C++ Special Math Functions

2.0

Generated by Doxygen 1.8.13

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`.

Follow-up proosals for new special functions have also been published: `A proposal to add special mathematical functions according to the ISO/IEC 80000-2:2009 standard, Vincent Reverdy`.

`A Proposal to add Mathematical Functions for Statistics to the C++ Standard Library, Paul A Bristow`.

`A proposal to add sincos to the standard library, Paul Dreik`.

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
- bernoulli - Bernoulli polynomials
- binomial - Binomial coefficients
- bose_einstein - Bose-Einstein integrals
- 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_cl - Clausen cosine integrals
- clausen_sl - Clausen sine integrals
- comp_ellint_d - Incomplete Legendre D elliptic integral
- conf_hyperg_lim - Confluent hypergeometric limit functions
- cos_pi - Reperiodized cosine function.
- cosh_pi - Reperiodized hyperbolic cosine function.
- 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
- debye - Debye functions
- dilog - Dilogarithm functions
- dirichlet_beta - Dirichlet beta function
- dirichlet_eta - Dirichlet beta function
- dirichlet_lambda - Dirichlet lambda function
- double_factorial - Double factorials
- ellint_d - Legendre D elliptic integrals
- ellint_rc - Carlson elliptic functions R_C
- ellint_rd - Carlson elliptic functions R_D
- ellint_rf - Carlson elliptic functions R_F
- ellint_rg - Carlson elliptic functions R_G
- ellint_rj - Carlson elliptic functions R_J
- ellnome - Elliptic nome
- euler - Euler numbers
- euler - Euler polynomials
- eulerian_1 - Eulerian numbers of the first kind
- eulerian_2 - Eulerian numbers of the second kind
- expint - Exponential integrals
- factorial - Factorials
- falling_factorial - Falling factorials
- fermi_dirac - Fermi-Dirac integrals
- fresnel_c - Fresnel cosine integrals

- fresnel_s - Fresnel sine integrals
- gamma_reciprocal - Reciprocal gamma function
- gegenbauer - Gegenbauer polynomials
- heuman_lambda - Heuman lambda functions
- hurwitz_zeta - Hurwitz zeta functions
- ibeta - Regularized incomplete beta functions
- jacobi - Jacobi polynomials
- jacobi_sn - Jacobi sine amplitude functions
- jacobi_cn - Jacobi cosine amplitude functions
- jacobi_dn - Jacobi delta amplitude functions
- jacobi_zeta - Jacobi zeta functions
- lbinomial - Log binomial coefficients
- ldouble_factorial - Log double factorials
- legendre_q - Legendre functions of the second kind
- lfactorial - Log factorials
- lfalling_factorial - Log falling factorials
- lgamma - Log gamma for complex arguments
- lrising_factorial - Log rising factorials
- owens_t - Owens T functions
- pgamma - Regularized lower incomplete gamma functions
- psi - Psi or digamma function
- qgamma - Regularized upper incomplete gamma functions
- radpoly - Radial polynomials
- rising_factorial - Rising factorials
- sinhc - Hyperbolic sinus cardinal function
- sinhc_pi - Reperiodized hyperbolic sinus cardinal function
- sinc - Normalized sinus cardinal function
- sincos - Sine + cosine function
- sincos_pi - Reperiodized sine + cosine function
- sin_pi - Reperiodized sine function.
- sinh_pi - Reperiodized hyperbolic sine function.
- sinc_pi - Sinus cardinal function
- 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
- stirling_1 - Stirling numbers of the first kind
- stirling_2 - Stirling numbers of the second kind
- tan_pi - Reperiodized tangent function.
- tanh_pi - Reperiodized hyperbolic tangent function.
- tgamma - Gamma for complex arguments
- tgamma - Upper incomplete gamma functions
- tgamma_lower - Lower incomplete gamma functions
- theta_1 - Exponential theta function 1
- theta_2 - Exponential theta function 2
- theta_3 - Exponential theta function 3
- theta_4 - Exponential theta function 4
- tricomi_u - Tricomi confluent hypergeometric function
- 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

Member __gnu_cxx::eulerian_1 (unsigned int __n, unsigned int __m)

Develop an iterator model for Eulerian numbers of the first kind.

Member __gnu_cxx::eulerian_2 (unsigned int __n, unsigned int __m)

Develop an iterator model for Eulerian numbers of the second kind.

Member __gnu_cxx::stirling_1 (unsigned int __n, unsigned int __m)

Develop an iterator model for Stirling numbers of the first kind.

Member __gnu_cxx::stirling_2 (unsigned int __n, unsigned int __m)

Develop an iterator model for Stirling numbers of the second kind.

page Mathematical Special Functions

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

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

: We should return both the Debye function and it's complement.

Member std::__detail::__euler_series (unsigned int __n)

Find a way to predict the maximum Euler number for a type.

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

Study arbitrary switch to large-n $E_n(x)$.

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

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

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_polylog (_Tp __s, std::complex$<$ _Tp $>$ __a)

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

Member std::__detail::__log_stirling_2 (unsigned int __n, unsigned int __m)

Look into asymptotic solutions.

Member std::__detail::__riemann_zeta (_Tp __s)

Global double sum or MacLaurin series in riemann_zeta?

Member std::__detail::__stirling_1 (unsigned int __n, unsigned int __m)

Find asymptotic solutions for the Stirling numbers of the first kind.

Develop an iterator model for Stirling numbers of the first kind.

Member std::__detail::__stirling_2 (unsigned int __n, unsigned int __m)

Find asymptotic solutions for Stirling numbers of the second kind.

Develop an iterator model for Stirling numbers of the second kind.

Member std::__detail::__stirling_2_series (unsigned int __n, unsigned int __m)

Find a way to predict the maximum Stirling number for a type.

Member std::__detail::_Airy_asymp< _Tp >::_S_absarg_lt_pio3 (_Cmplx __z) const

Revisit these numbers of terms for the Airy asymptotic expansions.

Member std::__detail::_Airy_series< _Tp >::_S_Scorer (_Cmplx __t)

Find out what is wrong with the Hi = fai + gai + hai scorer function.

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

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 6

Class Index

6.1 Class List

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

Chapter 7

File Index

7.1 File List

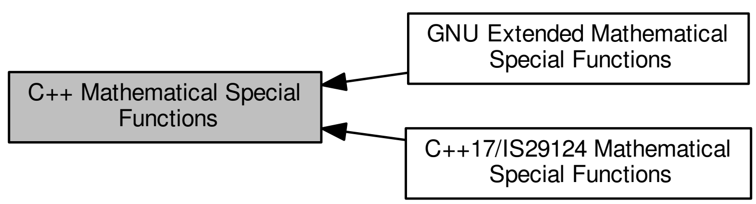
Here is a list of all files with brief descriptions:

Chapter 8

Module Documentation

8.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

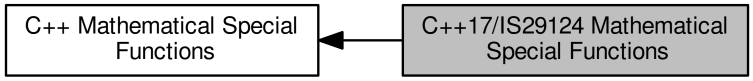
- C++17/IS29124 Mathematical Special Functions
- GNU Extended Mathematical Special Functions

8.1.1 Detailed Description

A collection of advanced mathematical special functions.

8.2 C++17/IS29124 Mathematical Special Functions

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



Functions

- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tpa, _Tpb > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp, _Tpn > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpn, _Tpp > 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_fp_t< _Tp > std::expint (_Tp __x)
- float std::expintf (float __x)
- long double std::expintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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)

8.2.1 Detailed Description

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

8.2.2 Function Documentation

8.2.2.1 assoc_laguerre()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 415 of file specfun.h.

8.2.2.2 assoc_laguerref()

```
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 367 of file specfun.h.

8.2.2.3 assoc_laguerrel()

```
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 378 of file specfun.h.

8.2.2.4 assoc_legendre()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 463 of file specfun.h.

8.2.2.5 assoc_legendref()

```
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 430 of file specfun.h.

8.2.2.6 assoc_legendrel()

```
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 441 of file specfun.h.

8.2.2.7 beta()

```
template<typename _Tpa , typename _Tpb >
__gnu_cxx::__promote_fp_t<_Tpa, _Tpb> 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 508 of file specfun.h.

8.2.2.8 betaf()

```
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 477 of file specfun.h.

8.2.2.9 betal()

```
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 487 of file specfun.h.

8.2.2.10 comp_ellint_1()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 556 of file specfun.h.

8.2.2.11 comp_ellint_1f()

```
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 523 of file specfun.h.

8.2.2.12 comp_ellint_1l()

```
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 533 of file specfun.h.

8.2.2.13 comp_ellint_2()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 603 of file specfun.h.

8.2.2.14 comp_ellint_2f()

```
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 571 of file specfun.h.

8.2.2.15 comp_ellint_2l()

```
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 581 of file specfun.h.

8.2.2.16 comp_ellint_3()

```
template<typename _Tp , typename _Tpn >
__gnu_cxx::__promote_fp_t<_Tp, _Tpn> 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 654 of file specfun.h.

8.2.2.17 comp_ellint_3f()

```
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 618 of file specfun.h.

8.2.2.18 comp_ellint_3l()

```
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 628 of file specfun.h.

8.2.2.19 cyl_bessel_i()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> 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 700 of file specfun.h.

8.2.2.20 cyl_bessel_if()

```
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 669 of file specfun.h.

8.2.2.21 cyl_bessel_il()

```
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 679 of file specfun.h.

8.2.2.22 cyl_bessel_j()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> 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 746 of file specfun.h.

8.2.2.23 cyl_bessel_jf()

```
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 715 of file specfun.h.

8.2.2.24 cyl_bessel_jl()

```
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 725 of file specfun.h.

8.2.2.25 cyl_bessel_k()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> 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 798 of file specfun.h.

8.2.2.26 cyl_bessel_kf()

```
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 761 of file specfun.h.

8.2.2.27 cyl_bessel_kl()

```
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 771 of file specfun.h.

8.2.2.28 cyl_neumann()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> 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 846 of file specfun.h.

8.2.2.29 cyl_neumannf()

```
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 813 of file specfun.h.

8.2.2.30 cyl_neumannl()

```
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 823 of file specfun.h.

8.2.2.31 ellint_1()

```
template<typename _Tp , typename _Tpp >
__gnu_cxx::__promote_fp_t<_Tp, _Tpp> 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 894 of file specfun.h.

8.2.2.32 ellint_1f()

```
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 861 of file specfun.h.

8.2.2.33 ellint_1l()

```
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 871 of file specfun.h.

8.2.2.34 ellint_2()

```
template<typename _Tp , typename _Tpp >
__gnu_cxx::__promote_fp_t<_Tp, _Tpp> 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 942 of file specfun.h.

8.2.2.35 ellint_2f()

```
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 909 of file specfun.h.

8.2.2.36 ellint_2l()

```
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 919 of file specfun.h.

8.2.2.37 ellint_3()

```
template<typename _Tp , typename _Tpn , typename _Tpp >
__gnu_cxx::__promote_fp_t<_Tp, _Tpn, _Tpp> 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 995 of file specfun.h.

8.2.2.38 ellint_3f()

```
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 957 of file specfun.h.

8.2.2.39 ellint_3l()

```
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 967 of file specfun.h.

8.2.2.40 expint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1035 of file specfun.h.

8.2.2.41 expintf()

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

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

See also

expint for details.

Definition at line 1009 of file specfun.h.

8.2.2.42 expintl()

```
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 1019 of file specfun.h.

8.2.2.43 hermite()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1083 of file specfun.h.

8.2.2.44 hermitef()

```
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 1050 of file specfun.h.

8.2.2.45 hermitel()

```
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 1060 of file specfun.h.

8.2.2.46 laguerre()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1127 of file specfun.h.

8.2.2.47 laguerref()

```
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 1098 of file specfun.h.

8.2.2.48 laguerrel()

```
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 1108 of file specfun.h.

8.2.2.49 legendre()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1172 of file specfun.h.

8.2.2.50 legendref()

```
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 1142 of file specfun.h.

8.2.2.51 legendrel()

```
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 1152 of file specfun.h.

8.2.2.52 riemann_zeta()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1223 of file specfun.h.

8.2.2.53 riemann_zetaf()

```
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 1187 of file specfun.h.

8.2.2.54 riemann_zetal()

```
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 1197 of file specfun.h.

8.2.2.55 sph_bessel()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1267 of file specfun.h.

8.2.2.56 sph_besself()

```
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 1238 of file specfun.h.

8.2.2.57 sph_bessell()

```
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 1248 of file specfun.h.

8.2.2.58 sph_legendre()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1314 of file specfun.h.

8.2.2.59 sph_legendref()

```
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 1282 of file specfun.h.

8.2.2.60 sph_legendrel()

```
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 1293 of file specfun.h.

8.2.2.61 sph_neumann()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> 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 1358 of file specfun.h.

8.2.2.62 sph_neumannf()

```
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 1329 of file specfun.h.

8.2.2.63 sph_neumannl()

```
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` $x >= 0$.

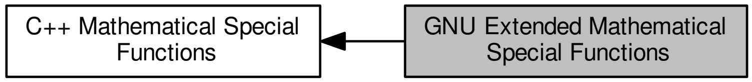
See also

sph_neumann for details.

Definition at line 1339 of file specfun.h.

8.3 GNU Extended Mathematical Special Functions

Collaboration diagram for GNU Extended Mathematical Special Functions:



Functions

- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::airy_ai (std::complex< _Tp > __x)
- float __gnu_cxx::airy_aif (float __x)
- long double __gnu_cxx::airy_ail (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::airy_bi (std::complex< _Tp > __x)
- float __gnu_cxx::airy_bif (float __x)
- long double __gnu_cxx::airy_bil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::bernoulli (unsigned int __n)
- template< typename _Tp >
  _Tp __gnu_cxx::bernoulli (unsigned int __n, _Tp __x)
- float __gnu_cxx::bernoullif (unsigned int __n)
- long double __gnu_cxx::bernoullil (unsigned int __n)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial (unsigned int __n, unsigned int __k)

  Return the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

  $$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial probability mass function.

- float __gnu_cxx::binomialf (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::binomiall (unsigned int __n, unsigned int __k)
- template<typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::bose_einstein (_Tps __s, _Tp __x)
- float __gnu_cxx::bose_einsteinf (float __s, float __x)
- long double __gnu_cxx::bose_einsteinl (long double __s, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_cl (unsigned int __m, _Tp __x)
- float __gnu_cxx::clausen_clf (unsigned int __m, float __x)
- long double __gnu_cxx::clausen_cll (unsigned int __m, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_sl (unsigned int __m, _Tp __x)
- float __gnu_cxx::clausen_slf (unsigned int __m, float __x)
- long double __gnu_cxx::clausen_sll (unsigned int __m, long double __x)
- float __gnu_cxx::clausenf (unsigned int __m, float __x)
- std::complex< float > __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __z)
- long double __gnu_cxx::clausenl (unsigned int __m, long double __x)
- std::complex< long double > __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __z)
- template<typename _Tk >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)

- template< typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __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_fp_t< _Tp > __gnu_cxx::cos_pi (_Tp __x)
- float __gnu_cxx::cos_pif (float __x)
- long double __gnu_cxx::cos_pil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::cosh_pi (_Tp __x)
- float __gnu_cxx::cosh_pif (float __x)
- long double __gnu_cxx::cosh_pil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tp > __gnu_cxx::debye (unsigned int __n, _Tp __x)
- float __gnu_cxx::debyef (unsigned int __n, float __x)
- long double __gnu_cxx::debyel (unsigned int __n, long double __x)

- template<typename _Tp >
  __gnu_cxx::__promote_fp_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 >
  _Tp __gnu_cxx::dirichlet_lambda (_Tp __s)
- float __gnu_cxx::dirichlet_lambdaf (float __s)
- long double __gnu_cxx::dirichlet_lambdal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::double_factorial (int __n)

  Return the double factorial $n!!$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

  for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

  for odd 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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 >
  _Tp __gnu_cxx::euler (unsigned int __n)

  This returns Euler number E_n.
- template<typename _Tp >
  _Tp __gnu_cxx::eulerian_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __gnu_cxx::eulerian_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::expint (unsigned int __n, _Tp __x)
- float __gnu_cxx::expintf (unsigned int __n, float __x)
- long double __gnu_cxx::expintl (unsigned int __n, long double __x)
- template<typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_cdf (_Tlam __lambda, _Tp __x)

  Return the exponential cumulative probability density function.
- template<typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_pdf (_Tlam __lambda, _Tp __x)

  Return the exponential probability density function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::factorial (unsigned int __n)

  Return the factorial $n!$ of the argument as a real number.

  $$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float __gnu_cxx::factorialf (unsigned int __n)
- long double __gnu_cxx::factoriall (unsigned int __n)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::falling_factorial (_Tp __a, _Tnu __nu)

Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

In particular, $n^{\underline{n}} = n!$.

- float __gnu_cxx::falling_factorialf (float __a, float __nu)
- long double __gnu_cxx::falling_factoriall (long double __a, long double __nu)
- template<typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::fermi_dirac (_Tps __s, _Tp __x)
- float __gnu_cxx::fermi_diracf (float __s, float __x)
- long double __gnu_cxx::fermi_diracl (long double __s, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::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 >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::fisher_f_pdf (_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 >
  __gnu_cxx::__promote_fp_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_fp_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 _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::gamma_cdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma cumulative propability distribution function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::gamma_pdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma propability distribution function.
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::gamma_reciprocal (_Ta __a)
- float __gnu_cxx::gamma_reciprocalf (float __a)
- long double __gnu_cxx::gamma_reciprocall (long double __a)
- template<typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Tp > __gnu_cxx::harmonic (unsigned int __n)
- template<typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tpa, _Tpb, _Tpc, _Tp > __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_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _←
  Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > __gnu_cxx::lbinomial (unsigned int __n, unsigned int __k)

  Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

  $$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- float __gnu_cxx::lbinomialf (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::lbinomiall (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)

Return the logarithm of the double factorial $ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

for odd n.

- float __gnu_cxx::ldouble_factorialf (int __n)
- long double __gnu_cxx::ldouble_factoriall (int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::legendre_q (unsigned int __l, _Tp __x)
- float __gnu_cxx::legendre_qf (unsigned int __l, float __x)
- long double __gnu_cxx::legendre_ql (unsigned int __l, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)

  Return the logarithm of the factorial $ln(n!)$ of the argument as a real number.

$$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float __gnu_cxx::lfactorialf (unsigned int __n)
- long double __gnu_cxx::lfactoriall (unsigned int __n)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lfalling_factorial (_Tp __a, _Tnu __nu)

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

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

  In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

  Many notations exist for this function: $(a)_\nu$,

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

  , and others.
- float __gnu_cxx::lfalling_factorialf (float __a, float __nu)
- long double __gnu_cxx::lfalling_factoriall (long double __a, long double __nu)
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::lgamma (_Ta __a)
- template<typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::lgamma (std::complex< _Ta > __a)
- float __gnu_cxx::lgammaf (float __a)
- std::complex< float > __gnu_cxx::lgammaf (std::complex< float > __a)
- long double __gnu_cxx::lgammal (long double __a)
- std::complex< long double > __gnu_cxx::lgammal (std::complex< long double > __a)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::logint (_Tp __x)
- float __gnu_cxx::logintf (float __x)
- long double __gnu_cxx::logintl (long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic cumulative distribution function.

- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic probability density function.
- template< typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal cumulative probability density function.
- template< typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal probability density function.
- template< typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lrising_factorial (_Tp __a, _Tnu __nu)

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

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

  Thus this function returns

  $$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[a^{\overline{0}}] = 0$$

  Many notations exist for this function: $(a)_\nu$ (especially in the literature of special functions),

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

  , and others.
- float __gnu_cxx::lrising_factorialf (float __a, float __nu)
- long double __gnu_cxx::lrising_factoriall (long double __a, long double __nu)
- template< typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal cumulative probability density function.
- template< typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal probability density function.
- template< typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_fp_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 _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::pgamma (_Ta __a, _Tp __x)
- float __gnu_cxx::pgammaf (float __a, float __x)
- long double __gnu_cxx::pgammal (long double __a, long double __x)
- template< typename _Tp , typename _Wp >
  __gnu_cxx::__promote_fp_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)
- template< typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > __gnu_cxx::psi (_Tp __x)
- float __gnu_cxx::psif (float __x)
- long double __gnu_cxx::psil (long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::qgamma (_Ta __a, _Tp __x)
- float __gnu_cxx::qgammaf (float __a, float __x)
- long double __gnu_cxx::qgammal (long double __a, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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 , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::rising_factorial (_Tp __a, _Tnu __nu)

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

$$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

  Many notations exist for this function: $(a)_\nu$, (especially in the literature of special functions),

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

  , and others.
- float __gnu_cxx::rising_factorialf (float __a, float __nu)
- long double __gnu_cxx::rising_factoriall (long double __a, long double __nu)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sin_pi (_Tp __x)
- float __gnu_cxx::sin_pif (float __x)
- long double __gnu_cxx::sin_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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)
- __gnu_cxx::__sincos_t< double > __gnu_cxx::sincos (double __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos_pi (_Tp __x)
- __gnu_cxx::__sincos_t< float > __gnu_cxx::sincos_pif (float __x)
- __gnu_cxx::__sincos_t< long double > __gnu_cxx::sincos_pil (long double __x)
- __gnu_cxx::__sincos_t< float > __gnu_cxx::sincosf (float __x)
- __gnu_cxx::__sincos_t< long double > __gnu_cxx::sincosl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinh_pi (_Tp __x)
- float __gnu_cxx::sinh_pif (float __x)
- long double __gnu_cxx::sinh_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinhc (_Tp __x)

- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_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 >
  _Tp __gnu_cxx::stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __gnu_cxx::stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_cdf (_Tt __t, unsigned int __nu)

Return the Students T probability function.

- template< typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_pdf (_Tt __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tan_pi (_Tp __x)
- float __gnu_cxx::tan_pif (float __x)
- long double __gnu_cxx::tan_pil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tanh_pi (_Tp __x)
- float __gnu_cxx::tanh_pif (float __x)
- long double __gnu_cxx::tanh_pil (long double __x)
- template< typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::tgamma (_Ta __a)
- template< typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma (std::complex< _Ta > __a)
- template< typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma (_Ta __a, _Tp __x)
- template< typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower (_Ta __a, _Tp __x)
- float __gnu_cxx::tgamma_lowerf (float __a, float __x)
- long double __gnu_cxx::tgamma_lowerl (long double __a, long double __x)
- float __gnu_cxx::tgammaf (float __a)
- std::complex< float > __gnu_cxx::tgammaf (std::complex< float > __a)
- float __gnu_cxx::tgammaf (float __a, float __x)
- long double __gnu_cxx::tgammal (long double __a)
- std::complex< long double > __gnu_cxx::tgammal (std::complex< long double > __a)
- long double __gnu_cxx::tgammal (long double __a, long double __x)
- template< typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __gnu_cxx::tricomi_u (_Tpa __a, _Tpc __c, _Tp __x)
- float __gnu_cxx::tricomi_uf (float __a, float __c, float __x)
- long double __gnu_cxx::tricomi_ul (long double __a, long double __c, long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull cumulative probability density function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull probability density function.
- template<typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_fp_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)

8.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

8.3.2 Function Documentation

8.3.2.1 airy_ai() [1/2]

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2808 of file specfun.h.

8.3.2.2 airy_ai() [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_t<_Tp> > __gnu_cxx::airy_ai (
            std::complex< _Tp > __x ) [inline]
```

Return the Airy function $Ai(x)$ of complex 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 complex argument

Definition at line 2828 of file specfun.h.

8.3.2.3 airy_aif()

```
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 2781 of file specfun.h.

8.3.2.4 airy_ail()

```
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 2791 of file specfun.h.

8.3.2.5 airy_bi() [1/2]

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2870 of file specfun.h.

8.3.2.6 airy_bi() [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_t<_Tp> > __gnu_cxx::airy_bi (
            std::complex< _Tp > __x ) [inline]
```

Return the Airy function $Bi(x)$ of complex 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 complex argument

Definition at line 2891 of file specfun.h.

8.3.2.7 airy_bif()

```
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 2842 of file specfun.h.

8.3.2.8 airy_bil()

```
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 2852 of file specfun.h.

8.3.2.9 bernoulli() [1/2]

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

Return the Bernoulli number of integer order n.

The Bernoulli numbers are defined by

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n), B_1 = -1/2$$

All odd Bernoulli numbers except B_1 are zero.

Parameters

__n	The order.

Definition at line 4264 of file specfun.h.

8.3.2.10 bernoulli() [2/2]

```
template<typename _Tp >
_Tp __gnu_cxx::bernoulli (
            unsigned int __n,
            _Tp __x ) [inline]
```

Return the Bernoulli polynomial $B_n(x)$ of order n at argument x.

The values at 0 and 1 are equal to the corresponding Bernoulli number:

$$B_n(0) = B_n(1) = B_n$$

The derivative is proportional to the previous polynomial:

$$B_n'(x) = n * B_{n-1}(x)$$

The series expansion for the Bernoulli polynomials is:

$$B_n(x) = \sum_{k=0}^{n} B_k \binom{n}{k} x^{n-k}$$

A useful argument promotion is:

$$B_n(x+1) - B_n(x) = n * x^{n-1}$$

Definition at line 6639 of file specfun.h.

References std::__detail::__bernoulli().

8.3.2.11 bernoullif()

```
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 4237 of file specfun.h.

8.3.2.12 bernoullil()

```
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 4247 of file specfun.h.

8.3.2.13 binomial()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::binomial (
            unsigned int __n,
            unsigned int __k ) [inline]
```

Return the binomial coefficient as a real number. The binomial coefficient is given by:

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

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

.

Parameters

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

Returns

The binomial coefficient.

Definition at line 4180 of file specfun.h.

8.3.2.14 binomial_cdf()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::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:

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

Parameters

__p	
__n	
__k	

Definition at line 6492 of file specfun.h.

8.3.2.15 binomial_pdf()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::binomial_pdf (
            _Tp __p,
            unsigned int __n,
            unsigned int __k )
```

Return the binomial probability mass function.

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

$$f(k|n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Parameters

__p	
__n	
__k	

Definition at line 6471 of file specfun.h.

8.3.2.16 binomialf()

```
float __gnu_cxx::binomialf (
            unsigned int __n,
            unsigned int __k ) [inline]
```

Return the binomial coefficient as a `float`.

See also

binomial for details.

Definition at line 4151 of file specfun.h.

8.3.2.17 binomiall()

```
long double __gnu_cxx::binomiall (
            unsigned int __n,
            unsigned int __k )  [inline]
```

Return the binomial coefficient as a `long double`.

See also

binomial for details.

Definition at line 4160 of file specfun.h.

8.3.2.18 bose_einstein()

```
template<typename _Tps , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tps, _Tp> __gnu_cxx::bose_einstein (
            _Tps __s,
            _Tp __x )  [inline]
```

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

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12.16

$$G_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x} - 1} dt = Li_{s+1}(e^x)$$

Parameters

__s	The order s >= 0.
__x	The real argument.

Returns

The real Bose-Einstein integral G_s(x),

Definition at line 5870 of file specfun.h.

8.3.2.19 bose_einsteinf()

```
float __gnu_cxx::bose_einsteinf (
            float __s,
            float __x ) [inline]
```

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

See also

bose_einstein for details.

Definition at line 5840 of file specfun.h.

8.3.2.20 bose_einsteinl()

```
long double __gnu_cxx::bose_einsteinl (
            long double __s,
            long double __x ) [inline]
```

Return the Bose-Einstein integral of `long double` order s and argument x.

See also

bose_einstein for details.

Definition at line 5850 of file specfun.h.

8.3.2.21 chebyshev_t()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2044 of file specfun.h.

8.3.2.22 chebyshev_tf()

```
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 2015 of file specfun.h.

8.3.2.23 chebyshev_tl()

```
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 2025 of file specfun.h.

8.3.2.24 chebyshev_u()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2088 of file specfun.h.

8.3.2.25 chebyshev_uf()

```
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 2059 of file specfun.h.

8.3.2.26 chebyshev_ul()

```
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 2069 of file specfun.h.

8.3.2.27 chebyshev_v()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2133 of file specfun.h.

8.3.2.28 chebyshev_vf()

```
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 2103 of file specfun.h.

8.3.2.29 chebyshev_vl()

```
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 2113 of file specfun.h.

8.3.2.30 chebyshev_w()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2178 of file specfun.h.

8.3.2.31 chebyshev_wf()

```
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 2148 of file specfun.h.

8.3.2.32 chebyshev_wl()

```
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 2158 of file specfun.h.

8.3.2.33 clausen() [1/2]

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::clausen (
            unsigned int __m,
            _Tp __x ) [inline]
```

Return the Clausen function $C_m(x)$ of integer order m and real argument x.

The Clausen function is defined by

$$C_m(x) = Sl_m(x) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^m} \text{ for even } m = Cl_m(x) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^m} \text{ for odd } m$$

Template Parameters

_Tp	The real type of the argument

Parameters

__m	The integral order
__x	The real argument

Definition at line 5293 of file specfun.h.

8.3.2.34 clausen() [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_t<_Tp> > __gnu_cxx::clausen (
            unsigned int __m,
            std::complex< _Tp > __z ) [inline]
```

Return the Clausen function $C_m(z)$ of integer order m and complex argument z.

The Clausen function is defined by

$$C_m(z) = Sl_m(z) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^m} \text{ for even } m = Cl_m(z) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^m} \text{ for odd } m$$

Template Parameters

_Tp	The real type of the complex components

Parameters

__m	The integral order
__z	The complex argument

Definition at line 5337 of file specfun.h.

8.3.2.35 clausen_cl()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::clausen_cl (
            unsigned int __m,
            _Tp __x ) [inline]
```

Return the Clausen cosine function $Cl_m(x)$ of order m and real argument x.

The Clausen cosine function is defined by

$$Cl_m(x) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^m}$$

Template Parameters

_Tp	The real type of the argument

Parameters

__m	The unsigned integer order
__x	The real argument

Definition at line 5248 of file specfun.h.

8.3.2.36 clausen_clf()

```
float __gnu_cxx::clausen_clf (
            unsigned int __m,
            float __x ) [inline]
```

Return the Clausen cosine function $Cl_m(x)$ of order m and `float` argument x.

See also

clausen_cl for details.

Definition at line 5220 of file specfun.h.

8.3.2.37 clausen_cll()

```
long double __gnu_cxx::clausen_cll (
            unsigned int __m,
            long double __x )  [inline]
```

Return the Clausen cosine function $Cl_m(x)$ of order m and `long double` argument x.

See also

clausen_cl for details.

Definition at line 5230 of file specfun.h.

8.3.2.38 clausen_sl()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::clausen_sl (
            unsigned int __m,
            _Tp __x )  [inline]
```

Return the Clausen sine function $Sl_m(x)$ of order m and real argument x.

The Clausen sine function is defined by

$$Sl_m(x) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^m}$$

Template Parameters

_Tp	The real type of the argument

Parameters

__m	The unsigned integer order
__x	The real argument

Definition at line 5205 of file specfun.h.

8.3.2.39 clausen_slf()

```
float __gnu_cxx::clausen_slf (
            unsigned int __m,
            float __x ) [inline]
```

Return the Clausen sine function $Sl_m(x)$ of order m and `float` argument x.

See also

clausen_sl for details.

Definition at line 5177 of file specfun.h.

8.3.2.40 clausen_sll()

```
long double __gnu_cxx::clausen_sll (
            unsigned int __m,
            long double __x ) [inline]
```

Return the Clausen sine function $Sl_m(x)$ of order m and `long double` argument x.

See also

clausen_sl for details.

Definition at line 5187 of file specfun.h.

8.3.2.41 clausenf() [1/2]

```
float __gnu_cxx::clausenf (
            unsigned int __m,
            float __x ) [inline]
```

Return the Clausen function $C_m(x)$ of integer order m and `float` argument x.

See also

clausen for details.

Definition at line 5263 of file specfun.h.

8.3.2.42 clausenf() [2/2]

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

Return the Clausen function $C_m(z)$ of integer order m and `std::complex<float>` argument z.

See also

clausen for details.

Definition at line 5308 of file specfun.h.

8.3.2.43 clausenl() [1/2]

```
long double __gnu_cxx::clausenl (
            unsigned int __m,
            long double __x ) [inline]
```

Return the Clausen function $C_m(x)$ of integer order m and `long double` argument x.

See also

clausen for details.

Definition at line 5273 of file specfun.h.

8.3.2.44 clausenl() [2/2]

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

Return the Clausen function $C_m(z)$ of integer order m and `std::complex<long double>` argument z.

See also

clausen for details.

Definition at line 5318 of file specfun.h.

8.3.2.45 comp_ellint_d()

```
template<typename _Tk >
__gnu_cxx::__promote_fp_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 4466 of file specfun.h.

8.3.2.46 comp_ellint_df()

```
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 4439 of file specfun.h.

8.3.2.47 comp_ellint_dl()

```
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 4449 of file specfun.h.

8.3.2.48 comp_ellint_rf() [1/3]

```
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 3151 of file specfun.h.

8.3.2.49 comp_ellint_rf() [2/3]

```
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 3161 of file specfun.h.

8.3.2.50 comp_ellint_rf() [3/3]

```
template<typename _Tx , typename _Ty >
__gnu_cxx::__promote_fp_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 3179 of file specfun.h.

8.3.2.51 comp_ellint_rg() [1/3]

```
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 3384 of file specfun.h.

8.3.2.52 comp_ellint_rg() [2/3]

```
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 3393 of file specfun.h.

8.3.2.53 comp_ellint_rg() [3/3]

```
template<typename _Tx , typename _Ty >
__gnu_cxx::__promote_fp_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:

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

Parameters

__x	The first argument.
__y	The second argument.

Definition at line 3412 of file specfun.h.

8.3.2.54 conf_hyperg()

```
template<typename _Tpa , typename _Tpc , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpa, _Tpc, _Tp> __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 1423 of file specfun.h.

8.3.2.55 conf_hyperg_lim()

```
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 1568 of file specfun.h.

8.3.2.56 conf_hyperg_limf()

```
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 1539 of file specfun.h.

8.3.2.57 conf_hyperg_liml()

```
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 1549 of file specfun.h.

8.3.2.58 conf_hypergf()

```
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 1391 of file specfun.h.

8.3.2.59 conf_hypergl()

```
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 1402 of file specfun.h.

8.3.2.60 cos_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cos_pi (
            _Tp __x ) [inline]
```

Return the reperiodized cosine function $\cos_\pi(x)$ for real argument x.

The reperiodized cosine function is defined by:

$$\cos_\pi(x) = \cos(\pi x)$$

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__x	The argument

Definition at line 5996 of file specfun.h.

8.3.2.61 cos_pif()

```
float __gnu_cxx::cos_pif (
            float __x ) [inline]
```

Return the reperiodized cosine function $\cos_\pi(x)$ for `float` argument x.

See also

cos_pi for more details.

Definition at line 5969 of file specfun.h.

8.3.2.62 cos_pil()

```
long double __gnu_cxx::cos_pil (
            long double __x ) [inline]
```

Return the reperiodized cosine function $\cos_\pi(x)$ for `long double` argument x.

See also

cos_pi for more details.

Definition at line 5979 of file specfun.h.

8.3.2.63 cosh_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cosh_pi (
            _Tp __x ) [inline]
```

Return the reperiodized hyperbolic cosine function $\cosh_\pi(x)$ for real argument x.

The reperiodized hyperbolic cosine function is defined by:

$$\cosh_\pi(x) = \cosh(\pi x)$$

Template Parameters

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

Parameters

_↩ _x	The argument

Definition at line 6038 of file specfun.h.

8.3.2.64 cosh_pif()

```
float __gnu_cxx::cosh_pif (
            float __x ) [inline]
```

Return the reperiodized hyperbolic cosine function $\cosh_\pi(x)$ for `float` argument x.

See also

cosh_pi for more details.

Definition at line 6011 of file specfun.h.

8.3.2.65 cosh_pil()

```
long double __gnu_cxx::cosh_pil (
            long double __x ) [inline]
```

Return the reperiodized hyperbolic cosine function $\cosh_\pi(x)$ for `long double` argument x.

See also

cosh_pi for more details.

Definition at line 6021 of file specfun.h.

8.3.2.66 coshint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 1850 of file specfun.h.

8.3.2.67 coshintf()

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

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

See also

coshint for details.

Definition at line 1822 of file specfun.h.

8.3.2.68 coshintl()

```
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 1832 of file specfun.h.

8.3.2.69 cosint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 1767 of file specfun.h.

8.3.2.70 cosintf()

```
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 1741 of file specfun.h.

8.3.2.71 cosintl()

```
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 1751 of file specfun.h.

8.3.2.72 cyl_hankel_1() [1/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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 spherical Hankel function of the first kind is defined by:

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

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 2535 of file specfun.h.

8.3.2.73 cyl_hankel_1() [2/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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 4743 of file specfun.h.

8.3.2.74 cyl_hankel_1f() [1/2]

```
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 2503 of file specfun.h.

8.3.2.75 cyl_hankel_1f() [2/2]

```
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 4712 of file specfun.h.

8.3.2.76 cyl_hankel_1l() [1/2]

```
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 2514 of file specfun.h.

8.3.2.77 cyl_hankel_1l() [2/2]

```
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 4723 of file specfun.h.

8.3.2.78 cyl_hankel_2() [1/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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:

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

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 2583 of file specfun.h.

8.3.2.79 cyl_hankel_2() [2/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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 4790 of file specfun.h.

8.3.2.80 cyl_hankel_2f() [1/2]

```
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 2551 of file specfun.h.

8.3.2.81 cyl_hankel_2f() [2/2]

```
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 4759 of file specfun.h.

8.3.2.82 cyl_hankel_2l() [1/2]

```
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 2562 of file specfun.h.

8.3.2.83 cyl_hankel_2l() [2/2]

```
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 4770 of file specfun.h.

8.3.2.84 dawson()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 3754 of file specfun.h.

8.3.2.85 dawsonf()

```
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 3725 of file specfun.h.

8.3.2.86 dawsonl()

```
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 3735 of file specfun.h.

8.3.2.87 debye()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::debye (
            unsigned int __n,
            _Tp __x ) [inline]
```

Return the Debye function $D_n(x)$ of positive order n and real argument x.

The Debye function is defined by:

$$D_n(x) = \frac{n}{x^n} \int_0^x \frac{t^n}{e^t - 1} dt$$

Template Parameters

_Tp	The real type of the argument

Parameters

_↩ _n	The positive integral order
_↩ _x	The real argument $x >= 0$

Definition at line 6608 of file specfun.h.

8.3.2.88 debyef()

```
float __gnu_cxx::debyef (
            unsigned int __n,
            float __x ) [inline]
```

Return the Debye function $D_n(x)$ of positive order n and `float` argument x.

See also

debye for details.

Definition at line 6580 of file specfun.h.

8.3.2.89 debyel()

```
long double __gnu_cxx::debyel (
            unsigned int __n,
            long double __x ) [inline]
```

Return the Debye function $D_n(x)$ of positive order n and real argument x.

See also

debye for details.

Definition at line 6590 of file specfun.h.

8.3.2.90 dilog()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 3136 of file specfun.h.

8.3.2.91 dilogf()

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

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

See also

dilog for details.

Definition at line 3110 of file specfun.h.

8.3.2.92 dilogl()

```
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 3120 of file specfun.h.

8.3.2.93 dirichlet_beta()

```
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:

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

The Dirichlet beta function, in terms of the polylogarithm, is

$$\beta(s) = \operatorname{Im} Li_s(i)$$

Parameters

__s	

Definition at line 5119 of file specfun.h.

8.3.2.94 dirichlet_betaf()

```
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 5084 of file specfun.h.

