)}(x) = 1$ and $P_1^{(\alpha,\beta)}(x) = ((\alpha-\beta)+(2+(\alpha+\beta))*x)/2$.

Template Parameters

_Talpha	The real type of the order α
_Tbeta	The real type of the order β
_Tp	The real type of the argument

Parameters

__n	The non-negative integral degree
__alpha	The real order
__beta	The real order
__x	The real argument

Definition at line 2129 of file specfun.h.

References std::__detail::__beta().

7.3.3.168 template<typename _Kp , typename _Up > __gnu_cxx::__promote_num_t<_Kp, _Up> __gnu_cxx::jacobi_cn (_Kp *__k*, _Up *__u*) [inline]

Return the Jacobi elliptic $cn(k, u)$ integral of real modulus k and argument u.

The Jacobi elliptic `cn` integral is defined by

$$\cos(\phi) = cn(k, F(k, \phi))$$

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

Template Parameters

_Kp	The type of the real modulus
_Up	The type of the real argument

Parameters

__k	The real modulus
__u	The real argument

Definition at line 1838 of file specfun.h.

7.3.3.169 float __gnu_cxx::jacobi_cnf (float *__k*, float *__u*) [inline]

Return the Jacobi elliptic $cn(k,u)$ integral of `float` modulus k and argument u.

See also

jacobi_cn for details.

Definition at line 1802 of file specfun.h.

7.3.3.170 long double __gnu_cxx::jacobi_cnl (long double *__k*, long double *__u*) [inline]

Return the Jacobi elliptic $cn(k,u)$ integral of `long double` modulus k and argument u.

See also

jacobi_cn for details.

Definition at line 1815 of file specfun.h.

7.3.3.171 template<typename _Kp , typename _Up > __gnu_cxx::__promote_num_t<_Kp, _Up> __gnu_cxx::jacobi_dn (_Kp *__k*, _Up *__u*) [inline]

Return the Jacobi elliptic $dn(k,u)$ integral of real modulus k and argument u.

The Jacobi elliptic `dn` integral is defined by

$$\sqrt{1-k^2\sin(\phi)} = dn(k, F(k,\phi))$$

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

Template Parameters

_Kp	The type of the real modulus
_Up	The type of the real argument

Parameters

__k	The real modulus

__u	The real argument

Definition at line 1890 of file specfun.h.

7.3.3.172 float __gnu_cxx::jacobi_dnf (float *__k*, float *__u*) [inline]

Return the Jacobi elliptic $dn(k,u)$ integral of `float` modulus k and argument u.

See also

jacobi_dn for details.

Definition at line 1854 of file specfun.h.

7.3.3.173 long double __gnu_cxx::jacobi_dnl (long double *__k*, long double *__u*) [inline]

Return the Jacobi elliptic $dn(k,u)$ integral of `long double` modulus k and argument u.

See also

jacobi_dn for details.

Definition at line 1867 of file specfun.h.

7.3.3.174 template<typename _Kp , typename _Up > __gnu_cxx::__promote_num_t<_Kp, _Up> __gnu_cxx::jacobi_sn (_Kp *__k*, _Up *__u*) [inline]

Return the Jacobi elliptic $sn(k,u)$ integral of real modulus k and argument u.

The Jacobi elliptic `sn` integral is defined by

$$\sin(\phi) = sn(k, F(k,\phi))$$

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

Template Parameters

_Kp	The type of the real modulus
_Up	The type of the real argument

Parameters

__k	The real modulus
__u	The real argument

Definition at line 1786 of file specfun.h.

7.3.3.175 float __gnu_cxx::jacobi_snf (float *__k*, float *__u*) [inline]

Return the Jacobi elliptic $sn(k,u)$ integral of `float` modulus k and argument u.

See also

jacobi_sn for details.

Definition at line 1750 of file specfun.h.

7.3.3.176 long double __gnu_cxx::jacobi_snl (long double *__k*, long double *__u*) [inline]

Return the Jacobi elliptic $sn(k, u)$ integral of `long double` modulus k and argument u.

See also

jacobi_sn for details.

Definition at line 1763 of file specfun.h.

7.3.3.177 template<typename _Tk , typename _Tphi > __gnu_cxx::__promote_num_t<_Tk, _Tphi> __gnu_cxx::jacobi_zeta (_Tk *__k*, _Tphi *__phi*) [inline]

Return the Jacobi zeta function of `k` and $@c\phi$.

The Jacobi zeta function is defined by

$$Z(m, \phi) = E(m, \phi) - \frac{E(m)F(m, \phi)}{K(m)}$$

where $E(m, \phi)$ is the elliptic function of the second kind, $E(m)$ is the complete ellitic function of the second kind, and $F(m, \phi)$ is the elliptic function of the first kind.

Template Parameters

_Tk	the real type of the modulus
_Tphi	the real type of the angle limit

Parameters

__k	The modulus
__phi	The angle

Definition at line 3813 of file specfun.h.

7.3.3.178 float __gnu_cxx::jacobi_zetaf (float *__k*, float *__phi*) [inline]

Definition at line 3788 of file specfun.h.

7.3.3.179 long double __gnu_cxx::jacobi_zetal (long double *__k*, long double *__phi*) [inline]

Definition at line 3792 of file specfun.h.

7.3.3.180 float __gnu_cxx::jacobif (unsigned *__n*, float *__alpha*, float *__beta*, float *__x*) [inline]

Return the Jacobi polynomial $P_n^{(\alpha,\beta)}(x)$ of degree `n` and `float` orders $\alpha, \beta > -1$ and argument `x`.

See also

jacobi for details.

Definition at line 2085 of file specfun.h.

References std::__detail::__beta().

7.3.3.181 long double __gnu_cxx::jacobil (unsigned __*n*, long double __*alpha*, long double __*beta*, long double __*x*) [inline]

Return the Jacobi polynomial $P_n^{(\alpha,\beta)}(x)$ of degree `n` and `long double` orders $\alpha, \beta > -1$ and argument `x`.

See also

jacobi for details.

Definition at line 2096 of file specfun.h.

References std::__detail::__beta().

7.3.3.182 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::lbincoef (unsigned int __*n*, unsigned int __*k*) [inline]

Definition at line 3665 of file specfun.h.

7.3.3.183 float __gnu_cxx::lbincoeff (unsigned int __*n*, unsigned int __*k*) [inline]

Definition at line 3653 of file specfun.h.

7.3.3.184 long double __gnu_cxx::lbincoefl (unsigned int __*n*, unsigned int __*k*) [inline]

Definition at line 3657 of file specfun.h.

7.3.3.185 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::ldouble_factorial (int __*n*) [inline]

Definition at line 3623 of file specfun.h.

7.3.3.186 float __gnu_cxx::ldouble_factorialf (int __*n*) [inline]

Definition at line 3611 of file specfun.h.

7.3.3.187 long double __gnu_cxx::ldouble_factoriall (int __*n*) [inline]

Definition at line 3615 of file specfun.h.

7.3.3.188 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::legendre_q (unsigned int __*n*, _Tp __*x*) [inline]

Definition at line 3737 of file specfun.h.

7.3.3.189 float __gnu_cxx::legendre_qf (unsigned int __*n*, float __*x*) [inline]

Return the Legendre function of the second kind $Q_l(x)$ for `float` argument.

See also

legendre_q for details.

Definition at line 3719 of file specfun.h.

7.3.3.190 long double __gnu_cxx::legendre_ql (unsigned int *__n*, long double *__x*) [inline]

Return the Legendre function of the second kind $Q_l(x)$ for `long double` argument.

See also

legendre_q for details.

Definition at line 3729 of file specfun.h.

7.3.3.191 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::lfactorial (unsigned int *__n*) [inline]

Definition at line 3602 of file specfun.h.

7.3.3.192 float __gnu_cxx::lfactorialf (unsigned int *__n*) [inline]

Definition at line 3590 of file specfun.h.

7.3.3.193 long double __gnu_cxx::lfactoriall (unsigned int *__n*) [inline]

Definition at line 3594 of file specfun.h.

7.3.3.194 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::logint (_Tp *__x*) [inline]

Return the logarithmic integral of argument `x`.

The logarithmic integral is defined by

$$li(x) = \int_0^x \frac{dt}{ln(t)}$$

Parameters

__x	The real upper integration limit

Definition at line 1565 of file specfun.h.

7.3.3.195 float __gnu_cxx::logintf (float *__x*) [inline]

Return the logarithmic integral of argument `x`.

See also

logint for details.

Definition at line 1541 of file specfun.h.

7.3.3.196 long double __gnu_cxx::logintl (long double __x) [inline]

Return the logarithmic integral of argument x.

See also

logint for details.

Definition at line 1550 of file specfun.h.

7.3.3.197 template<typename _Tp , typename _Tn > __gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::lpochhammer_l (_Tp __a, _Tn __n) [inline]

Definition at line 3497 of file specfun.h.

7.3.3.198 float __gnu_cxx::lpochhammer_lf (float __a, float __n) [inline]

Definition at line 3485 of file specfun.h.

7.3.3.199 long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n) [inline]

Definition at line 3489 of file specfun.h.

7.3.3.200 template<typename _Tp , typename _Tn > __gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::lpochhammer_u (_Tp __a, _Tn __n) [inline]

Definition at line 3476 of file specfun.h.

7.3.3.201 float __gnu_cxx::lpochhammer_uf (float __a, float __n) [inline]

Definition at line 3464 of file specfun.h.

7.3.3.202 long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n) [inline]

Definition at line 3468 of file specfun.h.

7.3.3.203 template<typename _Tph , typename _Tpa > __gnu_cxx::__promote_num_t<_Tph, _Tpa> __gnu_cxx::owens_t (_Tph __h, _Tpa __a) [inline]

Return the Owens T function $T(h,a)$ of shape factor h and integration limit a.

The Owens T function is defined by

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

Parameters

__h	The shape factor
__a	The integration limit

Definition at line 5091 of file specfun.h.

7.3.3.204 float __gnu_cxx::owens_tf (float *__h*, float *__a*) [inline]

Return the Owens T function $T(h,a)$ of shape factor h and integration limit a.

See also

owens_t for details.

Definition at line 5063 of file specfun.h.

7.3.3.205 long double __gnu_cxx::owens_tl (long double *__h*, long double *__a*) [inline]

Return the Owens T function $T(h,a)$ of `long double` shape factor h and integration limit a.

See also

owens_t for details.

Definition at line 5073 of file specfun.h.

7.3.3.206 template<typename _Tp , typename _Tn > __gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::pochhammer_l (_Tp *__a*, _Tn *__n*) [inline]

Definition at line 3539 of file specfun.h.

7.3.3.207 float __gnu_cxx::pochhammer_lf (float *__a*, float *__n*) [inline]

Definition at line 3527 of file specfun.h.

7.3.3.208 long double __gnu_cxx::pochhammer_ll (long double *__a*, long double *__n*) [inline]

Definition at line 3531 of file specfun.h.

7.3.3.209 template<typename _Tp , typename _Tn > __gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::pochhammer_u (_Tp *__a*, _Tn *__n*) [inline]

Definition at line 3518 of file specfun.h.

7.3.3.210 float __gnu_cxx::pochhammer_uf (float *__a*, float *__n*) [inline]

Definition at line 3506 of file specfun.h.

7.3.3.211 long double __gnu_cxx::pochhammer_ul (long double *__a*, long double *__n*) [inline]

Definition at line 3510 of file specfun.h.

7.3.3.212 template<typename _Tp , typename _Wp > __gnu_cxx::__promote_num_t<_Tp, _Wp> __gnu_cxx::polylog (_Tp *__s*, _Wp *__w*) [inline]

Return the complex polylogarithm function of real thing `s` and complex argument `w`.

The polylogarithm function is defined by

Parameters

__s	
__w	

Definition at line 4403 of file specfun.h.

7.3.3.213 template<typename _Tp , typename _Wp > std::complex<__gnu_cxx::__promote_num_t<_Tp, _Wp> > __gnu_cxx::polylog (_Tp *__s*, std::complex< _Tp > *__w*) [inline]

Return the complex polylogarithm function of real thing `s` and complex argument `w`.

The polylogarithm function is defined by

Parameters

__s	
__w	

Definition at line 4443 of file specfun.h.

7.3.3.214 float __gnu_cxx::polylogf (float *__s*, float *__w*) [inline]

Return the real polylogarithm function of real thing `s` and real argument `w`.

See also

polylog for details.

Definition at line 4376 of file specfun.h.

7.3.3.215 std::complex<float> __gnu_cxx::polylogf (float *__s*, std::complex< float > *__w*) [inline]

Return the complex polylogarithm function of real thing `s` and complex argument `w`.

See also

polylog for details.

Definition at line 4416 of file specfun.h.

7.3.3.216 long double __gnu_cxx::polylogl (long double __s, long double __w) [inline]

Return the complex polylogarithm function of real thing s and complex argument w.

See also

polylog for details.

Definition at line 4386 of file specfun.h.

7.3.3.217 std::complex<long double> __gnu_cxx::polylogl (long double __s, std::complex< long double > __w) [inline]

Return the complex polylogarithm function of real thing s and complex argument w.

See also

polylog for details.

Definition at line 4426 of file specfun.h.

7.3.3.218 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::psi (_Tp __x) [inline]

Return the psi or digamma function of argument x.

The the psi or digamma function is defined by

$$\psi(x) =$$

Parameters

__x	The parameter

Definition at line 3242 of file specfun.h.

7.3.3.219 float __gnu_cxx::psif (float __x) [inline]

Definition at line 3223 of file specfun.h.

7.3.3.220 long double __gnu_cxx::psil (long double __x) [inline]

Definition at line 3227 of file specfun.h.

7.3.3.221 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::radpoly (unsigned int __n, unsigned int __m, _Tp __rho) [inline]

Return the radial polynomial $R_n^m(\rho)$ for non-negative degree n, order $m <= n$, and real radial argument ρ.

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!(\frac{n+m}{2}-k)!(\frac{n-m}{2}-k)!} \rho^{n-2k}$$

for $n-m$ even and identically 0 for $n-m$ odd. The radial polynomials can be related to the Jacobi polynomials:

$$R_n^m(\rho) =$$

See also

jacobi for details on the Jacobi polynomials.

Template Parameters

_Tp	The real type of the radial coordinate

Parameters

__n	The non-negative degree.
__m	The non-negative azimuthal order
__rho	The radial argument

Definition at line 2287 of file specfun.h.

7.3.3.222 float __gnu_cxx::radpolyf (unsigned int *__n*, unsigned int *__m*, float *__rho*) [inline]

Return the radial polynomial $R_n^m(\rho)$ for non-negative degree n, order $m <= n$, and `float` radial argument ρ.

See also

radpoly for details.

Definition at line 2248 of file specfun.h.

References std::__detail::__poly_radial_jacobi().

7.3.3.223 long double __gnu_cxx::radpolyl (unsigned int *__n*, unsigned int *__m*, long double *__rho*) [inline]

Return the radial polynomial $R_n^m(\rho)$ for non-negative degree n, order $m <= n$, and `long double` radial argument ρ.

See also

radpoly for details.

Definition at line 2259 of file specfun.h.

References std::__detail::__poly_radial_jacobi().

7.3.3.224 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinc (_Tp *__x*) [inline]

Definition at line 1527 of file specfun.h.

7.3.3.225 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinc_pi (_Tp *__x*) [inline]

Definition at line 1500 of file specfun.h.

7.3.3.226 float __gnu_cxx::sinc_pif (float *__x*) [inline]

Definition at line 1485 of file specfun.h.

7.3.3.227 long double __gnu_cxx::sinc_pil (long double __x) [inline]

Definition at line 1492 of file specfun.h.

7.3.3.228 float __gnu_cxx::sincf (float __x) [inline]

Definition at line 1512 of file specfun.h.

7.3.3.229 long double __gnu_cxx::sincl (long double __x) [inline]

Definition at line 1519 of file specfun.h.

7.3.3.230 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinhc (_Tp __x) [inline]

Definition at line 2329 of file specfun.h.

7.3.3.231 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinhc_pi (_Tp __x) [inline]

Definition at line 2308 of file specfun.h.

7.3.3.232 float __gnu_cxx::sinhc_pif (float __x) [inline]

Definition at line 2296 of file specfun.h.

7.3.3.233 long double __gnu_cxx::sinhc_pil (long double __x) [inline]

Definition at line 2300 of file specfun.h.

7.3.3.234 float __gnu_cxx::sinhcf (float __x) [inline]

Definition at line 2317 of file specfun.h.

7.3.3.235 long double __gnu_cxx::sinhcl (long double __x) [inline]

Definition at line 2321 of file specfun.h.

7.3.3.236 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinhint (_Tp __x) [inline]

Return the hyperbolic sine integral $Shi(x)$ of real argument x.

The hyperbolic sine integral is defined by

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

Template Parameters

_Tp	The type of the real argument

Parameters

__x	The argument

Definition at line 1685 of file specfun.h.

7.3.3.237 float __gnu_cxx::sinhintf (float __x) [inline]

Return the hyperbolic sine integral of `float` argument x.

See also

sinhint for details.

Definition at line 1658 of file specfun.h.

7.3.3.238 long double __gnu_cxx::sinhintl (long double __x) [inline]

Return the hyperbolic sine integral $Shi(x)$ of `long double` argument x.

See also

sinhint for details.

Definition at line 1668 of file specfun.h.

7.3.3.239 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinint (_Tp __x) [inline]

Return the sine integral $Si(x)$ of real argument x.

The sine integral is defined by

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

Parameters

__x	The real upper integration limit

Definition at line 1604 of file specfun.h.

7.3.3.240 float __gnu_cxx::sinintf (float __x) [inline]

Return the sine integral $Si(x)$ of `float` argument x.

See also

sinint for details.

Definition at line 1579 of file specfun.h.

7.3.3.241 long double __gnu_cxx::sinintl (long double *__x*) [inline]

Return the sine integral $Si(x)$ of `long double` argument x.

See also

sinint for details.

Definition at line 1589 of file specfun.h.

7.3.3.242 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sph_bessel_i (unsigned int *__n,* _Tp *__x*) [inline]

Return the regular modified spherical Bessel function $i_n(x)$ of nonnegative order n and real argument $x >= 0$.

The spherical Bessel function is defined by:

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

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__n	The integral order `n >= 0`
__x	The real argument `x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 2567 of file specfun.h.

7.3.3.243 float __gnu_cxx::sph_bessel_if (unsigned int *__n,* float *__x*) [inline]

Return the regular modified spherical Bessel function $i_n(x)$ of nonnegative order n and `float` argument $x >= 0$.

See also

sph_bessel_i for details.

Definition at line 2528 of file specfun.h.

7.3.3.244 long double __gnu_cxx::sph_bessel_il (unsigned int *__n,* long double *__x*) [inline]

Return the regular modified spherical Bessel function $i_n(x)$ of nonnegative order n and `long double` argument $x >= 0$.

See also

sph_bessel_i for details.

Definition at line 2543 of file specfun.h.

7.3.3.245 template< typename _Tp > __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x) [inline]

Return the irregular modified spherical Bessel function $k_n(x)$ of nonnegative order n and real argument $x >= 0$.

The spherical Bessel function is defined by:

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

Template Parameters

_Tp	The floating-point type of the argument `__x`.

Parameters

__n	The integral order `n >= 0`
__x	The real argument `x >= 0`

Exceptions

std::domain_error	if `__x < 0` .

Definition at line 2624 of file specfun.h.

7.3.3.246 float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x) [inline]

Return the irregular modified spherical Bessel function $k_n(x)$ of nonnegative order n and `float` argument $x >= 0$.

See also

sph_bessel_k for more details.

Definition at line 2585 of file specfun.h.

7.3.3.247 long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x) [inline]

Return the irregular modified spherical Bessel function $k_n(x)$ of nonnegative order n and `long double` argument $x >= 0$.

See also

sph_bessel_k for more details.

Definition at line 2600 of file specfun.h.

7.3.3.248 template< typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z) [inline]

Return the spherical Hankel function of the first kind $h^{(1)}_n(x)$ of nonnegative order n and real argument $x >= 0$.

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

$$h^{(1)}_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} H^{(1)}_{n+1/2}(x)$$

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative order
__z	The real argument

Definition at line 2470 of file specfun.h.

7.3.3.249 template<typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __gnu_cxx::sph_hankel_1 (unsigned int *__n*, std::complex< _Tp > *__x*) [inline]

Return the complex spherical Hankel function of the first kind $h^{(1)}_n(x)$ of non-negative integral n and complex argument x.

The spherical Hankel function of the first kind is defined by

$$h^{(1)}_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} H^{(1)}_{n+1/2}(x) = j_n(x) + in_n(x)$$

where $j_n(x)$ and $n_n(x)$ are the spherical Bessel and Neumann functions respectively.

Parameters

__n	The integral order >= 0
__x	The complex argument

Definition at line 4262 of file specfun.h.

7.3.3.250 std::complex<float> __gnu_cxx::sph_hankel_1f (unsigned int *__n*, float *__z*) [inline]

Return the spherical Hankel function of the first kind $h^{(1)}_n(x)$ of nonnegative order n and `float` argument $x >= 0$.

See also

sph_hankel_1 for details.

Definition at line 2442 of file specfun.h.

7.3.3.251 std::complex<float> __gnu_cxx::sph_hankel_1f (unsigned int *__n*, std::complex< float > *__x*) [inline]

Return the complex spherical Hankel function of the first kind $h^{(1)}_n(x)$ of non-negative integral n and `std::complex<float>` argument x.

See also

sph_hankel_1 for more details.

Definition at line 4230 of file specfun.h.

7.3.3.252 std::complex<long double> __gnu_cxx::sph_hankel_1l (unsigned int *__n*, long double *__z*) [inline]

Return the spherical Hankel function of the first kind $h^{(1)}_n(x)$ of nonnegative order n and `long double` argument $x >= 0$.

See also

sph_hankel_1 for details.

Definition at line 2452 of file specfun.h.

7.3.3.253 std::complex<long double> __gnu_cxx::sph_hankel_1l (unsigned int *__n*, std::complex< long double > *__x*) [inline]

Return the complex spherical Hankel function of the first kind $h^{(1)}_n(x)$ of non-negative integral n and `std::complex<long double>` argument x.

See also

sph_hankel_1 for more details.

Definition at line 4241 of file specfun.h.

7.3.3.254 template<typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __gnu_cxx::sph_hankel_2 (unsigned int *__n*, _Tp *__z*) [inline]

Return the spherical Hankel function of the second kind $h^{(2)}_n(x)$ of nonnegative order n and real argument $x >= 0$.

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

$$h^{(2)}_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} H^{(2)}_{n+1/2}(x)$$

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative order
__z	The real argument

Definition at line 2513 of file specfun.h.

7.3.3.255 template<typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __gnu_cxx::sph_hankel_2 (unsigned int *__n*, std::complex< _Tp > *__x*) [inline]

Return the complex spherical Hankel function of the second kind $h^{(2)}_n(x)$ of nonnegative order n and complex argument `x`.

The spherical Hankel function of the second kind is defined by

$$h^{(2)}_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} H^{(2)}_{n+1/2}(x) = j_n(x) - i n_n(x)$$

where $j_n(x)$ and $n_n(x)$ are the spherical Bessel and Neumann functions respectively.

Parameters

__n	The integral order >= 0
__x	The complex argument

Definition at line 4310 of file specfun.h.

7.3.3.256 std::complex<float> __gnu_cxx::sph_hankel_2f (unsigned int *__n*, float *__z*) [inline]

Return the spherical Hankel function of the second kind $h^{(2)}_n(x)$ of nonnegative order n and `float` argument $x >= 0$.

See also

sph_hankel_2 for details.

Definition at line 2485 of file specfun.h.

7.3.3.257 std::complex<float> __gnu_cxx::sph_hankel_2f (unsigned int *__n*, std::complex< float > *__x*) [inline]

Return the complex spherical Hankel function of the second kind $h^{(2)}_n(x)$ of non-negative integral n and `std::complex<float>` argument x.

See also

sph_hankel_2 for more details.

Definition at line 4278 of file specfun.h.

7.3.3.258 std::complex<long double> __gnu_cxx::sph_hankel_2l (unsigned int *__n*, long double *__z*) [inline]

Return the spherical Hankel function of the second kind $h^{(2)}_n(x)$ of nonnegative order n and `long double` argument $x >= 0$.

See also

sph_hankel_2 for details.

Definition at line 2495 of file specfun.h.

7.3.3.259 std::complex<long double> __gnu_cxx::sph_hankel_2l (unsigned int *__n*, std::complex< long double > *__x*) [inline]

Return the complex spherical Hankel function of the second kind $h^{(2)}_n(x)$ of non-negative integral n and `std::complex<long double>` argument x.

See also

sph_hankel_2 for more details.

Definition at line 4289 of file specfun.h.

7.3.3.260 template<typename _Ttheta , typename _Tphi > std::complex<__gnu_cxx::__promote_num_t<_Ttheta, _Tphi> > __gnu_cxx::sph_harmonic (unsigned int *__l*, int *__m*, _Ttheta *__theta*, _Tphi *__phi*) [inline]

Return the complex spherical harmonic function of degree `l`, order `m`, and real zenith angle θ, and azimuth angle ϕ.

The spherical harmonic function 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
__m	The degree
__theta	The zenith angle in radians
__phi	The azimuth angle in radians

Definition at line 4361 of file specfun.h.

7.3.3.261 std::complex<float> __gnu_cxx::sph_harmonicf (unsigned int *__l*, int *__m*, float *__theta*, float *__phi*) [inline]

Return the complex spherical harmonic function of degree `l`, order `m`, and `float` zenith angle θ, and azimuth angle ϕ.

See also

sph_harmonic for details.

Definition at line 4325 of file specfun.h.

7.3.3.262 std::complex<long double> __gnu_cxx::sph_harmonicl (unsigned int *__l*, int *__m*, long double *__theta*, long double *__phi*) [inline]

Return the complex spherical harmonic function of degree `l`, order `m`, and `long double` zenith angle θ, and azimuth angle ϕ.

See also

sph_harmonic for details.

Definition at line 4337 of file specfun.h.

7.3.3.263 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_num_t<_Tpnu, _Tp> __gnu_cxx::theta_1 (_Tpnu *__nu*, _Tp *__x*) [inline]

Return the exponential theta-1 function $\theta_1(\nu,x)$ of period nu and argument x.

The Neville theta-1 function is defined by

$$\theta_1(\nu,x) = \frac{1}{\sqrt{\pi x}}\sum_{j=-\infty}^{+\infty}(-1)^j\exp\left(\frac{-(\nu+j-1/2)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 4711 of file specfun.h.

7.3.3.264 float __gnu_cxx::theta_1f (float *__nu*, float *__x*) [inline]

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period nu and argument x.

See also

theta_1 for details.

Definition at line 4683 of file specfun.h.

7.3.3.265 long double __gnu_cxx::theta_1l (long double *__nu*, long double *__x*) [inline]

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period nu and argument x.

See also

theta_1 for details.

Definition at line 4693 of file specfun.h.

7.3.3.266 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_num_t<_Tpnu, _Tp> __gnu_cxx::theta_2 (_Tpnu *__nu*, _Tp *__x*) [inline]

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period nu and argument x.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 4754 of file specfun.h.

7.3.3.267 float __gnu_cxx::theta_2f (float *__nu*, float *__x*) [inline]

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period nu and argument x.

See also

theta_2 for details.

Definition at line 4726 of file specfun.h.

7.3.3.268 long double __gnu_cxx::theta_2l (long double __nu, long double __x) [inline]

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period nu and argument x.

See also

theta_2 for details.

Definition at line 4736 of file specfun.h.

7.3.3.269 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_num_t<_Tpnu, _Tp> __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x) [inline]

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period nu and argument x.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 1) argument
__x	The argument

Definition at line 4797 of file specfun.h.

7.3.3.270 float __gnu_cxx::theta_3f (float __nu, float __x) [inline]

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period nu and argument x.

See also

theta_3 for details.

Definition at line 4769 of file specfun.h.

7.3.3.271 long double __gnu_cxx::theta_3l (long double __nu, long double __x) [inline]

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period nu and argument x.

See also

theta_3 for details.

Definition at line 4779 of file specfun.h.

7.3.3.272 template<typename _Tpnu , typename _Tp > __gnu_cxx::__promote_num_t<_Tpnu, _Tp> __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x) [inline]

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x.

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j + 1/2)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 1) argument
__x	The argument

Definition at line 4840 of file specfun.h.

7.3.3.273 float __gnu_cxx::theta_4f (float *__nu*, float *__x*) [inline]

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x.

See also

theta_4 for details.

Definition at line 4812 of file specfun.h.

7.3.3.274 long double __gnu_cxx::theta_4l (long double *__nu*, long double *__x*) [inline]

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x.

See also

theta_4 for details.

Definition at line 4822 of file specfun.h.

7.3.3.275 template<typename _Tpk , typename _Tp > __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_c (_Tpk *__k*, _Tp *__x*) [inline]

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x.

The Neville theta-c function is defined by

Parameters

__k	The modulus $-1 <= k <= +1$
__x	The argument

Definition at line 4964 of file specfun.h.

7.3.3.276 float __gnu_cxx::theta_cf (float *__k*, float *__x*) [inline]

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x.

See also

theta_c for details.

Definition at line 4937 of file specfun.h.

7.3.3.277 long double __gnu_cxx::theta_cl (long double __*k*, long double __*x*) [inline]

Return the Neville theta-c function $\theta_c(k,x)$ of `long double` modulus k and argument x.

See also

theta_c for details.

Definition at line 4947 of file specfun.h.

7.3.3.278 template<typename _Tpk , typename _Tp > __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_d (_Tpk __*k*, _Tp __*x*) [inline]

Return the Neville theta-d function $\theta_d(k,x)$ of modulus `k` and argument `x`.

The Neville theta-d function is defined by

$$\theta_d(k,x) =$$

Parameters

__*k*	The modulus $-1 <= k <= +1$
__*x*	The argument

Definition at line 5006 of file specfun.h.

7.3.3.279 float __gnu_cxx::theta_df (float __*k*, float __*x*) [inline]

Return the Neville theta-d function $\theta_d(k,x)$ of modulus `k` and argument `x`.

See also

theta_d for details.

Definition at line 4979 of file specfun.h.

7.3.3.280 long double __gnu_cxx::theta_dl (long double __*k*, long double __*x*) [inline]

Return the Neville theta-d function $\theta_d(k,x)$ of `long double` modulus `k` and argument `x`.

See also

theta_d for details.

Definition at line 4989 of file specfun.h.

7.3.3.281 template<typename _Tpk , typename _Tp > __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_n (_Tpk __*k*, _Tp __*x*) [inline]

Return the Neville theta-n function $\theta_n(k,x)$ of modulus k and argument x.

The Neville theta-n function is defined by

$$\theta_n(k,x) =$$

Parameters

__k	The modulus $-1 <= k <= +1$
__x	The argument

Definition at line 5048 of file specfun.h.

7.3.3.282 float __gnu_cxx::theta_nf (float __k, float __x) [inline]

Return the Neville theta-n function $\theta_n(k, x)$ of modulus k and argument x.

See also

theta_n for details.

Definition at line 5021 of file specfun.h.

7.3.3.283 long double __gnu_cxx::theta_nl (long double __k, long double __x) [inline]

Return the Neville theta-n function $\theta_n(k, x)$ of `long double` modulus k and argument x.

See also

theta_n for details.

Definition at line 5031 of file specfun.h.

7.3.3.284 template<typename _Tpk , typename _Tp > __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_s (_Tpk __k, _Tp __x) [inline]

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x.

The Neville theta-s function is defined by

Parameters

__k	The modulus $-1 <= k <= +1$
__x	The argument

Definition at line 4922 of file specfun.h.

7.3.3.285 float __gnu_cxx::theta_sf (float __k, float __x) [inline]

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x.

See also

theta_s for details.

Definition at line 4895 of file specfun.h.

7.3.3.286 long double __gnu_cxx::theta_sl (long double *__k*, long double *__x*) [inline]

Return the Neville theta-s function $\theta_s(k,x)$ of `long double` modulus k and argument x.

See also

theta_s for details.

Definition at line 4905 of file specfun.h.

7.3.3.287 template<typename _Trho , typename _Tphi > __gnu_cxx::__promote_num_t<_Trho, _Tphi> __gnu_cxx::zernike (unsigned int *__n*, int *__m*, _Trho *__rho*, _Tphi *__phi*) [inline]

Return the Zernicke polynomial $Z_n^m(\rho,\phi)$ for non-negative degree n, signed order m, and real radial argument ρ and azimuthal angle ϕ.

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho,\phi) = R_n^m(\rho)\cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho,\phi) = R_n^m(\rho)\sin(m\phi)$$

for non-negative degree m and $m <= n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

radpoly).

Template Parameters

_Trho	The real type of the radial coordinate
_Tphi	The real type of the azimuthal angle

Parameters

__n	The non-negative degree.
__m	The (signed) azimuthal order
__rho	The radial coordinate
__phi	The azimuthal angle

Definition at line 2232 of file specfun.h.

7.3.3.288 float __gnu_cxx::zernikef (unsigned int *__n*, int *__m*, float *__rho*, float *__phi*) [inline]

Return the Zernicke polynomial $Z_n^m(\rho,\phi)$ for non-negative degree n, signed order m, and real radial argument ρ and azimuthal angle ϕ.

See also

zernike for details.

Definition at line 2193 of file specfun.h.

7.3.3.289 long double __gnu_cxx::zernikel (unsigned int *__n*, int *__m*, long double *__rho*, long double *__phi*) [inline]

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n, signed order m, and real radial argument ρ and azimuthal angle ϕ.

See also

zernike for details.

Definition at line 2204 of file specfun.h.

Chapter 8

Namespace Documentation

8.1 __gnu_cxx Namespace Reference

Enumerations

- enum { _GLIBCXX_JACOBI_SN, _GLIBCXX_JACOBI_CN, _GLIBCXX_JACOBI_DN }

Functions

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > airy_ai (_Tp __x)
- float airy_aif (float __x)
- long double airy_ail (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > airy_bi (_Tp __x)
- float airy_bif (float __x)
- long double airy_bil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > bernoulli (unsigned int __n)
- float bernoullif (unsigned int __n)
- long double bernoullil (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > bincoef (unsigned int __n, unsigned int __k)
- float bincoeff (unsigned int __n, unsigned int __k)
- long double bincoefl (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > chebyshev_t (unsigned int __n, _Tp __x)
- float chebyshev_tf (unsigned int __n, float __x)
- long double chebyshev_tl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > chebyshev_u (unsigned int __n, _Tp __x)
- float chebyshev_uf (unsigned int __n, float __x)
- long double chebyshev_ul (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > chebyshev_v (unsigned int __n, _Tp __x)
- float chebyshev_vf (unsigned int __n, float __x)

- long double chebyshev_vl (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > chebyshev_w (unsigned int __n, _Tp __x)
- float chebyshev_wf (unsigned int __n, float __x)
- long double chebyshev_wl (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > clausen (unsigned int __m, _Tp __w)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > clausen (unsigned int __m, std::complex< _Tp > __w)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > clausen_c (unsigned int __m, _Tp __w)
- float clausen_cf (unsigned int __m, float __w)
- long double clausen_cl (unsigned int __m, long double __w)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > clausen_s (unsigned int __m, _Tp __w)
- float clausen_sf (unsigned int __m, float __w)
- long double clausen_sl (unsigned int __m, long double __w)
- float clausenf (unsigned int __m, float __w)
- std::complex< float > clausenf (unsigned int __m, std::complex< float > __w)
- long double clausenl (unsigned int __m, long double __w)
- std::complex< long double > clausenl (unsigned int __m, std::complex< long double > __w)
- template< typename _Tk >
  __gnu_cxx::__promote_num_t< _Tk > comp_ellint_d (_Tk __k)
- float comp_ellint_df (float __k)
- long double comp_ellint_dl (long double __k)
- float comp_ellint_rf (float __x, float __y)
- long double comp_ellint_rf (long double __x, long double __y)
- template< typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > comp_ellint_rf (_Tx __x, _Ty __y)
- float comp_ellint_rg (float __x, float __y)
- long double comp_ellint_rg (long double __x, long double __y)
- template< typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > comp_ellint_rg (_Tx __x, _Ty __y)
- template< typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)
- template< typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_2< _Tpc, _Tp >::__type conf_hyperg_lim (_Tpc __c, _Tp __x)
- float conf_hyperg_limf (float __c, float __x)
- long double conf_hyperg_liml (long double __c, long double __x)
- float conf_hypergf (float __a, float __c, float __x)
- long double conf_hypergl (long double __a, long double __c, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > coshint (_Tp __x)
- float coshintf (float __x)
- long double coshintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > cosint (_Tp __x)
- float cosintf (float __x)
- long double cosintl (long double __x)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_1 (_Tpnu __nu, _Tp __z)

- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > cyl_hankel_1f (float __nu, float __z)
- std::complex< float > cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > cyl_hankel_1l (long double __nu, long double __z)
- std::complex< long double > cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > cyl_hankel_2f (float __nu, float __z)
- std::complex< float > cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > cyl_hankel_2l (long double __nu, long double __z)
- std::complex< long double > cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > dawson (_Tp __x)
- float dawsonf (float __x)
- long double dawsonl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > digamma (_Tp __z)
- float digammaf (float __z)
- long double digammal (long double __z)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > dilog (_Tp __x)
- float dilogf (float __x)
- long double dilogl (long double __x)
- template<typename _Tp >
  _Tp dirichlet_beta (_Tp __s)
- float dirichlet_betaf (float __s)
- long double dirichlet_betal (long double __s)
- template<typename _Tp >
  _Tp dirichlet_eta (_Tp __s)
- float dirichlet_etaf (float __s)
- long double dirichlet_etal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > double_factorial (int __n)
- float double_factorialf (int __n)
- long double double_factoriall (int __n)
- template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tk, _Tp, _Ta, _Tb > ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float ellint_celf (float __k_c, float __p, float __a, float __b)
- long double ellint_cell (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > ellint_d (_Tk __k, _Tphi __phi)
- float ellint_df (float __k, float __phi)
- long double ellint_dl (long double __k, long double __phi)
- template<typename _Tp , typename _Tk >
  __gnu_cxx::__promote_num_t< _Tp, _Tk > ellint_el1 (_Tp __x, _Tk __k_c)

- float ellint_el1f (float __x, float __k_c)
- long double ellint_el1l (long double __x, long double __k_c)
- template<typename _Tp , typename _Tk , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float ellint_el2f (float __x, float __k_c, float __a, float __b)
- long double ellint_el2l (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx , typename _Tk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p)
- float ellint_el3f (float __x, float __k_c, float __p)
- long double ellint_el3l (long double __x, long double __k_c, long double __p)
- template<typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > ellint_rc (_Tp __x, _Up __y)
- float ellint_rcf (float __x, float __y)
- long double ellint_rcl (long double __x, long double __y)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > ellint_rd (_Tp __x, _Up __y, _Vp __z)
- float ellint_rdf (float __x, float __y, float __z)
- long double ellint_rdl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > ellint_rf (_Tp __x, _Up __y, _Vp __z)
- float ellint_rff (float __x, float __y, float __z)
- long double ellint_rfl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > ellint_rg (_Tp __x, _Up __y, _Vp __z)
- float ellint_rgf (float __x, float __y, float __z)
- long double ellint_rgl (long double __x, long double __y, long double __z)
- template<typename _Tp , typename _Up , typename _Vp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float ellint_rjf (float __x, float __y, float __z, float __p)
- long double ellint_rjl (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
  _Tp ellnome (_Tp __k)
- float ellnomef (float __k)
- long double ellnomel (long double __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > expint_e1 (_Tp __x)
- float expint_e1f (float __x)
- long double expint_e1l (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > expint_en (unsigned int __n, _Tp __x)
- float expint_enf (unsigned int __n, float __x)
- long double expint_enl (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > factorial (unsigned int __n)
- float factorialf (unsigned int __n)
- long double factoriall (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > fresnel_c (_Tp __x)
- float fresnel_cf (float __x)
- long double fresnel_cl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > fresnel_s (_Tp __x)

- float fresnel_sf (float __x)
- long double fresnel_sl (long double __x)
- template< typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > gamma_l (_Tn __n, _Tp __x)
- float gamma_lf (float __n, float __x)
- long double gamma_ll (long double __n, long double __x)
- template< typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > gamma_p (_Ta __a, _Tp __x)
- float gamma_pf (float __a, float __x)
- long double gamma_pl (long double __a, long double __x)
- template< typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > gamma_q (_Ta __a, _Tp __x)
- float gamma_qf (float __a, float __x)
- long double gamma_ql (long double __a, long double __x)
- template< typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > gamma_u (_Tn __n, _Tp __x)
- float gamma_uf (float __n, float __x)
- long double gamma_ul (long double __n, long double __x)
- template< typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tp > gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)
- float gegenbauerf (unsigned int __n, float __alpha, float __x)
- long double gegenbauerl (unsigned int __n, long double __alpha, long double __x)
- template< typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > heuman_lambda (_Tk __k, _Tphi __phi)
- float heuman_lambdaf (float __k, float __phi)
- long double heuman_lambdal (long double __k, long double __phi)
- template< typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > hurwitz_zeta (_Tp __s, _Up __a)
- template< typename _Tp , typename _Up >
  std::complex< _Tp > hurwitz_zeta (_Tp __s, std::complex< _Up > __a)
- float hurwitz_zetaf (float __s, float __a)
- long double hurwitz_zetal (long double __s, long double __a)
- template< typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float hypergf (float __a, float __b, float __c, float __x)
- long double hypergl (long double __a, long double __b, long double __c, long double __x)
- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > ibeta (_Ta __a, _Tb __b, _Tp __x)
- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > ibetac (_Ta __a, _Tb __b, _Tp __x)
- float ibetacf (float __a, float __b, float __x)
- long double ibetacl (long double __a, long double __b, long double __x)
- float ibetaf (float __a, float __b, float __x)
- long double ibetal (long double __a, long double __b, long double __x)
- template< typename _Talpha , typename _Tbeta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template< typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > jacobi_cn (_Kp __k, _Up __u)
- float jacobi_cnf (float __k, float __u)
- long double jacobi_cnl (long double __k, long double __u)

- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > jacobi_dn (_Kp __k, _Up __u)
- float jacobi_dnf (float __k, float __u)
- long double jacobi_dnl (long double __k, long double __u)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > jacobi_sn (_Kp __k, _Up __u)
- float jacobi_snf (float __k, float __u)
- long double jacobi_snl (long double __k, long double __u)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > jacobi_zeta (_Tk __k, _Tphi __phi)
- float jacobi_zetaf (float __k, float __phi)
- long double jacobi_zetal (long double __k, long double __phi)
- float jacobif (unsigned __n, float __alpha, float __beta, float __x)
- long double jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > lbincoef (unsigned int __n, unsigned int __k)
- float lbincoeff (unsigned int __n, unsigned int __k)
- long double lbincoefl (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > ldouble_factorial (int __n)
- float ldouble_factorialf (int __n)
- long double ldouble_factoriall (int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > legendre_q (unsigned int __n, _Tp __x)
- float legendre_qf (unsigned int __n, float __x)
- long double legendre_ql (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > lfactorial (unsigned int __n)
- float lfactorialf (unsigned int __n)
- long double lfactoriall (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > logint (_Tp __x)
- float logintf (float __x)
- long double logintl (long double __x)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_l (_Tp __a, _Tn __n)
- float lpochhammer_lf (float __a, float __n)
- long double lpochhammer_ll (long double __a, long double __n)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_u (_Tp __a, _Tn __n)
- float lpochhammer_uf (float __a, float __n)
- long double lpochhammer_ul (long double __a, long double __n)
- template<typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_num_t< _Tph, _Tpa > owens_t (_Tph __h, _Tpa __a)
- float owens_tf (float __h, float __a)
- long double owens_tl (long double __h, long double __a)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_l (_Tp __a, _Tn __n)
- float pochhammer_lf (float __a, float __n)
- long double pochhammer_ll (long double __a, long double __n)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_u (_Tp __a, _Tn __n)

- float pochhammer_uf (float __a, float __n)
- long double pochhammer_ul (long double __a, long double __n)
- template<typename _Tp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Wp > polylog (_Tp __s, _Wp __w)
- template<typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > polylog (_Tp __s, std::complex< _Tp > __w)
- float polylogf (float __s, float __w)
- std::complex< float > polylogf (float __s, std::complex< float > __w)
- long double polylogl (long double __s, long double __w)
- std::complex< long double > polylogl (long double __s, std::complex< long double > __w)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > psi (_Tp __x)
- float psif (float __x)
- long double psil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > radpoly (unsigned int __n, unsigned int __m, _Tp __rho)
- float radpolyf (unsigned int __n, unsigned int __m, float __rho)
- long double radpolyl (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinc_pi (_Tp __x)
- float sinc_pif (float __x)
- long double sinc_pil (long double __x)
- float sincf (float __x)
- long double sincl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinhc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinhc_pi (_Tp __x)
- float sinhc_pif (float __x)
- long double sinhc_pil (long double __x)
- float sinhcf (float __x)
- long double sinhcl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinhint (_Tp __x)
- float sinhintf (float __x)
- long double sinhintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sinint (_Tp __x)
- float sinintf (float __x)
- long double sinintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sph_bessel_i (unsigned int __n, _Tp __x)
- float sph_bessel_if (unsigned int __n, float __x)
- long double sph_bessel_il (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > sph_bessel_k (unsigned int __n, _Tp __x)
- float sph_bessel_kf (unsigned int __n, float __x)
- long double sph_bessel_kl (unsigned int __n, long double __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)

- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > sph_hankel_1f (unsigned int __n, float __z)
- std::complex< float > sph_hankel_1f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > sph_hankel_1l (unsigned int __n, long double __z)
- std::complex< long double > sph_hankel_1l (unsigned int __n, std::complex< long double > __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > sph_hankel_2f (unsigned int __n, float __z)
- std::complex< float > sph_hankel_2f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > sph_hankel_2l (unsigned int __n, long double __z)
- std::complex< long double > sph_hankel_2l (unsigned int __n, std::complex< long double > __x)
- template<typename _Ttheta , typename _Tphi >
  std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- std::complex< float > sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)
- std::complex< long double > sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_1 (_Tpnu __nu, _Tp __x)
- float theta_1f (float __nu, float __x)
- long double theta_1l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_2 (_Tpnu __nu, _Tp __x)
- float theta_2f (float __nu, float __x)
- long double theta_2l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_3 (_Tpnu __nu, _Tp __x)
- float theta_3f (float __nu, float __x)
- long double theta_3l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_4 (_Tpnu __nu, _Tp __x)
- float theta_4f (float __nu, float __x)
- long double theta_4l (long double __nu, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > theta_c (_Tpk __k, _Tp __x)
- float theta_cf (float __k, float __x)
- long double theta_cl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > theta_d (_Tpk __k, _Tp __x)
- float theta_df (float __k, float __x)
- long double theta_dl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > theta_n (_Tpk __k, _Tp __x)
- float theta_nf (float __k, float __x)
- long double theta_nl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > theta_s (_Tpk __k, _Tp __x)
- float theta_sf (float __k, float __x)

- long double theta_sl (long double __k, long double __x)
- template< typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Trho, _Tphi > zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float zernikef (unsigned int __n, int __m, float __rho, float __phi)
- long double zernikel (unsigned int __n, int __m, long double __rho, long double __phi)

8.2 std Namespace Reference

Namespaces

- __detail

Functions

- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)
- float assoc_laguerref (unsigned int __n, unsigned int __m, float __x)
- long double assoc_laguerrel (unsigned int __n, unsigned int __m, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type assoc_legendre (unsigned int __l, unsigned int __m, _Tp __x)
- float assoc_legendref (unsigned int __l, unsigned int __m, float __x)
- long double assoc_legendrel (unsigned int __l, unsigned int __m, long double __x)
- template< typename _Tpa , typename _Tpb >
  __gnu_cxx::__promote_2< _Tpa, _Tpb >::__type beta (_Tpa __a, _Tpb __b)
- float betaf (float __a, float __b)
- long double betal (long double __a, long double __b)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type comp_ellint_1 (_Tp __k)
- float comp_ellint_1f (float __k)
- long double comp_ellint_1l (long double __k)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type comp_ellint_2 (_Tp __k)
- float comp_ellint_2f (float __k)
- long double comp_ellint_2l (long double __k)
- template< typename _Tp , typename _Tpn >
  __gnu_cxx::__promote_2< _Tp, _Tpn >::__type comp_ellint_3 (_Tp __k, _Tpn __nu)
- float comp_ellint_3f (float __k, float __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `float` modulus k.
- long double comp_ellint_3l (long double __k, long double __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `long double` modulus k.
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_bessel_i (_Tpnu __nu, _Tp __x)
- float cyl_bessel_if (float __nu, float __x)
- long double cyl_bessel_il (long double __nu, long double __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_bessel_j (_Tpnu __nu, _Tp __x)
- float cyl_bessel_jf (float __nu, float __x)
- long double cyl_bessel_jl (long double __nu, long double __x)

- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_bessel_k (_Tpnu __nu, _Tp __x)
- float cyl_bessel_kf (float __nu, float __x)
- long double cyl_bessel_kl (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_neumann (_Tpnu __nu, _Tp __x)
- float cyl_neumannf (float __nu, float __x)
- long double cyl_neumannl (long double __nu, long double __x)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint_1 (_Tp __k, _Tpp __phi)
- float ellint_1f (float __k, float __phi)
- long double ellint_1l (long double __k, long double __phi)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint_2 (_Tp __k, _Tpp __phi)
- float ellint_2f (float __k, float __phi)

  Return the incomplete elliptic integral of the second kind $E(k,\phi)$ for `float` argument.
- long double ellint_2l (long double __k, long double __phi)

  Return the incomplete elliptic integral of the second kind $E(k,\phi)$.
- template<typename _Tp , typename _Tpn , typename _Tpp >
  __gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type ellint_3 (_Tp __k, _Tpn __nu, _Tpp __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$.
- float ellint_3f (float __k, float __nu, float __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$ for `float` argument.
- long double ellint_3l (long double __k, long double __nu, long double __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$.
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type expint (_Tp __x)
- float expintf (float __x)
- long double expintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type hermite (unsigned int __n, _Tp __x)
- float hermitef (unsigned int __n, float __x)
- long double hermitel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type laguerre (unsigned int __n, _Tp __x)
- float laguerref (unsigned int __n, float __x)
- long double laguerrel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type legendre (unsigned int __l, _Tp __x)
- float legendref (unsigned int __l, float __x)
- long double legendrel (unsigned int __l, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type riemann_zeta (_Tp __s)
- float riemann_zetaf (float __s)
- long double riemann_zetal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type sph_bessel (unsigned int __n, _Tp __x)
- float sph_besself (unsigned int __n, float __x)
- long double sph_bessell (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)

- float sph_legendref (unsigned int __l, unsigned int __m, float __theta)
- long double sph_legendrel (unsigned int __l, unsigned int __m, long double __theta)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type sph_neumann (unsigned int __n, _Tp __x)
- float sph_neumannf (unsigned int __n, float __x)
- long double sph_neumannl (unsigned int __n, long double __x)

8.3 std::__detail Namespace Reference

Classes

- struct _Factorial_table

Enumerations

- enum { SININT, COSINT }

Functions

- template< typename _Tp >
  void __airy (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)

  Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi(x)$ respectively.
- template< typename _Tp >
  void __airy (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)

  This function computes the Airy function $Ai(z)$ and its first derivative in the complex z-plane.
- template< typename _Tp >
  std::complex< _Tp > __airy_ai (std::complex< _Tp > __z)

  Return the complex Airy Ai function.
- template< typename _Tp >
  void __airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)

  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.
- template< typename _Tp >
  void __airy_asymp_absarg_ge_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, int __sign=-1)

  This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from abs(z).
- template< typename _Tp >
  void __airy_asymp_absarg_lt_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip)

  This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $|z|$.
- template< typename _Tp >
  void __airy_bessel_i (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ip1d3, std↩
  ::complex< _Tp > &_Im1d3, std::complex< _Tp > &_Ip2d3, std::complex< _Tp > &_Im2d3)

- template< typename _Tp >
void __airy_bessel_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)

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

- template< typename _Tp >
std::complex< _Tp > __airy_bi (std::complex< _Tp > __z)

  Return the complex Airy Bi function.