8.3.2.95 dirichlet_betal()

```
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 5093 of file specfun.h.

8.3.2.96 dirichlet_eta()

```
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:

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

The Dirichlet eta function, in terms of the polylogarithm, is

$$\eta(s) = -\operatorname{Re} Li_s(-1)$$

Parameters

__s	

Definition at line 5070 of file specfun.h.

8.3.2.97 dirichlet_etaf()

```
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 5034 of file specfun.h.

8.3.2.98 dirichlet_etal()

```
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 5043 of file specfun.h.

8.3.2.99 dirichlet_lambda()

```
template<typename _Tp >
_Tp __gnu_cxx::dirichlet_lambda (
            _Tp __s ) [inline]
```

Return the Dirichlet lambda function of real argument s.

The Dirichlet lambda function is defined by

$$\lambda(s) = \sum_{k=0}^{\infty} \frac{1}{(2k+1)^s} = \left(1 - 2^{-s}\right) \zeta(s)$$

In terms of the Riemann zeta and the Dirichlet eta functions

$$\lambda(s) = \frac{1}{2}(\zeta(s) + \eta(s))$$

Parameters

_↩ _s	

Definition at line 5162 of file specfun.h.

8.3.2.100 dirichlet_lambdaf()

```
float __gnu_cxx::dirichlet_lambdaf (
            float __s )  [inline]
```

Return the Dirichlet lambda function of real argument s.

See also

dirichlet_lambda for details.

Definition at line 5133 of file specfun.h.

8.3.2.101 dirichlet_lambdal()

```
long double __gnu_cxx::dirichlet_lambdal (
            long double __s )  [inline]
```

Return the Dirichlet lambda function of real argument s.

See also

dirichlet_lambda for details.

Definition at line 5142 of file specfun.h.

8.3.2.102 double_factorial()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::double_factorial (
            int __n )  [inline]
```

Return the double factorial $n!!$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

for odd n.

Definition at line 4058 of file specfun.h.

8.3.2.103 double_factorialf()

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

Return the double factorial $n!!$ of the argument as a `float`.

See also

double_factorial for more details

Definition at line 4031 of file specfun.h.

8.3.2.104 double_factoriall()

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

Return the double factorial $n!!$ of the argument as a `long double` .

See also

double_factorial for more details

Definition at line 4041 of file specfun.h.

8.3.2.105 ellint_cel()

```
template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
__gnu_cxx::__promote_fp_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 4696 of file specfun.h.

8.3.2.106 ellint_celf()

```
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 4664 of file specfun.h.

8.3.2.107 ellint_cell()

```
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 4673 of file specfun.h.

8.3.2.108 ellint_d()

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::__promote_fp_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 4509 of file specfun.h.

8.3.2.109 ellint_df()

```
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 4481 of file specfun.h.

8.3.2.110 ellint_dl()

```
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 4491 of file specfun.h.

8.3.2.111 ellint_el1()

```
template<typename _Tp , typename _Tk >
__gnu_cxx::__promote_fp_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 4555 of file specfun.h.

8.3.2.112 ellint_el1f()

```
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 4525 of file specfun.h.

8.3.2.113 ellint_el1l()

```
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 4536 of file specfun.h.

8.3.2.114 ellint_el2()

```
template<typename _Tp , typename _Tk , typename _Ta , typename _Tb >
__gnu_cxx::__promote_fp_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 4601 of file specfun.h.

8.3.2.115 ellint_el2f()

```
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 4570 of file specfun.h.

8.3.2.116 ellint_el2l()

```
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 4580 of file specfun.h.

8.3.2.117 ellint_el3()

```
template<typename _Tx , typename _Tk , typename _Tp >
__gnu_cxx::__promote_fp_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 4648 of file specfun.h.

8.3.2.118 ellint_el3f()

```
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 4617 of file specfun.h.

8.3.2.119 ellint_el3l()

```
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 4628 of file specfun.h.

8.3.2.120 ellint_rc()

```
template<typename _Tp , typename _Up >
__gnu_cxx::__promote_fp_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 3271 of file specfun.h.

8.3.2.121 ellint_rcf()

```
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 3237 of file specfun.h.

8.3.2.122 ellint_rcl()

```
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 3246 of file specfun.h.

8.3.2.123 ellint_rd()

```
template<typename _Tp , typename _Up , typename _Vp >
__gnu_cxx::__promote_fp_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 3370 of file specfun.h.

8.3.2.124 ellint_rdf()

```
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 3334 of file specfun.h.

8.3.2.125 ellint_rdl()

```
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 3343 of file specfun.h.

8.3.2.126 ellint_rf()

```
template<typename _Tp , typename _Up , typename _Vp >
__gnu_cxx::__promote_fp_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 3223 of file specfun.h.

8.3.2.127 ellint_rff()

```
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 3194 of file specfun.h.

8.3.2.128 ellint_rfl()

```
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 3204 of file specfun.h.

8.3.2.129 ellint_rg()

```
template<typename _Tp , typename _Up , typename _Vp >
__gnu_cxx::__promote_fp_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 3461 of file specfun.h.

8.3.2.130 ellint_rgf()

```
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 3426 of file specfun.h.

8.3.2.131 ellint_rgl()

```
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 3435 of file specfun.h.

8.3.2.132 ellint_rj()

```
template<typename _Tp , typename _Up , typename _Vp , typename _Wp >
__gnu_cxx::__promote_fp_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 3320 of file specfun.h.

8.3.2.133 ellint_rjf()

```
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 3285 of file specfun.h.

8.3.2.134 ellint_rjl()

```
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 3294 of file specfun.h.

8.3.2.135 ellnome()

```
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

$$q(k) = \exp\left(-\pi\frac{K(\sqrt{1-k^2})}{K(k)}\right)$$

where $K(k)$ is the complete elliptic function of the first kind.

Template Parameters

_Tp	The real type of the modulus

Parameters

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

Definition at line 5551 of file specfun.h.

8.3.2.136 ellnomef()

```
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 5524 of file specfun.h.

8.3.2.137 ellnomel()

```
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 5534 of file specfun.h.

8.3.2.138 euler()

```
template<typename _Tp >
_Tp __gnu_cxx::euler (
            unsigned int __n ) [inline]
```

This returns Euler number E_n.

Parameters

___n	the order n of the Euler number.

Returns

The Euler number of order n.

Definition at line 6650 of file specfun.h.

8.3.2.139 eulerian_1()

```
template<typename _Tp >
_Tp __gnu_cxx::eulerian_1 (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle \begin{matrix} n \\ m \end{matrix} \right\rangle = (n-m)\left\langle \begin{matrix} n-1 \\ m-1 \end{matrix} \right\rangle + (m+1)\left\langle \begin{matrix} n-1 \\ m \end{matrix} \right\rangle \text{ for } n > 0$$

Note that $A(n,m)$ is a common older notation.

Todo Develop an iterator model for Eulerian numbers of the first kind.

Definition at line 6668 of file specfun.h.

8.3.2.140 eulerian_2()

```
template<typename _Tp >
_Tp __gnu_cxx::eulerian_2 (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the Eulerian number of the second kind. The Eulerian numbers of the second kind are defined by recursion:

$$\left\langle\!\!\left\langle \begin{matrix} n \\ m \end{matrix} \right\rangle\!\!\right\rangle = (2n-m-1)\left\langle\!\!\left\langle \begin{matrix} n-1 \\ m-1 \end{matrix} \right\rangle\!\!\right\rangle + (m+1)\left\langle\!\!\left\langle \begin{matrix} n-1 \\ m \end{matrix} \right\rangle\!\!\right\rangle \text{ for } n > 0$$

Todo Develop an iterator model for Eulerian numbers of the second kind.

Definition at line 6686 of file specfun.h.

8.3.2.141 expint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::expint (
            unsigned int __n,
            _Tp __x ) [inline]
```

Return the exponential integral $E_n(x)$ of integral order n and real argument x. The exponential integral is defined by:

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

In particular

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

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The integral order
__x	The real argument

Definition at line 3800 of file specfun.h.

8.3.2.142 expintf()

```
float __gnu_cxx::expintf (
            unsigned int __n,
            float __x ) [inline]
```

Return the exponential integral $E_n(x)$ for integral order n and `float` argument x.

See also

expint for details.

Definition at line 3769 of file specfun.h.

8.3.2.143 expintl()

```
long double __gnu_cxx::expintl (
            unsigned int __n,
            long double __x ) [inline]
```

Return the exponential integral $E_n(x)$ for integral order n and `long double` argument x.

See also

expint for details.

Definition at line 3779 of file specfun.h.

8.3.2.144 exponential_cdf()

```
template<typename _Tlam , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tlam, _Tp> __gnu_cxx::exponential_cdf (
            _Tlam __lambda,
            _Tp __x ) [inline]
```

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-\lambda x} \text{ for } x >= 0$$

Definition at line 6327 of file specfun.h.

8.3.2.145 exponential_pdf()

```
template<typename _Tlam , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tlam, _Tp> __gnu_cxx::exponential_pdf (
            _Tlam __lambda,
            _Tp __x ) [inline]
```

Return the exponential probability density function.

The formula for the exponential probability density function is

$$f(x|\lambda) = \lambda e^{-\lambda x} \text{ for } x >= 0$$

Definition at line 6311 of file specfun.h.

8.3.2.146 factorial()

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

Return the factorial $n!$ of the argument as a real number.

$$n! = 1 \times 2 \times ... \times n, 0! = 1$$

.

Definition at line 4017 of file specfun.h.

8.3.2.147 factorialf()

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

Return the factorial $n!$ of the argument as a `float`.

See also

factorial for more details

Definition at line 3997 of file specfun.h.

8.3.2.148 factoriall()

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

Return the factorial $n!$ of the argument as a `long double`.

See also

factorial for more details

Definition at line 4006 of file specfun.h.

8.3.2.149 falling_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::__promote_fp_t<_Tp, _Tnu> __gnu_cxx::falling_factorial (
            _Tp __a,
            _Tnu __nu ) [inline]
```

Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

In particular, $n^{\underline{n}} = n!$.

Definition at line 3983 of file specfun.h.

8.3.2.150 falling_factorialf()

```
float __gnu_cxx::falling_factorialf (
            float __a,
            float __nu ) [inline]
```

Return the falling factorial $a^{\underline{\nu}}$ for float arguments.

See also

falling_factorial for details.

Definition at line 3957 of file specfun.h.

8.3.2.151 falling_factoriall()

```
long double __gnu_cxx::falling_factoriall (
            long double __a,
            long double __nu ) [inline]
```

Return the falling factorial $a^{\underline{\nu}}$ for `long double` arguments.

See also

falling_factorial for details.

Definition at line 3967 of file specfun.h.

8.3.2.152 fermi_dirac()

```
template<typename _Tps , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tps, _Tp> __gnu_cxx::fermi_dirac (
            _Tps __s,
            _Tp __x ) [inline]
```

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

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12.16

$$F_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x}+1} dt = -Li_{s+1}(-e^x)$$

Parameters

__s	The order s > -1.
__x	The real argument.

Returns

The real Fermi-Dirac integral F_s(x),

Definition at line 5826 of file specfun.h.

8.3.2.153 fermi_diracf()

```
float __gnu_cxx::fermi_diracf (
            float __s,
            float __x ) [inline]
```

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

See also

fermi_dirac for details.

Definition at line 5796 of file specfun.h.

8.3.2.154 fermi_diracl()

```
long double __gnu_cxx::fermi_diracl (
            long double __s,
            long double __x )  [inline]
```

Return the Fermi-Dirac integral of `long double` order s and argument x.

See also

fermi_dirac for details.

Definition at line 5806 of file specfun.h.

8.3.2.155 fisher_f_cdf()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::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 6425 of file specfun.h.

8.3.2.156 fisher_f_pdf()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::fisher_f_pdf (
```

```
_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 6450 of file specfun.h.

8.3.2.157 fresnel_c()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 3711 of file specfun.h.

8.3.2.158 fresnel_cf()

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

Definition at line 3692 of file specfun.h.

8.3.2.159 fresnel_cl()

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

Definition at line 3696 of file specfun.h.

8.3.2.160 fresnel_s()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 3683 of file specfun.h.

8.3.2.161 fresnel_sf()

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

Definition at line 3664 of file specfun.h.

8.3.2.162 fresnel_sl()

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

Definition at line 3668 of file specfun.h.

8.3.2.163 gamma_cdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::gamma_cdf (
            _Ta __alpha,
            _Tb __beta,
            _Tp __x ) [inline]
```

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)}(x/\beta)^{\alpha-1}e^{-x/\beta}$$

Definition at line 6229 of file specfun.h.

References std::__detail::__beta().

8.3.2.164 gamma_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::gamma_pdf (
            _Ta __alpha,
            _Tb __beta,
            _Tp __x ) [inline]
```

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)}(x/\beta)^{\alpha-1}e^{-x/\beta}$$

Definition at line 6212 of file specfun.h.

References std::__detail::__beta().

8.3.2.165 gamma_reciprocal()

```
template<typename _Ta >
__gnu_cxx::__promote_fp_t<_Ta> __gnu_cxx::gamma_reciprocal (
            _Ta __a ) [inline]
```

Return the reciprocal gamma function for real argument.

The reciprocal of the Gamma function is what you'd expect:

$$\Gamma_r(a) = \frac{1}{\Gamma(a)}$$

But unlike the Gamma function this function has no singularities and is exponentially decreasing for increasing argument.

Definition at line 6565 of file specfun.h.

8.3.2.166 gamma_reciprocalf()

```
float __gnu_cxx::gamma_reciprocalf (
            float __a ) [inline]
```

Return the reciprocal gamma function for `float` argument.

See also

gamma_reciprocal for details.

Definition at line 6540 of file specfun.h.

8.3.2.167 gamma_reciprocall()

```
long double __gnu_cxx::gamma_reciprocall (
            long double __a ) [inline]
```

Return the reciprocal gamma function for `long double` argument.

See also

gamma_reciprocal for details.

Definition at line 6550 of file specfun.h.

8.3.2.168 gegenbauer()

```
template<typename _Talpha , typename _Tp >
__gnu_cxx::__promote_fp_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 polynomial is 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 2295 of file specfun.h.

8.3.2.169 gegenbauerf()

```
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 2262 of file specfun.h.

8.3.2.170 gegenbauerl()

```
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 2273 of file specfun.h.

8.3.2.171 harmonic()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::harmonic (
             unsigned int __n ) [inline]
```

Return the harmonic number H_n.

The the harmonic number is defined by

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

Parameters

_↩ _n	The parameter

Definition at line 3575 of file specfun.h.

8.3.2.172 heuman_lambda()

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::__promote_fp_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 4424 of file specfun.h.

8.3.2.173 heuman_lambdaf()

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

Definition at line 4398 of file specfun.h.

8.3.2.174 heuman_lambdal()

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

Definition at line 4402 of file specfun.h.

8.3.2.175 hurwitz_zeta() [1/2]

```
template<typename _Tp , typename _Up >
__gnu_cxx::__promote_fp_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 3503 of file specfun.h.

8.3.2.176 hurwitz_zeta() [2/2]

```
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 3517 of file specfun.h.

8.3.2.177 hurwitz_zetaf()

```
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 3476 of file specfun.h.

8.3.2.178 hurwitz_zetal()

```
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 3486 of file specfun.h.

8.3.2.179 hyperg()

```
template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpa, _Tpb, _Tpc, _Tp> __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 1522 of file specfun.h.

8.3.2.180 hypergf()

```
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 1489 of file specfun.h.

8.3.2.181 hypergl()

```
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 1500 of file specfun.h.

8.3.2.182 ibeta()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_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 incomplete 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 3624 of file specfun.h.

8.3.2.183 ibetac()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_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 3655 of file specfun.h.

8.3.2.184 ibetacf()

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

Definition at line 3633 of file specfun.h.

References __gnu_cxx::ibetaf().

8.3.2.185 ibetacl()

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

Definition at line 3637 of file specfun.h.

References __gnu_cxx::ibetal().

8.3.2.186 ibetaf()

```
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 3590 of file specfun.h.

Referenced by __gnu_cxx::ibetacf().

8.3.2.187 ibetal()

```
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 3600 of file specfun.h.

Referenced by __gnu_cxx::ibetacl().

8.3.2.188 jacobi()

```
template<typename _Talpha , typename _Tbeta , typename _Tp >
__gnu_cxx::__promote_fp_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 2245 of file specfun.h.

References std::__detail::__beta().

8.3.2.189 jacobi_cn()

```
template<typename _Kp , typename _Up >
__gnu_cxx::__promote_fp_t<_Kp, _Up> __gnu_cxx::jacobi_cn (
            _Kp __k,
            _Up __u )  [inline]
```

Return the Jacobi elliptic cosine amplitude function $cn(k, u)$ 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 Legendre elliptic integral of the first kind (

See also

ellint_1).

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 1950 of file specfun.h.

8.3.2.190 jacobi_cnf()

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

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

See also

jacobi_cn for details.

Definition at line 1915 of file specfun.h.

8.3.2.191 jacobi_cnl()

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

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

See also

jacobi_cn for details.

Definition at line 1927 of file specfun.h.

8.3.2.192 jacobi_dn()

```
template<typename _Kp , typename _Up >
__gnu_cxx::__promote_fp_t<_Kp, _Up> __gnu_cxx::jacobi_dn (
            _Kp __k,
            _Up __u ) [inline]
```

Return the Jacobi elliptic delta amplitude function $dn(k,u)$ 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 Legendre elliptic integral of the first kind (

See also

ellint_1).

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 2000 of file specfun.h.

8.3.2.193 jacobi_dnf()

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

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

See also

jacobi_dn for details.

Definition at line 1965 of file specfun.h.

8.3.2.194 jacobi_dnl()

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

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

See also

jacobi_dn for details.

Definition at line 1977 of file specfun.h.

8.3.2.195 jacobi_sn()

```
template<typename _Kp , typename _Up >
__gnu_cxx::__promote_fp_t<_Kp, _Up> __gnu_cxx::jacobi_sn (
            _Kp __k,
            _Up __u )  [inline]
```

Return the Jacobi elliptic sine amplitude function $sn(k, u)$ 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 Legendre elliptic integral of the first kind (

See also

ellint_1).

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 1900 of file specfun.h.

8.3.2.196 jacobi_snf()

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

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

See also

jacobi_sn for details.

Definition at line 1865 of file specfun.h.

8.3.2.197 jacobi_snl()

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

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

See also

jacobi_sn for details.

Definition at line 1877 of file specfun.h.

8.3.2.198 jacobi_zeta()

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::__promote_fp_t<_Tk, _Tphi> __gnu_cxx::jacobi_zeta (
            _Tk __k,
            _Tphi __phi ) [inline]
```

Return the Jacobi zeta function of k and ϕ.

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 4389 of file specfun.h.

8.3.2.199 jacobi_zetaf()

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

Definition at line 4364 of file specfun.h.

8.3.2.200 jacobi_zetal()

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

Definition at line 4368 of file specfun.h.

8.3.2.201 jacobif()

```
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 2194 of file specfun.h.

References std::__detail::__beta().

8.3.2.202 jacobil()

```
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 2208 of file specfun.h.

References std::__detail::__beta().

8.3.2.203 lbinomial()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::lbinomial (
            unsigned int __n,
            unsigned int __k ) [inline]
```

Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

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

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

.

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 4223 of file specfun.h.

8.3.2.204 lbinomialf()

```
float __gnu_cxx::lbinomialf (
            unsigned int __n,
            unsigned int __k )  [inline]
```

Return the logarithm of the binomial coefficient as a `float`.

See also

lbinomial for details.

Definition at line 4194 of file specfun.h.

8.3.2.205 lbinomiall()

```
long double __gnu_cxx::lbinomiall (
            unsigned int __n,
            unsigned int __k )  [inline]
```

Return the logarithm of the binomial coefficient as a `long double`.

See also

lbinomial for details.

Definition at line 4203 of file specfun.h.

8.3.2.206 ldouble_factorial()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::ldouble_factorial (
            int __n )  [inline]
```

Return the logarithm of the double factorial $ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

for odd n.

Definition at line 4137 of file specfun.h.

8.3.2.207 ldouble_factorialf()

```
float __gnu_cxx::ldouble_factorialf (
            int __n )  [inline]
```

Return the logarithm of the double factorial $ln(n!!)$ of the argument as a `float`.

See also

ldouble_factorial for more details

Definition at line 4110 of file specfun.h.

8.3.2.208 ldouble_factoriall()

```
long double __gnu_cxx::ldouble_factoriall (
            int __n )  [inline]
```

Return the logarithm of the double factorial $ln(n!!)$ of the argument as a `long double` .

See also

double_factorial for more details

Definition at line 4120 of file specfun.h.

8.3.2.209 legendre_q()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::legendre_q (
            unsigned int __l,
            _Tp __x )  [inline]
```

Return the Legendre function of the second kind $Q_l(x)$ of nonnegative degree l and real argument $|x| <= 0$.

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

$$Q_l(x) = \frac{1}{2}\log\frac{x+1}{x-1}P_l(x) - \sum_{k=0}^{l-1}\frac{(l+k)!}{(l-k)!(k!)^2 s^k}\left[\psi(l+1) - \psi(k+1)\right](x-1)^k$$

where $P_l(x)$ is the Legendre polynomial of degree l and $\psi(x)$ is the psi or dilogarithm function.

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 4313 of file specfun.h.

8.3.2.210 legendre_qf()

```
float __gnu_cxx::legendre_qf (
            unsigned int __l,
            float __x ) [inline]
```

Return the Legendre function of the second kind $Q_l(x)$ of nonnegative degree l and `float` argument.

See also

legendre_q for details.

Definition at line 4279 of file specfun.h.

8.3.2.211 legendre_ql()

```
long double __gnu_cxx::legendre_ql (
            unsigned int __l,
            long double __x ) [inline]
```

Return the Legendre function of the second kind $Q_l(x)$ of nonnegative degree l and `long double` argument.

See also

legendre_q for details.

Definition at line 4289 of file specfun.h.

8.3.2.212 lfactorial()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::lfactorial (
            unsigned int __n ) [inline]
```

Return the logarithm of the factorial $ln(n!)$ of the argument as a real number.

$$n! = 1 \times 2 \times ... \times n, 0! = 1$$

.

Definition at line 4095 of file specfun.h.

8.3.2.213 lfactorialf()

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

Return the logarithm of the factorial $ln(n!)$ of the argument as a `float`.

See also

lfactorial for more details

Definition at line 4073 of file specfun.h.

8.3.2.214 lfactoriall()

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

Return the logarithm of the factorial $ln(n!)$ of the argument as a `long double`.

See also

lfactorial for more details

Definition at line 4083 of file specfun.h.

8.3.2.215 lfalling_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::__promote_fp_t<_Tp, _Tnu> __gnu_cxx::lfalling_factorial (
            _Tp __a,
            _Tnu __nu ) [inline]
```

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

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

In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

Many notations exist for this function: $(a)_\nu$,

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

, and others.

Definition at line 3899 of file specfun.h.

8.3.2.216 lfalling_factorialf()

```
float __gnu_cxx::lfalling_factorialf (
            float __a,
            float __nu ) [inline]
```

Return the logarithm of the falling factorial $ln(a^{\overline{\nu}})$ for float arguments.

See also

lfalling_factorial for details.

Definition at line 3864 of file specfun.h.

8.3.2.217 lfalling_factoriall()

```
long double __gnu_cxx::lfalling_factoriall (
            long double __a,
            long double __nu ) [inline]
```

Return the logarithm of the falling factorial $ln(a^{\overline{\nu}})$ for float arguments.

See also

lfalling_factorial for details.

Definition at line 3874 of file specfun.h.

8.3.2.218 lgamma() [1/2]

```
template<typename _Ta >
__gnu_cxx::__promote_fp_t<_Ta> __gnu_cxx::lgamma (
            _Ta __a )  [inline]
```

Return the logarithm of the gamma function for real argument.

Definition at line 2924 of file specfun.h.

Referenced by std::__detail::__gegenbauer_zeros(), std::__detail::__jacobi_zeros(), and std::__detail::__laguerre_↩
zeros().

8.3.2.219 lgamma() [2/2]

```
template<typename _Ta >
std::complex<__gnu_cxx::__promote_fp_t<_Ta> > __gnu_cxx::lgamma (
            std::complex< _Ta > __a )  [inline]
```

Return the logarithm of the gamma function for complex argument.

Definition at line 2957 of file specfun.h.

8.3.2.220 lgammaf() [1/2]

```
float __gnu_cxx::lgammaf (
            float __a )  [inline]
```

Return the logarithm of the gamma function for `float` argument.

See also

lgamma for details.

Definition at line 2906 of file specfun.h.

8.3.2.221 lgammaf() [2/2]

```
std::complex<float> __gnu_cxx::lgammaf (
            std::complex< float > __a )  [inline]
```

Return the logarithm of the gamma function for `std::complex<float>` argument.

See also

lgamma for details.

Definition at line 2939 of file specfun.h.

8.3.2.222 lgammal() [1/2]

```
long double __gnu_cxx::lgammal (
            long double __a )  [inline]
```

Return the logarithm of the gamma function for `long double` argument.

See also

lgamma for details.

Definition at line 2916 of file specfun.h.

8.3.2.223 lgammal() [2/2]

```
std::complex<long double> __gnu_cxx::lgammal (
            std::complex< long double > __a )  [inline]
```

Return the logarithm of the gamma function for `std::complex<long double>` argument.

See also

lgamma for details.

Definition at line 2949 of file specfun.h.

8.3.2.224 logint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 1688 of file specfun.h.

8.3.2.225 logintf()

```
float __gnu_cxx::logintf (
            float __x ) [inline]
```

Return the logarithmic integral of argument x.

See also

logint for details.

Definition at line 1664 of file specfun.h.

8.3.2.226 logintl()

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

Return the logarithmic integral of argument x.

See also

logint for details.

Definition at line 1673 of file specfun.h.

8.3.2.227 logistic_cdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::logistic_cdf (
            _Ta __a,
            _Tb __b,
            _Tp __x ) [inline]
```

Return the logistic cumulative distribution function.

The formula for the logistic probability function is

$$P(x|a,b) = \frac{e^{(x-a)/b}}{1 + e^{(x-a)/b}}$$

where $b > 0$.

Definition at line 6526 of file specfun.h.

8.3.2.228 logistic_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::logistic_pdf (
            _Ta __a,
            _Tb __b,
            _Tp __x ) [inline]
```

Return the logistic probability density function.

The formula for the logistic probability density function is

$$f(x|a,b) = \frac{e^{(x-a)/b}}{b[1 + e^{(x-a)/b}]^2}$$

where $b > 0$.

Definition at line 6509 of file specfun.h.

8.3.2.229 lognormal_cdf()

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tmu, _Tsig, _Tp> __gnu_cxx::lognormal_cdf (
            _Tmu __mu,
            _Tsig __sigma,
            _Tp __x ) [inline]
```

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

$$F(x|\mu,\sigma) = \frac{1}{2}\left[1 - erf(\frac{\ln x - \mu}{\sqrt{2}\sigma})\right]$$

Definition at line 6295 of file specfun.h.

8.3.2.230 lognormal_pdf()

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tmu, _Tsig, _Tp> __gnu_cxx::lognormal_pdf (
            _Tmu __mu,
            _Tsig __sigma,
            _Tp __x ) [inline]
```

Return the lognormal probability density function.

The formula for the lognormal probability density function is

$$f(x|\mu,\sigma) = \frac{e^{(\ln x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 6278 of file specfun.h.

8.3.2.231 lrising_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::__promote_fp_t<_Tp, _Tnu> __gnu_cxx::lrising_factorial (
            _Tp __a,
            _Tnu __nu ) [inline]
```

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

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

Thus this function returns

$$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[a^{\overline{0}}] = 0$$

Many notations exist for this function: $(a)_\nu$ (especially in the literature of special functions),

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

, and others.

Definition at line 3849 of file specfun.h.

8.3.2.232 lrising_factorialf()

```
float __gnu_cxx::lrising_factorialf (
            float __a,
            float __nu ) [inline]
```

Return the logarithm of the rising factorial $a^{\overline{\nu}}$ for float arguments.

See also

lrising_factorial for details.

Definition at line 3815 of file specfun.h.

8.3.2.233 lrising_factoriall()

```
long double __gnu_cxx::lrising_factoriall (
            long double __a,
            long double __nu ) [inline]
```

Return the logarithm of the rising factorial $ln(a^{\overline{\nu}})$ for `long double` arguments.

See also

lrising_factorial for details.

Definition at line 3825 of file specfun.h.

8.3.2.234 normal_cdf()

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tmu, _Tsig, _Tp> __gnu_cxx::normal_cdf (
            _Tmu __mu,
            _Tsig __sigma,
            _Tp __x ) [inline]
```

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

$$F(x|\mu,\sigma) = \frac{1}{2}\left[1 - erf(\frac{x-\mu}{\sqrt{2}\sigma})\right]$$

Definition at line 6262 of file specfun.h.

8.3.2.235 normal_pdf()

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tmu, _Tsig, _Tp> __gnu_cxx::normal_pdf (
            _Tmu __mu,
            _Tsig __sigma,
            _Tp __x ) [inline]
```

Return the normal probability density function.

The formula for the normal probability density function is

$$f(x|\mu,\sigma) = \frac{e^{(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 6245 of file specfun.h.

8.3.2.236 owens_t()

```
template<typename _Tph , typename _Tpa >
__gnu_cxx::__promote_fp_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 5782 of file specfun.h.

8.3.2.237 owens_tf()

```
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 5754 of file specfun.h.

8.3.2.238 owens_tl()

```
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 5764 of file specfun.h.

8.3.2.239 pgamma()

```
template<typename _Ta , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::pgamma (
            _Ta __a,
            _Tp __x ) [inline]
```

Definition at line 4334 of file specfun.h.

8.3.2.240 pgammaf()

```
float __gnu_cxx::pgammaf (
            float __a,
            float __x ) [inline]
```

Definition at line 4322 of file specfun.h.

8.3.2.241 pgammal()

```
long double __gnu_cxx::pgammal (
            long double __a,
            long double __x ) [inline]
```

Definition at line 4326 of file specfun.h.

8.3.2.242 polylog() [1/2]

```
template<typename _Tp , typename _Wp >
__gnu_cxx::__promote_fp_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 4980 of file specfun.h.

8.3.2.243 polylog() [2/2]

```
template<typename _Tp , typename _Wp >
std::complex<__gnu_cxx::__promote_fp_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 5020 of file specfun.h.

8.3.2.244 polylogf() [1/2]

```
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 4953 of file specfun.h.

8.3.2.245 polylogf() [2/2]

```
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 4993 of file specfun.h.

8.3.2.246 polylogl() [1/2]

```
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 4963 of file specfun.h.

8.3.2.247 polylogl() [2/2]

```
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 5003 of file specfun.h.

8.3.2.248 psi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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

$$\psi(x) = \frac{d}{dx} log\left(\Gamma(x)\right) = \frac{\Gamma'(x)}{\Gamma(x)}$$

Parameters

__x	The parameter

Definition at line 3557 of file specfun.h.

8.3.2.249 psif()

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

Return the psi or digamma function of `float` argument x.

See also

psi for details.

Definition at line 3531 of file specfun.h.

8.3.2.250 psil()

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

Return the psi or digamma function of `long double` argument x.

See also

psi for details.

Definition at line 3541 of file specfun.h.

8.3.2.251 qgamma()

```
template<typename _Ta , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::qgamma (
            _Ta __a,
            _Tp __x ) [inline]
```

Definition at line 4355 of file specfun.h.

8.3.2.252 qgammaf()

```
float __gnu_cxx::qgammaf (
            float __a,
            float __x )  [inline]
```

Definition at line 4343 of file specfun.h.

8.3.2.253 qgammal()

```
long double __gnu_cxx::qgammal (
            long double __a,
            long double __x )  [inline]
```

Definition at line 4347 of file specfun.h.

8.3.2.254 radpoly()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2405 of file specfun.h.

8.3.2.255 radpolyf()

```
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 2366 of file specfun.h.

References std::__detail::__poly_radial_jacobi().

8.3.2.256 radpolyl()

```
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 2377 of file specfun.h.

References std::__detail::__poly_radial_jacobi().

8.3.2.257 rising_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::__promote_fp_t<_Tp, _Tnu> __gnu_cxx::rising_factorial (
            _Tp __a,
            _Tnu __nu ) [inline]
```

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

$$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

Many notations exist for this function: $(a)_\nu$, (especially in the literature of special functions),

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

, and others.

Definition at line 3942 of file specfun.h.

8.3.2.258 rising_factorialf()

```
float __gnu_cxx::rising_factorialf (
            float __a,
            float __nu ) [inline]
```

Return the rising factorial $a^{\overline{\nu}}$ for float arguments.

See also

rising_factorial for details.

Definition at line 3914 of file specfun.h.

8.3.2.259 rising_factoriall()

```
long double __gnu_cxx::rising_factoriall (
            long double __a,
            long double __nu ) [inline]
```

Return the rising factorial $a^{\overline{\nu}}$ for `long double` arguments.

See also

rising_factorial for details.

Definition at line 3924 of file specfun.h.

8.3.2.260 sin_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sin_pi (
            _Tp __x ) [inline]
```

Return the reperiodized sine function $\sin_\pi(x)$ for real argument x.

The reperiodized sine function is defined by:

$$\sin_\pi(x) = \sin(\pi x)$$

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__x	The argument

Definition at line 5912 of file specfun.h.

8.3.2.261 sin_pif()

```
float __gnu_cxx::sin_pif (
            float __x ) [inline]
```

Return the reperiodized sine function $\sin_\pi(x)$ for `float` argument x.

See also

sin_pi for more details.

Definition at line 5885 of file specfun.h.

8.3.2.262 sin_pil()

```
long double __gnu_cxx::sin_pil (
            long double __x ) [inline]
```

Return the reperiodized sine function $\sin_\pi(x)$ for `long double` argument x.

See also

sin_pi for more details.

Definition at line 5895 of file specfun.h.

8.3.2.263 sinc()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sinc (
            _Tp __x ) [inline]
```

Return the sinus cardinal function $sinc_\pi(x)$ for real argument `__x`. The sinus cardinal function is defined by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

__x	The argument

Definition at line 1609 of file specfun.h.

8.3.2.264 sinc_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sinc_pi (
            _Tp __x ) [inline]
```

Return the reperiodized sinus cardinal function $sinc(x)$ for real argument `__x`. The normalized sinus cardinal function is defined by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

__x	The argument

Definition at line 1650 of file specfun.h.

8.3.2.265 sinc_pif()

```
float __gnu_cxx::sinc_pif (
            float __x ) [inline]
```

Return the reperiodized sinus cardinal function $sinc(x)$ for `float` argument `__x`.

See also

sinc for details.

Definition at line 1624 of file specfun.h.

8.3.2.266 sinc_pil()

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

Return the reperiodized sinus cardinal function $sinc(x)$ for `long double` argument `__x`.

See also

sinc for details.

Definition at line 1634 of file specfun.h.

8.3.2.267 sincf()

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

Return the sinus cardinal function $sinc_\pi(x)$ for `float` argument `__x`.

See also

sinc_pi for details.

Definition at line 1583 of file specfun.h.

8.3.2.268 sincl()

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

Return the sinus cardinal function $sinc_\pi(x)$ for `long double` argument __x.

See also

sinc_pi for details.

Definition at line 1593 of file specfun.h.

8.3.2.269 sincos() [1/2]

```
__gnu_cxx::__sincos_t<double> __gnu_cxx::sincos (
            double __x )  [inline]
```

Return both the sine and the cosine of a `double` argument.

See also

sincos for details.

Definition at line 6150 of file specfun.h.

8.3.2.270 sincos() [2/2]

```
template<typename _Tp >
__gnu_cxx::__sincos_t<__gnu_cxx::__promote_fp_t<_Tp> > __gnu_cxx::sincos (
            _Tp __x )  [inline]
```

Return both the sine and the cosine of a reperiodized argument.

$$sincos(x) = \sin(x), \cos(x)$$

Definition at line 6161 of file specfun.h.

8.3.2.271 sincos_pi()

```
template<typename _Tp >
__gnu_cxx::__sincos_t<__gnu_cxx::__promote_fp_t<_Tp> > __gnu_cxx::sincos_pi (
            _Tp __x ) [inline]
```

Return both the sine and the cosine of a reperiodized real argument.

$$sincos_\pi(x) = \sin(\pi x), \cos(\pi x)$$

Definition at line 6195 of file specfun.h.

8.3.2.272 sincos_pif()

```
__gnu_cxx::__sincos_t<float> __gnu_cxx::sincos_pif (
            float __x ) [inline]
```

Return both the sine and the cosine of a reperiodized `float` argument.

See also

sincos_pi for details.

Definition at line 6173 of file specfun.h.

8.3.2.273 sincos_pil()

```
__gnu_cxx::__sincos_t<long double> __gnu_cxx::sincos_pil (
            long double __x ) [inline]
```

Return both the sine and the cosine of a reperiodized `long double` argument.

See also

sincos_pi for details.

Definition at line 6183 of file specfun.h.

8.3.2.274 sincosf()

```
__gnu_cxx::__sincos_t<float> __gnu_cxx::sincosf (
            float __x ) [inline]
```

Return both the sine and the cosine of a `float` argument.

Definition at line 6132 of file specfun.h.

8.3.2.275 sincosl()

```
__gnu_cxx::__sincos_t<long double> __gnu_cxx::sincosl (
            long double __x ) [inline]
```

Return both the sine and the cosine of a `long double` argument.

See also

sincos for details.

Definition at line 6141 of file specfun.h.

8.3.2.276 sinh_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sinh_pi (
            _Tp __x ) [inline]
```

Return the reperiodized hyperbolic sine function $\sinh_\pi(x)$ for real argument x.

The reperiodized hyperbolic sine function is defined by:

$$\sinh_\pi(x) = \sinh(\pi x)$$

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__x	The argument

Definition at line 5954 of file specfun.h.

8.3.2.277 sinh_pif()

```
float __gnu_cxx::sinh_pif (
            float __x ) [inline]
```

Return the reperiodized hyperbolic sine function $\sinh_\pi(x)$ for `float` argument x.

See also

sinh_pi for more details.

Definition at line 5927 of file specfun.h.

8.3.2.278 sinh_pil()

```
long double __gnu_cxx::sinh_pil (
            long double __x ) [inline]
```

Return the reperiodized hyperbolic sine function $\sinh_\pi(x)$ for `long double` argument x.

See also

sinh_pi for more details.

Definition at line 5937 of file specfun.h.

8.3.2.279 sinhc()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sinhc (
            _Tp __x ) [inline]
```

Return the normalized hyperbolic sinus cardinal function $sinhc(x)$ for real argument __x. The normalized hyperbolic sinus cardinal function is defined by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

_↩ _x	The argument

Definition at line 2487 of file specfun.h.

8.3.2.280 sinhc_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::sinhc_pi (
            _Tp __x ) [inline]
```

Return the hyperbolic sinus cardinal function $sinhc_\pi(x)$ for real argument `__x`. The sinus cardinal function is defined by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

_↩ _x	The argument

Definition at line 2446 of file specfun.h.

8.3.2.281 sinhc_pif()

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

Return the hyperbolic sinus cardinal function $sinhc_\pi(x)$ for `float` argument `__x`.

See also

sinhc_pi for details.

Definition at line 2420 of file specfun.h.

8.3.2.282 sinhc_pil()

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

Return the hyperbolic sinus cardinal function $sinhc_\pi(x)$ for `long double` argument `__x`.

See also

sinhc_pi for details.

Definition at line 2430 of file specfun.h.

8.3.2.283 sinhcf()

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

Return the normalized hyperbolic sinus cardinal function $sinhc(x)$ for `float` argument `__x`.

See also

sinhc for details.

Definition at line 2461 of file specfun.h.

8.3.2.284 sinhcl()

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

Return the normalized hyperbolic sinus cardinal function $sinhc(x)$ for `long double` argument `__x`.

See also

sinhc for details.

Definition at line 2471 of file specfun.h.

8.3.2.285 sinhint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 1808 of file specfun.h.