- template< typename _Tp >
void __airy_hyperg_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)

  This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = +-1/3$ *and* $\nu + -2/3$. *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.*

- template< typename _Tp >
_Tp __assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)

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

- template< typename _Tp >
_Tp __assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)

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

- template< typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp __bernoulli (int __n)

  This returns Bernoulli number B_n.

- template< typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp __bernoulli_2n (int __n)

  This returns Bernoulli number B_n.

- template< typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)

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

- template< typename _Tp >
_Tp __beta (_Tp __a, _Tp __b)

  Return the beta function $B(a,b)$.

- template< typename _Tp >
_Tp __beta_gamma (_Tp __a, _Tp __b)

  Return the beta function: $B(a,b)$.

- template< typename _Tp >
_Tp __beta_inc (_Tp __a, _Tp __b, _Tp __x)

- template< typename _Tp >
_Tp __beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)

- template< typename _Tp >
_Tp __beta_lgamma (_Tp __a, _Tp __b)

  Return the beta function $B(a,b)$ *using the log gamma functions.*

- template< typename _Tp >
_Tp __beta_product (_Tp __a, _Tp __b)

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

- template< typename _Tp >
_Tp __bincoef (unsigned int __n, unsigned int __k)

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

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

.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)

  Return the complementary binomial cumulative distribution function.
- template<typename _Tp >
  _Tp __bose_einstein (_Tp __s, _Tp __x)
- template<typename _Tp >
  _Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)
- template<typename _Tp >
  _Tp __chebyshev_t (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp __chebyshev_u (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp __chebyshev_v (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp __chebyshev_w (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __chi_squared_pdf (_Tp __chi2, unsigned int __nu)

  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.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __chi_squared_pdfc (_Tp __chi2, unsigned int __nu)

  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.
- template<typename _Tp >
  std::pair< _Tp, _Tp > __chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)

  This function returns the hyperbolic cosine $Ci(x)$ *and hyperbolic sine* $Si(x)$ *integrals as a pair.*
- template<typename _Tp >
  void __chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)

  This function computes the hyperbolic cosine $Chi(x)$ *and hyperbolic sine* $Shi(x)$ *integrals by continued fraction for positive argument.*
- template<typename _Tp >
  void __chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)

  This function computes the hyperbolic cosine $Chi(x)$ *and hyperbolic sine* $Shi(x)$ *integrals by series summation for positive argument.*
- template<typename _Tp >
  std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __clamp_pi (std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp __clausen (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp __clausen_c (unsigned int __m, std::complex< _Tp > __w)

- template<typename _Tp >
  _Tp __clausen_c (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp __clausen_s (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp __clausen_s (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp __comp_ellint_1 (_Tp __k)

  Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __comp_ellint_2 (_Tp __k)

  Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __comp_ellint_3 (_Tp __k, _Tp __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu) = \Pi(k,\nu,\pi/2)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __comp_ellint_d (_Tp __k)
- template<typename _Tp >
  _Tp __comp_ellint_rf (_Tp __x, _Tp __y)
- template<typename _Tp >
  _Tp __comp_ellint_rg (_Tp __x, _Tp __y)
- template<typename _Tp >
  _Tp __conf_hyperg (_Tp __a, _Tp __c, _Tp __x)

  Return the confluent hypergeometric function ${}_1F_1(a;c;x)$.
- template<typename _Tp >
  _Tp __conf_hyperg_lim (_Tp __c, _Tp __x)

  Return the confluent hypergeometric limit function ${}_0F_1(-;c;x)$.
- template<typename _Tp >
  _Tp __conf_hyperg_lim_series (_Tp __c, _Tp __x)

  This routine returns the confluent hypergeometric limit function by series expansion.
- template<typename _Tp >
  _Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)

  Return the hypergeometric function ${}_1F_1(a;c;x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- template<typename _Tp >
  _Tp __conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)

  This routine returns the confluent hypergeometric function by series expansion.
- template<typename _Tp >
  _Tp __coshint (const _Tp __x)

  Return the hyperbolic cosine integral $li(x)$.
- template<typename _Tp >
  std::complex< _Tp > __cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Bessel function.
- template<typename _Tp >
  _Tp __cyl_bessel_i (_Tp __nu, _Tp __x)

  Return the regular modified Bessel function of order ν: $I_\nu(x)$.
- template<typename _Tp >
  _Tp __cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)

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

- template<typename _Tp >
  void __cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)

  Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- template<typename _Tp >
  void __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)

  This routine computes the asymptotic modified cylindrical Bessel and functions of order nu: $I_\nu(x)$, $N_\nu(x)$. Use this for $x >> nu^2 + 1$.
- template<typename _Tp >
  void __cyl_bessel_ik_steed (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)

  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.
- template<typename _Tp >
  _Tp __cyl_bessel_j (_Tp __nu, _Tp __x)

  Return the Bessel function of order ν: $J_\nu(x)$.
- template<typename _Tp >
  void __cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)

  Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- template<typename _Tp >
  void __cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)

  This routine computes the asymptotic cylindrical Bessel and Neumann functions of order nu: $J_\nu(x)$, $N_\nu(x)$. Use this for $x >> nu^2 + 1$.
- template<typename _Tp >
  void __cyl_bessel_jn_steed (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)

  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.
- template<typename _Tp >
  _Tp __cyl_bessel_k (_Tp __nu, _Tp __x)

  Return the irregular modified Bessel function $K_\nu(x)$ of order ν.
- template<typename _Tp >
  std::complex< _Tp > __cyl_hankel_1 (_Tp __nu, _Tp __x)

  Return the cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$.
- template<typename _Tp >
  std::complex< _Tp > __cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Hankel function of the first kind.
- template<typename _Tp >
  std::complex< _Tp > __cyl_hankel_2 (_Tp __nu, _Tp __x)

  Return the cylindrical Hankel function of the second kind $H^{(2)}_n u(x)$.
- template<typename _Tp >
  std::complex< _Tp > __cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Hankel function of the second kind.
- template<typename _Tp >
  std::complex< _Tp > __cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Neumann function.
- template<typename _Tp >
  _Tp __cyl_neumann_n (_Tp __nu, _Tp __x)

  Return the Neumann function of order ν: $N_\nu(x)$.
- template<typename _Tp >
  _Tp __dawson (_Tp __x)

  Return the Dawson integral, $F(x)$, for real argument x.

- template<typename _Tp >
  _Tp __dawson_cont_frac (_Tp __x)

  Compute the Dawson integral using a sampling theorem representation.
- template<typename _Tp >
  _Tp __dawson_series (_Tp __x)

  Compute the Dawson integral using the series expansion.
- template<typename _Tp >
  void __debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)
- template<typename _Tp >
  _Tp __dilog (_Tp __x)

  Compute the dilogarithm function $Li_2(x)$ by summation for x <= 1.
- template<typename _Tp >
  _Tp __dirichlet_beta (std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp __dirichlet_beta (_Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp __dirichlet_eta (_Tp __w)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)

  Return the double factorial of the integer n.
- template<typename _Tp >
  _Tp __ellint_1 (_Tp __k, _Tp __phi)

  Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __ellint_2 (_Tp __k, _Tp __phi)

  Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)
- template<typename _Tp >
  _Tp __ellint_d (_Tp __k, _Tp __phi)
- template<typename _Tp >
  _Tp __ellint_el1 (_Tp __x, _Tp __k_c)
- template<typename _Tp >
  _Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)
- template<typename _Tp >
  _Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)
- template<typename _Tp >
  _Tp __ellint_rc (_Tp __x, _Tp __y)

  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.
- template<typename _Tp >
  _Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)

  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.
- template<typename _Tp >
  _Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)

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

- template<typename _Tp >
  _Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)

  Return the symmetric Carlson elliptic function of the second kind $R_G(x,y,z)$.
- template<typename _Tp >
  _Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)

  Return the Carlson elliptic function $R_J(x,y,z,p)$ of the third kind.
- template<typename _Tp >
  _Tp __ellnome (_Tp __k)
- template<typename _Tp >
  _Tp __ellnome_k (_Tp __k)
- template<typename _Tp >
  _Tp __ellnome_series (_Tp __k)
- template<typename _Tp >
  _Tp __expint (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$.
- template<typename _Tp >
  _Tp __expint (_Tp __x)

  Return the exponential integral $Ei(x)$.
- template<typename _Tp >
  _Tp __expint_asymp (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large argument.
- template<typename _Tp >
  _Tp __expint_E1 (_Tp __x)

  Return the exponential integral $E_1(x)$.
- template<typename _Tp >
  _Tp __expint_E1_asymp (_Tp __x)

  Return the exponential integral $E_1(x)$ by asymptotic expansion.
- template<typename _Tp >
  _Tp __expint_E1_series (_Tp __x)

  Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.
- template<typename _Tp >
  _Tp __expint_Ei (_Tp __x)

  Return the exponential integral $Ei(x)$.
- template<typename _Tp >
  _Tp __expint_Ei_asymp (_Tp __x)

  Return the exponential integral $Ei(x)$ by asymptotic expansion.
- template<typename _Tp >
  _Tp __expint_Ei_series (_Tp __x)

  Return the exponential integral $Ei(x)$ by series summation.
- template<typename _Tp >
  _Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ by continued fractions.
- template<typename _Tp >
  _Tp __expint_En_recursion (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.
- template<typename _Tp >
  _Tp __expint_En_series (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp __expint_large_n (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large order.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)

  Return the factorial of the integer n.
- template<typename _Tp >
  _Tp __fermi_dirac (_Tp __s, _Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)

  Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)

  Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2.
- template<typename _Tp >
  void __fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)

  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.

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

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

  .
- template<typename _Tp >
  bool __fpequal (const _Tp &__a, const _Tp &__b)
- template<typename _Tp >
  bool __fpimag (const std::complex< _Tp > &__w)
- template<typename _Tp >
  bool __fpimag (const _Tp)
- template<typename _Tp >
  bool __fpreal (const std::complex< _Tp > &__w)
- template<typename _Tp >
  bool __fpreal (const _Tp)
- template<typename _Tp >
  std::complex< _Tp > __fresnel (const _Tp __x)

  Return the Fresnel cosine and sine integrals as a complex number $f[C(x) + iS(x) $f].
- template<typename _Tp >
  void __fresnel_cont_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)

  This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- template<typename _Tp >
  void __fresnel_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)

  This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.
- template<typename _Tp >
  _Tp __gamma (_Tp __x)

  Return $\Gamma(x)$.
- template<typename _Tp >
  std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)
- template<typename _Tp >
  _Tp __gamma_l (_Tp __a, _Tp __x)

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

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

.

- template<typename _Tp >
  _Tp __gamma_p (_Tp __a, _Tp __x)

  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.
- template<typename _Tp >
  _Tp __gamma_q (_Tp __a, _Tp __x)

  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.
- template<typename _Tp >
  std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)
- template<typename _Tp >
  void __gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)

  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.*
- template<typename _Tp >
  _Tp __gamma_u (_Tp __a, _Tp __x)

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

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

  .
- template<typename _Tp >
  _Tp __gauss (_Tp __x)
- template<typename _Tp >
  _Tp __gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)
- template<typename _Tp >
  void __hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std↩ ::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

- template<typename _Tp >
  void __hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)
- template<typename _Tp >
  void __hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< ↩ _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)

  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.
- template<typename _Tp >
  void __hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

  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.
- template<typename _Tp >
  void __hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &↩ _H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

  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.
- template<typename _Tp >
  void __hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std↩ ::complex< _Tp > &__etrat, std::complex< _Tp > &_Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &_Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)

  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.
- template<typename _Tp >
  void __hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > ↩ __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > _Aip, std::complex< _Tp > __o4dp, std↩ ::complex< _Tp > _Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &↩ _H1sum, std::complex< _Tp > &_H1psum, std::complex< _Tp > &_H2sum, std::complex< _Tp > &_H2psum)

  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.
- template<typename _Tp >
  _Tp __heuman_lambda (_Tp __k, _Tp __phi)
- template<typename _Tp >
  _Tp __hurwitz_zeta (_Tp __s, _Tp __a)

  Return the Hurwitz zeta function $\zeta(s,a)$ for all s != 1 and a > -1.
- template<typename _Tp >
  std::complex< _Tp > __hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
  _Tp __hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)

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

- template<typename _Tp >
  std::complex< _Tp > __hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp _Z, const _Tp __r, const _Tp __theta, const _Tp __phi)
- template<typename _Tp >
  _Tp __hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

  Return the hypergeometric function $_2F_1(a, b; c; x)$.
- template<typename _Tp >
  _Tp __hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)

  Return the hypergeometric function $_2F_1(a, b; c; x)$ *by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.*
- template<typename _Tp >
  _Tp __hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

  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.*
- template<typename _Tp >
  _Tp __hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

  Return the hypergeometric function $_2F_1(a, b; c; x)$ *by series expansion.*
- template<typename _Tp >
  std::tuple< _Tp, _Tp, _Tp > __jacobi_sncndn (_Tp __k, _Tp __u)
- template<typename _Tp >
  _Tp __jacobi_zeta (_Tp __k, _Tp __phi)
- template<typename _Tp >
  _Tp __laguerre (unsigned int __n, _Tp __x)

  This routine returns the Laguerre polynomial of order n: $L_n(x)$.
- template<typename _Tp >
  _Tp __legendre_q (unsigned int __l, _Tp __x)

  Return the Legendre function of the second kind by upward recursion on order l.
- template<typename _Tp >
  _Tp __log_bincoef (unsigned int __n, unsigned int __k)

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

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

  .
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)

  Return the logarithm of the double factorial of the integer n.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)

  Return the logarithm of the factorial of the integer n.
- template<typename _Tp >
  _Tp __log_gamma (_Tp __x)

  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.*
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)

  Return $log(\Gamma(x))$ *by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.*
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_gamma_lanczos (_Tp __x)

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

- template<typename _Tp >
  _Tp __log_gamma_sign (_Tp __x)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __log_gamma_spouge (_Tp __z)

  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

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

  .

- template<typename _Tp >
  _Tp __log_pochhammer_l (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp __log_pochhammer_u (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp __logint (const _Tp __x)

  Return the logarithmic integral $li(x)$.

- template<typename _Tp >
  _Tp __owens_t (_Tp __h, _Tp __a)

- template<typename _Tp >
  _Tp __pochhammer_l (_Tp __a, _Tp __n)

  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].

- template<typename _Tp >
  _Tp __pochhammer_u (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp __poly_hermite (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp __poly_hermite_asymp (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp __poly_hermite_recursion (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp __poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)
- template<typename _Tpa , typename _Tp >
  _Tp __poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)

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

- template<typename _Tpa , typename _Tp >
  _Tp __poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)

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

- template<typename _Tpa , typename _Tp >
  _Tp __poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)

  This routine returns the associated Laguerre polynomial of order n, degree $\alpha > -1$ for large n. Abramowitz & Stegun, 13.5.21.

- template<typename _Tpa , typename _Tp >
  _Tp __poly_laguerre_recursion (unsigned int __n, _Tpa __alpha1, _Tp __x)

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

- template<typename _Tp >
  _Tp __poly_legendre_p (unsigned int __l, _Tp __x)

  Return the Legendre polynomial by upward recursion on order l.

- template<typename _Tp >
  _Tp __poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)
- template<typename _Tp >
  _Tp __polylog (_Tp __s, _Tp __x)

- template<typename _Tp >
  std::complex< _Tp > __polylog (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp , typename ArgType >
  __gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > __polylog_exp (_Tp __s, ArgType __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_int_neg (int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_int_neg (const int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg (int __s, std::complex< _Tp > __w)
- template<typename _Tp , int __sigma>
  std::complex< _Tp > __polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)
- template<typename _Tp , int __sigma>
  std::complex< _Tp > __polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)
- template<typename _PowTp , typename _Tp >
  _Tp __polylog_exp_negative_real_part (_PowTp __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos (unsigned int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, _Tp __w)
- template<typename _Tp >
  _Tp __psi (_Tp __x)

  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:

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

  .
- template<typename _Tp >
  _Tp __psi (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp __psi_asymp (_Tp __x)

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

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

  .
- template<typename _Tp >
  _Tp __psi_series (_Tp __x)

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

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

  .
- template<typename _Tp >
  _Tp __riemann_zeta (_Tp __s)

  Return the Riemann zeta function $\zeta(s)$.
- template<typename _Tp >
  _Tp __riemann_zeta_alt (_Tp __s)

  Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s $>$ 0.
- template<typename _Tp >
  _Tp __riemann_zeta_euler_maclaurin (_Tp __s)

  Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s $>$ 0.
- template<typename _Tp >
  _Tp __riemann_zeta_glob (_Tp __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.
- template<typename _Tp >
  _Tp __riemann_zeta_m_1 (_Tp __s)

  Return the Riemann zeta function $\zeta(s) - 1$.
- template<typename _Tp >
  _Tp __riemann_zeta_m_1_sum (_Tp __s)

  Return the Riemann zeta function $\zeta(s) - 1$ by summation for s $>$ 1. This is a small remainder for large s.
- template<typename _Tp >
  _Tp __riemann_zeta_product (_Tp __s)

  Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.
- template<typename _Tp >
  _Tp __riemann_zeta_sum (_Tp __s)

  Compute the Riemann zeta function $\zeta(s)$ by summation for s $>$ 1.
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinc (_Tp __a, _Tp __x)

  Return the generalized sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinc (_Tp __x)

  Return the normalized sinus cardinal function

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

  .

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinc_pi (_Tp __x)

  Return the unnormalized sinus cardinal function

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

  .
- template<typename _Tp >
  std::pair< _Tp, _Tp > __sincosint (_Tp __x)

  This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.
- template<typename _Tp >
  void __sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)

  This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- template<typename _Tp >
  void __sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)

  This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- template<typename _Tp >
  void __sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)

  This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinhc (_Tp __a, _Tp __x)

  Return the generalized hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinhc (_Tp __x)

  Return the normalized hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __sinhc_pi (_Tp __x)

  Return the unnormalized hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  _Tp __sinhint (const _Tp __x)

  Return the hyperbolic sine integral $li(x)$.
- template<typename _Tp >
  _Tp __sph_bessel (unsigned int __n, _Tp __x)

  Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x.
- template<typename _Tp >
  std::complex< _Tp > __sph_bessel (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Bessel function.
- template<typename _Tp >
  void __sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)

  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.

- template<typename _Tp >
  void __sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)

  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.
- template<typename _Tp >
  void __sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)

  Helper to compute complex spherical Hankel functions and their derivatives.
- template<typename _Tp >
  std::complex< _Tp > __sph_hankel_1 (unsigned int __n, _Tp __x)

  Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- template<typename _Tp >
  std::complex< _Tp > __sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Hankel function of the first kind.
- template<typename _Tp >
  std::complex< _Tp > __sph_hankel_2 (unsigned int __n, _Tp __x)

  Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.
- template<typename _Tp >
  std::complex< _Tp > __sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Hankel function of the second kind.
- template<typename _Tp >
  std::complex< _Tp > __sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)

  Return the spherical harmonic function.
- template<typename _Tp >
  _Tp __sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)

  Return the spherical associated Legendre function.
- template<typename _Tp >
  _Tp __sph_neumann (unsigned int __n, _Tp __x)

  Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x.
- template<typename _Tp >
  std::complex< _Tp > __sph_neumann (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Neumann function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __student_t_cdf (_Tp __t, unsigned int __nu)

  Return the Students T probability function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __student_t_cdfc (_Tp __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template<typename _Tp >
  _Tp __theta_1 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_2 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_2_asymp (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_2_sum (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_3 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_3_asymp (_Tp __nu, _Tp __x)

- template<typename _Tp >
  _Tp __theta_3_sum (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_4 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp __theta_c (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp __theta_d (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp __theta_n (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp __theta_s (_Tp __k, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)
- template<typename _Tp >
  _Tp __znorm1 (_Tp __x)
- template<typename _Tp >
  _Tp __znorm2 (_Tp __x)
- template<typename _Tp = double>
  _Tp evenzeta (unsigned int __k)

Variables

- constexpr size_t _Num_Euler_Maclaurin_zeta = 100
- constexpr _Factorial_table< long double > _S_double_factorial_table [301]
- constexpr long double _S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]
- constexpr _Factorial_table< long double > _S_factorial_table [171]
- constexpr _Factorial_table< long double > _S_neg_double_factorial_table [999]
- template<typename _Tp >
  constexpr std::size_t _S_num_double_factorials = 0
- template<>
  constexpr std::size_t _S_num_double_factorials< double > = 301
- template<>
  constexpr std::size_t _S_num_double_factorials< float > = 57
- template<>
  constexpr std::size_t _S_num_double_factorials< long double > = 301
- template<typename _Tp >
  constexpr std::size_t _S_num_factorials = 0
- template<>
  constexpr std::size_t _S_num_factorials< double > = 171
- template<>
  constexpr std::size_t _S_num_factorials< float > = 35
- template<>
  constexpr std::size_t _S_num_factorials< long double > = 171
- template<typename _Tp >
  constexpr std::size_t _S_num_neg_double_factorials = 0
- template<>
  constexpr std::size_t _S_num_neg_double_factorials< double > = 150
- template<>
  constexpr std::size_t _S_num_neg_double_factorials< float > = 27
- template<>
  constexpr std::size_t _S_num_neg_double_factorials< long double > = 999

- constexpr size_t _S_num_zetam1 = 33
- constexpr long double _S_zetam1 [_S_num_zetam1]

8.3.1 Enumeration Type Documentation

8.3.1.1 anonymous enum

Enumerator

SININT

COSINT

Definition at line 42 of file sf_trigint.tcc.

8.3.2 Function Documentation

8.3.2.1 template<typename _Tp > void std::__detail::__airy (_Tp *__z,* _Tp & *_Ai,* _Tp & *_Bi,* _Tp & *_Aip,* _Tp & *_Bip*)

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

Parameters

__z	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.

Definition at line 497 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik(), and __cyl_bessel_jn().

8.3.2.2 template<typename _Tp > void std::__detail::__airy (const std::complex< _Tp > & *__z,* _Tp *__eps,* std::complex< _Tp > & *_Ai,* std::complex< _Tp > & *_Aip,* std::complex< _Tp > & *_Bi,* std::complex< _Tp > & *_Bip*)

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 $\zeta = -\frac{2}{3}z^{3/2}$ 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
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

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

and

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

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).
out	*_Bi*	The value computed for Bi(z).
out	*_Bip*	The value computed for Bi'(z).

Definition at line 1002 of file sf_airy.tcc.

References __airy_asymp_absarg_ge_pio3(), __airy_asymp_absarg_lt_pio3(), __airy_bessel_i(), __airy_bessel_k(), and __airy_hyperg_rational().

Referenced by __airy_ai(), __airy_bi(), __hankel_uniform_outer(), and __poly_hermite_asymp().

8.3.2.3 template<typename _Tp > std::complex<_Tp> std::__detail::__airy_ai (std::complex< _Tp > *__z*)

Return the complex Airy Ai function.

Definition at line 1139 of file sf_airy.tcc.

References __airy().

8.3.2.4 template<typename _Tp > void std::__detail::__airy_arg (std::complex< _Tp > *__num2d3,* std::complex< _Tp > *__zeta,* std::complex< _Tp > & *__argp,* std::complex< _Tp > & *__argm*)

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.	

Definition at line 241 of file sf_hankel.tcc.

Referenced by __hankel_uniform_outer().

8.3.2.5 template<typename _Tp > void std::__detail::__airy_asymp_absarg_ge_pio3 (std::complex< _Tp > *__z*, std::complex< _Tp > & *_Ai*, std::complex< _Tp > & *_Aip*, int *__sign* = -1)

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

Note that for speed and since this function is called by another, checks for valid arguments are not made.

See also

Digital Library of Mathematical Functions 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 $\|z\| > 15$ and $\|arg(z)\| < 2\pi/3$.
`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 $\|arg(z)\| < \pi$. The value +1 gives the Airy Bi functions for $\|arg(z)\| < \pi/3$.

Definition at line 71 of file sf_airy.tcc.

Referenced by __airy().

8.3.2.6 template<typename _Tp > void std::__detail::__airy_asymp_absarg_lt_pio3 (std::complex< _Tp > *__z*, std::complex< _Tp > & *_Ai*, std::complex< _Tp > & *_Aip*)

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

Note that for speed and since this function is called by another, checks for valid arguments are not made. This function assumes $|z| > 15$ and $|arg(-z)| < pi/3$.

Parameters

`in`	*__z*	The value at which the Airy function and its derivative are evaluated.
`out`	*_Ai*	The computed value of the Airy function $Ai(z)$.
`out`	*_Aip*	The computed value of the Airy function derivative $Ai'(z)$.

Definition at line 186 of file sf_airy.tcc.

Referenced by __airy().

8.3.2.7 template<typename _Tp > void std::__detail::__airy_bessel_i (const std::complex< _Tp > & *__z*, _Tp *__eps*, std::complex< _Tp > & *_Ip1d3*, std::complex< _Tp > & *_Im1d3*, std::complex< _Tp > & *_Ip2d3*, std::complex< _Tp > & *_Im2d3*)

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 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 $I_(-2/3)(z)$.

Definition at line 390 of file sf_airy.tcc.

Referenced by __airy().

8.3.2.8 template<typename _Tp > void std::__detail::__airy_bessel_k (const std::complex< _Tp > & *__z*, _Tp *__eps*, std::complex< _Tp > & *_Kp1d3*, std::complex< _Tp > & *_Kp2d3*)

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 $|\arg(z)| <= pi$, 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 $|\arg(z)| <= 3pi/4$ 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 $|\arg(z)| <= 3pi/4$.
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 $\arg(z) = 3pi/4$, 20 iterations should give 7 or 8 decimals of accuracy.

Definition at line 604 of file sf_airy.tcc.

Referenced by __airy().

8.3.2.9 template<typename _Tp > std::complex<_Tp> std::__detail::__airy_bi (std::complex< _Tp > *__z*)

Return the complex Airy Bi function.

Definition at line 1152 of file sf_airy.tcc.

References __airy().

8.3.2.10 template<typename _Tp > void std::__detail::__airy_hyperg_rational (const std::complex< _Tp > & *__z,* std::complex< _Tp > & *_Ai,* std::complex< _Tp > & *_Aip,* std::complex< _Tp > & *_Bi,* std::complex< _Tp > & *_Bip*)

This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = + - 1/3$ and $\nu + -2/3$. 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 poly- nomials 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 speed and since this function is called 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, $|z|$ must be small to ensure sufficient accuracy of the computed results.

out	*_Ai*	The Airy function $Ai(z)$.
out	*_Aip*	The Airy function derivative $Ai'(z)$.
out	*_Bi*	The Airy function $Bi(z)$.
out	*_Bip*	The Airy function derivative $Bi'(z)$.

Definition at line 787 of file sf_airy.tcc.

Referenced by __airy().

8.3.2.11 template<typename _Tp > _Tp std::__detail::__assoc_laguerre (unsigned int *__n*, unsigned int *__m*, _Tp *__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:

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

Template Parameters

_Tp	The type of the parameter

Parameters

__n	The order
__m	The degree
__x	The argument

Returns

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

Definition at line 301 of file sf_laguerre.tcc.

Referenced by __hydrogen().

8.3.2.12 template<typename _Tp > _Tp std::__detail::__assoc_legendre_p (unsigned int *__l*, unsigned int *__m*, _Tp *__x*)

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. $\|x\| <= 1$.

Definition at line 175 of file sf_legendre.tcc.

References __poly_legendre_p().

8.3.2.13 template< typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int *__n*)

This returns Bernoulli number B_n.

Parameters

__n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

Definition at line 1673 of file sf_gamma.tcc.

8.3.2.14 template< typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int *__n*)

This returns Bernoulli number B_n.

Parameters

__n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

Definition at line 1685 of file sf_gamma.tcc.

8.3.2.15 template< typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int *__n*)

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.

Definition at line 1608 of file sf_gamma.tcc.

8.3.2.16 template<typename _Tp > _Tp std::__detail::__beta (_Tp *__a*, _Tp *__b*)

Return the beta function $B(a, b)$.

The beta function is defined by

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

Parameters

__a	The first argument of the beta function.
__b	The second argument of the beta function.

Returns

The beta function.

Definition at line 173 of file sf_beta.tcc.

References __beta_lgamma().

Referenced by __poly_jacobi(), __gnu_cxx::jacobi(), __gnu_cxx::jacobif(), and __gnu_cxx::jacobil().

8.3.2.17 template<typename _Tp > _Tp std::__detail::__beta_gamma (_Tp *__a*, _Tp *__b*)

Return the beta function: $B(a, b)$.

The beta function is defined by

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

Parameters

__a	The first argument of the beta function.
__b	The second argument of the beta function.

Returns

The beta function.

Definition at line 75 of file sf_beta.tcc.

References __gamma().

8.3.2.18 template<typename _Tp > _Tp std::__detail::__beta_inc (_Tp *__a*, _Tp *__b*, _Tp *__x*)

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

The regularized incomplete beta function is defined by:

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)}$$

where

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

is the non-regularized beta function and $B(a, b)$ is the usual beta function.

Parameters

__a	The first parameter
__b	The second parameter
__x	The argument

Definition at line 262 of file sf_beta.tcc.

References __beta_inc_cont_frac().

Referenced by __binomial_cdf(), __binomial_cdfc(), __fisher_f_cdf(), __fisher_f_cdfc(), __student_t_cdf(), and __↩ student_t_cdfc().

8.3.2.19 template<typename _Tp > _Tp std::__detail::__beta_inc_cont_frac (_Tp *__a*, _Tp *__b*, _Tp *__x*)

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

Parameters

__a	The first parameter
__b	The second parameter
__x	The argument

Definition at line 193 of file sf_beta.tcc.

Referenced by __beta_inc().

8.3.2.20 template<typename _Tp > _Tp std::__detail::__beta_lgamma (_Tp *__a*, _Tp *__b*)

Return the beta function $B(a, b)$ using the log gamma functions.

The beta function is defined by

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

Parameters

__a	The first argument of the beta function.
__b	The second argument of the beta function.

Returns

The beta function.

Definition at line 109 of file sf_beta.tcc.

References __log_gamma().

Referenced by __beta().

8.3.2.21 template<typename _Tp > _Tp std::__detail::__beta_product (_Tp *__a*, _Tp *__b*)

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

The beta function is defined by

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

Here, we employ the product form:

$$B(a,b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1+(a+b)/k}{(1+a/k)(1+b/k)}$$

Parameters

__a	The first argument of the beta function.
__b	The second argument of the beta function.

Returns

The beta function.

Definition at line 140 of file sf_beta.tcc.

8.3.2.22 template<typename _Tp > _Tp std::__detail::__bincoef (unsigned int *__n*, unsigned int *__k*)

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.

Definition at line 1888 of file sf_gamma.tcc.

8.3.2.23 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf (_Tp *__p*, unsigned int *__n*, unsigned int *__k*)

Return the binomial cumulative distribution function.

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

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

Parameters

__p	
__n	
__k	

Definition at line 405 of file sf_beta.tcc.

References __beta_inc().

8.3.2.24 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)

Return the complementary binomial cumulative distribution function.

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

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

Parameters

__p	
__n	
__k	

Definition at line 435 of file sf_beta.tcc.

References __beta_inc().

8.3.2.25 template<typename _Tp > _Tp std::__detail::__bose_einstein (_Tp __s, _Tp __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),

Definition at line 1401 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.26 template<typename _Tp > _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)

Return a Chebyshev polynomial of non-negative order n and real argument x by the recursion

$$C_n(x) = 2xC_{n-1} - C_{n-2}$$

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$
_C0	The value of the zeroth-order Chebyshev polynomial at x
_C1	The value of the first-order Chebyshev polynomial at x

Definition at line 57 of file sf_chebyshev.tcc.

Referenced by __chebyshev_t(), __chebyshev_u(), __chebyshev_v(), and __chebyshev_w().

8.3.2.27 template$<$typename _Tp $>$ _Tp std::__detail::__chebyshev_t (unsigned int *__n*, _Tp *__x*)

Return the Chebyshev polynomial of the first kind $T_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the first kind is defined by:

$$T_n(x) = \cos(n\theta)$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 85 of file sf_chebyshev.tcc.

References __chebyshev_recur().

8.3.2.28 template$<$typename _Tp $>$ _Tp std::__detail::__chebyshev_u (unsigned int *__n*, _Tp *__x*)

Return the Chebyshev polynomial of the second kind $U_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the second kind is defined by:

$$U_n(x) = \frac{\sin\left[(n+1)\theta\right]}{\sin(\theta)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 114 of file sf_chebyshev.tcc.

References __chebyshev_recur().

8.3.2.29 template<typename _Tp > _Tp std::__detail::__chebyshev_v (unsigned int *__n*, _Tp *__x*)

Return the Chebyshev polynomial of the third kind $V_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the third kind is defined by:

$$V_n(x) = \frac{\cos\left[\left(n+\frac{1}{2}\right)\theta\right]}{\cos\left(\frac{\theta}{2}\right)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 144 of file sf_chebyshev.tcc.

References __chebyshev_recur().

8.3.2.30 template<typename _Tp > _Tp std::__detail::__chebyshev_w (unsigned int *__n*, _Tp *__x*)

Return the Chebyshev polynomial of the fourth kind $W_n(x)$ of non-negative order n and real argument x.

The Chebyshev polynomial of the fourth kind is defined by:

$$W_n(x) = \frac{\sin\left[\left(n+\frac{1}{2}\right)\theta\right]}{\sin\left(\frac{\theta}{2}\right)}$$

where $\theta = \arccos(x)$, $-1 <= x <= +1$.

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative integral order
__x	The real argument $-1 <= x <= +1$

Definition at line 174 of file sf_chebyshev.tcc.

References __chebyshev_recur().

8.3.2.31 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__chi_squared_pdf (_Tp *__chi2*, unsigned int *__nu*)

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:

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

Definition at line 2544 of file sf_gamma.tcc.

References __gamma_p().

8.3.2.32 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__chi_squared_pdfc (_Tp *__chi2*, unsigned int *__nu*)

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:

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

Definition at line 2568 of file sf_gamma.tcc.

References __gamma_q().

8.3.2.33 template<typename _Tp > std::pair<_Tp, _Tp> std::__detail::__chshint (_Tp *__x*, _Tp & *_Chi*, _Tp & *_Shi*)

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:

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

Definition at line 162 of file sf_hypint.tcc.

References __chshint_cont_frac(), and __chshint_series().

8.3.2.34 template<typename _Tp > void std::__detail::__chshint_cont_frac (_Tp *__t*, _Tp & *_Chi*, _Tp & *_Shi*)

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

Definition at line 50 of file sf_hypint.tcc.

Referenced by __chshint().

8.3.2.35 template<typename _Tp > void std::__detail::__chshint_series (_Tp *__t*, _Tp & *_Chi*, _Tp & *_Shi*)

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

Definition at line 93 of file sf_hypint.tcc.

Referenced by __chshint().

8.3.2.36 template<typename _Tp > std::complex<_Tp> std::__detail::__clamp_0_m2pi (std::complex< _Tp > *__w*)

Definition at line 136 of file sf_polylog.tcc.

Referenced by __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_real_neg(), and __polylog_exp_↩
real_pos().

8.3.2.37 template<typename _Tp > std::complex<_Tp> std::__detail::__clamp_pi (std::complex< _Tp > __w)

Definition at line 123 of file sf_polylog.tcc.

Referenced by __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_real_neg(), and __polylog_exp_real_pos().

8.3.2.38 template<typename _Tp > std::complex<_Tp> std::__detail::__clausen (unsigned int __m, std::complex< _Tp > __w)

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

Parameters

__m	The non-negative integral order.
__w	The complex argument.

Returns

The complex Clausen function.

Definition at line 1230 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.39 template<typename _Tp > _Tp std::__detail::__clausen (unsigned int __m, _Tp __w)

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.

Definition at line 1254 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.40 template<typename _Tp > _Tp std::__detail::__clausen_c (unsigned int __m, std::complex< _Tp > __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),

Definition at line 1329 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.41 template<typename _Tp > _Tp std::__detail::__clausen_c (unsigned int *__m*, _Tp *__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),

Definition at line 1354 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.42 template<typename _Tp > _Tp std::__detail::__clausen_s (unsigned int *__m*, std::complex< _Tp > *__w*)

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),

Definition at line 1279 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.43 template<typename _Tp > _Tp std::__detail::__clausen_s (unsigned int *__m*, _Tp *__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),

Definition at line 1304 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.44 template<typename _Tp > _Tp std::__detail::__comp_ellint_1 (_Tp *__k*)

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 modulus of the complete elliptic function.

Returns

The complete elliptic function of the first kind.

Definition at line 565 of file sf_ellint.tcc.

References __comp_ellint_rf().

Referenced by __ellint_1(), __ellnome_k(), __jacobi_zeta(), __theta_c(), __theta_d(), __theta_n(), and __theta_s().

8.3.2.45 template<typename _Tp > _Tp std::__detail::__comp_ellint_2 (_Tp *__k*)

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 modulus of the complete elliptic function.

Returns

The complete elliptic function of the second kind.

Definition at line 638 of file sf_ellint.tcc.

References __ellint_rd(), and __ellint_rf().

Referenced by __ellint_2().

8.3.2.46 template<typename _Tp > _Tp std::__detail::__comp_ellint_3 (_Tp *__k*, _Tp *__nu*)

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.

Definition at line 727 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

Referenced by __ellint_3().

8.3.2.47 template<typename _Tp > _Tp std::__detail::__comp_ellint_d (_Tp *__k*)

Return the complete Legendre elliptic integral D.

Definition at line 832 of file sf_ellint.tcc.

References __ellint_rd().

8.3.2.48 template<typename _Tp > _Tp std::__detail::__comp_ellint_rf (_Tp *__x*, _Tp *__y*)

Definition at line 235 of file sf_ellint.tcc.

Referenced by __comp_ellint_1(), and __ellint_rf().

8.3.2.49 template<typename _Tp > _Tp std::__detail::__comp_ellint_rg (_Tp *__x*, _Tp *__y*)

Definition at line 346 of file sf_ellint.tcc.

Referenced by __ellint_rg().

8.3.2.50 template<typename _Tp > _Tp std::__detail::__conf_hyperg (_Tp __*a*, _Tp __*c*, _Tp __*x*)

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.

Definition at line 281 of file sf_hyperg.tcc.

References __conf_hyperg_luke(), and __conf_hyperg_series().

8.3.2.51 template<typename _Tp > _Tp std::__detail::__conf_hyperg_lim (_Tp __*c*, _Tp __*x*)

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.

Definition at line 109 of file sf_hyperg.tcc.

References __conf_hyperg_lim_series().

8.3.2.52 template<typename _Tp > _Tp std::__detail::__conf_hyperg_lim_series (_Tp __*c*, _Tp __*x*)

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.

Definition at line 76 of file sf_hyperg.tcc.

Referenced by __conf_hyperg_lim().

8.3.2.53 template<typename _Tp > _Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)

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.

Definition at line 176 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

8.3.2.54 template<typename _Tp > _Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)

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.

Definition at line 141 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

8.3.2.55 template<typename _Tp > _Tp std::__detail::__coshint (const _Tp __x)

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.

Definition at line 558 of file sf_expint.tcc.

References __expint_E1(), and __expint_Ei().

8.3.2.56 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)

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.

Definition at line 1222 of file sf_hankel.tcc.

References __hankel().

8.3.2.57 template<typename _Tp > _Tp std::__detail::__cyl_bessel_i (_Tp *__nu,* _Tp *__x*)

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.

Definition at line 386 of file sf_mod_bessel.tcc.

References __cyl_bessel_ij_series(), and __cyl_bessel_ik().

8.3.2.58 template<typename _Tp > _Tp std::__detail::__cyl_bessel_ij_series (_Tp *__nu,* _Tp *__x,* _Tp *__sgn,* unsigned int *__max_iter*)

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.
__max_iter	The maximum number of iterations for sum.

Returns

The output Bessel function.

Definition at line 413 of file sf_bessel.tcc.

References __log_gamma().

Referenced by __cyl_bessel_i(), and __cyl_bessel_j().

8.3.2.59 template<typename _Tp > void std::__detail::__cyl_bessel_ik (_Tp *__nu*, _Tp *__x*, _Tp & *_Inu*, _Tp & *_Knu*, _Tp & *_Ipnu*, _Tp & *_Kpnu*)

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

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.

Definition at line 316 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik_asymp(), and __cyl_bessel_ik_steed().

Referenced by __airy(), __cyl_bessel_i(), __cyl_bessel_k(), and __sph_bessel_ik().

8.3.2.60 template<typename _Tp > void std::__detail::__cyl_bessel_ik_asymp (_Tp *__nu*, _Tp *__x*, _Tp & *_Inu*, _Tp & *_Knu*, _Tp & *_Ipnu*, _Tp & *_Kpnu*)

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.
_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.

Definition at line 81 of file sf_mod_bessel.tcc.

Referenced by __cyl_bessel_ik(), and __cyl_bessel_ik_steed().

8.3.2.61 template< typename _Tp > void std::__detail::__cyl_bessel_ik_steed (_Tp *__nu*, _Tp *__x*, _Tp & *_Inu*, _Tp & *_Knu*, _Tp & *_Ipnu*, _Tp & *_Kpnu*)

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.

Definition at line 152 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik_asymp(), and __gamma_temme().

Referenced by __cyl_bessel_ik().

8.3.2.62 template< typename _Tp > _Tp std::__detail::__cyl_bessel_j (_Tp *__nu*, _Tp *__x*)

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.

Definition at line 523 of file sf_bessel.tcc.

References __cyl_bessel_ij_series(), and __cyl_bessel_jn().

8.3.2.63 template< typename _Tp > void std::__detail::__cyl_bessel_jn (_Tp *__nu*, _Tp *__x*, _Tp & *_Jnu*, _Tp & *_Nnu*, _Tp & *_Jpnu*, _Tp & *_Npnu*)

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

Definition at line 442 of file sf_bessel.tcc.

References __cyl_bessel_jn_asymp(), and __cyl_bessel_jn_steed().

Referenced by __airy(), __cyl_bessel_j(), __cyl_hankel_1(), __cyl_hankel_2(), __cyl_neumann_n(), and __sph_↩ bessel_jn().

8.3.2.64 template<typename _Tp > void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp & _Jnu, _Tp & _Nnu, _Tp & _Jpnu, _Tp & _Npnu)

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.
out	*_Jnu*	The Bessel function of the first kind.
out	*_Nnu*	The Neumann function (Bessel function of the second kind).
out	*_Jpnu*	The Bessel function of the first kind.
out	*_Npnu*	The Neumann function (Bessel function of the second kind).

Definition at line 79 of file sf_bessel.tcc.

Referenced by __cyl_bessel_jn(), and __cyl_bessel_jn_steed().

8.3.2.65 template<typename _Tp > void std::__detail::__cyl_bessel_jn_steed (_Tp __nu, _Tp __x, _Tp & _Jnu, _Tp & _Nnu, _Tp & _Jpnu, _Tp & _Npnu)

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.
out	*_Jnu*	The output Bessel function of the first kind.
out	*_Nnu*	The output Neumann function (Bessel function of the second kind).
out	*_Jpnu*	The output derivative of the Bessel function of the first kind.
out	*_Npnu*	The output derivative of the Neumann function.

Definition at line 197 of file sf_bessel.tcc.

References __cyl_bessel_jn_asymp(), and __gamma_temme().

Referenced by __cyl_bessel_jn().

8.3.2.66 template<typename _Tp > _Tp std::__detail::__cyl_bessel_k (_Tp __nu, _Tp __x)

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.

Definition at line 424 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik().

8.3.2.67 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_hankel_1 (_Tp *__nu*, _Tp *__x*)

Return the cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$.

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

$$H^{(1)}_\nu(x) = J_\nu(x) + iN_\nu(x)$$

Parameters

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

Returns

The output spherical Neumann function.

Definition at line 588 of file sf_bessel.tcc.

References __cyl_bessel_jn().

8.3.2.68 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_hankel_1 (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*)

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.

Definition at line 1190 of file sf_hankel.tcc.

References __hankel().

8.3.2.69 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_hankel_2 (_Tp *__nu*, _Tp *__x*)

Return the cylindrical Hankel function of the second kind $H^{(2)}_nu(x)$.

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

$$H^{(2)}_\nu(x) = J_\nu(x) - iN_\nu(x)$$

Parameters

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

Returns

The output spherical Neumann function.

Definition at line 623 of file sf_bessel.tcc.

References __cyl_bessel_jn().

8.3.2.70 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_hankel_2 (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*)

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.

Definition at line 1206 of file sf_hankel.tcc.

References __hankel().

8.3.2.71 template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_neumann (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*)

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.

Definition at line 1238 of file sf_hankel.tcc.

References __hankel().

8.3.2.72 template<typename _Tp > _Tp std::__detail::__cyl_neumann_n (_Tp *__nu*, _Tp *__x*)

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.

Definition at line 558 of file sf_bessel.tcc.

References __cyl_bessel_jn().

8.3.2.73 template<typename _Tp > _Tp std::__detail::__dawson (_Tp *__x*)

Return the Dawson integral, $F(x)$, for real argument x.

The Dawson integral is defined by:

$$F(x) = e^{-x^2} \int_0^x e^{y^2} dy$$

and it's derivative is:

$$F'(x) = 1 - 2xF(x)$$

Parameters

__x	The argument $-inf < x < inf$.

Definition at line 233 of file sf_dawson.tcc.

References __dawson_cont_frac(), and __dawson_series().

8.3.2.74 template<typename _Tp > _Tp std::__detail::__dawson_cont_frac (_Tp *__x*)

Compute the Dawson integral using a sampling theorem representation.

Todo this needs some compile-time construction!

Definition at line 71 of file sf_dawson.tcc.

Referenced by __dawson().

8.3.2.75 template<typename _Tp > _Tp std::__detail::__dawson_series (_Tp *__x*)

Compute the Dawson integral using the series expansion.

Definition at line 47 of file sf_dawson.tcc.

Referenced by __dawson().

8.3.2.76 template<typename _Tp > void std::__detail::__debye_region (std::complex< _Tp > *__alpha,* int & *__indexr,* char & *__aorb*)

Compute the Debye region in te complex plane.

Definition at line 54 of file sf_hankel.tcc.

Referenced by __hankel().

8.3.2.77 template<typename _Tp > _Tp std::__detail::__dilog (_Tp __x)

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:

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

Definition at line 194 of file sf_zeta.tcc.

8.3.2.78 template<typename _Tp > _Tp std::__detail::__dirichlet_beta (std::complex< _Tp > __w)

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.

Definition at line 1192 of file sf_polylog.tcc.

References __fpequal(), and __polylog().

8.3.2.79 template<typename _Tp > _Tp std::__detail::__dirichlet_beta (_Tp __w)

Return the Dirichlet beta function for real argument.

Parameters

__w	The real argument.

Returns

The Dirichlet Beta function of real argument.

Definition at line 1211 of file sf_polylog.tcc.

References __polylog().

8.3.2.80 template<typename _Tp > std::complex<_Tp> std::__detail::__dirichlet_eta (std::complex< _Tp > *__w*)

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.

Definition at line 1155 of file sf_polylog.tcc.

References __fpequal(), and __polylog().

8.3.2.81 template<typename _Tp > _Tp std::__detail::__dirichlet_eta (_Tp __w)

Return the Dirichlet eta function for real argument.

Parameters

__w	The real argument.

Returns

The Dirichlet eta function.

Definition at line 1173 of file sf_polylog.tcc.

References __polylog().

8.3.2.82 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)

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].

Definition at line 2480 of file sf_gamma.tcc.

References __factorial(), __log_double_factorial(), _S_double_factorial_table, and _S_neg_double_factorial_table.

8.3.2.83 template<typename _Tp > _Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)

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.

Definition at line 594 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rf().

8.3.2.84 template<typename _Tp > _Tp std::__detail::__ellint_2 (_Tp *__k*, _Tp *__phi*)

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.

Definition at line 673 of file sf_ellint.tcc.

References __comp_ellint_2(), __ellint_rd(), and __ellint_rf().

8.3.2.85 template<typename _Tp > _Tp std::__detail::__ellint_3 (_Tp *__k*, _Tp *__nu*, _Tp *__phi*)

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.

Definition at line 768 of file sf_ellint.tcc.

References __comp_ellint_3(), __ellint_rf(), and __ellint_rj().

8.3.2.86 template<typename _Tp > _Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)

Return the Bulirsch complete elliptic integrals.

Definition at line 920 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

8.3.2.87 template<typename _Tp > _Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)

Return the Legendre elliptic integral D.

Definition at line 809 of file sf_ellint.tcc.

References __ellint_rd().

8.3.2.88 template<typename _Tp > _Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)

Return the Bulirsch elliptic integrals of the first kind.

Definition at line 848 of file sf_ellint.tcc.

References __ellint_rf().

8.3.2.89 template<typename _Tp > _Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)

Return the Bulirsch elliptic integrals of the second kind.

Definition at line 869 of file sf_ellint.tcc.

References __ellint_rd(), and __ellint_rf().

8.3.2.90 template<typename _Tp > _Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)

Return the Bulirsch elliptic integrals of the third kind.

Definition at line 894 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

8.3.2.91 template<typename _Tp > _Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)

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.

Definition at line 81 of file sf_ellint.tcc.

Referenced by __ellint_rf(), and __ellint_rj().

8.3.2.92 template<typename _Tp > _Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)

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.

Definition at line 163 of file sf_ellint.tcc.

Referenced by __comp_ellint_2(), __comp_ellint_d(), __ellint_2(), __ellint_d(), __ellint_el2(), __ellint_rg(), and __↩ ellint_rj().

8.3.2.93 template<typename _Tp > _Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)

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.

Definition at line 277 of file sf_ellint.tcc.

References __comp_ellint_rf(), and __ellint_rc().

Referenced by __comp_ellint_2(), __comp_ellint_3(), __ellint_1(), __ellint_2(), __ellint_3(), __ellint_cel(), __ellint_el1(), __ellint_el2(), __ellint_el3(), and __heuman_lambda().

8.3.2.94 template< typename _Tp > _Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)

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.

Definition at line 408 of file sf_ellint.tcc.

References __comp_ellint_rg(), and __ellint_rd().

8.3.2.95 template< typename _Tp > _Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)

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.

Definition at line 456 of file sf_ellint.tcc.

References __ellint_rc(), and __ellint_rd().

Referenced by __comp_ellint_3(), __ellint_3(), __ellint_cel(), __ellint_el3(), __heuman_lambda(), and __jacobi_zeta().

8.3.2.96 template<typename _Tp > _Tp std::__detail::__ellnome (_Tp *__k*)

Return the elliptic nome given the modulus k.

Definition at line 292 of file sf_theta.tcc.

References __ellnome_k(), and __ellnome_series().

Referenced by __theta_c(), __theta_d(), __theta_n(), and __theta_s().

8.3.2.97 template<typename _Tp > _Tp std::__detail::__ellnome_k (_Tp *__k*)

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

Definition at line 278 of file sf_theta.tcc.

References __comp_ellint_1().

Referenced by __ellnome().

8.3.2.98 template<typename _Tp > _Tp std::__detail::__ellnome_series (_Tp *__k*)

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

Definition at line 262 of file sf_theta.tcc.

Referenced by __ellnome().

8.3.2.99 template<typename _Tp > _Tp std::__detail::__expint (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 474 of file sf_expint.tcc.

References __expint_E1(), and __expint_En_recursion().

Referenced by __logint().

8.3.2.100 template<typename _Tp > _Tp std::__detail::__expint (_Tp *__x*)

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.