8.3.2.286 sinhintf()

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

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

See also

sinhint for details.

Definition at line 1781 of file specfun.h.

8.3.2.287 sinhintl()

```
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 1791 of file specfun.h.

8.3.2.288 sinint()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 1727 of file specfun.h.

8.3.2.289 sinintf()

```
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 1702 of file specfun.h.

8.3.2.290 sinintl()

```
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 1712 of file specfun.h.

8.3.2.291 sph_bessel_i()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2723 of file specfun.h.

8.3.2.292 sph_bessel_if()

```
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 2694 of file specfun.h.

8.3.2.293 sph_bessel_il()

```
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 2704 of file specfun.h.

8.3.2.294 sph_bessel_k()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 2767 of file specfun.h.

8.3.2.295 sph_bessel_kf()

```
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 2738 of file specfun.h.

8.3.2.296 sph_bessel_kl()

```
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 2748 of file specfun.h.

8.3.2.297 sph_hankel_1() [1/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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:

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

or in terms of the cylindrical Bessel and Neumann functions by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

_↩ _n	The non-negative order
_↩ _z	The real argument

Definition at line 2631 of file specfun.h.

8.3.2.298 sph_hankel_1() [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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 4838 of file specfun.h.

8.3.2.299 sph_hankel_1f() [1/2]

```
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 2598 of file specfun.h.

8.3.2.300 sph_hankel_1f() [2/2]

```
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 4806 of file specfun.h.

8.3.2.301 sph_hankel_1l() [1/2]

```
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 2608 of file specfun.h.

8.3.2.302 sph_hankel_1l() [2/2]

```
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 4817 of file specfun.h.

8.3.2.303 sph_hankel_2() [1/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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:

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

or in terms of the cylindrical Bessel and Neumann functions by:

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

Template Parameters

_Tp	The real type of the argument

Parameters

__n	The non-negative order
__z	The real argument

Definition at line 2679 of file specfun.h.

8.3.2.304 sph_hankel_2() [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::__promote_fp_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 $h_n^{(2)}(x)$ of nonnegative order n and complex argument x.

The spherical Hankel function of the second kind is defined by

$$h_n^{(2)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} H_{n+1/2}^{(2)}(x) = j_n(x) - i n_n(x)$$

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 4886 of file specfun.h.

8.3.2.305 sph_hankel_2f() [1/2]

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

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

See also

sph_hankel_2 for details.

Definition at line 2646 of file specfun.h.

8.3.2.306 sph_hankel_2f() [2/2]

```
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 4854 of file specfun.h.

8.3.2.307 sph_hankel_2l() [1/2]

```
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 2656 of file specfun.h.

8.3.2.308 sph_hankel_2l() [2/2]

```
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 4865 of file specfun.h.

8.3.2.309 sph_harmonic()

```
template<typename _Ttheta , typename _Tphi >
std::complex<__gnu_cxx::__promote_fp_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 4938 of file specfun.h.

8.3.2.310 sph_harmonicf()

```
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 4902 of file specfun.h.

8.3.2.311 sph_harmonicl()

```
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 4914 of file specfun.h.

8.3.2.312 stirling_1()

```
template<typename _Tp >
_Tp __gnu_cxx::stirling_1 (
            unsigned int __n,
            unsigned int __m ) [inline]
```

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the Pocchammer polynomials or the rising factorials:

$$(x)_n = \sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

The recursion is

$$\begin{bmatrix} n+1 \\ m \end{bmatrix} = \begin{bmatrix} n \\ m-1 \end{bmatrix} - n \begin{bmatrix} n \\ m \end{bmatrix}$$

with starting values

$$\begin{bmatrix} 0 \\ 0 \to m \end{bmatrix} = 1, 0, 0, ..., 0$$

and

$$\begin{bmatrix} 0 \to n \\ 0 \end{bmatrix} = 1, 0, 0, ..., 0$$

The Stirling number of the first kind is denoted by other symbols in the literature, usually $S_n^{(m)}$.

Todo Develop an iterator model for Stirling numbers of the first kind.

Definition at line 6722 of file specfun.h.

8.3.2.313 stirling_2()

```
template<typename _Tp >
_Tp __gnu_cxx::stirling_2 (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the Stirling number of the second kind by series expansion or by recursion.

The series is:

$$\sigma_n^{(m)} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^{m} \frac{(-1)^{m-k} k^n}{(m-k)!k!}$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Todo Develop an iterator model for Stirling numbers of the second kind.

Definition at line 6745 of file specfun.h.

8.3.2.314 student_t_cdf()

```
template<typename _Tt , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::student_t_cdf (
            _Tt __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 6382 of file specfun.h.

8.3.2.315 student_t_pdf()

```
template<typename _Tt , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::student_t_pdf (
            _Tt __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 6402 of file specfun.h.

8.3.2.316 tan_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::tan_pi (
            _Tp __x ) [inline]
```

Return the reperiodized tangent function $\tan_\pi(x)$ for real argument x.

The reperiodized tangent function is defined by:

$$\tan_\pi(x) = \tan(\pi x)$$

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__x	The argument

Definition at line 6080 of file specfun.h.

8.3.2.317 tan_pif()

```
float __gnu_cxx::tan_pif (
            float __x ) [inline]
```

Return the reperiodized tangent function $\tan_\pi(x)$ for `float` argument x.

See also

tan_pi for more details.

Definition at line 6053 of file specfun.h.

8.3.2.318 tan_pil()

```
long double __gnu_cxx::tan_pil (
            long double __x ) [inline]
```

Return the reperiodized tangent function $\tan_\pi(x)$ for `long double` argument x.

See also

tan_pi for more details.

Definition at line 6063 of file specfun.h.

8.3.2.319 tanh_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::tanh_pi (
            _Tp __x ) [inline]
```

Return the reperiodized hyperbolic tangent function $\tanh_\pi(x)$ for real argument x.

The reperiodized hyperbolic tangent function is defined by:

$$\tanh_\pi(x) = \tanh(\pi x)$$

Template Parameters

_Tp	The floating-point type of the argument __x.

Parameters

__x	The argument

Definition at line 6122 of file specfun.h.

8.3.2.320 tanh_pif()

```
float __gnu_cxx::tanh_pif (
            float __x ) [inline]
```

Return the reperiodized hyperbolic tangent function $\tanh_\pi(x)$ for `float` argument x.

See also

tanh_pi for more details.

Definition at line 6095 of file specfun.h.

8.3.2.321 tanh_pil()

```
long double __gnu_cxx::tanh_pil (
            long double __x ) [inline]
```

Return the reperiodized hyperbolic tangent function $\tanh_\pi(x)$ for `long double` argument x.

See also

tanh_pi for more details.

Definition at line 6105 of file specfun.h.

8.3.2.322 tgamma() [1/3]

```
template<typename _Ta >
__gnu_cxx::__promote_fp_t<_Ta> __gnu_cxx::tgamma (
            _Ta __a ) [inline]
```

Return the gamma function for real argument.

Definition at line 2989 of file specfun.h.

Referenced by std::__detail::__tricomi_u_naive().

8.3.2.323 tgamma() [2/3]

```
template<typename _Ta >
std::complex<__gnu_cxx::__promote_fp_t<_Ta> > __gnu_cxx::tgamma (
            std::complex< _Ta > __a ) [inline]
```

Return the gamma function for complex argument.

Definition at line 3021 of file specfun.h.

8.3.2.324 tgamma() [3/3]

```
template<typename _Ta , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::tgamma (
            _Ta __a,
            _Tp __x ) [inline]
```

Return the upper incomplete gamma function $\Gamma(a,x)$. The (upper) incomplete gamma function is defined by

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

Definition at line 3058 of file specfun.h.

8.3.2.325 tgamma_lower()

```
template<typename _Ta , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::tgamma_lower (
            _Ta __a,
            _Tp __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a,x)$. The lower incomplete gamma function is defined by

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

Definition at line 3095 of file specfun.h.

8.3.2.326 tgamma_lowerf()

```
float __gnu_cxx::tgamma_lowerf (
            float __a,
            float __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a, x)$ for `float` argument.

See also

tgamma_lower for details.

Definition at line 3073 of file specfun.h.

8.3.2.327 tgamma_lowerl()

```
long double __gnu_cxx::tgamma_lowerl (
            long double __a,
            long double __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a, x)$ for `long double` argument.

See also

tgamma_lower for details.

Definition at line 3083 of file specfun.h.

8.3.2.328 tgammaf() [1/3]

```
float __gnu_cxx::tgammaf (
            float __a ) [inline]
```

Return the gamma function for `float` argument.

See also

lgamma for details.

Definition at line 2971 of file specfun.h.

8.3.2.329 tgammaf() [2/3]

```
std::complex<float> __gnu_cxx::tgammaf (
            std::complex< float > __a ) [inline]
```

Return the gamma function for `std::complex<float>` argument.

See also

lgamma for details.

Definition at line 3003 of file specfun.h.

8.3.2.330 tgammaf() [3/3]

```
float __gnu_cxx::tgammaf (
            float __a,
            float __x ) [inline]
```

Return the upper incomplete gamma function $\Gamma(a, x)$ for `float` argument.

See also

tgamma for details.

Definition at line 3036 of file specfun.h.

8.3.2.331 tgammal() [1/3]

```
long double __gnu_cxx::tgammal (
            long double __a ) [inline]
```

Return the gamma function for `long double` argument.

See also

lgamma for details.

Definition at line 2981 of file specfun.h.

8.3.2.332 tgammal() [2/3]

```
std::complex<long double> __gnu_cxx::tgammal (
            std::complex< long double > __a )  [inline]
```

Return the gamma function for `std::complex<long double>` argument.

See also

lgamma for details.

Definition at line 3013 of file specfun.h.

8.3.2.333 tgammal() [3/3]

```
long double __gnu_cxx::tgammal (
            long double __a,
            long double __x )  [inline]
```

Return the upper incomplete gamma function $\Gamma(a, x)$ for `long double` argument.

See also

tgamma for details.

Definition at line 3046 of file specfun.h.

8.3.2.334 theta_1()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_1 (
            _Tpnu __nu,
            _Tp __x )  [inline]
```

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period ν 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 5380 of file specfun.h.

8.3.2.335 theta_1f()

```
float __gnu_cxx::theta_1f (
            float __nu,
            float __x ) [inline]
```

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period ν and argument x.

See also

theta_1 for details.

Definition at line 5352 of file specfun.h.

8.3.2.336 theta_1l()

```
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 ν and argument x.

See also

theta_1 for details.

Definition at line 5362 of file specfun.h.

8.3.2.337 theta_2()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_2 (
            _Tpnu __nu,
            _Tp __x ) [inline]
```

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period ν 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 5423 of file specfun.h.

8.3.2.338 theta_2f()

```
float __gnu_cxx::theta_2f (
            float __nu,
            float __x )  [inline]
```

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period ν and argument x.

See also

theta_2 for details.

Definition at line 5395 of file specfun.h.

8.3.2.339 theta_2l()

```
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 ν and argument x.

See also

theta_2 for details.

Definition at line 5405 of file specfun.h.

8.3.2.340 theta_3()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_3 (
            _Tpnu __nu,
            _Tp __x )  [inline]
```

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period ν 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 5466 of file specfun.h.

8.3.2.341 theta_3f()

```
float __gnu_cxx::theta_3f (
            float __nu,
            float __x )  [inline]
```

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period ν and argument x.

See also

theta_3 for details.

Definition at line 5438 of file specfun.h.

8.3.2.342 theta_3l()

```
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 ν and argument x.

See also

theta_3 for details.

Definition at line 5448 of file specfun.h.

8.3.2.343 theta_4()

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_4 (
            _Tpnu __nu,
            _Tp __x )  [inline]
```

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period ν 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 5509 of file specfun.h.

8.3.2.344 theta_4f()

```
float __gnu_cxx::theta_4f (
            float __nu,
            float __x ) [inline]
```

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period ν and argument x.

See also

theta_4 for details.

Definition at line 5481 of file specfun.h.

8.3.2.345 theta_4l()

```
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 ν and argument x.

See also

theta_4 for details.

Definition at line 5491 of file specfun.h.

8.3.2.346 theta_c()

```
template<typename _Tpk , typename _Tp >
__gnu_cxx::__promote_fp_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

$$\theta_c(k,x) = \sqrt{\frac{\pi}{2kK(k)}}\theta_1\left(q(k),\frac{\pi x}{2K(k)}\right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_1(\nu,x)$ is the exponential theta-1 function.

See also

ellnome, std::comp_ellint_1, and theta_1 for details.

Parameters

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

Definition at line 5645 of file specfun.h.

8.3.2.347 theta_cf()

```
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 5613 of file specfun.h.

8.3.2.348 theta_cl()

```
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 5623 of file specfun.h.

8.3.2.349 theta_d()

```
template<typename _Tpk , typename _Tp >
__gnu_cxx::__promote_fp_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

$$\theta_d(k, x) = \sqrt{\frac{\pi}{2K(k)}}\theta_3\left(q(k), \frac{\pi x}{2K(k)}\right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_3(\nu, x)$ is the exponential theta-3 function.

See also

ellnome, std::comp_ellint_1, and theta_3 for details.

Parameters

_↩_k	The modulus $-1 <= k <= +1$
_↩_x	The argument

Definition at line 5692 of file specfun.h.

8.3.2.350 theta_df()

```
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 5660 of file specfun.h.

8.3.2.351 theta_dl()

```
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 5670 of file specfun.h.

8.3.2.352 theta_n()

```
template<typename _Tpk , typename _Tp >
__gnu_cxx::__promote_fp_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

$$\theta_n(k, x) = \sqrt{\frac{\pi}{2k' K(k)}} \theta_4 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_4(\nu, x)$ is the exponential theta-4 function.

See also

ellnome, std::comp_ellint_1, and theta_4 for details.

Parameters

_↩_k	The modulus $-1 <= k <= +1$
_↩_x	The argument

Definition at line 5739 of file specfun.h.

8.3.2.353 theta_nf()

```
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 5707 of file specfun.h.

8.3.2.354 theta_nl()

```
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 5717 of file specfun.h.

8.3.2.355 theta_s()

```
template<typename _Tpk , typename _Tp >
__gnu_cxx::__promote_fp_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

$$\theta_s(k,x) = \sqrt{\frac{\pi}{2kk'K(k)}}\theta_1\left(q(k), \frac{\pi x}{2K(k)}\right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_1(\nu,x)$ is the exponential theta-1 function.

See also

ellnome, std::comp_ellint_1, and theta_1 for details.

Parameters

_↩ _k	The modulus $-1 <= k <= +1$
_↩ _x	The argument

Definition at line 5598 of file specfun.h.

8.3.2.356 theta_sf()

```
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 5566 of file specfun.h.

8.3.2.357 theta_sl()

```
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 5576 of file specfun.h.

8.3.2.358 tricomi_u()

```
template<typename _Tpa , typename _Tpc , typename _Tp >
__gnu_cxx::__promote_fp_t<_Tpa, _Tpc, _Tp> __gnu_cxx::tricomi_u (
            _Tpa __a,
            _Tpc __c,
            _Tp __x ) [inline]
```

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of real numeratorial parameter a, denominatorial parameter c, and argument x.

The Tricomi confluent hypergeometric function is defined by

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

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

See also

conf_hyperg.

Parameters

_↩ _a	The numeratorial parameter
_↩ _c	The denominatorial parameter
_↩ _x	The argument

Definition at line 1473 of file specfun.h.

8.3.2.359 tricomi_uf()

```
float __gnu_cxx::tricomi_uf (
            float __a,
            float __c,
            float __x ) [inline]
```

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of `float` numeratorial parameter a, denominatorial parameter c, and argument x.

See also

tricomi_u for details.

Definition at line 1439 of file specfun.h.

8.3.2.360 tricomi_ul()

```
long double __gnu_cxx::tricomi_ul (
            long double __a,
            long double __c,
            long double __x ) [inline]
```

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of `long double` numeratorial parameter a, denominatorial parameter c, and argument x.

See also

tricomi_u for details.

Definition at line 1450 of file specfun.h.

8.3.2.361 weibull_cdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::weibull_cdf (
            _Ta __a,
            _Tb __b,
            _Tp __x ) [inline]
```

Return the Weibull cumulative probability density function.

The formula for the Weibull cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-(x/b)^a} \text{ for } x >= 0$$

Definition at line 6362 of file specfun.h.

8.3.2.362 weibull_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::weibull_pdf (
            _Ta __a,
            _Tb __b,
            _Tp __x ) [inline]
```

Return the Weibull probability density function.

The formula for the Weibull probability density function is

$$f(x|a,b) = \frac{a}{b}\left(\frac{x}{b}\right)^{a-1}\exp-\left(\frac{x}{b}\right)^{a} \text{ for } x >= 0$$

Definition at line 6346 of file specfun.h.

8.3.2.363 zernike()

```
template<typename _Trho , typename _Tphi >
__gnu_cxx::__promote_fp_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 2350 of file specfun.h.

8.3.2.364 zernikef()

```
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 2311 of file specfun.h.

8.3.2.365 zernikel()

```
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 2322 of file specfun.h.

Chapter 9

Namespace Documentation

9.1 __gnu_cxx Namespace Reference

Classes

- struct __airy_t
- struct __cyl_bessel_t
- struct __cyl_coulomb_t
- struct __cyl_hankel_t
- struct __cyl_mod_bessel_t
- struct __fock_airy_t
- struct __fp_is_integer_t
- struct __gamma_inc_t
- struct __gamma_temme_t

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

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

  and

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

  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.
- struct __hermite_he_t
- struct __hermite_t
- struct __jacobi_ellint_t
- struct __jacobi_t
- struct __laguerre_t
- struct __legendre_p_t
- struct __lgamma_t
- struct __pqgamma_t
- struct __quadrature_point_t
- struct __sincos_t
- struct __sph_bessel_t
- struct __sph_hankel_t
- struct __sph_mod_bessel_t

Functions

- template<typename _Tp >
  bool __fp_is_equal (_Tp __a, _Tp __b, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __fp_is_even_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __fp_is_half_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __fp_is_half_odd_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __fp_is_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __fp_is_odd_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  bool __fp_is_zero (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  _Tp __fp_max_abs (_Tp __a, _Tp __b)
- template<typename _Tp , typename _IntTp >
  _Tp __parity (_IntTp __k)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > airy_ai (_Tp __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > airy_ai (std::complex< _Tp > __x)
- float airy_aif (float __x)
- long double airy_ail (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > airy_bi (_Tp __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > airy_bi (std::complex< _Tp > __x)
- float airy_bif (float __x)
- long double airy_bil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > bernoulli (unsigned int __n)
- template<typename _Tp >
  _Tp bernoulli (unsigned int __n, _Tp __x)
- float bernoullif (unsigned int __n)
- long double bernoullil (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > binomial (unsigned int __n, unsigned int __k)

  Return the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

  $$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.

- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial probability mass function.
- float binomialf (unsigned int __n, unsigned int __k)
- long double binomiall (unsigned int __n, unsigned int __k)
- template<typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > bose_einstein (_Tps __s, _Tp __x)
- float bose_einsteinf (float __s, float __x)
- long double bose_einsteinl (long double __s, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > clausen (unsigned int __m, _Tp __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > clausen (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > clausen_cl (unsigned int __m, _Tp __x)
- float clausen_clf (unsigned int __m, float __x)
- long double clausen_cll (unsigned int __m, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > clausen_sl (unsigned int __m, _Tp __x)
- float clausen_slf (unsigned int __m, float __x)
- long double clausen_sll (unsigned int __m, long double __x)
- float clausenf (unsigned int __m, float __x)
- std::complex< float > clausenf (unsigned int __m, std::complex< float > __z)
- long double clausenl (unsigned int __m, long double __x)
- std::complex< long double > clausenl (unsigned int __m, std::complex< long double > __z)
- template<typename _Tk >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tx, _Ty > comp_ellint_rg (_Tx __x, _Ty __y)
- template<typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > 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_fp_t< _Tp > cos_pi (_Tp __x)
- float cos_pif (float __x)
- long double cos_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > cosh_pi (_Tp __x)
- float cosh_pif (float __x)
- long double cosh_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > coshint (_Tp __x)
- float coshintf (float __x)
- long double coshintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Tpnu, _Tp > > cyl_hankel_1 (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tpnu, _Tp > > cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > dawson (_Tp __x)
- float dawsonf (float __x)
- long double dawsonl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > debye (unsigned int __n, _Tp __x)

- float debyef (unsigned int __n, float __x)
- long double debyel (unsigned int __n, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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 >
  _Tp dirichlet_lambda (_Tp __s)
- float dirichlet_lambdaf (float __s)
- long double dirichlet_lambdal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > double_factorial (int __n)

  Return the double factorial $n!!$ *of the argument as a real number.*

  $$n!! = n(n-2)...(2), 0!! = 1$$

  for even n *and*

  $$n!! = n(n-2)...(1), (-1)!! = 1$$

  for odd n*.*
- float double_factorialf (int __n)
- long double double_factoriall (int __n)
- template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 >
  _Tp euler (unsigned int __n)

  This returns Euler number E_n.
- template<typename _Tp >
  _Tp eulerian_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp eulerian_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > expint (unsigned int __n, _Tp __x)
- float expintf (unsigned int __n, float __x)
- long double expintl (unsigned int __n, long double __x)
- template<typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > exponential_cdf (_Tlam __lambda, _Tp __x)

  Return the exponential cumulative probability density function.
- template<typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > exponential_pdf (_Tlam __lambda, _Tp __x)

  Return the exponential probability density function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > factorial (unsigned int __n)

  Return the factorial $n!$ of the argument as a real number.

  $$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float factorialf (unsigned int __n)
- long double factoriall (unsigned int __n)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > falling_factorial (_Tp __a, _Tnu __nu)

Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

In particular, $n^{\underline{n}} = n!$.

- float falling_factorialf (float __a, float __nu)
- long double falling_factoriall (long double __a, long double __nu)
- template< typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > fermi_dirac (_Tps __s, _Tp __x)
- float fermi_diracf (float __s, float __x)
- long double fermi_diracl (long double __s, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _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 >
  __gnu_cxx::__promote_fp_t< _Tp > fisher_f_pdf (_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 >
  __gnu_cxx::__promote_fp_t< _Tp > fresnel_c (_Tp __x)
- float fresnel_cf (float __x)
- long double fresnel_cl (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > fresnel_s (_Tp __x)
- float fresnel_sf (float __x)
- long double fresnel_sl (long double __x)
- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > gamma_cdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma cumulative propability distribution function.
- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > gamma_pdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma propability distribution function.
- template< typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > gamma_reciprocal (_Ta __a)
- float gamma_reciprocalf (float __a)
- long double gamma_reciprocall (long double __a)
- template< typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_fp_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 _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > harmonic (unsigned int __n)
- template< typename _Tk , typename _Tphi >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tpa, _Tpb, _Tpc, _Tp > 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_fp_t< _Ta, _Tb, _Tp > ibeta (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Talpha, _Tbeta, _Tp > jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > lbinomial (unsigned int __n, unsigned int __k)

  Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- float lbinomialf (unsigned int __n, unsigned int __k)
- long double lbinomiall (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > ldouble_factorial (int __n)

Return the logarithm of the double factorial $ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

for odd n.

- float ldouble_factorialf (int __n)
- long double ldouble_factoriall (int __n)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > legendre_q (unsigned int __l, _Tp __x)
- float legendre_qf (unsigned int __l, float __x)
- long double legendre_ql (unsigned int __l, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > lfactorial (unsigned int __n)

  Return the logarithm of the factorial $ln(n!)$ of the argument as a real number.

  $$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float lfactorialf (unsigned int __n)
- long double lfactoriall (unsigned int __n)
- template< typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > lfalling_factorial (_Tp __a, _Tnu __nu)

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

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

  In particular, $n^{\underline{n}} = n!$. Thus this function returns

  $$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

  Many notations exist for this function: $(a)_\nu$,

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

  , and others.
- float lfalling_factorialf (float __a, float __nu)
- long double lfalling_factoriall (long double __a, long double __nu)
- template< typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > lgamma (_Ta __a)
- template< typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > lgamma (std::complex< _Ta > __a)
- float lgammaf (float __a)
- std::complex< float > lgammaf (std::complex< float > __a)
- long double lgammal (long double __a)
- std::complex< long double > lgammal (std::complex< long double > __a)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > logint (_Tp __x)
- float logintf (float __x)
- long double logintl (long double __x)
- template< typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > logistic_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic cumulative distribution function.

- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > logistic_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal cumulative probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal probability density function.
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > lrising_factorial (_Tp __a, _Tnu __nu)

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

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

  Thus this function returns

  $$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[a^{\overline{0}}] = 0$$

  Many notations exist for this function: $(a)_\nu$ (especially in the literature of special functions),

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

  , and others.
- float lrising_factorialf (float __a, float __nu)
- long double lrising_factoriall (long double __a, long double __nu)
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal cumulative probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal probability density function.
- template<typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_fp_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 _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > pgamma (_Ta __a, _Tp __x)
- float pgammaf (float __a, float __x)
- long double pgammal (long double __a, long double __x)
- template<typename _Tp , typename _Wp >
  __gnu_cxx::__promote_fp_t< _Tp, _Wp > polylog (_Tp __s, _Wp __w)
- template<typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > psi (_Tp __x)
- float psif (float __x)
- long double psil (long double __x)

- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > qgamma (_Ta __a, _Tp __x)
- float qgammaf (float __a, float __x)
- long double qgammal (long double __a, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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 , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > rising_factorial (_Tp __a, _Tnu __nu)

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

$$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

  Many notations exist for this function: $(a)_\nu$*, (especially in the literature of special functions),*

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

  , and others.
- float rising_factorialf (float __a, float __nu)
- long double rising_factoriall (long double __a, long double __nu)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > sin_pi (_Tp __x)
- float sin_pif (float __x)
- long double sin_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > sinc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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)
- __gnu_cxx::__sincos_t< double > sincos (double __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > sincos (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > sincos_pi (_Tp __x)
- __gnu_cxx::__sincos_t< float > sincos_pif (float __x)
- __gnu_cxx::__sincos_t< long double > sincos_pil (long double __x)
- __gnu_cxx::__sincos_t< float > sincosf (float __x)
- __gnu_cxx::__sincos_t< long double > sincosl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > sinh_pi (_Tp __x)
- float sinh_pif (float __x)
- long double sinh_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > sinhc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Tp > sinhint (_Tp __x)
- float sinhintf (float __x)
- long double sinhintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > sinint (_Tp __x)
- float sinintf (float __x)
- long double sinintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > > sph_hankel_2 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_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 _Tp >
  _Tp stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > student_t_cdf (_Tt __t, unsigned int __nu)

  Return the Students T probability function.
- template<typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > student_t_pdf (_Tt __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > tan_pi (_Tp __x)

- float tan_pif (float __x)
- long double tan_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > tanh_pi (_Tp __x)
- float tanh_pif (float __x)
- long double tanh_pil (long double __x)
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > tgamma (_Ta __a)
- template<typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > tgamma (std::complex< _Ta > __a)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > tgamma (_Ta __a, _Tp __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > tgamma_lower (_Ta __a, _Tp __x)
- float tgamma_lowerf (float __a, float __x)
- long double tgamma_lowerl (long double __a, long double __x)
- float tgammaf (float __a)
- std::complex< float > tgammaf (std::complex< float > __a)
- float tgammaf (float __a, float __x)
- long double tgammal (long double __a)
- std::complex< long double > tgammal (std::complex< long double > __a)
- long double tgammal (long double __a, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > tricomi_u (_Tpa __a, _Tpc __c, _Tp __x)
- float tricomi_uf (float __a, float __c, float __x)
- long double tricomi_ul (long double __a, long double __c, long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > weibull_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull cumulative probability density function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > weibull_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull probability density function.
- template<typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_fp_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)

9.1.1 Function Documentation

9.1.1.1 __fp_is_equal()

```
template<typename _Tp >
bool __gnu_cxx::__fp_is_equal (
            _Tp __a,
            _Tp __b,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably compare two floating point numbers.

Parameters

__a	The left hand side
__b	The right hand side
__mul	The multiplier for numeric epsilon for comparison

Returns

`true` if a and b are equal to zero or differ only by $max(a, b) * mul * epsilon$

Definition at line 81 of file math_util.h.

References __fp_max_abs().

Referenced by __fp_is_half_integer(), __fp_is_half_odd_integer(), __fp_is_integer(), std::__detail::__polylog(), std::__detail::__polylog_exp_neg(), std::__detail::__polylog_exp_neg_int(), std::__detail::__polylog_exp_pos_int(), and std::__detail::__polylog_exp_pos_real().

9.1.1.2 __fp_is_even_integer()

```
template<typename _Tp >
__fp_is_integer_t __gnu_cxx::__fp_is_even_integer (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably detect if a floating point number is an even integer.

Parameters

__a	The floating point number
__mul	The multiplier of machine epsilon for the tolerance

Returns

`true` if a is an even integer within mul $*$ epsilon.

Definition at line 217 of file math_util.h.

References __fp_is_integer().

Referenced by std::__detail::__riemann_zeta_glob().

9.1.1.3 __fp_is_half_integer()

```
template<typename _Tp >
__fp_is_integer_t __gnu_cxx::__fp_is_half_integer (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably detect if a floating point number is a half-integer.

Parameters

__a	The floating point number
__mul	The multiplier of machine epsilon for the tolerance

Returns

`true` if 2a is an integer within mul $*$ epsilon and the returned value is half the integer, int(a) / 2.

Definition at line 172 of file math_util.h.

References __fp_is_equal().

9.1.1.4 __fp_is_half_odd_integer()

```
template<typename _Tp >
__fp_is_integer_t __gnu_cxx::__fp_is_half_odd_integer (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably detect if a floating point number is a half-odd-integer.

Parameters

__a	The floating point number
__mul	The multiplier of machine epsilon for the tolerance

Returns

`true` if 2a is an odd integer within mul $*$ epsilon and the returned value is int(a - 1) / 2.

Definition at line 195 of file math_util.h.

References __fp_is_equal().

Referenced by std::__detail::__psi().

9.1.1.5 __fp_is_integer()

```
template<typename _Tp >
__fp_is_integer_t __gnu_cxx::__fp_is_integer (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably detect if a floating point number is an integer.

Parameters

__a	The floating point number
__mul	The multiplier of machine epsilon for the tolerance

Returns

`true` if a is an integer within mul $*$ epsilon.

Definition at line 150 of file math_util.h.

References __fp_is_equal().

Referenced by std::__detail::__dirichlet_eta(), std::__detail::__falling_factorial(), __fp_is_even_integer(), __fp_is_odd_integer(), std::__detail::__gamma(), std::__detail::__gamma_reciprocal(), std::__detail::__gamma_series(), std::__detail::__log_falling_factorial(), std::__detail::__log_gamma(), std::__detail::__pgamma(), std::__detail::__polylog(), std::__detail::__polylog_exp(), std::__detail::__psi(), std::__detail::__qgamma(), std::__detail::__riemann_zeta(), std::__detail::__riemann_zeta_m_1(), std::__detail::__tgamma(), std::__detail::__tgamma_lower(), and std::__detail::__tricomi_u_naive().

9.1.1.6 __fp_is_odd_integer()

```
template<typename _Tp >
__fp_is_integer_t __gnu_cxx::__fp_is_odd_integer (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably detect if a floating point number is an odd integer.

Parameters

__a	The floating point number
__mul	The multiplier of machine epsilon for the tolerance

Returns

`true` if a is an odd integer within mul $*$ epsilon.

Definition at line 237 of file math_util.h.

References __fp_is_integer().

9.1.1.7 __fp_is_zero()

```
template<typename _Tp >
bool __gnu_cxx::__fp_is_zero (
            _Tp __a,
            _Tp __mul = _Tp{1} )  [inline]
```

A function to reliably compare a floating point number with zero.

Parameters

__a	The floating point number
__mul	The multiplier for numeric epsilon for comparison

Returns

`true` if a and b are equal to zero or differ only by $max(a, b) * mul * epsilon$

Definition at line 106 of file math_util.h.

Referenced by std::__detail::__polylog(), std::__detail::__polylog_exp_neg(), std::__detail::__polylog_exp_neg_int(), std::__detail::__polylog_exp_pos_int(), std::__detail::__polylog_exp_pos_real(), and std::__detail::__theta_1().

9.1.1.8 __fp_max_abs()

```
template<typename _Tp >
_Tp __gnu_cxx::__fp_max_abs (
            _Tp __a,
            _Tp __b )  [inline]
```

A function to return the max of the absolute values of two numbers ... so we won't include everything.

Parameters

__a	The left hand side
__b	The right hand side

Definition at line 58 of file math_util.h.

Referenced by __fp_is_equal().

9.1.1.9 __parity()

```
template<typename _Tp , typename _IntTp >
_Tp __gnu_cxx::__parity (
            _IntTp __k )  [inline]
```

Return -1 if the integer argument is odd and +1 if it is even.

Definition at line 47 of file math_util.h.

Referenced by std::__detail::__stirling_1_series().

9.2 std Namespace Reference

Namespaces

- __detail

Functions

- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tpa, _Tpb > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp, _Tpn > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpn, _Tpp > 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_fp_t< _Tp > expint (_Tp __x)
- float expintf (float __x)
- long double expintl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > riemann_zeta (_Tp __s)
- float riemann_zetaf (float __s)
- long double riemann_zetal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > 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)

9.3 std::__detail Namespace Reference

Classes

- struct __gamma_lanczos_data

- struct __gamma_lanczos_data< double >
- struct __gamma_lanczos_data< float >
- struct __gamma_lanczos_data< long double >
- struct __gamma_spouge_data
- struct __gamma_spouge_data< double >
- struct __gamma_spouge_data< float >
- struct __gamma_spouge_data< long double >
- struct __jacobi_theta_0_t
- class _Airy
- class _Airy_asymp
- struct _Airy_asymp_data
- struct _Airy_asymp_data< double >
- struct _Airy_asymp_data< float >
- struct _Airy_asymp_data< long double >
- class _Airy_asymp_series
- struct _Airy_default_radii
- struct _Airy_default_radii< double >
- struct _Airy_default_radii< float >
- struct _Airy_default_radii< long double >
- class _Airy_series
- struct _AiryAuxilliaryState
- struct _AiryState
- class _AsympTerminator
- struct _Factorial_table
- class _Terminator

Functions

- template< typename _Tp >
  __gnu_cxx::__airy_t< _Tp, _Tp > __airy (_Tp __z)

  Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi(x)$ respectively.
- 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 >
  std::complex< _Tp > __airy_bi (std::complex< _Tp > __z)

  Return the complex Airy Bi function.
- 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 (unsigned int __n)

  This returns Bernoulli number B_n.
- template<typename _Tp >
  _Tp __bernoulli (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __bernoulli_2n (unsigned int __n)

  This returns Bernoulli number B_2n *at even integer arguments* $2n$.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)

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

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

  .
- 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_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 __binomial (unsigned int __n, unsigned int __k)

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

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

  The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  _Tp __binomial (_Tp __nu, unsigned int __k)

  Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- template<typename _Tp >
  _Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

Return the binomial cumulative distribution function.

- template<typename _Tp >
  _Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)

  Return the complementary binomial cumulative distribution function.
- template<typename _Tp >
  _Tp __binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial probability mass function.
- template<typename _Sp , typename _Tp >
  _Tp __bose_einstein (_Sp __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 >
  _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 >
  _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 > __z)
- template<typename _Tp >
  std::complex< _Tp > __clamp_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __clausen (unsigned int __m, _Tp __x)
- template<typename _Tp >
  _Tp __clausen_cl (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __clausen_cl (unsigned int __m, _Tp __x)
- template<typename _Tp >
  _Tp __clausen_sl (unsigned int __m, std::complex< _Tp > __z)

- template<typename _Tp >
  _Tp __clausen_sl (unsigned int __m, _Tp __x)
- 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 $_1F_1(a;c;x) = M(a,c,x)$.
- 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 __cos_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __cos_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __cosh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __cosh_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __coshint (const _Tp __x)

  Return the hyperbolic cosine integral $Chi(x)$.
- template<typename _Tp >
  std::pair< _Tp, _Tp > __coulomb_CF1 (unsigned int __l, _Tp __eta, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __coulomb_CF2 (unsigned int __l, _Tp __eta, _Tp __x)
- template<typename _Tp >
  std::pair< _Tp, _Tp > __coulomb_f_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _F_l_max, _Tp _Fp_l_max)

- template<typename _Tp >
  std::pair< _Tp, _Tp > __coulomb_g_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _G_l_min, _Tp _Gp_l_min)
- template<typename _Tp >
  _Tp __coulomb_norm (unsigned int __l, _Tp __eta)
- 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 >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik (_Tp __nu, _Tp __x)

  Return the modified cylindrical Bessel functions and their derivatives of order ν *by various means.*
- template<typename _Tp >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik_steed (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn (_Tp __nu, _Tp __x)

  Return the cylindrical Bessel functions and their derivatives of order ν *by various means.*
- template<typename _Tp >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn_asymp (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, std::complex< _Tp > > __cyl_bessel_jn_neg_arg (_Tp __nu, _Tp __x)

  Return the cylindrical Bessel functions and their derivatives of order ν *and argument* $x < 0$.
- template<typename _Tp >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn_steed (_Tp __nu, _Tp __x)

  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 >
_Tp __debye (unsigned int __n, _Tp __x)
- 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 > __s)
- template<typename _Tp >
_Tp __dirichlet_beta (_Tp __s)
- template<typename _Tp >
std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __s)
- template<typename _Tp >
_Tp __dirichlet_eta (_Tp __s)
- template<typename _Tp >
_Tp __dirichlet_lambda (_Tp __s)
- 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 __euler (unsigned int __n)

  This returns Euler number E_n.
- template<typename _Tp >
  _Tp __euler (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp __euler_series (unsigned int __n)
- template<typename _Tp >
  _Tp __eulerian_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __eulerian_1_recur (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __eulerian_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __eulerian_2_recur (unsigned int __n, unsigned int __m)
- 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_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_asymp (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large argument.
- 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_large_n (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large order.
- 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 __exponential_cdf (_Tp __lambda, _Tp __x)

  Return the exponential cumulative probability density function.
- template<typename _Tp >
  _Tp __exponential_cdfc (_Tp __lambda, _Tp __x)

  Return the complement of the exponential cumulative probability density function.
- template<typename _Tp >
  _Tp __exponential_pdf (_Tp __lambda, _Tp __x)

  Return the exponential probability density function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)

Return the factorial of the integer n.

- template<typename _Tp >
  _Tp __falling_factorial (_Tp __a, int __n)

  Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

  In particular, $n^{\underline{n}} = n!$.

- template<typename _Tp >
  _Tp __falling_factorial (_Tp __a, _Tp __nu)

  Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν. The falling factorial function is defined by

  $$a^{\underline{\nu}} = \Gamma(a+1)/\Gamma(a-\nu+1)$$

  .

- template<typename _Sp , typename _Tp >
  _Tp __fermi_dirac (_Sp __s, _Tp __x)
- template<typename _Tp >
  _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 >
  _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 >
  _Tp __fisher_f_pdf (_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 >
  __gnu_cxx::__fock_airy_t< _Tp, std::complex< _Tp > > __fock_airy (_Tp __x)

  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 >
  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 __a)

  Return the gamma function $\Gamma(a)$. The gamma function is defined by:

  $$\Gamma(a) = \int_0^\infty e^{-t}t^{a-1}dt (a > 0)$$

  .