Definition at line 514 of file sf_expint.tcc.

References __expint_Ei().

8.3.2.101 template<typename _Tp > _Tp std::__detail::__expint_asymp (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 406 of file sf_expint.tcc.

8.3.2.102 template<typename _Tp > _Tp std::__detail::__expint_E1 (_Tp __x)

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.

Todo Find a good asymptotic switch point in $E_1(x)$.

Todo Find a good asymptotic switch point in $E_1(x)$.

Definition at line 375 of file sf_expint.tcc.

References __expint_E1_asymp(), __expint_E1_series(), __expint_Ei(), and __expint_En_cont_frac().

Referenced by __coshint(), __expint(), __expint_Ei(), __expint_En_recursion(), and __sinhint().

8.3.2.103 template<typename _Tp > _Tp std::__detail::__expint_E1_asymp (_Tp __x)

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.

Definition at line 112 of file sf_expint.tcc.

Referenced by __expint_E1().

8.3.2.104 template<typename _Tp > _Tp std::__detail::__expint_E1_series (_Tp __x)

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.

Definition at line 75 of file sf_expint.tcc.

Referenced by __expint_E1().

8.3.2.105 template<typename _Tp > _Tp std::__detail::__expint_Ei (_Tp *__x*)

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.

Definition at line 351 of file sf_expint.tcc.

References __expint_E1(), __expint_Ei_asymp(), and __expint_Ei_series().

Referenced by __coshint(), __expint(), __expint_E1(), and __sinhint().

8.3.2.106 template<typename _Tp > _Tp std::__detail::__expint_Ei_asymp (_Tp *__x*)

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.

Definition at line 318 of file sf_expint.tcc.

Referenced by __expint_Ei().

8.3.2.107 template< typename _Tp > _Tp std::__detail::__expint_Ei_series (_Tp *__x*)

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.

Definition at line 286 of file sf_expint.tcc.

Referenced by __expint_Ei().

8.3.2.108 template< typename _Tp > _Tp std::__detail::__expint_En_cont_frac (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 195 of file sf_expint.tcc.

Referenced by __expint_E1().

8.3.2.109 template< typename _Tp > _Tp std::__detail::__expint_En_recursion (unsigned int *__n*, _Tp *__x*)

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.

Todo Find a principled starting number for the $E_n(x)$ downward recursion.

Definition at line 240 of file sf_expint.tcc.

References __expint_E1().

Referenced by __expint().

8.3.2.110 template<typename _Tp > _Tp std::__detail::__expint_En_series (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 149 of file sf_expint.tcc.

References __psi().

8.3.2.111 template<typename _Tp > _Tp std::__detail::__expint_large_n (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 440 of file sf_expint.tcc.

8.3.2.112 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int *__n*)

Return the factorial of the integer n.

The factorial is:

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

Definition at line 2422 of file sf_gamma.tcc.

References _S_factorial_table.

Referenced by __double_factorial().

8.3.2.113 template<typename _Tp > _Tp std::__detail::__fermi_dirac (_Tp *__s*, _Tp *__x*)

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),

Definition at line 1379 of file sf_polylog.tcc.

References __polylog_exp().

8.3.2.114 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf (_Tp *__F*, unsigned int *__nu1*, unsigned int *__nu2*)

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

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

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

Parameters

__nu1	The number of degrees of freedom of sample 1
__nu2	The number of degrees of freedom of sample 2
__F	The F statistic

Definition at line 350 of file sf_beta.tcc.

References __beta_inc().

8.3.2.115 template< typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)

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

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

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

Parameters

__F	
__nu1	
__nu2	

Definition at line 379 of file sf_beta.tcc.

References __beta_inc().

8.3.2.116 template< typename _Tp > void std::__detail::__fock_airy (_Tp __x, std::complex< _Tp > & __w1, std::complex< _Tp > & __w2, std::complex< _Tp > & __w1p, std::complex< _Tp > & __w2p)

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.

Definition at line 580 of file sf_mod_bessel.tcc.

8.3.2.117 template< typename _Tp > bool std::__detail::__fpequal (const _Tp & __a, const _Tp & __b)

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 $max(a, b) * 5 * eps$

Definition at line 62 of file sf_polylog.tcc.

Referenced by __dirichlet_beta(), __dirichlet_eta(), __fpimag(), __fpreal(), __polylog(), __polylog_exp_asymp(), __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_neg(), __polylog_exp_neg_even(), __polylog_exp_neg_odd(), __polylog_exp_negative_real_part(), __polylog_exp_pos(), and __polylog_exp_real_pos().

8.3.2.118 template< typename _Tp > bool std::__detail::__fpimag (const std::complex< _Tp > & __w)

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.

Definition at line 107 of file sf_polylog.tcc.

References __fpequal().

8.3.2.119 template< typename _Tp > bool std::__detail::__fpimag (const _Tp)

Definition at line 117 of file sf_polylog.tcc.

8.3.2.120 template< typename _Tp > bool std::__detail::__fpreal (const std::complex< _Tp > & __w)

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.

Definition at line 84 of file sf_polylog.tcc.

References __fpequal().

Referenced by __polylog_exp_int_pos(), and __polylog_exp_real_pos().

8.3.2.121 template< typename _Tp > bool std::__detail::__fpreal (const _Tp)

Definition at line 94 of file sf_polylog.tcc.

8.3.2.122 template< typename _Tp > std::complex<_Tp> std::__detail::__fresnel (const _Tp __x)

Return 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:

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

Parameters

__x	The argument

Definition at line 166 of file sf_fresnel.tcc.

References __fresnel_cont_frac(), and __fresnel_series().

8.3.2.123 template<typename _Tp > void std::__detail::__fresnel_cont_frac (const _Tp *__ax*, _Tp & *_Cf*, _Tp & *_Sf*)

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

Definition at line 105 of file sf_fresnel.tcc.

Referenced by __fresnel().

8.3.2.124 template<typename _Tp > void std::__detail::__fresnel_series (const _Tp *__ax*, _Tp & *_Cf*, _Tp & *_Sf*)

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

Definition at line 48 of file sf_fresnel.tcc.

Referenced by __fresnel().

8.3.2.125 template<typename _Tp > _Tp std::__detail::__gamma (_Tp *__x*)

Return $\Gamma(x)$.

Parameters

__x	The argument of the gamma function.

Returns

The gamma function.

Definition at line 1918 of file sf_gamma.tcc.

References __log_gamma().

Referenced by __beta_gamma(), and __riemann_zeta().

8.3.2.126 template<typename _Tp > std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp *__a*, _Tp *__x*)

Definition at line 1964 of file sf_gamma.tcc.

Referenced by __gamma_l(), __gamma_p(), __gamma_q(), and __gamma_u().

8.3.2.127 template<typename _Tp > _Tp std::__detail::__gamma_l (_Tp *__a*, _Tp *__x*)

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

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

.

Definition at line 2070 of file sf_gamma.tcc.

References __gamma_cont_frac(), and __gamma_series().

8.3.2.128 template<typename _Tp > _Tp std::__detail::__gamma_p (_Tp *__a*, _Tp *__x*)

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.

Definition at line 2013 of file sf_gamma.tcc.

References __gamma_cont_frac(), and __gamma_series().

Referenced by __chi_squared_pdf().

8.3.2.129 template<typename _Tp > _Tp std::__detail::__gamma_q (_Tp *__a*, _Tp *__x*)

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.

Definition at line 2044 of file sf_gamma.tcc.

References __gamma_cont_frac(), and __gamma_series().

Referenced by __chi_squared_pdfc().

8.3.2.130 template<typename _Tp > std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp *__a*, _Tp *__x*)

Definition at line 1930 of file sf_gamma.tcc.

Referenced by __gamma_l(), __gamma_p(), __gamma_q(), and __gamma_u().

8.3.2.131 template<typename _Tp > void std::__detail::__gamma_temme (_Tp *__mu*, _Tp & *__gam1*, _Tp & *__gam2*, _Tp & *__gampl*, _Tp & *__gammi*)

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 $\nu.$ 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.
out	*__gam1*	The output function $\Gamma_1(\mu)$
out	*__gam2*	The output function $\Gamma_2(\mu)$
out	*__gampl*	The output function $\Gamma(1+\mu)$
out	*__gammi*	The output function $\Gamma(1-\mu)$

Definition at line 163 of file sf_bessel.tcc.

Referenced by __cyl_bessel_ik_steed(), and __cyl_bessel_jn_steed().

8.3.2.132 template<typename _Tp > _Tp std::__detail::__gamma_u (_Tp *__a*, _Tp *__x*)

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

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

.

Definition at line 2102 of file sf_gamma.tcc.

References __gamma_cont_frac(), and __gamma_series().

8.3.2.133 template<typename _Tp > _Tp std::__detail::__gauss (_Tp *__x*)

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

Definition at line 70 of file sf_owens_t.tcc.

8.3.2.134 template<typename _Tp > _Tp std::__detail::__gegenbauer_poly (unsigned int *__n*, _Tp *__alpha*, _Tp *__x*)

Return the Gegenbauer polynomial $C_n^\alpha(x)$ of degree `n` and real order α and argument `x`.

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^\alpha(x) = \frac{1}{n}\left[2x(n+\alpha-1)C_{n-1}^\alpha(x) - (n+2\alpha-2)C_{n-2}^\alpha(x)\right]$$

and $C_0^\alpha(x) = 1$, $C_1^\alpha(x) = 2\alpha x$.

Template Parameters

_Talpha	The real type of the order
_Tp	The real type of the argument

Parameters

__n	The non-negative integral degree
__alpha	The real order
__x	The real argument

Definition at line 61 of file sf_gegenbauer.tcc.

8.3.2.135 template< typename _Tp > void std::__detail::__hankel (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*, std::complex< _Tp > & *_H1*, std::complex< _Tp > & *_H2*, std::complex< _Tp > & *_H1p*, std::complex< _Tp > & *_H2p*)

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.

Definition at line 1127 of file sf_hankel.tcc.

References __debye_region(), __hankel_debye(), and __hankel_uniform().

Referenced by __cyl_bessel(), __cyl_hankel_1(), __cyl_hankel_2(), __cyl_neumann(), and __sph_hankel().

8.3.2.136 template< typename _Tp > void std::__detail::__hankel_debye (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*, std::complex< _Tp > *__alpha*, int *__indexr*, char & *__aorb*, int & *__morn*, std::complex< _Tp > & *_H1*, std::complex< _Tp > & *_H2*, std::complex< _Tp > & *_H1p*, std::complex< _Tp > & *_H2p*)

Parameters

in	*__nu*	The order for which the Hankel functions are evaluated.
in	*__z*	The argument at which the Hankel functions are evaluated.
in	*__alpha*	
in	*__indexr*	
out	*__aorb*	
out	*__morn*	
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.

Definition at line 959 of file sf_hankel.tcc.

Referenced by __hankel().

8.3.2.137 template< typename _Tp > void std::__detail::__hankel_params (std::complex< _Tp > *__nu*, std::complex< _Tp > *__zhat*, std::complex< _Tp > & *__p*, std::complex< _Tp > & *__p2*, std::complex< _Tp > & *__nup2*, std::complex< _Tp > & *__num2*, std::complex< _Tp > & *__num1d3*, std::complex< _Tp > & *__num2d3*, std::complex< _Tp > & *__num4d3*, std::complex< _Tp > & *__zeta*, std::complex< _Tp > & *__zetaphf*, std::complex< _Tp > & *__zetamhf*, std::complex< _Tp > & *__zetam3hf*, std::complex< _Tp > & *__zetrat*)

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.

Definition at line 110 of file sf_hankel.tcc.

Referenced by __hankel_uniform_outer().

8.3.2.138 template< typename _Tp > void std::__detail::__hankel_uniform (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*, std::complex< _Tp > & *_H1*, std::complex< _Tp > & *_H2*, std::complex< _Tp > & *_H1p*, std::complex< _Tp > & *_H2p*)

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.

Definition at line 904 of file sf_hankel.tcc.

References __hankel_uniform_olver().

Referenced by __hankel().

8.3.2.139 template< typename _Tp > void std::__detail::__hankel_uniform_olver (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*, std::complex< _Tp > & *_H1*, std::complex< _Tp > & *_H2*, std::complex< _Tp > & *_H1p*, std::complex< _Tp > & *_H2p*)

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.

Definition at line 818 of file sf_hankel.tcc.

References __hankel_uniform_outer(), and __hankel_uniform_sum().

Referenced by __hankel_uniform().

8.3.2.140 template< typename _Tp > void std::__detail::__hankel_uniform_outer (std::complex< _Tp > *__nu*, std::complex< _Tp > *__z*, _Tp *__eps*, std::complex< _Tp > & *__zhat*, std::complex< _Tp > & *__1dnsq*, std::complex< _Tp > & *__num1d3*, std::complex< _Tp > & *__num2d3*, std::complex< _Tp > & *__p*, std::complex< _Tp > & *__p2*, std::complex< _Tp > & *__etm3h*, std::complex< _Tp > & *__etrat*, std::complex< _Tp > & *_Aip*, std::complex< _Tp > & *__o4dp*, std::complex< _Tp > & *_Aim*, std::complex< _Tp > & *__o4dm*, std::complex< _Tp > & *__od2p*, std::complex< _Tp > & *__od0dp*, std::complex< _Tp > & *__od2m*, std::complex< _Tp > & *__od0dm*)

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.

Definition at line 273 of file sf_hankel.tcc.

References __airy(), __airy_arg(), and __hankel_params().

Referenced by __hankel_uniform_olver().

8.3.2.141 template< typename _Tp > void std::__detail::__hankel_uniform_sum (std::complex< _Tp > *__p*, std::complex< _Tp > *__p2*, std::complex< _Tp > *__num2*, std::complex< _Tp > *__zetam3hf*, std::complex< _Tp > *_Aip*, std::complex< _Tp > *__o4dp*, std::complex< _Tp > *_Aim*, std::complex< _Tp > *__o4dm*, std::complex< _Tp > *__od2p*, std::complex< _Tp > *__od0dp*, std::complex< _Tp > *__od2m*, std::complex< _Tp > *__od0dm*, _Tp *__eps*, std::complex< _Tp > & *_H1sum*, std::complex< _Tp > & *_H1psum*, std::complex< _Tp > & *_H2sum*, std::complex< _Tp > & *_H2psum*)

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

in	*__p*	
in	*__p2*	
in	*__num2*	
in	*__zetam3hf*	
in	*_Aip*	The Airy function value $Ai()$.
in	*__o4dp*	
in	*_Aim*	The Airy function value $Ai()$.
in	*__o4dm*	
in	*__od2p*	
in	*__od0dp*	
in	*__od2m*	
in	*__od0dm*	
in	*__eps*	The error tolerance
out	*_H1sum*	The Hankel function of the first kind.
out	*_H1psum*	The derivative of the Hankel function of the first kind.
out	*_H2sum*	The Hankel function of the second kind.
out	*_H2psum*	The derivative of the Hankel function of the second kind.

Definition at line 351 of file sf_hankel.tcc.

Referenced by __hankel_uniform_olver().

8.3.2.142 template< typename _Tp > _Tp std::__detail::__heuman_lambda (_Tp *__k*, _Tp *__phi*)

Return the Heuman lambda function.

Definition at line 941 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

8.3.2.143 template<typename _Tp > _Tp std::__detail::__hurwitz_zeta (_Tp __s, _Tp __a)

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:

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

Parameters

__s	The argument $s! = 1$
__a	The scale parameter $a > -1$

Definition at line 702 of file sf_zeta.tcc.

References __hurwitz_zeta_euler_maclaurin().

8.3.2.144 template<typename _Tp > std::complex<_Tp> std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)

Return the Hurwitz Zeta function for real s and complex a.

Parameters

__s	The real argument
__a	The complex parameter

Todo This __hurwitz_zeta prefactor is prone to overflow. positive integer orders s?

Definition at line 1119 of file sf_polylog.tcc.

References __polylog_exp().

Referenced by __psi().

8.3.2.145 template<typename _Tp > _Tp std::__detail::__hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)

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

See also

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

Parameters

__s	The argument $s != 1$
__a	The scale parameter $a > -1$

Definition at line 560 of file sf_zeta.tcc.

References _S_Euler_Maclaurin_zeta.

Referenced by __hurwitz_zeta().

8.3.2.146 template<typename _Tp > std::complex<_Tp> std::__detail::__hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp _Z, const _Tp __r, const _Tp __theta, const _Tp __phi)

Definition at line 44 of file sf_hydrogen.tcc.

References __assoc_laguerre(), __psi(), and __sph_legendre().

8.3.2.147 template<typename _Tp > _Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

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

Returns

The confluent hypergeometric function.

Definition at line 776 of file sf_hyperg.tcc.

References __hyperg_luke(), __hyperg_reflect(), __hyperg_series(), __log_gamma(), and __log_gamma_sign().

8.3.2.148 template<typename _Tp > _Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)

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

Definition at line 352 of file sf_hyperg.tcc.

Referenced by __hyperg().

8.3.2.149 template<typename _Tp > _Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

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:

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

Definition at line 486 of file sf_hyperg.tcc.

References __hyperg_series(), __log_gamma(), __log_gamma_sign(), and __psi().

Referenced by __hyperg().

8.3.2.150 template<typename _Tp > _Tp std::__detail::__hyperg_series (_Tp *__a*, _Tp *__b*, _Tp *__c*, _Tp *__x*)

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

Returns

The confluent hypergeometric function.

Definition at line 321 of file sf_hyperg.tcc.

Referenced by __hyperg(), and __hyperg_reflect().

8.3.2.151 template<typename _Tp > std::tuple<_Tp, _Tp, _Tp> std::__detail::__jacobi_sncndn (_Tp *__k*, _Tp *__u*)

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

Definition at line 414 of file sf_theta.tcc.

8.3.2.152 template<typename _Tp > _Tp std::__detail::__jacobi_zeta (_Tp __*k*, _Tp __*phi*)

Return the Jacobi zeta function.

Definition at line 971 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rj().

8.3.2.153 template<typename _Tp > _Tp std::__detail::__laguerre (unsigned int __*n*, _Tp __*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.

Definition at line 321 of file sf_laguerre.tcc.

8.3.2.154 template<typename _Tp > _Tp std::__detail::__legendre_q (unsigned int __*l*, _Tp __*x*)

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

The Legendre function of the second kind 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 function. $l >= 0$.
__*x*	The argument of the Legendre function. $\|x\| <= 1$.

Definition at line 123 of file sf_legendre.tcc.

8.3.2.155 template<typename _Tp > _Tp std::__detail::__log_bincoef (unsigned int __*n*, unsigned int __*k*)

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.

Definition at line 1862 of file sf_gamma.tcc.

8.3.2.156 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp *__x*)

Definition at line 2450 of file sf_gamma.tcc.

References __log_gamma().

Referenced by __double_factorial(), and __log_double_factorial().

8.3.2.157 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int *__n*)

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].

Definition at line 2516 of file sf_gamma.tcc.

References __log_double_factorial(), __log_factorial(), _S_double_factorial_table, and _S_neg_double_factorial_table.

8.3.2.158 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int *__n*)

Return the logarithm of the factorial of the integer n.

The factorial is:

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

Definition at line 2440 of file sf_gamma.tcc.

References __log_gamma(), and _S_factorial_table.

Referenced by __log_double_factorial().

8.3.2.159 template<typename _Tp > _Tp std::__detail::__log_gamma (_Tp *__x*)

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.

Definition at line 1800 of file sf_gamma.tcc.

References __log_gamma_lanczos().

Referenced by __beta_lgamma(), __cyl_bessel_ij_series(), __gamma(), __hyperg(), __hyperg_reflect(), __log_double_factorial(), __log_factorial(), __log_pochhammer_u(), __poly_laguerre_large_n(), __psi(), __riemann_zeta(), __riemann_zeta_glob(), and __sph_legendre().

8.3.2.160 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)

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.

Definition at line 1699 of file sf_gamma.tcc.

8.3.2.161 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)

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.

Definition at line 1755 of file sf_gamma.tcc.

Referenced by __log_gamma().

8.3.2.162 template<typename _Tp > _Tp std::__detail::__log_gamma_sign (_Tp __x)

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.

Definition at line 1831 of file sf_gamma.tcc.

Referenced by __hyperg(), __hyperg_reflect(), and __pochhammer_l().

8.3.2.163 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp *__z*)

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.

Definition at line 1739 of file sf_gamma.tcc.

8.3.2.164 template<typename _Tp > _Tp std::__detail::__log_pochhammer_l (_Tp *__a*, _Tp *__n*)

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.

Definition at line 2209 of file sf_gamma.tcc.

8.3.2.165 template<typename _Tp > _Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)

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.

Definition at line 2144 of file sf_gamma.tcc.

References __log_gamma().

8.3.2.166 template<typename _Tp > _Tp std::__detail::__logint (const _Tp __x)

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.

Definition at line 535 of file sf_expint.tcc.

References __expint().

8.3.2.167 template<typename _Tp > _Tp std::__detail::__owens_t (_Tp __h, _Tp __a)

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.

Definition at line 92 of file sf_owens_t.tcc.

References __znorm1(), and __znorm2().

8.3.2.168 template<typename _Tp > _Tp std::__detail::__pochhammer_l (_Tp *__a*, _Tp *__n*)

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].

Definition at line 2232 of file sf_gamma.tcc.

References __log_gamma_sign().

8.3.2.169 template<typename _Tp > _Tp std::__detail::__pochhammer_u (_Tp *__a*, _Tp *__n*)

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.

Definition at line 2170 of file sf_gamma.tcc.

8.3.2.170 template<typename _Tp > _Tp std::__detail::__poly_hermite (unsigned int *__n*, _Tp *__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.

Definition at line 179 of file sf_hermite.tcc.

References __poly_hermite_asymp(), and __poly_hermite_recursion().

8.3.2.171 template<typename _Tp > _Tp std::__detail::__poly_hermite_asymp (unsigned int *__n*, _Tp *__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 "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.

Definition at line 113 of file sf_hermite.tcc.

References __airy().

Referenced by __poly_hermite().

8.3.2.172 template<typename _Tp > _Tp std::__detail::__poly_hermite_recursion (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 69 of file sf_hermite.tcc.

Referenced by __poly_hermite().

8.3.2.173 template<typename _Tp > _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)

Compute the Jacobi polynomial by recursion on `n`:

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

Definition at line 57 of file sf_jacobi.tcc.

References __beta().

Referenced by __poly_radial_jacobi().

8.3.2.174 template<typename _Tpa , typename _Tp > _Tp std::__detail::__poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __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:

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

Template Parameters

_Tpa	The type of the degree.
_Tp	The type of the parameter.

Parameters

__n	The order of the Laguerre function.
__alpha1	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.

Definition at line 248 of file sf_laguerre.tcc.

References __poly_laguerre_hyperg(), __poly_laguerre_large_n(), and __poly_laguerre_recursion().

8.3.2.175 template<typename _Tpa , typename _Tp > _Tp std::__detail::__poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)

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.

Template Parameters

_Tpa	The type of the degree.
_Tp	The type of the parameter.

Parameters

__n	The order of the Laguerre function.
__alpha1	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.

Definition at line 129 of file sf_laguerre.tcc.

Referenced by __poly_laguerre().

8.3.2.176 template<typename _Tpa , typename _Tp > _Tp std::__detail::__poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)

This routine returns the associated Laguerre polynomial of order n, degree $\alpha > -1$ for large n. Abramowitz & Stegun, 13.5.21.

Template Parameters

_Tpa	The type of the degree.
_Tp	The type of the parameter.

Parameters

__n	The order of the Laguerre function.
__alpha1	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.

Definition at line 72 of file sf_laguerre.tcc.

References __log_gamma().

Referenced by __poly_laguerre().

8.3.2.177 template<typename _Tpa , typename _Tp > _Tp std::__detail::__poly_laguerre_recursion (unsigned int *__n*, _Tpa *__alpha1*, _Tp *__x*)

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:

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

Template Parameters

_Tpa	The type of the degree.
_Tp	The type of the parameter.

Parameters

__n	The order of the Laguerre function.
__alpha1	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.

Definition at line 187 of file sf_laguerre.tcc.

Referenced by __poly_laguerre().

8.3.2.178 template<typename _Tp > _Tp std::__detail::__poly_legendre_p (unsigned int *__l*, _Tp *__x*)

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. $\|x\| <= 1$.

Definition at line 73 of file sf_legendre.tcc.

Referenced by __assoc_legendre_p(), and __sph_legendre().

8.3.2.179 template<typename _Tp > _Tp std::__detail::__poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)

Return the radial polynomial $R_n^m(\rho)$ for non-negative degree n, order $m <= n$, and real radial argument ρ.

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k!(\frac{n+m}{2}-k)!(\frac{n-m}{2}-k)!} \rho^{n-2k}$$

for $n-m$ even and identically 0 for $n-m$ odd. The radial polynomials can be related to the Zernike polynomials:

$$Z_n^m(\rho,\phi) = R_n^m(\rho)\cos(m\phi)$$

$$Z_n^{-m}(\rho,\phi) = R_n^m(\rho)\sin(m\phi)$$

for non-negative m, n.

See also

zernike for details on the Zernike polynomials.

Template Parameters

_Tp	The real type of the radial coordinate

Parameters

__n	The non-negative degree.
__m	The non-negative azimuthal order
__rho	The radial argument

Definition at line 138 of file sf_jacobi.tcc.

References __poly_jacobi().

Referenced by __zernike(), __gnu_cxx::radpolyf(), and __gnu_cxx::radpolyl().

8.3.2.180 template<typename _Tp > _Tp std::__detail::__polylog (_Tp __s, _Tp __x)

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.

Definition at line 1072 of file sf_polylog.tcc.

References __fpequal(), and __polylog_exp().

Referenced by __dirichlet_beta(), __dirichlet_eta(), and __polylog().

8.3.2.181 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)

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.

Definition at line 1102 of file sf_polylog.tcc.

References __fpequal(), __polylog(), and __polylog_exp().

8.3.2.182 template<typename _Tp , typename ArgType > __gnu_cxx::__promote_num_t<std::complex<_Tp>, ArgType> std::__detail::__polylog_exp (_Tp __s, ArgType __w)

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 ArgType.

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).

Definition at line 1039 of file sf_polylog.tcc.

References __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_negative_real_part(), __polylog_exp_real_neg(), and __polylog_exp_real_pos().

Referenced by __bose_einstein(), __clausen(), __clausen_c(), __clausen_s(), __fermi_dirac(), __hurwitz_zeta(), and __polylog().

8.3.2.183 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)

This function implements the asymptotic series for the polylog. It is given by

$$2\sum_{k=0}^{\infty}\zeta(2k)w^{s-2k}/\Gamma(s-2k+1)-i\pi w^(s-1)/\Gamma(s)$$

for $Re(w) >> 1$

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

Parameters

__s	the real index s.
__w	the large complex argument w.

Returns

the value of the polylogarithm.

Definition at line 686 of file sf_polylog.tcc.

References __fpequal().

Referenced by __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_real_neg(), and __polylog_exp_↩ real_pos().

8.3.2.184 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_int_neg (int __s, std::complex< _Tp > __w)

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,.

Definition at line 856 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __fpequal(), __polylog_exp_asymp(), __polylog_exp_neg(), and __↩ polylog_exp_negative_real_part().

Referenced by __polylog_exp().

8.3.2.185 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)

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.

Definition at line 898 of file sf_polylog.tcc.

References __fpequal(), __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_negative_real_part().

8.3.2.186 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)

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.

Definition at line 767 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __fpequal(), __fpreal(), __polylog_exp_asymp(), __polylog_exp_↩ negative_real_part(), __polylog_exp_pos(), and __riemann_zeta().

Referenced by __polylog_exp().

8.3.2.187 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_int_pos (unsigned int *__s*, _Tp *__w*)

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.

Definition at line 815 of file sf_polylog.tcc.

References __fpequal(), __polylog_exp_asymp(), __polylog_exp_negative_real_part(), __polylog_exp_pos(), and __↩ riemann_zeta().

8.3.2.188 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_neg (_Tp *__s*, std::complex< _Tp > *__w*)

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

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

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

Parameters

__s	The real index
__w	The complex argument

Returns

The value of the polylogarithm.

Definition at line 346 of file sf_polylog.tcc.

References __fpequal(), __riemann_zeta(), and __riemann_zeta_m_1().

Referenced by __polylog_exp_int_neg(), and __polylog_exp_real_neg().

8.3.2.189 template< typename _Tp > std::complex< _Tp> std::__detail::__polylog_exp_neg (int __s, std::complex< _Tp > __w)

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.

Definition at line 564 of file sf_polylog.tcc.

References __polylog_exp_neg_even(), and __polylog_exp_neg_odd().

8.3.2.190 template<typename _Tp , int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_even (unsigned int *__n*, std::complex< _Tp > *__w*)

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

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

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 integral index $n = 4k$.
__w	The complex argument w

Returns

the value of the Polylogarithm.

Definition at line 450 of file sf_polylog.tcc.

References __fpequal().

Referenced by __polylog_exp_neg().

8.3.2.191 template<typename _Tp , int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_odd (unsigned int *__n*, std::complex< _Tp > *__w*)

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

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

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)

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

Parameters

__n	the integral index n = 4k.
__w	The complex argument w.

Returns

The value of the Polylogarithm.

Definition at line 517 of file sf_polylog.tcc.

References __fpequal().

Referenced by __polylog_exp_neg().

8.3.2.192 template<typename _PowTp , typename _Tp > _Tp std::__detail::__polylog_exp_negative_real_part (_PowTp *__s*, _Tp *__w*)

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, integral or float.
__w	something with a negative real part.

Returns

the value of the polylogarithm.

Definition at line 737 of file sf_polylog.tcc.

References __fpequal().

Referenced by __polylog_exp(), __polylog_exp_int_neg(), __polylog_exp_int_pos(), __polylog_exp_real_neg(), and ↩ __polylog_exp_real_pos().

8.3.2.193 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)

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

Parameters

__s	the index s.
__w	the argument w.

Returns

the value of the polylogarithm.

Definition at line 206 of file sf_polylog.tcc.

References __fpequal(), and __riemann_zeta().

Referenced by __polylog_exp_int_pos(), and __polylog_exp_real_pos().

8.3.2.194 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)

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 $\log(e^(i\pi))$: gcc: $\log(e^(+ -i\pi)) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^(+ -i * \pi)) = +i\pi$

Parameters

__s	the index.
__w	the argument

Returns

the value of the Polylogarithm

Definition at line 279 of file sf_polylog.tcc.

References __fpequal(), and __riemann_zeta().

8.3.2.195 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)

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 complex argument w.

Returns

the value of the polylogarithm.

Definition at line 603 of file sf_polylog.tcc.

References __fpequal(), and __riemann_zeta().

8.3.2.196 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp *__s*, std::complex< _Tp > *__w*)

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.

Definition at line 985 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_negative_real_part().

Referenced by __polylog_exp().

8.3.2.197 template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp *__s*, _Tp *__w*)

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.

Definition at line 1013 of file sf_polylog.tcc.

References __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_negative_real_part().

8.3.2.198 template$<$typename _Tp $>$ std::complex$<$_Tp$>$ std::__detail::__polylog_exp_real_pos (_Tp *__s*, std::complex$<$ _Tp $>$ *__w*)

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.

Definition at line 922 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __fpequal(), __fpreal(), __polylog_exp_asymp(), __polylog_exp_negative_real_part(), __polylog_exp_pos(), and __riemann_zeta().

Referenced by __polylog_exp().

8.3.2.199 template$<$typename _Tp $>$ std::complex$<$_Tp$>$ std::__detail::__polylog_exp_real_pos (_Tp *__s*, _Tp *__w*)

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.

Definition at line 956 of file sf_polylog.tcc.

References __fpequal(), __polylog_exp_asymp(), __polylog_exp_negative_real_part(), __polylog_exp_pos(), and __riemann_zeta().

8.3.2.200 template$<$typename _Tp $>$ _Tp std::__detail::__psi (_Tp *__x*)

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:

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

.

Definition at line 2330 of file sf_gamma.tcc.

References __psi_asymp().

Referenced by __expint_En_series(), __hydrogen(), __hyperg_reflect(), and __psi().

8.3.2.201 template<typename _Tp > _Tp std::__detail::__psi (unsigned int __n, _Tp __x)

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

The polygamma function is related to the Hurwitz zeta function:

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

Definition at line 2395 of file sf_gamma.tcc.

References __hurwitz_zeta(), __log_gamma(), and __psi().

8.3.2.202 template<typename _Tp > _Tp std::__detail::__psi_asymp (_Tp __x)

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:

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

Definition at line 2299 of file sf_gamma.tcc.

Referenced by __psi().

8.3.2.203 template<typename _Tp > _Tp std::__detail::__psi_series (_Tp __x)

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:

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

Definition at line 2268 of file sf_gamma.tcc.

8.3.2.204 template<typename _Tp > _Tp std::__detail::__riemann_zeta (_Tp __s)

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

The Riemann zeta function is defined by:

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

For s $<$ 1 use the reflection formula:

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

Parameters

__s	The argument

Definition at line 505 of file sf_zeta.tcc.

References __gamma(), __log_gamma(), __riemann_zeta_glob(), __riemann_zeta_product(), and __riemann_zeta_↩ sum().

Referenced by __polylog_exp_int_pos(), __polylog_exp_neg(), __polylog_exp_pos(), __polylog_exp_real_pos(), and evenzeta().

8.3.2.205 template<typename _Tp > _Tp std::__detail::__riemann_zeta_alt (_Tp __s)

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:

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

Definition at line 329 of file sf_zeta.tcc.

8.3.2.206 template<typename _Tp > _Tp std::__detail::__riemann_zeta_euler_maclaurin (_Tp __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.

Definition at line 282 of file sf_zeta.tcc.

References _S_Euler_Maclaurin_zeta.

8.3.2.207 template<typename _Tp > _Tp std::__detail::__riemann_zeta_glob (_Tp __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:

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

Definition at line 374 of file sf_zeta.tcc.

References __log_gamma().

Referenced by __riemann_zeta().

8.3.2.208 template$<$typename _Tp $>$ _Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)

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

Parameters

__s	The argument $s! = 1$

Definition at line 672 of file sf_zeta.tcc.

References __riemann_zeta_m_1_sum(), _S_num_zetam1, and _S_zetam1.

Referenced by __polylog_exp_neg().

8.3.2.209 template$<$typename _Tp $>$ _Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)

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:

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

Parameters

__s	The argument $s! = 1$

Definition at line 645 of file sf_zeta.tcc.

Referenced by __riemann_zeta_m_1().

8.3.2.210 template$<$typename _Tp $>$ _Tp std::__detail::__riemann_zeta_product (_Tp __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:

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

Parameters

__s	The argument

Definition at line 463 of file sf_zeta.tcc.

Referenced by __riemann_zeta().

8.3.2.211 template<typename _Tp > _Tp std::__detail::__riemann_zeta_sum (_Tp *__s*)

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:

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

Definition at line 254 of file sf_zeta.tcc.

Referenced by __riemann_zeta().

8.3.2.212 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp *__a*, _Tp *__x*)

Return the generalized sinus cardinal function

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

.

Definition at line 51 of file sf_cardinal.tcc.

8.3.2.213 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp *__x*)

Return the normalized sinus cardinal function

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

.

Definition at line 98 of file sf_cardinal.tcc.

8.3.2.214 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc_pi (_Tp *__x*)

Return the unnormalized sinus cardinal function

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

.

Definition at line 78 of file sf_cardinal.tcc.

8.3.2.215 template<typename _Tp > std::pair<_Tp, _Tp> std::__detail::__sincosint (_Tp __x)

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:

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

Definition at line 227 of file sf_trigint.tcc.

References __sincosint_asymp(), __sincosint_cont_frac(), and __sincosint_series().

8.3.2.216 template<typename _Tp > void std::__detail::__sincosint_asymp (_Tp __t, _Tp & _Si, _Tp & _Ci)

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.

Definition at line 163 of file sf_trigint.tcc.

Referenced by __sincosint().

8.3.2.217 template<typename _Tp > void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp & _Si, _Tp & _Ci)

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

Definition at line 55 of file sf_trigint.tcc.

Referenced by __sincosint().

8.3.2.218 template<typename _Tp > void std::__detail::__sincosint_series (_Tp __t, _Tp & _Si, _Tp & _Ci)

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

Definition at line 98 of file sf_trigint.tcc.

Referenced by __sincosint().

8.3.2.219 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __a, _Tp __x)

Return the generalized hyperbolic sinus cardinal function

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

.

Definition at line 124 of file sf_cardinal.tcc.

8.3.2.220 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __x)

Return the normalized hyperbolic sinus cardinal function

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

.

Definition at line 167 of file sf_cardinal.tcc.

8.3.2.221 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc_pi (_Tp __x)

Return the unnormalized hyperbolic sinus cardinal function

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

.

Definition at line 149 of file sf_cardinal.tcc.

8.3.2.222 template<typename _Tp > _Tp std::__detail::__sinhint (const _Tp __x)

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.

Definition at line 581 of file sf_expint.tcc.

References __expint_E1(), and __expint_Ei().

8.3.2.223 template<typename _Tp > _Tp std::__detail::__sph_bessel (unsigned int __n, _Tp __x)

Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x.

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 non-negative integral order
__x	The non-negative real argument

Returns

The output spherical Bessel function.

Definition at line 693 of file sf_bessel.tcc.

References __sph_bessel_jn().

8.3.2.224 template< typename _Tp > std::complex<_Tp> std::__detail::__sph_bessel (unsigned int *__n*, std::complex< _Tp > *__z*)

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.

Definition at line 1314 of file sf_hankel.tcc.

References __sph_hankel().

8.3.2.225 template<typename _Tp > void std::__detail::__sph_bessel_ik (unsigned int *__n*, _Tp *__x*, _Tp & *__i_n*, _Tp & *__k_n*, _Tp & *__ip_n*, _Tp & *__kp_n*)

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.

Definition at line 456 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik().

8.3.2.226 template<typename _Tp > void std::__detail::__sph_bessel_jn (unsigned int *__n*, _Tp *__x*, _Tp & *__j_n*, _Tp & *__n_n*, _Tp & *__jp_n*, _Tp & *__np_n*)

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.
out	*__j_n*	The output spherical Bessel function.
out	*__n_n*	The output spherical Neumann function.
out	*__jp_n*	The output derivative of the spherical Bessel function.
out	*__np_n*	The output derivative of the spherical Neumann function.

Definition at line 658 of file sf_bessel.tcc.

References __cyl_bessel_jn().

Referenced by __sph_bessel(), __sph_hankel_1(), __sph_hankel_2(), and __sph_neumann().

8.3.2.227 template< typename _Tp > void std::__detail::__sph_hankel (unsigned int *__n*, std::complex< _Tp > *__z*, std::complex< _Tp > & *_H1*, std::complex< _Tp > & *_H1p*, std::complex< _Tp > & *_H2*, std::complex< _Tp > & *_H2p*)

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.

Definition at line 1258 of file sf_hankel.tcc.

References __hankel().

Referenced by __sph_bessel(), __sph_hankel_1(), __sph_hankel_2(), and __sph_neumann().

8.3.2.228 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_hankel_1 (unsigned int *__n*, _Tp *__x*)

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.

Definition at line 762 of file sf_bessel.tcc.

References __sph_bessel_jn().

8.3.2.229 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_hankel_1 (unsigned int *__n*, std::complex< _Tp > *__z*)

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.

Definition at line 1282 of file sf_hankel.tcc.

References __sph_hankel().

8.3.2.230 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_hankel_2 (unsigned int __n, _Tp __x)

Return the spherical Hankel function of the second kind $h^{(2)}_n(x)$.

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

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

Parameters

__n	The non-negative integral order
__x	The non-negative real argument

Returns

The output spherical Neumann function.

Definition at line 794 of file sf_bessel.tcc.

References __sph_bessel_jn().

8.3.2.231 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)

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.

Definition at line 1298 of file sf_hankel.tcc.

References __sph_hankel().

8.3.2.232 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)

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.

Definition at line 350 of file sf_legendre.tcc.

References __sph_legendre().

8.3.2.233 template< typename _Tp > _Tp std::__detail::__sph_legendre (unsigned int *__l*, unsigned int *__m*, _Tp *__theta*)

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.

Definition at line 253 of file sf_legendre.tcc.

References __log_gamma(), and __poly_legendre_p().

Referenced by __hydrogen(), and __sph_harmonic().

8.3.2.234 template< typename _Tp > _Tp std::__detail::__sph_neumann (unsigned int *__n*, _Tp *__x*)

Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument 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.

Definition at line 730 of file sf_bessel.tcc.

References __sph_bessel_jn().

8.3.2.235 template<typename _Tp > std::complex<_Tp> std::__detail::__sph_neumann (unsigned int *__n*, std::complex< _Tp > *__z*)

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.

Definition at line 1330 of file sf_hankel.tcc.

References __sph_hankel().

8.3.2.236 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf (_Tp *__t*, unsigned int *__nu*)

Return the Students T probability function.

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

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

Parameters

__t	
__nu	

Definition at line 301 of file sf_beta.tcc.

References __beta_inc().

8.3.2.237 template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc (_Tp *__t*, unsigned int *__nu*)

Return the complement of the Students T probability function.

The complement of the students T propability function is:

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

Parameters

__t	
__nu	

Definition at line 324 of file sf_beta.tcc.

References __beta_inc().

8.3.2.238 template<typename _Tp > _Tp std::__detail::__theta_1 (_Tp *__nu*, _Tp *__x*)

Return the exponential theta-1 function of period `nu` and argument `x`.

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 190 of file sf_theta.tcc.

References __theta_2().

Referenced by __theta_s().

8.3.2.239 template<typename _Tp > _Tp std::__detail::__theta_2 (_Tp *__nu*, _Tp *__x*)

Return the exponential theta-2 function of period nu and argument x.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 162 of file sf_theta.tcc.

References __theta_2_asymp(), and __theta_2_sum().

Referenced by __theta_1(), and __theta_c().

8.3.2.240 template<typename _Tp > _Tp std::__detail::__theta_2_asymp (_Tp *__nu*, _Tp *__x*)

Compute and return the θ_2 function by series expansion.

Definition at line 103 of file sf_theta.tcc.

Referenced by __theta_2().

8.3.2.241 template<typename _Tp > _Tp std::__detail::__theta_2_sum (_Tp *__nu*, _Tp *__x*)

Compute and return the θ_1 function by series expansion.

Definition at line 49 of file sf_theta.tcc.

Referenced by __theta_2().

8.3.2.242 template<typename _Tp > _Tp std::__detail::__theta_3 (_Tp *__nu*, _Tp *__x*)

Return the exponential theta-3 function of period nu and argument x.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 1) argument
__x	The argument

Definition at line 216 of file sf_theta.tcc.

References __theta_3_asymp(), and __theta_3_sum().

Referenced by __theta_4(), and __theta_d().

8.3.2.243 template<typename _Tp > _Tp std::__detail::__theta_3_asymp (_Tp *__nu*, _Tp *__x*)

Compute and return the θ_3 function by asymptotic series expansion.

Definition at line 128 of file sf_theta.tcc.

Referenced by __theta_3().

8.3.2.244 template<typename _Tp > _Tp std::__detail::__theta_3_sum (_Tp *__nu*, _Tp *__x*)

Compute and return the θ_3 function by series expansion.

Definition at line 77 of file sf_theta.tcc.

Referenced by __theta_3().

8.3.2.245 template<typename _Tp > _Tp std::__detail::__theta_4 (_Tp *__nu*, _Tp *__x*)

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x} \right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 244 of file sf_theta.tcc.

References __theta_3().

Referenced by __theta_n().

8.3.2.246 template<typename _Tp > _Tp std::__detail::__theta_c (_Tp *__k*, _Tp *__x*)

Return the Neville θ_c function

Definition at line 337 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_2().

8.3.2.247 template<typename _Tp > _Tp std::__detail::__theta_d (_Tp __k, _Tp __x)

Return the Neville θ_d function

Definition at line 362 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_3().

8.3.2.248 template<typename _Tp > _Tp std::__detail::__theta_n (_Tp __k, _Tp __x)

Return the Neville θ_n function

Definition at line 387 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_4().

8.3.2.249 template<typename _Tp > _Tp std::__detail::__theta_s (_Tp __k, _Tp __x)

Return the Neville θ_s function

Definition at line 311 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_1().

8.3.2.250 template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n, signed order m, and real radial argument ρ and azimuthal angle ϕ.

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m <= n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

__poly_radial_jacobi).

Template Parameters

_Tp	The real type of the radial coordinate and azimuthal angle

Parameters

__n	The non-negative degree.
__m	The azimuthal order

__rho	The radial coordinate
__phi	The azimuthal angle

Definition at line 183 of file sf_jacobi.tcc.

References __poly_radial_jacobi().

8.3.2.251 template<typename _Tp > _Tp std::__detail::__znorm1 (_Tp __x)

Definition at line 58 of file sf_owens_t.tcc.

Referenced by __owens_t().

8.3.2.252 template<typename _Tp > _Tp std::__detail::__znorm2 (_Tp __x)

Definition at line 47 of file sf_owens_t.tcc.

Referenced by __owens_t().

8.3.2.253 template<typename _Tp = double> _Tp std::__detail::evenzeta (unsigned int __k)

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

$zeta(k)$

Definition at line 156 of file sf_polylog.tcc.

References __riemann_zeta().

8.3.3 Variable Documentation

8.3.3.1 constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100

Coefficients for Euler-Maclaurin summation of zeta functions.

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

where B_k are the Bernoulli numbers.

Definition at line 65 of file sf_zeta.tcc.

8.3.3.2 constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]

Definition at line 274 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

8.3.3.3 constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]

Definition at line 68 of file sf_zeta.tcc.

Referenced by __hurwitz_zeta_euler_maclaurin(), and __riemann_zeta_euler_maclaurin().

8.3.3.4 constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]

Definition at line 84 of file sf_gamma.tcc.

Referenced by __factorial(), and __log_factorial().

8.3.3.5 constexpr _Factorial_table<long double> std::__detail::_S_neg_double_factorial_table[999]

Definition at line 595 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

8.3.3.6 template<typename _Tp > constexpr std::size_t std::__detail::_S_num_double_factorials = 0

Definition at line 259 of file sf_gamma.tcc.

8.3.3.7 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301

Definition at line 264 of file sf_gamma.tcc.

8.3.3.8 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57

Definition at line 262 of file sf_gamma.tcc.

8.3.3.9 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301

Definition at line 266 of file sf_gamma.tcc.

8.3.3.10 template<typename _Tp > constexpr std::size_t std::__detail::_S_num_factorials = 0

Definition at line 69 of file sf_gamma.tcc.

8.3.3.11 template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171

Definition at line 74 of file sf_gamma.tcc.

8.3.3.12 template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35

Definition at line 72 of file sf_gamma.tcc.

8.3.3.13 template< > constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171

Definition at line 76 of file sf_gamma.tcc.

8.3.3.14 template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0

Definition at line 579 of file sf_gamma.tcc.

8.3.3.15 template< > constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150

Definition at line 584 of file sf_gamma.tcc.

8.3.3.16 template< > constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27

Definition at line 582 of file sf_gamma.tcc.

8.3.3.17 template< > constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999

Definition at line 586 of file sf_gamma.tcc.

8.3.3.18 constexpr size_t std::__detail::_S_num_zetam1 = 33

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

Definition at line 592 of file sf_zeta.tcc.

Referenced by __riemann_zeta_m_1().

8.3.3.19 constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]

Definition at line 596 of file sf_zeta.tcc.

Referenced by __riemann_zeta_m_1().

Chapter 9

Class Documentation

9.1 std::__detail::_Factorial_table< _Tp > Struct Template Reference

Public Attributes

- _Tp __factorial
- _Tp __log_factorial
- unsigned int __n

9.1.1 Detailed Description

template<typename _Tp>struct std::__detail::_Factorial_table< _Tp >

Definition at line 61 of file sf_gamma.tcc.

9.1.2 Member Data Documentation

9.1.2.1 template<typename _Tp > _Tp std::__detail::_Factorial_table< _Tp >::__factorial

Definition at line 64 of file sf_gamma.tcc.

9.1.2.2 template<typename _Tp > _Tp std::__detail::_Factorial_table< _Tp >::__log_factorial

Definition at line 65 of file sf_gamma.tcc.

9.1.2.3 template<typename _Tp > unsigned int std::__detail::_Factorial_table< _Tp >::__n

Definition at line 63 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

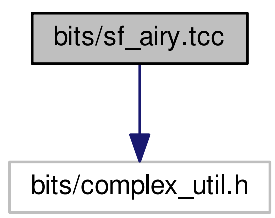
Chapter 10

File Documentation

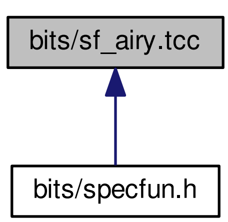
10.1 bits/sf_airy.tcc File Reference

`#include <bits/complex_util.h>`

Include dependency graph for sf_airy.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_AIRY_TCC 1

Functions

- template<typename _Tp >
  void std::__detail::__airy (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)

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

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__airy_ai (std::complex< _Tp > __z)

  Return the complex Airy Ai function.

- template<typename _Tp >
  void std::__detail::__airy_asymp_absarg_ge_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, int __sign=-1)

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

- template<typename _Tp >
  void std::__detail::__airy_asymp_absarg_lt_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip)

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

- template<typename _Tp >
  void std::__detail::__airy_bessel_i (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ip1d3, std::complex< _Tp > &_Im1d3, std::complex< _Tp > &_Ip2d3, std::complex< _Tp > &_Im2d3)

- template<typename _Tp >
  void std::__detail::__airy_bessel_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)

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

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__airy_bi (std::complex< _Tp > __z)

  Return the complex Airy Bi function.

- template<typename _Tp >
  void std::__detail::__airy_hyperg_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)

  This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = +-1/3$ and $\nu + -2/3$. 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.

10.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.1.2 Macro Definition Documentation

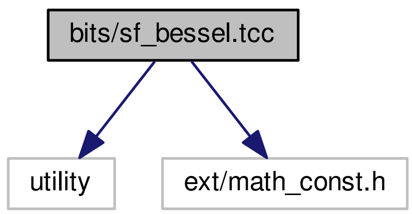
10.1.2.1 #define _GLIBCXX_BITS_SF_AIRY_TCC 1

Definition at line 31 of file sf_airy.tcc.

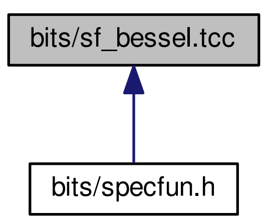
10.2 bits/sf_bessel.tcc File Reference

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

Include dependency graph for sf_bessel.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std

- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_BESSEL_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)

  This routine returns the cylindrical Bessel functions of order ν: J_ν or I_ν by series expansion.
- template<typename _Tp >
  _Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)

  Return the Bessel function of order ν: $J_\nu(x)$.
- template<typename _Tp >
  void std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)

  Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- template<typename _Tp >
  void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_←
  Npnu)

  This routine computes the asymptotic cylindrical Bessel and Neumann functions of order nu: $J_\nu(x)$, $N_\nu(x)$. Use this for $x >> nu^2 + 1$.
- template<typename _Tp >
  void std::__detail::__cyl_bessel_jn_steed (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_←
  Npnu)

  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.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)

  Return the cylindrical Hankel function of the first kind $H^{(1)}_\nu(x)$.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)

  Return the cylindrical Hankel function of the second kind $H^{(2)}_n u(x)$.
- template<typename _Tp >
  _Tp std::__detail::__cyl_neumann_n (_Tp __nu, _Tp __x)

  Return the Neumann function of order ν: $N_\nu(x)$.
- template<typename _Tp >
  void std::__detail::__gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)

  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.
- template<typename _Tp >
  _Tp std::__detail::__sph_bessel (unsigned int __n, _Tp __x)

  Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x.