- template<typename _Tp >
  std::pair< _Tp, _Tp > __gamma (_Tp __a, _Tp __x)

  Return the incomplete gamma functions.
- template<typename _Tp >
  _Tp __gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma cumulative propability distribution function.
- template<typename _Tp >
  _Tp __gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma complementary cumulative propability distribution function.
- template<typename _Tp >
  std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)

  Return the incomplete gamma function by continued fraction.
- template<typename _Tp >
  _Tp __gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma propability distribution function.
- template<typename _Tp >
  _Tp __gamma_reciprocal (_Tp __a)
- template<typename _Tp >
  _Tp __gamma_reciprocal_series (_Tp __a)
- template<typename _Tp >
  std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)

  Return the incomplete gamma function by series summation.

  $$\gamma(a,x) = x^a e^{-z} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

  .
- template<typename _Tp >
  __gnu_cxx::__gamma_temme_t< _Tp > __gamma_temme (_Tp __mu)

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

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

  and

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

  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 __gauss (_Tp __x)
- template<typename _Tp >
  _Tp __gegenbauer_poly (unsigned int __n, _Tp __alpha1, _Tp __x)
- template<typename _Tp >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __gegenbauer_zeros (unsigned int __n, _Tp __↩ alpha1)
- template<typename _Tp >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel↩ _debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn)

- 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 >
__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  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 >
__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  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 __harmonic_number (unsigned int __n)

- template<typename _Tp >
_Tp __hermite (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
_Tp __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 >
__gnu_cxx::__hermite_t< _Tp > __hermite_recur (unsigned int __n, _Tp __x)

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

- template<typename _Tp >
std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __hermite_zeros (unsigned int __n, _Tp __proto=_Tp{})

- 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 >
  _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 > __hurwitz_zeta_polylog (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
  std::complex< _Tp > __hydrogen (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _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 >
  _Tp __ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__jacobi_ellint_t< _Tp > __jacobi_ellint (_Tp __k, _Tp __u)
- template<typename _Tp >
  __gnu_cxx::__jacobi_t< _Tp > __jacobi_recur (unsigned int __n, _Tp __alpha1, _Tp __beta1, _Tp __x)
- template<typename _Tp >
  __jacobi_theta_0_t< _Tp > __jacobi_theta_0 (_Tp __q)
- template<typename _Tp >
  std::complex< _Tp > __jacobi_theta_1 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template<typename _Tp >
  _Tp __jacobi_theta_1 (_Tp __q, const _Tp __x)
- template<typename _Tp >
  _Tp __jacobi_theta_1_sum (_Tp __q, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __jacobi_theta_2 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template<typename _Tp >
  _Tp __jacobi_theta_2 (_Tp __q, const _Tp __x)
- template<typename _Tp >
  _Tp __jacobi_theta_2_prod0 (_Tp __q)
- template<typename _Tp >
  _Tp __jacobi_theta_2_sum (_Tp __q, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __jacobi_theta_3 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)

- template<typename _Tp >
  _Tp __jacobi_theta_3 (_Tp __q, const _Tp __x)
- template<typename _Tp >
  _Tp __jacobi_theta_3_prod0 (_Tp __q)
- template<typename _Tp >
  _Tp __jacobi_theta_3_sum (_Tp __q, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __jacobi_theta_4 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template<typename _Tp >
  _Tp __jacobi_theta_4 (_Tp __q, const _Tp __x)
- template<typename _Tp >
  _Tp __jacobi_theta_4_prod0 (_Tp __q)
- template<typename _Tp >
  _Tp __jacobi_theta_4_sum (_Tp __q, _Tp __x)
- template<typename _Tp >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __jacobi_zeros (unsigned int __n, _Tp __alpha1, _Tp __beta1)
- template<typename _Tp >
  _Tp __jacobi_zeta (_Tp __k, _Tp __phi)
- template<typename _Tpa , typename _Tp >
  _Tp __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 _Tp >
  _Tp __laguerre (unsigned int __n, _Tp __x)

  This routine returns the Laguerre polynomial of order n: $L_n(x)$.
- template<typename _Tpa , typename _Tp >
  _Tp __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 __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 >
  __gnu_cxx::__laguerre_t< _Tpa, _Tp > __laguerre_recur (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 >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __laguerre_zeros (unsigned int __n, _Tp __alpha1)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __lanczos_binet1p (_Tp __z)

  Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $log(\Gamma^*(z))$ defined by

  $$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

  or

  $$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

  where $\Gamma(z)$ is the gamma function.
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __lanczos_log_gamma1p (_Tp __z)

  Return the logarithm of the gamma function $log(\Gamma(1+z))$ by the Lanczos method.
- template<typename _Tp >
  __gnu_cxx::__legendre_p_t< _Tp > __legendre_p (unsigned int __l, _Tp __x)

Return the Legendre polynomial by upward recursion on order l.

- 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 >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __legendre_zeros (unsigned int __l, _Tp proto=_Tp{})
- template<typename _Tp >
  _Tp __log_binomial (unsigned int __n, unsigned int __k)

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

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

  The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  _Tp __log_binomial (_Tp __nu, unsigned int __k)

  Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- template<typename _Tp >
  _Tp __log_binomial_sign (_Tp __nu, unsigned int __k)

  Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- template<typename _Tp >
  std::complex< _Tp > __log_binomial_sign (std::complex< _Tp > __nu, unsigned int __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_falling_factorial (_Tp __a, _Tp __nu)

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

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

In particular, $n^{\underline{n}} = n!$*. Thus this function returns*

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

Many notations exist for this function:

$$(a)_\nu$$

,

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

, and others.

- template<typename _Tp >
  _Tp __log_gamma (_Tp __a)

  Return $log(|\Gamma(a)|)$*. This will return values even for* $a < 0$*. To recover the sign of* $\Gamma(a)$ *for any argument use __log_gamma_sign.*
- template<typename _Tp >
  std::complex< _Tp > __log_gamma (std::complex< _Tp > __a)

  Return $log(\Gamma(a))$ *for complex argument.*
- 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 >
  _Tp __log_gamma_sign (_Tp __a)

  Return the sign of $\Gamma(x)$*. At nonpositive integers zero is returned indicating* $\Gamma(x)$ *is undefined.*
- template<typename _Tp >
  std::complex< _Tp > __log_gamma_sign (std::complex< _Tp > __a)
- template<typename _Tp >
  _Tp __log_rising_factorial (_Tp __a, _Tp __nu)

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

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

  Thus this function returns

  $$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[(a)_0] = 0$$

  Many notations exist for this function:

  $$(a)_\nu$$

  (especially in the literature of special functions),

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

  , and others.
- template<typename _Tp >
  _Tp __log_stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __log_stirling_1_sign (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __log_stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __logint (const _Tp __x)

  Return the logarithmic integral $li(x)$*.*

- template<typename _Tp >
  _Tp __logistic_cdf (_Tp __a, _Tp __b, _Tp __x)

  Return the logistic cumulative distribution function.
- template<typename _Tp >
  _Tp __logistic_pdf (_Tp __a, _Tp __b, _Tp __x)

  Return the logistic probability density function.
- template<typename _Tp >
  _Tp __lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the lognormal cumulative probability density function.
- template<typename _Tp >
  _Tp __lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)

  Return the lognormal probability density function.
- template<typename _Tp >
  _Tp __normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the normal cumulative probability density function.
- template<typename _Tp >
  _Tp __normal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the normal probability density function.
- template<typename _Tp >
  _Tp __owens_t (_Tp __h, _Tp __a)
- template<typename _Tp >
  _Tp __pgamma (_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 >
  std::complex< _Tp > __polar_pi (_Tp __rho, _Tp __phi_pi)
- template<typename _Tp >
  std::complex< _Tp > __polar_pi (_Tp __rho, const std::complex< _Tp > &__phi_pi)
- 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_fp_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_neg (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg (int __n, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg_int (int __s, std::complex< _Tp > __w)

- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg_int (int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg_real (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_neg_real (_Tp __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_pos_int (unsigned int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos_int (unsigned int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos_real (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > __polylog_exp_pos_real (_Tp __s, _Tp __w)
- template<typename _PowTp , typename _Tp >
  _Tp __polylog_exp_sum (_PowTp __s, _Tp __w)
- template<typename _Tp >
  __gnu_cxx::__hermite_he_t< _Tp > __prob_hermite_recursion (unsigned int __n, _Tp __x)

  This routine returns the Probabilists Hermite polynomial of order n: $He_n(x)$ by recursion on n.
- template<typename _Tp >
  _Tp __psi (unsigned int __n)

  Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

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

  The digamma series for integral argument is given by:

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

  The latter sum is called the harmonic number, H_n.
- 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 __qgamma (_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 >
  _Tp __rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)

  Return the Rice probability density function.

- template<typename _Tp >
  _Tp __riemann_zeta (_Tp __s)

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

- 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)

- 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_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_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 >
  _Tp __rising_factorial (_Tp __a, int __n)

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

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

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

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

, and others.

- template<typename _Tp >
  _Tp __rising_factorial (_Tp __a, _Tp __nu)

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

  $$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

  Many notations exist for this function:

  $$(a)_\nu$$

  , (especially in the literature of special functions),

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

  , and others.
- template<typename _Tp >
  _Tp __sin_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __sin_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __sinc (_Tp __x)

  Return the sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __sinc_pi (_Tp __x)

  Return the reperiodized sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__sincos_t< _Tp > __sincos (_Tp __x)
- template<>
  __gnu_cxx::__sincos_t< float > __sincos (float __x)
- template<>
  __gnu_cxx::__sincos_t< double > __sincos (double __x)
- template<>
  __gnu_cxx::__sincos_t< long double > __sincos (long double __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< _Tp > __sincos_pi (_Tp __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 >
  _Tp __sinh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __sinh_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __sinhc (_Tp __x)

  Return the hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __sinhc_pi (_Tp __x)

  Return the reperiodized hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  _Tp __sinhint (const _Tp __x)

  Return the hyperbolic sine integral $Shi(x)$.
- 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 >
  __gnu_cxx::__sph_mod_bessel_t< unsigned int, _Tp, _Tp > __sph_bessel_ik (unsigned int __n, _Tp __x)

  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 >
  __gnu_cxx::__sph_bessel_t< unsigned int, _Tp, _Tp > __sph_bessel_jn (unsigned int __n, _Tp __x)

  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 >
  __gnu_cxx::__sph_bessel_t< unsigned int, _Tp, std::complex< _Tp > > __sph_bessel_jn_neg_arg (unsigned int __n, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__sph_hankel_t< unsigned int, std::complex< _Tp >, std::complex< _Tp > > __sph_hankel (unsigned int __n, std::complex< _Tp > __z)

  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 $h^{(1)}_n(x)$.
- 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 $h^{(2)}_n(x)$.
- 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 __spouge_binet1p (_Tp __z)

  Return the Binet function $J(1+z)$ *by the Spouge method. The Binet function is the log of the scaled Gamma function* $log(\Gamma^*(z))$ *defined by*

  $$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

  or

  $$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

  where $\Gamma(z)$ *is the gamma function.*
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp __spouge_log_gamma1p (_Tp __z)

  Return the logarithm of the gamma function $log(\Gamma(1+z))$ *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 __stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __stirling_1_recur (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __stirling_1_series (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __stirling_2_recur (unsigned int __n, unsigned int __m)

- template<typename _Tp >
  _Tp __stirling_2_series (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __student_t_cdf (_Tp __t, unsigned int __nu)

  Return the Students T probability function.
- template<typename _Tp >
  _Tp __student_t_cdfc (_Tp __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template<typename _Tp >
  _Tp __student_t_pdf (_Tp __t, unsigned int __nu)

  Return the Students T probability density.
- template<typename _Tp >
  _Tp __tan_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __tan_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __tanh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > __tanh_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp __tgamma (_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 __tgamma_lower (_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 __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 >
  _Tp __tricomi_u (_Tp __a, _Tp __c, _Tp __x)

  Return the Tricomi confluent hypergeometric function

$$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1;2-c;x)$$

  .
- template<typename _Tp >
  _Tp __tricomi_u_naive (_Tp __a, _Tp __c, _Tp __x)

  Return the Tricomi confluent hypergeometric function

$$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1;2-c;x)$$

  .
- template<typename _Tp >
  _Tp __weibull_cdf (_Tp __a, _Tp __b, _Tp __x)

  Return the Weibull cumulative probability density function.
- template<typename _Tp >
  _Tp __weibull_pdf (_Tp __a, _Tp __b, _Tp __x)

  Return the Weibull probability density function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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)

Variables

- template<typename _Tp >
  constexpr int __max_FGH = _Airy_series<_Tp>::_N_FGH
- template<>
  constexpr int __max_FGH< double > = 79
- template<>
  constexpr int __max_FGH< float > = 15
- 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 unsigned long long _S_harmonic_denom [_S_num_harmonic_numer]
- constexpr unsigned long long _S_harmonic_numer [_S_num_harmonic_numer]
- 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
- constexpr unsigned long long _S_num_harmonic_numer = 29
- 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 = 121
- constexpr long double _S_zetam1 [_S_num_zetam1]

9.3.1 Function Documentation

9.3.1.1 __airy()

```
template<typename _Tp >
__gnu_cxx::__airy_t<_Tp, _Tp> std::__detail::__airy (
            _Tp __z )
```

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.

Returns

A struct containing the Airy functions of the first and second kinds and their derivatives.

Definition at line 466 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik(), and __cyl_bessel_jn().

Referenced by __airy_ai(), __airy_bi(), __fock_airy(), and __hermite_asymp().

9.3.1.2 __airy_ai()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__airy_ai (
            std::complex< _Tp > __z )
```

Return the complex Airy Ai function.

Definition at line 2629 of file sf_airy.tcc.

References __airy().

9.3.1.3 __airy_arg()

```
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*	$e^{+i2\pi/3}\nu^{2/3}\zeta$
out	*__argm*	$e^{-i2\pi/3}\nu^{2/3}\zeta$

Exceptions

std::runtime_error	if unable to compute Airy function arguments

Definition at line 215 of file sf_hankel.tcc.

Referenced by __hankel_uniform_outer().

9.3.1.4 __airy_bi()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__airy_bi (
            std::complex< _Tp > __z )
```

Return the complex Airy Bi function.

Definition at line 2641 of file sf_airy.tcc.

References __airy().

9.3.1.5 __assoc_laguerre()

```
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 366 of file sf_laguerre.tcc.

Referenced by __hydrogen().

9.3.1.6 __assoc_legendre_p()

```
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.

Definition at line 195 of file sf_legendre.tcc.

References __legendre_p().

9.3.1.7 __bernoulli() [1/2]

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (
            unsigned 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 128 of file sf_bernoulli.tcc.

Referenced by __euler(), and __gnu_cxx::bernoulli().

9.3.1.8 __bernoulli() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__bernoulli (
            unsigned int __n,
            _Tp __x )
```

Return the Bernoulli polynomial $B_n(x)$ of order n at argument x.

The values at 0 and 1 are equal to the corresponding Bernoulli number:

$$B_n(0) = B_n(1) = B_n$$

The derivative is proportional to the previous polynomial:

$$B_n'(x) = n * B_{n-1}(x)$$

The series expansion is:

$$B_n(x) = \sum_{k=0}^{n} B_k binomnk x^{n-k}$$

A useful argument promotion is:

$$B_n(x+1) - B_n(x) = n * x^{n-1}$$

Definition at line 168 of file sf_bernoulli.tcc.

References __binomial().

9.3.1.9 __bernoulli_2n()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (
            unsigned int __n )
```

This returns Bernoulli number B_2n at even integer arguments $2n$.

Parameters

__n	the half-order n of the Bernoulli number.

Returns

The Bernoulli number of order 2n.

Definition at line 140 of file sf_bernoulli.tcc.

9.3.1.10 __bernoulli_series()

```
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.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

.

Note that

$$\zeta(2n) - 1 = (-1)^{n+1} \frac{(2\pi)^{2n}}{(2n)!} B_{2n} - 2$$

are small and rapidly decreasing finctions of n.

Parameters

_↩ _n	the order n of the Bernoulli number.

Returns

The Bernoulli number of order n.

Definition at line 65 of file sf_bernoulli.tcc.

9.3.1.11 __beta()

```
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 215 of file sf_beta.tcc.

References __beta_gamma(), and __beta_lgamma().

Referenced by __fisher_f_pdf(), __gnu_cxx::gamma_cdf(), __gnu_cxx::gamma_pdf(), __gnu_cxx::jacobi(), __gnu_↩ cxx::jacobif(), __gnu_cxx::jacobil(), and std::__detail::_Airy$<$ _Tp $>$::operator()().

9.3.1.12 __beta_gamma()

```
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 77 of file sf_beta.tcc.

References __gamma().

Referenced by __beta().

9.3.1.13 __beta_inc()

```
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 311 of file sf_beta.tcc.

References __ibeta_cont_frac(), __log_gamma(), and __log_gamma_sign().

Referenced by __binomial_cdf(), __binomial_cdfc(), __fisher_f_cdf(), __fisher_f_cdfc(), __student_t_cdf(), and __student_t_cdfc().

9.3.1.14 __beta_lgamma()

```
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 125 of file sf_beta.tcc.

References __log_gamma(), and __log_gamma_sign().

Referenced by __beta().

9.3.1.15 __beta_product()

```
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:

$$B(a, b) = \frac{a+b}{ab}\prod_{k=1}^{\infty}\frac{1+(a+b)/k}{(1+a/k)(1+b/k)} = \frac{a+b}{ab}\prod_{k=1}^{\infty}\left[1 - \frac{ab}{(a+k)(b+k)}\right]$$

Parameters

_↩ _a	The first argument of the beta function.
_↩ _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 179 of file sf_beta.tcc.

9.3.1.16 __binomial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__binomial (
            unsigned int __n,
            unsigned int __k )
```

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

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

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

.

Parameters

_↩ _n	The first argument of the binomial coefficient.
_↩ _k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 2515 of file sf_gamma.tcc.

References std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __bernoulli().

9.3.1.17 __binomial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__binomial (
            _Tp __nu,
            unsigned int __k )
```

Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^\infty \binom{\nu}{k} t^k$$

.

Parameters

__nu	The real first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 2575 of file sf_gamma.tcc.

References __gamma(), __log_binomial(), __log_binomial_sign(), and std::__detail::_Factorial_table< _Tp >::__n.

9.3.1.18 __binomial_cdf()

```
template<typename _Tp >
_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:

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

Parameters

_↩ _p	
_↩ _n	
_↩ _k	

Definition at line 614 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.19 __binomial_cdfc()

```
template<typename _Tp >
_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:

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

Parameters

__p	
__n	
__k	

Definition at line 644 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.20 __binomial_pdf()

```
template<typename _Tp >
_Tp std::__detail::__binomial_pdf (
            _Tp __p,
            unsigned int __n,
            unsigned int __k )
```

Return the binomial probability mass function.

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

$$f(k|n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Parameters

_↩ _p	
_↩ _n	
_↩ _k	

Definition at line 578 of file sf_distributions.tcc.

9.3.1.21 __bose_einstein()

```
template<typename _Sp , typename _Tp >
_Tp std::__detail::__bose_einstein (
            _Sp __s,
            _Tp __x )
```

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

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12.16

$$G_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x} - 1} dt = Li_{s+1}(e^x)$$

Parameters

_↩ _s	The order s >= 0.
_↩ _x	The real argument.

Returns

The real Bose-Einstein integral G_s(x),

Definition at line 1462 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.22 __chebyshev_recur()

```
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 59 of file sf_chebyshev.tcc.

Referenced by __chebyshev_t(), __chebyshev_u(), __chebyshev_v(), and __chebyshev_w().

9.3.1.23 __chebyshev_t()

```
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 87 of file sf_chebyshev.tcc.

References __chebyshev_recur().

9.3.1.24 __chebyshev_u()

```
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 116 of file sf_chebyshev.tcc.

References __chebyshev_recur().

9.3.1.25 __chebyshev_v()

```
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 146 of file sf_chebyshev.tcc.

References __chebyshev_recur().

9.3.1.26 __chebyshev_w()

```
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 176 of file sf_chebyshev.tcc.

References __chebyshev_recur().

9.3.1.27 __chi_squared_pdf()

```
template<typename _Tp >
_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 75 of file sf_distributions.tcc.

References __pgamma().

9.3.1.28 __chi_squared_pdfc()

```
template<typename _Tp >
_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 99 of file sf_distributions.tcc.

References __qgamma().