- template<typename _Tp >
  void std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)

  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.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)

  Return the spherical Hankel function of the first kind $h^{(1)}_n(x)$.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, _Tp __x)

  Return the spherical Hankel function of the second kind $h^{(2)}_n(x)$.
- template<typename _Tp >
  _Tp std::__detail::__sph_neumann (unsigned int __n, _Tp __x)

  Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x.

10.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

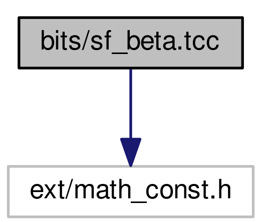
10.2.2 Macro Definition Documentation

10.2.2.1 #define _GLIBCXX_BITS_SF_BESSEL_TCC 1

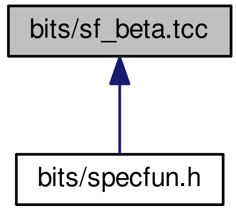
Definition at line 47 of file sf_bessel.tcc.

10.3 bits/sf_beta.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_BETA_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__beta (_Tp __a, _Tp __b)

  Return the beta function $B(a,b)$.
- template<typename _Tp >
  _Tp std::__detail::__beta_gamma (_Tp __a, _Tp __b)

  Return the beta function: $B(a,b)$.
- template<typename _Tp >
  _Tp std::__detail::__beta_inc (_Tp __a, _Tp __b, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__beta_lgamma (_Tp __a, _Tp __b)

  Return the beta function $B(a,b)$ using the log gamma functions.
- template<typename _Tp >
  _Tp std::__detail::__beta_product (_Tp __a, _Tp __b)

  Return the beta function $B(x,y)$ using the product form.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)

  Return the complementary binomial cumulative distribution function.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __↩
  nu2)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)

  Return the Students T probability function.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)

  Return the complement of the Students T probability function.

10.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.3.2 Macro Definition Documentation

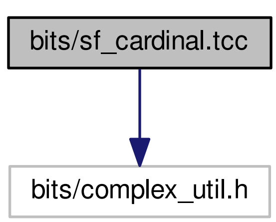
10.3.2.1 #define _GLIBCXX_BITS_SF_BETA_TCC 1

Definition at line 49 of file sf_beta.tcc.

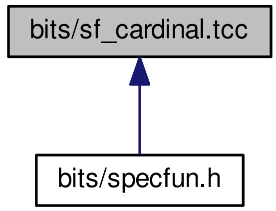
10.4 bits/sf_cardinal.tcc File Reference

`#include <bits/complex_util.h>`

Include dependency graph for sf_cardinal.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_CARDINAL_TCC 1

Functions

- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinc (_Tp __a, _Tp __x)

  Return the generalized sinus cardinal function

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

  .
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinc (_Tp __x)

  Return the normalized sinus cardinal function

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

  .
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinc_pi (_Tp __x)

  Return the unnormalized sinus cardinal function

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

  .
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __a, _Tp __x)

Return the generalized hyperbolic sinus cardinal function

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

.

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __x)

  Return the normalized hyperbolic sinus cardinal function

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

.

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)

  Return the unnormalized hyperbolic sinus cardinal function

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

.

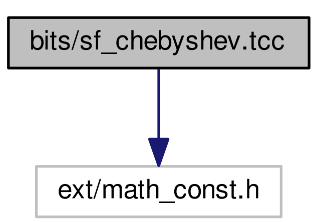
10.4.1 Macro Definition Documentation

10.4.1.1 #define _GLIBCXX_BITS_SF_CARDINAL_TCC 1

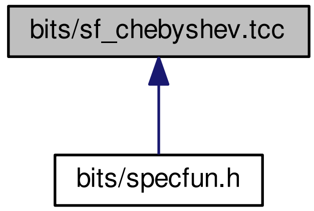
Definition at line 30 of file sf_cardinal.tcc.

10.5 bits/sf_chebyshev.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_CHEBYSHEV_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)
- template<typename _Tp >
  _Tp std::__detail::__chebyshev_t (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__chebyshev_u (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)

10.5.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.5.2 Macro Definition Documentation

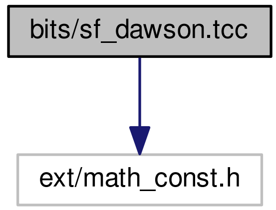
10.5.2.1 #define _GLIBCXX_SF_CHEBYSHEV_TCC 1

Definition at line 31 of file sf_chebyshev.tcc.

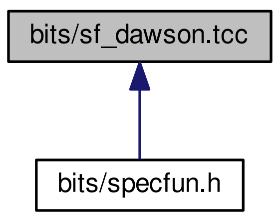
10.6 bits/sf_dawson.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_dawson.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_DAWSON_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__dawson (_Tp __x)

    Return the Dawson integral, $F(x)$, for real argument x.

- template<typename _Tp >
  _Tp std::__detail::__dawson_cont_frac (_Tp __x)

  Compute the Dawson integral using a sampling theorem representation.

- template<typename _Tp >
  _Tp std::__detail::__dawson_series (_Tp __x)

  Compute the Dawson integral using the series expansion.

10.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.6.2 Macro Definition Documentation

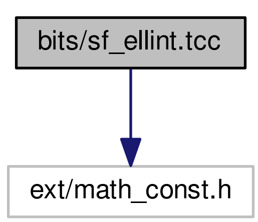
10.6.2.1 #define _GLIBCXX_SF_DAWSON_TCC 1

Definition at line 31 of file sf_dawson.tcc.

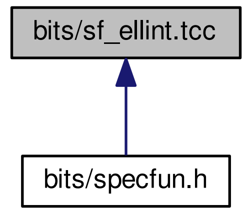
10.7 bits/sf_ellint.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_ellint.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_ELLINT_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_1 (_Tp __k)

  Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_2 (_Tp __k)

  Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_3 (_Tp __k, _Tp __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu) = \Pi(k,\nu,\pi/2)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_d (_Tp __k)
- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)
- template<typename _Tp >
  _Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)
- template<typename _Tp >
  _Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)

  Return the incomplete elliptic integral of the first kind $F(k,\phi)$ using the Carlson formulation.
- template<typename _Tp >
  _Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)

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

- template<typename _Tp >
  _Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k,\nu,\phi)$ *using the Carlson formulation.*
- template<typename _Tp >
  _Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)
- template<typename _Tp >
  _Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)
- template<typename _Tp >
  _Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)
- template<typename _Tp >
  _Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)
- template<typename _Tp >
  _Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)
- template<typename _Tp >
  _Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)

  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.*
- template<typename _Tp >
  _Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)

  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.*
- template<typename _Tp >
  _Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)

  Return the Carlson elliptic function $R_F(x,y,z)$ *of the first kind.*
- template<typename _Tp >
  _Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)

  Return the symmetric Carlson elliptic function of the second kind $R_G(x,y,z)$.
- template<typename _Tp >
  _Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)

  Return the Carlson elliptic function $R_J(x,y,z,p)$ *of the third kind.*
- template<typename _Tp >
  _Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)
- template<typename _Tp >
  _Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)

10.7.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

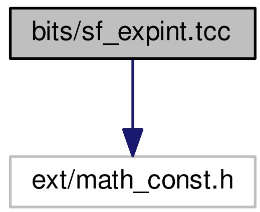
10.7.2 Macro Definition Documentation

10.7.2.1 #define _GLIBCXX_BITS_SF_ELLINT_TCC 1

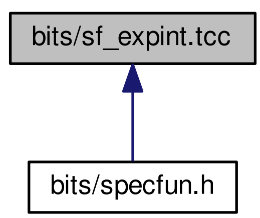
Definition at line 47 of file sf_ellint.tcc.

10.8 bits/sf_expint.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_EXPINT_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__coshint (const _Tp __x)

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

- template<typename _Tp >
  _Tp std::__detail::__expint (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$.
- template<typename _Tp >
  _Tp std::__detail::__expint (_Tp __x)

  Return the exponential integral $Ei(x)$.
- template<typename _Tp >
  _Tp std::__detail::__expint_asymp (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large argument.
- template<typename _Tp >
  _Tp std::__detail::__expint_E1 (_Tp __x)

  Return the exponential integral $E_1(x)$.
- template<typename _Tp >
  _Tp std::__detail::__expint_E1_asymp (_Tp __x)

  Return the exponential integral $E_1(x)$ by asymptotic expansion.
- template<typename _Tp >
  _Tp std::__detail::__expint_E1_series (_Tp __x)

  Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.
- template<typename _Tp >
  _Tp std::__detail::__expint_Ei (_Tp __x)

  Return the exponential integral $Ei(x)$.
- template<typename _Tp >
  _Tp std::__detail::__expint_Ei_asymp (_Tp __x)

  Return the exponential integral $Ei(x)$ by asymptotic expansion.
- template<typename _Tp >
  _Tp std::__detail::__expint_Ei_series (_Tp __x)

  Return the exponential integral $Ei(x)$ by series summation.
- template<typename _Tp >
  _Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ by continued fractions.
- template<typename _Tp >
  _Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.
- template<typename _Tp >
  _Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ by series summation.
- template<typename _Tp >
  _Tp std::__detail::__expint_large_n (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large order.
- template<typename _Tp >
  _Tp std::__detail::__logint (const _Tp __x)

  Return the logarithmic integral $li(x)$.
- template<typename _Tp >
  _Tp std::__detail::__sinhint (const _Tp __x)

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

10.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.8.2 Macro Definition Documentation

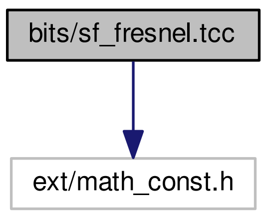
10.8.2.1 #define _GLIBCXX_BITS_SF_EXPINT_TCC 1

Definition at line 47 of file sf_expint.tcc.

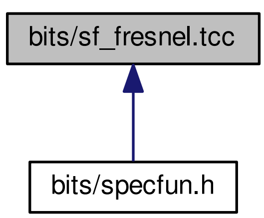
10.9 bits/sf_fresnel.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_fresnel.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std

- [std::__detail]

Macros

- #define _GLIBCXX_SF_FRESNEL_TCC 1

Functions

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__fresnel (const _Tp __x)

  Return the Fresnel cosine and sine integrals as a complex number $f[C(x) + iS(x) $f].
- template<typename _Tp >
  void std::__detail::__fresnel_cont_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)

  This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- template<typename _Tp >
  void std::__detail::__fresnel_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)

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

10.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.9.2 Macro Definition Documentation

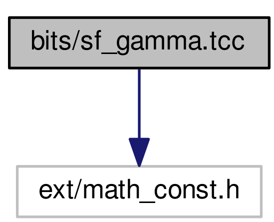
10.9.2.1 #define _GLIBCXX_SF_FRESNEL_TCC 1

Definition at line 31 of file sf_fresnel.tcc.

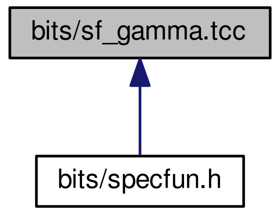
10.10 bits/sf_gamma.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_gamma.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- struct std::__detail::_Factorial_table< _Tp >

Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_GAMMA_TCC 1

Functions

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int __n)

  This returns Bernoulli number B_n.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int __n)

  This returns Bernoulli number B_n.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)

  This returns Bernoulli numbers from a table or by summation for larger values.
- template<typename _Tp >
  _Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)

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

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

  .
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__chi_squared_pdf (_Tp __chi2, unsigned int __nu)

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*.*

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__chi_squared_pdfc (_Tp __chi2, unsigned int __nu)

  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*.*
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)

  Return the double factorial of the integer n.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)

  Return the factorial of the integer n.
- template<typename _Tp >
  _Tp std::__detail::__gamma (_Tp __x)

  Return $\Gamma(x)$*.*
- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)

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

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

  .
- template<typename _Tp >
  _Tp std::__detail::__gamma_p (_Tp __a, _Tp __x)

  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.
- template<typename _Tp >
  _Tp std::__detail::__gamma_q (_Tp __a, _Tp __x)

  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.
- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__gamma_series (_Tp __a, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)

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

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

  .

- template<typename _Tp >
  _Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)

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

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

  .

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)

  Return the logarithm of the factorial of the integer n.

- template<typename _Tp >
  _Tp std::__detail::__log_gamma (_Tp __x)

  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.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)

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

- template<typename _Tp >
  _Tp std::__detail::__log_gamma_sign (_Tp __x)

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

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)

  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

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

  .

- template<typename _Tp >
  _Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp std::__detail::__pochhammer_l (_Tp __a, _Tp __n)

  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].

- template<typename _Tp >
  _Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)

  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.

- template<typename _Tp >
  _Tp std::__detail::__psi (_Tp __x)

  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:

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

  .

- template<typename _Tp >
  _Tp std::__detail::__psi (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp std::__detail::__psi_asymp (_Tp __x)

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

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

  .
- template<typename _Tp >
  _Tp std::__detail::__psi_series (_Tp __x)

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

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

  .

Variables

- constexpr _Factorial_table< long double > std::__detail::_S_double_factorial_table [301]
- constexpr _Factorial_table< long double > std::__detail::_S_factorial_table [171]
- constexpr _Factorial_table< long double > std::__detail::_S_neg_double_factorial_table [999]
- template<typename _Tp >
  constexpr std::size_t std::__detail::_S_num_double_factorials = 0
- template<>
  constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301
- template<>
  constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57
- template<>
  constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301
- template<typename _Tp >
  constexpr std::size_t std::__detail::_S_num_factorials = 0
- template<>
  constexpr std::size_t std::__detail::_S_num_factorials< double > = 171
- template<>
  constexpr std::size_t std::__detail::_S_num_factorials< float > = 35
- template<>
  constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171
- template<typename _Tp >
  constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0
- template<>
  constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150
- template<>
  constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27
- template<>
  constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999

10.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

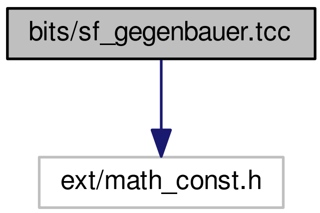
10.10.2 Macro Definition Documentation

10.10.2.1 #define _GLIBCXX_BITS_SF_GAMMA_TCC 1

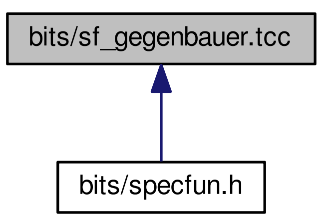
Definition at line 49 of file sf_gamma.tcc.

10.11 bits/sf_gegenbauer.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_GEGENBAUER_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)

10.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.11.2 Macro Definition Documentation

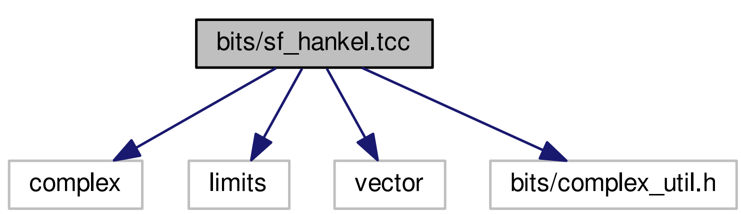
10.11.2.1 #define _GLIBCXX_SF_GEGENBAUER_TCC 1

Definition at line 31 of file sf_gegenbauer.tcc.

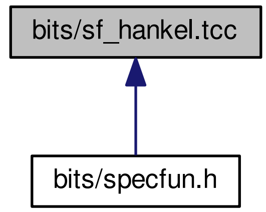
10.12 bits/sf_hankel.tcc File Reference

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

Include dependency graph for sf_hankel.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_HANKEL_TCC 1

Functions

- template<typename _Tp >
  void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)

  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.*
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Bessel function.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Hankel function of the first kind.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Hankel function of the second kind.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  Return the complex cylindrical Neumann function.
- template<typename _Tp >
  void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)
- template<typename _Tp >
  void std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

- template<typename _Tp >
  void std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)
- template<typename _Tp >
  void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)

  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.
- template<typename _Tp >
  void std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

  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.
- template<typename _Tp >
  void std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)

  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.
- template<typename _Tp >
  void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &_Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &_Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)

  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.
- template<typename _Tp >
  void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > _Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > _Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &_H1sum, std::complex< _Tp > &_H1psum, std::complex< _Tp > &_H2sum, std::complex< _Tp > &_H2psum)

  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`.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Bessel function.
- template<typename _Tp >
  void std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)

  Helper to compute complex spherical Hankel functions and their derivatives.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)

Return the complex spherical Hankel function of the first kind.

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Hankel function of the second kind.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)

  Return the complex spherical Neumann function.

10.12.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.12.2 Macro Definition Documentation

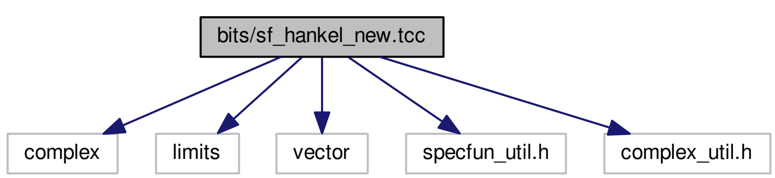
10.12.2.1 #define _GLIBCXX_BITS_SF_HANKEL_TCC 1

Definition at line 31 of file sf_hankel.tcc.

10.13 bits/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

10.13.1 Macro Definition Documentation

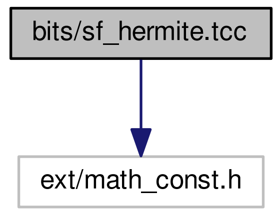
10.13.1.1 #define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1

Definition at line 31 of file sf_hankel_new.tcc.

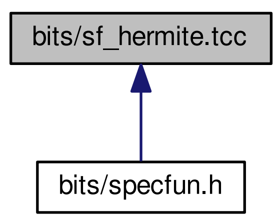
10.14 bits/sf_hermite.tcc File Reference

#include <ext/math_const.h>

Include dependency graph for sf_hermite.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_HERMITE_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__poly_hermite (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
  _Tp std::__detail::__poly_hermite_asymp (unsigned int __n, _Tp __x)

  This routine returns the Hermite polynomial of large order n: $H_n(x)$. We assume here that x >= 0.
- template<typename _Tp >
  _Tp std::__detail::__poly_hermite_recursion (unsigned int __n, _Tp __x)

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

10.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

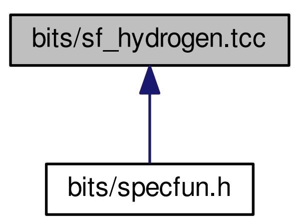
10.14.2 Macro Definition Documentation

10.14.2.1 #define _GLIBCXX_BITS_SF_HERMITE_TCC 1

Definition at line 42 of file sf_hermite.tcc.

10.15 bits/sf_hydrogen.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1

Functions

- template< typename _Tp >
  std::complex< _Tp > std::__detail::__hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp _Z, const _Tp __r, const _Tp __theta, const _Tp __phi)

10.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

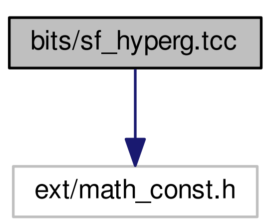
10.15.2 Macro Definition Documentation

10.15.2.1 #define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1

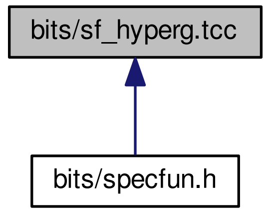
Definition at line 31 of file sf_hydrogen.tcc.

10.16 bits/sf_hyperg.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_HYPERG_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)

  Return the confluent hypergeometric function ${}_1F_1(a;c;x)$.
- template<typename _Tp >
  _Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)

  Return the confluent hypergeometric limit function ${}_0F_1(-;c;x)$.
- template<typename _Tp >
  _Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)

  This routine returns the confluent hypergeometric limit function by series expansion.
- template<typename _Tp >
  _Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)

  Return the hypergeometric function ${}_1F_1(a;c;x)$ *by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.*
- template<typename _Tp >
  _Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)

  This routine returns the confluent hypergeometric function by series expansion.
- template<typename _Tp >
  _Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

  Return the hypergeometric function ${}_2F_1(a,b;c;x)$.
- template<typename _Tp >
  _Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)

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

- template<typename _Tp >
  _Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

  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.

- template<typename _Tp >
  _Tp std::__detail::__hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

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

10.16.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.16.2 Macro Definition Documentation

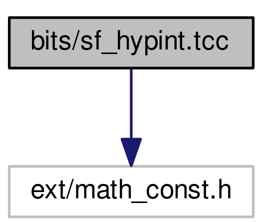
10.16.2.1 #define _GLIBCXX_BITS_SF_HYPERG_TCC 1

Definition at line 44 of file sf_hyperg.tcc.

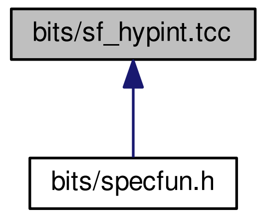
10.17 bits/sf_hypint.tcc File Reference

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

Include dependency graph for sf_hypint.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_HYPINT_TCC 1

Functions

- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)

  This function returns the hyperbolic cosine $Ci(x)$ and hyperbolic sine $Si(x)$ integrals as a pair.
- template<typename _Tp >
  void std::__detail::__chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)

  This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- template<typename _Tp >
  void std::__detail::__chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)

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

10.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.17.2 Macro Definition Documentation

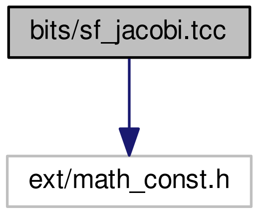
10.17.2.1 #define _GLIBCXX_SF_HYPINT_TCC 1

Definition at line 31 of file sf_hypint.tcc.

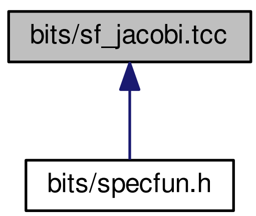
10.18 bits/sf_jacobi.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_jacobi.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_JACOBI_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)

- template<typename _Tp >
  _Tp std::__detail::__poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)

10.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.18.2 Macro Definition Documentation

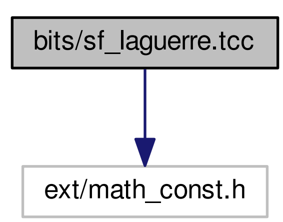
10.18.2.1 #define _GLIBCXX_SF_JACOBI_TCC 1

Definition at line 31 of file sf_jacobi.tcc.

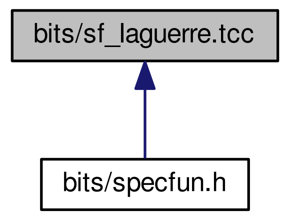
10.19 bits/sf_laguerre.tcc File Reference

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

Include dependency graph for sf_laguerre.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)

  This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.
- template<typename _Tp >
  _Tp std::__detail::__laguerre (unsigned int __n, _Tp __x)

  This routine returns the Laguerre polynomial of order n: $L_n(x)$.
- template<typename _Tpa , typename _Tp >
  _Tp std::__detail::__poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)

  This routine returns the associated Laguerre polynomial of order n, degree α*:* $L_n^alpha(x)$.
- template<typename _Tpa , typename _Tp >
  _Tp std::__detail::__poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)

  Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- template<typename _Tpa , typename _Tp >
  _Tp std::__detail::__poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)

  This routine returns the associated Laguerre polynomial of order n*, degree* $\alpha > -1$ *for large n. Abramowitz & Stegun, 13.5.21.*
- template<typename _Tpa , typename _Tp >
  _Tp std::__detail::__poly_laguerre_recursion (unsigned int __n, _Tpa __alpha1, _Tp __x)

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

10.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.19.2 Macro Definition Documentation

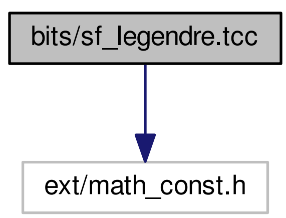
10.19.2.1 #define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1

Definition at line 44 of file sf_laguerre.tcc.

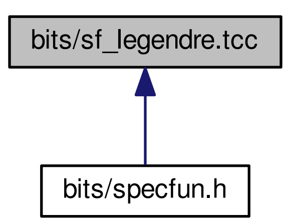
10.20 bits/sf_legendre.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_legendre.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std

- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1

Functions

- template< typename _Tp >
  _Tp std::__detail::__assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)

  Return the associated Legendre function by recursion on l and downward recursion on m.
- template< typename _Tp >
  _Tp std::__detail::__legendre_q (unsigned int __l, _Tp __x)

  Return the Legendre function of the second kind by upward recursion on order l.
- template< typename _Tp >
  _Tp std::__detail::__poly_legendre_p (unsigned int __l, _Tp __x)

  Return the Legendre polynomial by upward recursion on order l.
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)

  Return the spherical harmonic function.
- template< typename _Tp >
  _Tp std::__detail::__sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)

  Return the spherical associated Legendre function.

10.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.20.2 Macro Definition Documentation

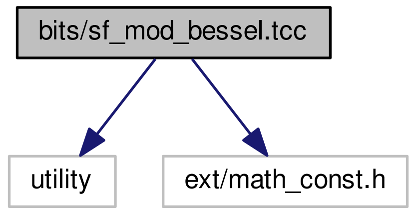
10.20.2.1 #define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1

Definition at line 47 of file sf_legendre.tcc.

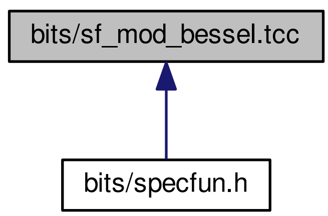
10.21 bits/sf_mod_bessel.tcc File Reference

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

Include dependency graph for sf_mod_bessel.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1

Functions

- template<typename _Tp >
  void std::__detail::__airy (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)

  Compute the Airy functions $Ai(x)$ *and* $Bi(x)$ *and their first derivatives* $Ai'(x)$ *and* $Bi(x)$ *respectively.*
- template<typename _Tp >
  _Tp std::__detail::__cyl_bessel_i (_Tp __nu, _Tp __x)

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

- template<typename _Tp >
  void std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)

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

- template<typename _Tp >
  void std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_↩ Kpnu)

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

- template<typename _Tp >
  void std::__detail::__cyl_bessel_ik_steed (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)

  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.

- template<typename _Tp >
  _Tp std::__detail::__cyl_bessel_k (_Tp __nu, _Tp __x)

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

- template<typename _Tp >
  void std::__detail::__fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std↩ ::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)

  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.

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

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

  .

- template<typename _Tp >
  void std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)

  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.

10.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

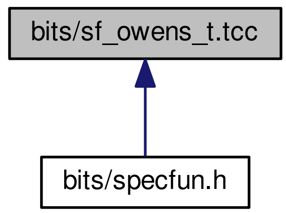
10.21.2 Macro Definition Documentation

10.21.2.1 #define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1

Definition at line 47 of file sf_mod_bessel.tcc.

10.22 bits/sf_owens_t.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_OWENS_T_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__gauss (_Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__owens_t (_Tp __h, _Tp __a)
- template<typename _Tp >
  _Tp std::__detail::__znorm1 (_Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__znorm2 (_Tp __x)

10.22.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.22.2 Macro Definition Documentation

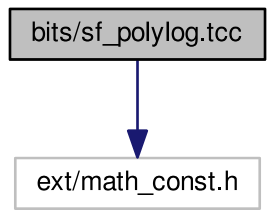
10.22.2.1 #define _GLIBCXX_BITS_SF_OWENS_T_TCC 1

Definition at line 31 of file sf_owens_t.tcc.

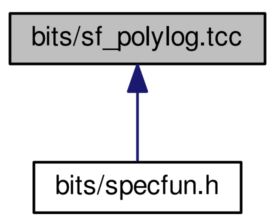
10.23 bits/sf_polylog.tcc File Reference

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

Include dependency graph for sf_polylog.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_POLYLOG_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__bose_einstein (_Tp __s, _Tp __x)

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clamp_0_m2pi (std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clamp_pi (std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clausen (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp std::__detail::__clausen (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp std::__detail::__clausen_c (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp std::__detail::__clausen_c (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp std::__detail::__clausen_s (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp std::__detail::__clausen_s (unsigned int __m, _Tp __w)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_beta (std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_beta (_Tp __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __w)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_eta (_Tp __w)
- template<typename _Tp >
  _Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)
- template<typename _Tp >
  bool std::__detail::__fpequal (const _Tp &__a, const _Tp &__b)
- template<typename _Tp >
  bool std::__detail::__fpimag (const std::complex< _Tp > &__w)
- template<typename _Tp >
  bool std::__detail::__fpimag (const _Tp)
- template<typename _Tp >
  bool std::__detail::__fpreal (const std::complex< _Tp > &__w)
- template<typename _Tp >
  bool std::__detail::__fpreal (const _Tp)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
  _Tp std::__detail::__polylog (_Tp __s, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp , typename ArgType >
  __gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > std::__detail::__polylog_exp (_Tp __s, Arg↩
  Type __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_int_neg (int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)

- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, _Tp __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_neg (int __s, std::complex< _Tp > __w)
- template< typename _Tp , int __sigma>
std::complex< _Tp > std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)
- template< typename _Tp , int __sigma>
std::complex< _Tp > std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)
- template< typename _PowTp , typename _Tp >
_Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)
- template< typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)
- template< typename _Tp = double>
_Tp std::__detail::evenzeta (unsigned int __k)

10.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.23.2 Macro Definition Documentation

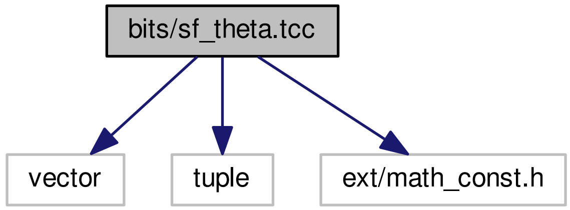
10.23.2.1 #define _GLIBCXX_BITS_SF_POLYLOG_TCC 1

Definition at line 41 of file sf_polylog.tcc.

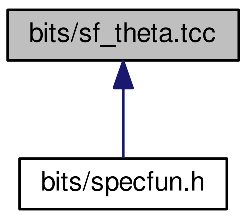
10.24 bits/sf_theta.tcc File Reference

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

Include dependency graph for sf_theta.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_THETA_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__ellnome (_Tp __k)
- template<typename _Tp >
  _Tp std::__detail::__ellnome_k (_Tp __k)
- template<typename _Tp >
  _Tp std::__detail::__ellnome_series (_Tp __k)

- template<typename _Tp >
  std::tuple< _Tp, _Tp, _Tp > std::__detail::__jacobi_sncndn (_Tp __k, _Tp __u)
- template<typename _Tp >
  _Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_2_asymp (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_2_sum (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_3_asymp (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_c (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_d (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_n (_Tp __k, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__theta_s (_Tp __k, _Tp __x)

10.24.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.24.2 Macro Definition Documentation

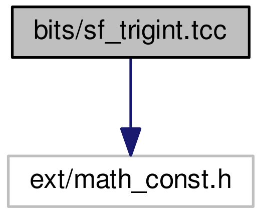
10.24.2.1 #define _GLIBCXX_SF_THETA_TCC 1

Definition at line 31 of file sf_theta.tcc.

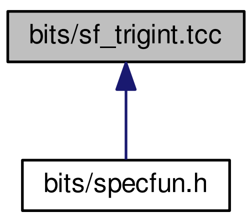
10.25 bits/sf_trigint.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for sf_trigint.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_SF_TRIGINT_TCC 1

Enumerations

- enum { std::__detail::SININT, std::__detail::COSINT }

Functions

- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__sincosint (_Tp __x)

  This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.
- template<typename _Tp >
  void std::__detail::__sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)

  This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- template<typename _Tp >
  void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)

  This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- template<typename _Tp >
  void std::__detail::__sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)

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

10.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.25.2 Macro Definition Documentation

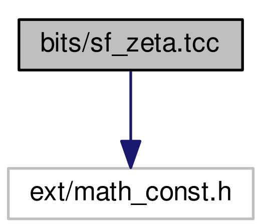
10.25.2.1 #define _GLIBCXX_SF_TRIGINT_TCC 1

Definition at line 31 of file sf_trigint.tcc.

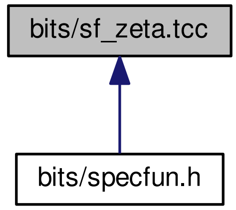
10.26 bits/sf_zeta.tcc File Reference

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

Include dependency graph for sf_zeta.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_ZETA_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__dilog (_Tp __x)

  Compute the dilogarithm function $Li_2(x)$ by summation for x <= 1.
- template<typename _Tp >
  _Tp std::__detail::__hurwitz_zeta (_Tp __s, _Tp __a)

  Return the Hurwitz zeta function $\zeta(s,a)$ for all s != 1 and a > -1.
- template<typename _Tp >
  _Tp std::__detail::__hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)

  Return the Hurwitz zeta function $\zeta(s,a)$ for all s != 1 and a > -1.
- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta (_Tp __s)

  Return the Riemann zeta function $\zeta(s)$.
- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_alt (_Tp __s)

  Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s > 0.
- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_euler_maclaurin (_Tp __s)

  Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for s > 0.
- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_glob (_Tp __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.

- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)

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

- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)

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

- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_product (_Tp __s)

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

- template<typename _Tp >
  _Tp std::__detail::__riemann_zeta_sum (_Tp __s)

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

Variables

- constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100
- constexpr long double std::__detail::_S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]
- constexpr size_t std::__detail::_S_num_zetam1 = 33
- constexpr long double std::__detail::_S_zetam1 [_S_num_zetam1]

10.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.26.2 Macro Definition Documentation

10.26.2.1 #define _GLIBCXX_BITS_SF_ZETA_TCC 1

Definition at line 46 of file sf_zeta.tcc.

10.27 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_gamma.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_dawson.tcc>
#include <bits/sf_ellint.tcc>
#include <bits/sf_expint.tcc>
#include <bits/sf_fresnel.tcc>
#include <bits/sf_gegenbauer.tcc>
#include <bits/sf_hyperg.tcc>
#include <bits/sf_hypint.tcc>
#include <bits/sf_jacobi.tcc>
#include <bits/sf_laguerre.tcc>
#include <bits/sf_legendre.tcc>
#include <bits/sf_hydrogen.tcc>
#include <bits/sf_mod_bessel.tcc>
#include <bits/sf_hermite.tcc>
#include <bits/sf_theta.tcc>
#include <bits/sf_trigint.tcc>
#include <bits/sf_zeta.tcc>
#include <bits/sf_owens_t.tcc>
#include <bits/sf_polylog.tcc>
#include <bits/sf_airy.tcc>
#include <bits/sf_hankel.tcc>
```

Include dependency graph for specfun.h:

Namespaces

- __gnu_cxx
- std

Macros

- #define __cpp_lib_math_special_functions 201603L
- #define __STDCPP_MATH_SPEC_FUNCS__ 201003L

Enumerations

- enum { __gnu_cxx::_GLIBCXX_JACOBI_SN, __gnu_cxx::_GLIBCXX_JACOBI_CN, __gnu_cxx::_GLIBCXX_J↩ ACOBI_DN }

Functions

- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)
- float __gnu_cxx::airy_aif (float __x)
- long double __gnu_cxx::airy_ail (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)
- float __gnu_cxx::airy_bif (float __x)
- long double __gnu_cxx::airy_bil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)
- float std::assoc_laguerref (unsigned int __n, unsigned int __m, float __x)
- long double std::assoc_laguerrel (unsigned int __n, unsigned int __m, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::assoc_legendre (unsigned int __l, unsigned int __m, _Tp __x)
- float std::assoc_legendref (unsigned int __l, unsigned int __m, float __x)
- long double std::assoc_legendrel (unsigned int __l, unsigned int __m, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::bernoulli (unsigned int __n)
- float __gnu_cxx::bernoullif (unsigned int __n)
- long double __gnu_cxx::bernoullil (unsigned int __n)
- template< typename _Tpa , typename _Tpb >
  __gnu_cxx::__promote_2< _Tpa, _Tpb >::__type std::beta (_Tpa __a, _Tpb __b)
- float std::betaf (float __a, float __b)
- long double std::betal (long double __a, long double __b)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::bincoef (unsigned int __n, unsigned int __k)
- float __gnu_cxx::bincoeff (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::bincoefl (unsigned int __n, unsigned int __k)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_t (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_tf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_tl (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_u (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_uf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_ul (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x)
- float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x)
- long double __gnu_cxx::chebyshev_wl (unsigned int __n, long double __x)

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __w)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __w)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c (unsigned int __m, _Tp __w)
- float __gnu_cxx::clausen_cf (unsigned int __m, float __w)
- long double __gnu_cxx::clausen_cl (unsigned int __m, long double __w)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s (unsigned int __m, _Tp __w)
- float __gnu_cxx::clausen_sf (unsigned int __m, float __w)
- long double __gnu_cxx::clausen_sl (unsigned int __m, long double __w)
- float __gnu_cxx::clausenf (unsigned int __m, float __w)
- std::complex< float > __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __w)
- long double __gnu_cxx::clausenl (unsigned int __m, long double __w)
- std::complex< long double > __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __w)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1 (_Tp __k)
- float std::comp_ellint_1f (float __k)
- long double std::comp_ellint_1l (long double __k)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2 (_Tp __k)
- float std::comp_ellint_2f (float __k)
- long double std::comp_ellint_2l (long double __k)
- template<typename _Tp , typename _Tpn >
  __gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3 (_Tp __k, _Tpn __nu)
- float std::comp_ellint_3f (float __k, float __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `float` modulus k.
- long double std::comp_ellint_3l (long double __k, long double __nu)

  Return the complete elliptic integral of the third kind $\Pi(k,\nu)$ for `long double` modulus k.
- template<typename _Tk >
  __gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d (_Tk __k)
- float __gnu_cxx::comp_ellint_df (float __k)
- long double __gnu_cxx::comp_ellint_dl (long double __k)
- float __gnu_cxx::comp_ellint_rf (float __x, float __y)
- long double __gnu_cxx::comp_ellint_rf (long double __x, long double __y)
- template<typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf (_Tx __x, _Ty __y)
- float __gnu_cxx::comp_ellint_rg (float __x, float __y)
- long double __gnu_cxx::comp_ellint_rg (long double __x, long double __y)
- template<typename _Tx , typename _Ty >
  __gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)
- template<typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)
- template<typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_2< _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg_lim (_Tpc __c, _Tp __x)
- float __gnu_cxx::conf_hyperg_limf (float __c, float __x)
- long double __gnu_cxx::conf_hyperg_liml (long double __c, long double __x)
- float __gnu_cxx::conf_hypergf (float __a, float __c, float __x)
- long double __gnu_cxx::conf_hypergl (long double __a, long double __c, long double __x)

- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::coshint (_Tp __x)
- float __gnu_cxx::coshintf (float __x)
- long double __gnu_cxx::coshintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::cosint (_Tp __x)
- float __gnu_cxx::cosintf (float __x)
- long double __gnu_cxx::cosintl (long double __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_if (float __nu, float __x)
- long double std::cyl_bessel_il (long double __nu, long double __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_jf (float __nu, float __x)
- long double std::cyl_bessel_jl (long double __nu, long double __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_k (_Tpnu __nu, _Tp __x)
- float std::cyl_bessel_kf (float __nu, float __x)
- long double std::cyl_bessel_kl (long double __nu, long double __x)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)
- std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)
- std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)
- std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)
- std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_neumann (_Tpnu __nu, _Tp __x)
- float std::cyl_neumannf (float __nu, float __x)
- long double std::cyl_neumannl (long double __nu, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)
- float __gnu_cxx::dawsonf (float __x)
- long double __gnu_cxx::dawsonl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)

- float __gnu_cxx::digammaf (float __z)
- long double __gnu_cxx::digammal (long double __z)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dilog (_Tp __x)
- float __gnu_cxx::dilogf (float __x)
- long double __gnu_cxx::dilogl (long double __x)
- template<typename _Tp >
  _Tp __gnu_cxx::dirichlet_beta (_Tp __s)
- float __gnu_cxx::dirichlet_betaf (float __s)
- long double __gnu_cxx::dirichlet_betal (long double __s)
- template<typename _Tp >
  _Tp __gnu_cxx::dirichlet_eta (_Tp __s)
- float __gnu_cxx::dirichlet_etaf (float __s)
- long double __gnu_cxx::dirichlet_etal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::double_factorial (int __n)
- float __gnu_cxx::double_factorialf (int __n)
- long double __gnu_cxx::double_factoriall (int __n)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_1 (_Tp __k, _Tpp __phi)
- float std::ellint_1f (float __k, float __phi)
- long double std::ellint_1l (long double __k, long double __phi)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_2 (_Tp __k, _Tpp __phi)
- float std::ellint_2f (float __k, float __phi)

  Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for `float` argument.
- long double std::ellint_2l (long double __k, long double __phi)

  Return the incomplete elliptic integral of the second kind $E(k, \phi)$.
- template<typename _Tp , typename _Tpn , typename _Tpp >
  __gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type std::ellint_3 (_Tp __k, _Tpn __nu, _Tpp __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.
- float std::ellint_3f (float __k, float __nu, float __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for `float` argument.
- long double std::ellint_3l (long double __k, long double __nu, long double __phi)

  Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.
- template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tk, _Tp, _Ta, _Tb > __gnu_cxx::ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float __gnu_cxx::ellint_celf (float __k_c, float __p, float __a, float __b)
- long double __gnu_cxx::ellint_cell (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::ellint_d (_Tk __k, _Tphi __phi)
- float __gnu_cxx::ellint_df (float __k, float __phi)
- long double __gnu_cxx::ellint_dl (long double __k, long double __phi)
- template<typename _Tp , typename _Tk >
  __gnu_cxx::__promote_num_t< _Tp, _Tk > __gnu_cxx::ellint_el1 (_Tp __x, _Tk __k_c)
- float __gnu_cxx::ellint_el1f (float __x, float __k_c)
- long double __gnu_cxx::ellint_el1l (long double __x, long double __k_c)
- template<typename _Tp , typename _Tk , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > __gnu_cxx::ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)

- float __gnu_cxx::ellint_el2f (float __x, float __k_c, float __a, float __b)
- long double __gnu_cxx::ellint_el2l (long double __x, long double __k_c, long double __a, long double __b)
- template< typename _Tx , typename _Tk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > __gnu_cxx::ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p)
- float __gnu_cxx::ellint_el3f (float __x, float __k_c, float __p)
- long double __gnu_cxx::ellint_el3l (long double __x, long double __k_c, long double __p)
- template< typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::ellint_rc (_Tp __x, _Up __y)
- float __gnu_cxx::ellint_rcf (float __x, float __y)
- long double __gnu_cxx::ellint_rcl (long double __x, long double __y)
- template< typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rd (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rdf (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rdl (long double __x, long double __y, long double __z)
- template< typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rf (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rff (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rfl (long double __x, long double __y, long double __z)
- template< typename _Tp , typename _Up , typename _Vp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rg (_Tp __x, _Up __y, _Vp __z)
- float __gnu_cxx::ellint_rgf (float __x, float __y, float __z)
- long double __gnu_cxx::ellint_rgl (long double __x, long double __y, long double __z)
- template< typename _Tp , typename _Up , typename _Vp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > __gnu_cxx::ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float __gnu_cxx::ellint_rjf (float __x, float __y, float __z, float __p)
- long double __gnu_cxx::ellint_rjl (long double __x, long double __y, long double __z, long double __p)
- template< typename _Tp >
  _Tp __gnu_cxx::ellnome (_Tp __k)
- float __gnu_cxx::ellnomef (float __k)
- long double __gnu_cxx::ellnomel (long double __k)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::expint (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_e1 (_Tp __x)
- float __gnu_cxx::expint_e1f (float __x)
- long double __gnu_cxx::expint_e1l (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_en (unsigned int __n, _Tp __x)
- float __gnu_cxx::expint_enf (unsigned int __n, float __x)
- long double __gnu_cxx::expint_enl (unsigned int __n, long double __x)
- float std::expintf (float __x)
- long double std::expintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::factorial (unsigned int __n)
- float __gnu_cxx::factorialf (unsigned int __n)
- long double __gnu_cxx::factoriall (unsigned int __n)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_c (_Tp __x)
- float __gnu_cxx::fresnel_cf (float __x)
- long double __gnu_cxx::fresnel_cl (long double __x)

- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_s (_Tp __x)
- float __gnu_cxx::fresnel_sf (float __x)
- long double __gnu_cxx::fresnel_sl (long double __x)
- template<typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_l (_Tn __n, _Tp __x)
- float __gnu_cxx::gamma_lf (float __n, float __x)
- long double __gnu_cxx::gamma_ll (long double __n, long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_p (_Ta __a, _Tp __x)
- float __gnu_cxx::gamma_pf (float __a, float __x)
- long double __gnu_cxx::gamma_pl (long double __a, long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_q (_Ta __a, _Tp __x)
- float __gnu_cxx::gamma_qf (float __a, float __x)
- long double __gnu_cxx::gamma_ql (long double __a, long double __x)
- template<typename _Tn , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_u (_Tn __n, _Tp __x)
- float __gnu_cxx::gamma_uf (float __n, float __x)
- long double __gnu_cxx::gamma_ul (long double __n, long double __x)
- template<typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)
- float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)
- long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)
- float std::hermitef (unsigned int __n, float __x)
- long double std::hermitel (unsigned int __n, long double __x)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi)
- float __gnu_cxx::heuman_lambdaf (float __k, float __phi)
- long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)
- template<typename _Tp , typename _Up >
  __gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta (_Tp __s, _Up __a)
- template<typename _Tp , typename _Up >
  std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)
- float __gnu_cxx::hurwitz_zetaf (float __s, float __a)
- long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)
- template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)
- long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)
- float __gnu_cxx::ibetacf (float __a, float __b, float __x)
- long double __gnu_cxx::ibetacl (long double __a, long double __b, long double __x)
- float __gnu_cxx::ibetaf (float __a, float __b, float __x)
- long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)

- template<typename _Talpha , typename _Tbeta , typename _Tp >
  __gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_cnf (float __k, float __u)
- long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_dnf (float __k, float __u)
- long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn (_Kp __k, _Up __u)
- float __gnu_cxx::jacobi_snf (float __k, float __u)
- long double __gnu_cxx::jacobi_snl (long double __k, long double __u)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta (_Tk __k, _Tphi __phi)
- float __gnu_cxx::jacobi_zetaf (float __k, float __phi)
- long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)
- float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)
- long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::laguerre (unsigned int __n, _Tp __x)
- float std::laguerref (unsigned int __n, float __x)
- long double std::laguerrel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)
- float __gnu_cxx::lbincoeff (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::lbincoefl (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)
- float __gnu_cxx::ldouble_factorialf (int __n)
- long double __gnu_cxx::ldouble_factoriall (int __n)
- template<typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::legendre (unsigned int __l, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q (unsigned int __n, _Tp __x)
- float __gnu_cxx::legendre_qf (unsigned int __n, float __x)
- long double __gnu_cxx::legendre_ql (unsigned int __n, long double __x)
- float std::legendref (unsigned int __l, float __x)
- long double std::legendrel (unsigned int __l, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)
- float __gnu_cxx::lfactorialf (unsigned int __n)
- long double __gnu_cxx::lfactoriall (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint (_Tp __x)
- float __gnu_cxx::logintf (float __x)
- long double __gnu_cxx::logintl (long double __x)
- template<typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l (_Tp __a, _Tn __n)
- float __gnu_cxx::lpochhammer_lf (float __a, float __n)

- long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n)
- template< typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_u (_Tp __a, _Tn __n)
- float __gnu_cxx::lpochhammer_uf (float __a, float __n)
- long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n)
- template< typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_num_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)
- float __gnu_cxx::owens_tf (float __h, float __a)
- long double __gnu_cxx::owens_tl (long double __h, long double __a)
- template< typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::pochhammer_l (_Tp __a, _Tn __n)
- float __gnu_cxx::pochhammer_lf (float __a, float __n)
- long double __gnu_cxx::pochhammer_ll (long double __a, long double __n)
- template< typename _Tp , typename _Tn >
  __gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::pochhammer_u (_Tp __a, _Tn __n)
- float __gnu_cxx::pochhammer_uf (float __a, float __n)
- long double __gnu_cxx::pochhammer_ul (long double __a, long double __n)
- template< typename _Tp , typename _Wp >
  __gnu_cxx::__promote_num_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)
- template< typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > __gnu_cxx::polylog (_Tp __s, std::complex< _Tp > __w)
- float __gnu_cxx::polylogf (float __s, float __w)
- std::complex< float > __gnu_cxx::polylogf (float __s, std::complex< float > __w)
- long double __gnu_cxx::polylogl (long double __s, long double __w)
- std::complex< long double > __gnu_cxx::polylogl (long double __s, std::complex< long double > __w)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::psi (_Tp __x)
- float __gnu_cxx::psif (float __x)
- long double __gnu_cxx::psil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::radpoly (unsigned int __n, unsigned int __m, _Tp __rho)
- float __gnu_cxx::radpolyf (unsigned int __n, unsigned int __m, float __rho)
- long double __gnu_cxx::radpolyl (unsigned int __n, unsigned int __m, long double __rho)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::riemann_zeta (_Tp __s)
- float std::riemann_zetaf (float __s)
- long double std::riemann_zetal (long double __s)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc_pi (_Tp __x)
- float __gnu_cxx::sinc_pif (float __x)
- long double __gnu_cxx::sinc_pil (long double __x)
- float __gnu_cxx::sincf (float __x)
- long double __gnu_cxx::sincl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc_pi (_Tp __x)
- float __gnu_cxx::sinhc_pif (float __x)
- long double __gnu_cxx::sinhc_pil (long double __x)

- float __gnu_cxx::sinhcf (float __x)
- long double __gnu_cxx::sinhcl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhint (_Tp __x)
- float __gnu_cxx::sinhintf (float __x)
- long double __gnu_cxx::sinhintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinint (_Tp __x)
- float __gnu_cxx::sinintf (float __x)
- long double __gnu_cxx::sinintl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_bessel (unsigned int __n, _Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)
- float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)
- long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)
- float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)
- long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x)
- float std::sph_besself (unsigned int __n, float __x)
- long double std::sph_bessell (unsigned int __n, long double __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)
- std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)
- std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)
- std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)
- std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)
- std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)
- template< typename _Ttheta , typename _Tphi >
  std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)
- std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)
- float std::sph_legendref (unsigned int __l, unsigned int __m, float __theta)
- long double std::sph_legendrel (unsigned int __l, unsigned int __m, long double __theta)
- template< typename _Tp >
  __gnu_cxx::__promote< _Tp >::__type std::sph_neumann (unsigned int __n, _Tp __x)

- float std::sph_neumannf (unsigned int __n, float __x)
- long double std::sph_neumannl (unsigned int __n, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_1f (float __nu, float __x)
- long double __gnu_cxx::theta_1l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_2f (float __nu, float __x)
- long double __gnu_cxx::theta_2l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_3f (float __nu, float __x)
- long double __gnu_cxx::theta_3l (long double __nu, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)
- float __gnu_cxx::theta_4f (float __nu, float __x)
- long double __gnu_cxx::theta_4l (long double __nu, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_c (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_cf (float __k, float __x)
- long double __gnu_cxx::theta_cl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_d (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_df (float __k, float __x)
- long double __gnu_cxx::theta_dl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_n (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_nf (float __k, float __x)
- long double __gnu_cxx::theta_nl (long double __k, long double __x)
- template<typename _Tpk , typename _Tp >
  __gnu_cxx::__promote_num_t< _Tpk, _Tp > __gnu_cxx::theta_s (_Tpk __k, _Tp __x)
- float __gnu_cxx::theta_sf (float __k, float __x)
- long double __gnu_cxx::theta_sl (long double __k, long double __x)
- template<typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_num_t< _Trho, _Tphi > __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)
- long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)

10.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

10.27.2 Macro Definition Documentation

10.27.2.1 #define __cpp_lib_math_special_functions 201603L

Definition at line 39 of file specfun.h.

10.27.2.2 #define __STDCPP_MATH_SPEC_FUNCS__ 201003L

Definition at line 37 of file specfun.h.

Index