9.3.1.29 __chshint()

```
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 166 of file sf_hypint.tcc.

References __chshint_cont_frac(), and __chshint_series().

9.3.1.30 __chshint_cont_frac()

```
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 53 of file sf_hypint.tcc.

Referenced by __chshint().

9.3.1.31 __chshint_series()

```
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 96 of file sf_hypint.tcc.

Referenced by __chshint().

9.3.1.32 __clamp_0_m2pi()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clamp_0_m2pi (
            std::complex< _Tp > __z )
```

Definition at line 185 of file sf_polylog.tcc.

Referenced by __polylog_exp_neg_int(), __polylog_exp_neg_real(), __polylog_exp_pos_int(), and __polylog_exp_↩
pos_real().

9.3.1.33 __clamp_pi()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clamp_pi (
            std::complex< _Tp > __z )
```

Definition at line 172 of file sf_polylog.tcc.

Referenced by __polylog_exp_neg_int(), __polylog_exp_neg_real(), __polylog_exp_pos_int(), and __polylog_exp_↩
pos_real().

9.3.1.34 __clausen() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clausen (
            unsigned int __m,
            std::complex< _Tp > __z )
```

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

Parameters

_↩ _m	The non-negative integral order.
_↩ _z	The complex argument.

Returns

The complex Clausen function.

Definition at line 1257 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.35 __clausen() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen (
            unsigned int __m,
            _Tp __x )
```

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

Parameters

__m	The integer order m >= 1.
__x	The real argument.

Returns

The Clausen function.

Definition at line 1284 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.36 __clausen_cl() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_cl (
            unsigned int __m,
            std::complex< _Tp > __z )
```

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.
_↩ _z	The complex argument.

Returns

The Clausen cosine sum Cl_m(w),

Definition at line 1368 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.37 __clausen_cl() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_cl (
            unsigned int __m,
            _Tp __x )
```

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.
_↩ _x	The real argument.

Returns

The real Clausen cosine sum Cl_m(w),

Definition at line 1396 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.38 __clausen_sl() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_sl (
            unsigned int __m,
            std::complex< _Tp > __z )
```

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

See also

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

Parameters

__m	The integer order m >= 1.
__z	The complex argument.

Returns

The Clausen sine sum Sl_m(w),

Definition at line 1312 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.39 __clausen_sl() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_sl (
            unsigned int __m,
            _Tp __x )
```

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

See also

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

Parameters

__m	The integer order m >= 1.
__x	The real argument.

Returns

The Clausen sine sum Sl_m(w),

Definition at line 1340 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.40 __comp_ellint_1()

```
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 568 of file sf_ellint.tcc.

References __comp_ellint_rf().

Referenced by __ellint_1(), __ellnome_k(), __heuman_lambda(), __jacobi_zeta(), __theta_c(), __theta_d(), __theta_↩
n(), and __theta_s().

9.3.1.41 __comp_ellint_2()

```
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 642 of file sf_ellint.tcc.

References __ellint_rd(), and __ellint_rf().

Referenced by __ellint_2().

9.3.1.42 __comp_ellint_3()

```
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 732 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

Referenced by __ellint_3().

9.3.1.43 __comp_ellint_d()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_d (
            _Tp __k )
```

Return the complete Legendre elliptic integral D.

Definition at line 840 of file sf_ellint.tcc.

References __ellint_rd().

9.3.1.44 __comp_ellint_rf()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_rf (
            _Tp __x,
            _Tp __y )
```

Definition at line 238 of file sf_ellint.tcc.

Referenced by __comp_ellint_1(), and __ellint_rf().

9.3.1.45 __comp_ellint_rg()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_rg (
            _Tp __x,
            _Tp __y )
```

Definition at line 349 of file sf_ellint.tcc.

Referenced by __ellint_rg().

9.3.1.46 __conf_hyperg()

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg (
            _Tp __a,
            _Tp __c,
            _Tp __x )
```

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

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().

Referenced by __tricomi_u_naive().

9.3.1.47 __conf_hyperg_lim()

```
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().

9.3.1.48 __conf_hyperg_lim_series()

```
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().

9.3.1.49 __conf_hyperg_luke()

```
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().

9.3.1.50 __conf_hyperg_series()

```
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().

9.3.1.51 __cos_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__cos_pi (
            _Tp __x )
```

Return the reperiodized cosine of argument x:

$$ \cos_\pi(x) = \cos(\pi x) $$

Definition at line 102 of file sf_trig.tcc.

Referenced by __cos_pi(), __cosh_pi(), __cyl_bessel_jn(), __cyl_bessel_jn_neg_arg(), __log_double_factorial(), __sin_pi(), and __sinh_pi().

9.3.1.52 __cos_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cos_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized cosine of complex argument z:

$$\cos_\pi(z) = \cos(\pi z) = \cos_\pi(x)\cosh_\pi(y) - i\sin_\pi(x)\sinh_\pi(y)$$

Definition at line 227 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.1.53 __cosh_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__cosh_pi (
            _Tp __x )
```

Return the reperiodized hyperbolic cosine of argument x:

$$\cosh_\pi(x) = \cosh(\pi x)$$

Definition at line 130 of file sf_trig.tcc.

9.3.1.54 __cosh_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cosh_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized hyperbolic cosine of complex argument z:

$$\cosh_\pi(z) = \cosh_\pi(z) = \cosh_\pi(x)\cos_\pi(y) + i\sinh_\pi(x)\sin_\pi(y)$$

Definition at line 249 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.1.55 __coshint()

```
template<typename _Tp >
_Tp std::__detail::__coshint (
            const _Tp __x )
```

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

The hyperbolic cosine integral is given by

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

Parameters

_↩ _x	The argument of the hyperbolic cosine integral function.

Returns

The hyperbolic cosine integral.

Definition at line 561 of file sf_expint.tcc.

References __expint_E1(), and __expint_Ei().

9.3.1.56 __coulomb_CF1()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_CF1 (
            unsigned int __l,
            _Tp __eta,
            _Tp __x )
```

Evaluate the first continued fraction, giving the ratio F'/F at the upper l value. We also determine the sign of F at that point, since it is the sign of the last denominator in the continued fraction.

Definition at line 143 of file sf_coulomb.tcc.

9.3.1.57 __coulomb_CF2()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__coulomb_CF2 (
            unsigned int __l,
            _Tp __eta,
            _Tp __x )
```

Evaluate the second continued fraction to obtain the ratio

$$(G' + iF')/(G + iF) := P + iQ$$

at the specified l value.

Definition at line 201 of file sf_coulomb.tcc.

9.3.1.58 __coulomb_f_recur()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_f_recur (
            unsigned int __l_min,
            unsigned int __k_max,
            _Tp __eta,
            _Tp __x,
            _Tp _F_l_max,
            _Tp _Fp_l_max )
```

Evolve the backwards recurrence for F, F'.

$$F_{l-1} = (S_l F_l + F_l')/R_l F_{l-1}' = (S_l F_{l-1} - R_l F_l)$$

where

$$R_l = \sqrt{1 + (\eta/l)^2} S_l = l/x + \eta/l$$

Definition at line 74 of file sf_coulomb.tcc.

9.3.1.59 __coulomb_g_recur()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_g_recur (
            unsigned int __l_min,
            unsigned int __k_max,
            _Tp __eta,
            _Tp __x,
            _Tp _G_l_min,
            _Tp _Gp_l_min )
```

Evolve the forward recurrence for G, G'.

$$G_{l+1} = (S_l G_l - G_l')/R_l G_{l+1}' = R_{l+1} G_l - S_l G_{l+1}$$

where

$$R_l = \sqrt{1 + (\eta/l)^2} S_l = l/x + \eta/l$$

Definition at line 112 of file sf_coulomb.tcc.

9.3.1.60 __coulomb_norm()

```
template<typename _Tp >
_Tp std::__detail::__coulomb_norm (
            unsigned int __l,
            _Tp __eta )
```

Definition at line 46 of file sf_coulomb.tcc.

9.3.1.61 __cyl_bessel()

```
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 1174 of file sf_hankel.tcc.

References __hankel().

9.3.1.62 __cyl_bessel_i()

```
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 362 of file sf_mod_bessel.tcc.

References __cyl_bessel_ij_series(), and __cyl_bessel_ik().

Referenced by __rice_pdf().

9.3.1.63 __cyl_bessel_ij_series()

```
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 399 of file sf_bessel.tcc.

References __log_gamma().

Referenced by __cyl_bessel_i(), and __cyl_bessel_j().

9.3.1.64 __cyl_bessel_ik()

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik (
            _Tp __nu,
            _Tp __x )
```

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.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 300 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik_asymp(), __cyl_bessel_ik_steed(), and __sin_pi().

Referenced by __airy(), __cyl_bessel_i(), __cyl_bessel_k(), and __sph_bessel_ik().

9.3.1.65 __cyl_bessel_ik_asymp()

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_asymp (
            _Tp __nu,
            _Tp __x )
```

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.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 79 of file sf_mod_bessel.tcc.

Referenced by __cyl_bessel_ik(), and __cyl_bessel_ik_steed().

9.3.1.66 __cyl_bessel_ik_steed()

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_steed (
            _Tp __nu,
            _Tp __x )
```

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.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 145 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik_asymp(), and __gamma_temme().

Referenced by __cyl_bessel_ik().

9.3.1.67 __cyl_bessel_j()

```
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 546 of file sf_bessel.tcc.

References __cyl_bessel_ij_series(), and __cyl_bessel_jn().

9.3.1.68 __cyl_bessel_jn()

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn (
            _Tp __nu,
            _Tp __x )
```

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

Definition at line 438 of file sf_bessel.tcc.

References __cos_pi(), __cyl_bessel_jn_asymp(), __cyl_bessel_jn_steed(), and __sin_pi().

Referenced by __airy(), __cyl_bessel_j(), __cyl_bessel_jn_neg_arg(), __cyl_hankel_1(), __cyl_hankel_2(), __cyl_↩ neumann_n(), and __sph_bessel_jn().

9.3.1.69 __cyl_bessel_jn_asymp()

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_asymp (
            _Tp __nu,
            _Tp __x )
```

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.

Returns

A struct containing the cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 79 of file sf_bessel.tcc.

Referenced by __cyl_bessel_jn(), and __cyl_bessel_jn_steed().

9.3.1.70 __cyl_bessel_jn_neg_arg()

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, std::complex<_Tp> > std::__detail::__cyl_bessel_jn_neg_arg (
            _Tp __nu,
            _Tp __x )
```

Return the cylindrical Bessel functions and their derivatives of order ν and argument $x < 0$.

Definition at line 504 of file sf_bessel.tcc.

References __cos_pi(), __cyl_bessel_jn(), and __polar_pi().

Referenced by __cyl_hankel_1(), __cyl_hankel_2(), and __sph_bessel_jn_neg_arg().

9.3.1.71 __cyl_bessel_jn_steed()

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_steed (
            _Tp __nu,
            _Tp __x )
```

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.

Returns

A struct containing the cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 199 of file sf_bessel.tcc.

References __cyl_bessel_jn_asymp(), and __gamma_temme().

Referenced by __cyl_bessel_jn().

9.3.1.72 __cyl_bessel_k()

```
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 396 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik().

9.3.1.73 __cyl_hankel_1() [1/2]

```
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 603 of file sf_bessel.tcc.

References __cyl_bessel_jn(), __cyl_bessel_jn_neg_arg(), and __polar_pi().

9.3.1.74 __cyl_hankel_1() [2/2]

```
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 1140 of file sf_hankel.tcc.

References __hankel().

9.3.1.75 __cyl_hankel_2() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_hankel_2 (
            _Tp __nu,
            _Tp __x )
```

Return the cylindrical Hankel function of the second kind $H^{(2)}_n u(x)$.

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 642 of file sf_bessel.tcc.

References __cyl_bessel_jn(), __cyl_bessel_jn_neg_arg(), and __polar_pi().

9.3.1.76 __cyl_hankel_2() [2/2]

```
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 1157 of file sf_hankel.tcc.

References __hankel().

9.3.1.77 __cyl_neumann()

```
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 1191 of file sf_hankel.tcc.

References __hankel().

9.3.1.78 __cyl_neumann_n()

```
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 577 of file sf_bessel.tcc.

References __cyl_bessel_jn().

9.3.1.79 __dawson()

```
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 235 of file sf_dawson.tcc.

References __dawson_cont_frac(), and __dawson_series().

9.3.1.80 __dawson_cont_frac()

```
template<typename _Tp >
_Tp std::__detail::__dawson_cont_frac (
            _Tp __x )
```

Compute the Dawson integral using a sampling theorem representation.

This array could be built on a thread-local basis.

Definition at line 73 of file sf_dawson.tcc.

Referenced by __dawson().

9.3.1.81 __dawson_series()

```
template<typename _Tp >
_Tp std::__detail::__dawson_series (
            _Tp __x )
```

Compute the Dawson integral using the series expansion.

Definition at line 49 of file sf_dawson.tcc.

Referenced by __dawson().

9.3.1.82 __debye()

```
template<typename _Tp >
_Tp std::__detail::__debye (
            unsigned int __n,
            _Tp __x )
```

Return the Debye function. The Debye functions are related to the incomplete Riemann zeta function:

$$\zeta_x(s) = \frac{1}{\Gamma(s)} \int_0^x \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^\infty \frac{P(s, kx)}{k^s}$$

$$Z_x(s) = \frac{1}{\Gamma(s)} \int_x^\infty \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^\infty \frac{Q(s, kx)}{k^s}$$

where $P(a,x), Q(a,x)$ is the incomplete gamma function ratios. The Debye functions are:

$$D_n(x) = \frac{n}{x^n} \int_0^x \frac{t^n}{e^t - 1} dt = \Gamma(n+1)\zeta_x(n+1)$$

and

$$\int_0^x \frac{t^n}{e^t - 1} dt = \Gamma(n+1)\zeta_x(n+1)$$

Todo : We should return both the Debye function and it's complement.

Compute the Debye function:

$$D_n(x) = 1 - \sum_{k=1}^{\infty} e^{-kx} \frac{n}{k} \sum_{m=0}^{n} \frac{n!}{(n-m)!} frac1(kx)^m$$

Abramowitz & Stegun 27.1.2

Compute the Debye function:

$$D_n(x) = 1 - \frac{nx}{2(n+1)} + n \sum_{k=1}^{\infty} \frac{B_{2k} x^{2k}}{(2k+n)(2k)!}$$

for $|x| < 2\pi$. Abramowitz-Stegun 27.1.1

Definition at line 820 of file sf_zeta.tcc.

9.3.1.83 __debye_region()

```
template<typename _Tp >
void std::__detail::__debye_region (
            std::complex< _Tp > __alpha,
            int & __indexr,
            char & __aorb )
```

Compute the Debye region in the complex plane.

Definition at line 54 of file sf_hankel.tcc.

Referenced by __hankel().

9.3.1.84 __dilog()

```
template<typename _Tp >
_Tp std::__detail::__dilog (
            _Tp __x )
```

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

The dilogarithm function is defined by:

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

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 196 of file sf_zeta.tcc.

9.3.1.85 __dirichlet_beta() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_beta (
            std::complex< _Tp > __s )
```

Return the Dirichlet beta function. Currently, s must be real (complex type but negligible imaginary part.) Otherwise std::domain_error is thrown. The Dirichlet beta function, in terms of the polylogarithm, is

$$\beta(s) = \operatorname{Im} Li_s(i)$$

Parameters

_↩ _s	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 1194 of file sf_polylog.tcc.

References __polylog().

9.3.1.86 __dirichlet_beta() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_beta (
            _Tp __s )
```

Return the Dirichlet beta function for real argument. The Dirichlet beta function, in terms of the polylogarithm, is

$$\beta(s) = \operatorname{Im} Li_s(i)$$

Parameters

_↩ _s	The real argument.

Returns

The Dirichlet Beta function of real argument.

Definition at line 1219 of file sf_polylog.tcc.

References __polylog().

9.3.1.87 __dirichlet_eta() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__dirichlet_eta (
            std::complex< _Tp > __s )
```

Return the Dirichlet eta function. Currently, s must be real (complex type but negligible imaginary part.) Otherwise std::domain_error is thrown. The Dirichlet eta function, in terms of the polylogarithm, is

$$\eta(s) = -\operatorname{Re} Li_s(-1)$$

Parameters

_↩ _s	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 1130 of file sf_polylog.tcc.

References __polylog().

Referenced by __dirichlet_eta(), and __dirichlet_lambda().

9.3.1.88 __dirichlet_eta() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_eta (
            _Tp __s )
```

Return the Dirichlet eta function for real argument. The Dirichlet eta function, in terms of the polylogarithm, is

$$\eta(s) = -\operatorname{Re} Li_s(-1)$$

Parameters

_↩ _s	The real argument.

Returns

The Dirichlet eta function.

Definition at line 1154 of file sf_polylog.tcc.

References __dirichlet_eta(), __gnu_cxx::__fp_is_integer(), __gamma(), __polylog(), and __sin_pi().

9.3.1.89 __dirichlet_lambda()

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_lambda (
            _Tp __s )
```

Return the Dirichlet lambda function for real argument.

$$\lambda(s) = \frac{1}{2}(\zeta(s) + \eta(s))$$

Parameters

_↩ _s	The real argument.

Returns

The Dirichlet lambda function.

Definition at line 1239 of file sf_polylog.tcc.

References __dirichlet_eta(), and __riemann_zeta().

9.3.1.90 __double_factorial()

```
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 1673 of file sf_gamma.tcc.

References std::__detail::_Factorial_table< _Tp >::__factorial, __log_double_factorial(), std::__detail::_Factorial_table< _Tp >::__n, _S_double_factorial_table, and _S_neg_double_factorial_table.

9.3.1.91 __ellint_1()

```
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 597 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rf().

Referenced by __heuman_lambda().

9.3.1.92 __ellint_2()

```
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 678 of file sf_ellint.tcc.

References __comp_ellint_2(), __ellint_rd(), and __ellint_rf().

9.3.1.93 __ellint_3()

```
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 773 of file sf_ellint.tcc.

References __comp_ellint_3(), __ellint_rf(), and __ellint_rj().

9.3.1.94 __ellint_cel()

```
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 928 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

9.3.1.95 __ellint_d()

```
template<typename _Tp >
_Tp std::__detail::__ellint_d (
            _Tp __k,
            _Tp __phi )
```

Return the Legendre elliptic integral D.

Definition at line 814 of file sf_ellint.tcc.

References __ellint_rd().

9.3.1.96 __ellint_el1()

```
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 856 of file sf_ellint.tcc.

References __ellint_rf().

9.3.1.97 __ellint_el2()

```
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 877 of file sf_ellint.tcc.

References __ellint_rd(), and __ellint_rf().

9.3.1.98 __ellint_el3()

```
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 902 of file sf_ellint.tcc.

References __ellint_rf(), and __ellint_rj().

9.3.1.99 __ellint_rc()

```
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 84 of file sf_ellint.tcc.

Referenced by __ellint_rf(), and __ellint_rj().

9.3.1.100 __ellint_rd()

```
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 166 of file sf_ellint.tcc.

Referenced by __comp_ellint_2(), __comp_ellint_d(), __ellint_2(), __ellint_d(), __ellint_el2(), __ellint_rg(), and __ellint_rj().

9.3.1.101 __ellint_rf()

```
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 280 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().

9.3.1.102 __ellint_rg()

```
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 411 of file sf_ellint.tcc.

References __comp_ellint_rg(), and __ellint_rd().

9.3.1.103 __ellint_rj()

```
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 459 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().

9.3.1.104 __ellnome()

```
template<typename _Tp >
_Tp std::__detail::__ellnome (
            _Tp __k )
```

Return the elliptic nome given the modulus k.

$$q(k) = exp\left(-\pi\frac{K(k')}{K(k)}\right)$$

Definition at line 382 of file sf_theta.tcc.

References __ellnome_k(), and __ellnome_series().

Referenced by __theta_c(), __theta_d(), __theta_n(), and __theta_s().

9.3.1.105 __ellnome_k()

```
template<typename _Tp >
_Tp std::__detail::__ellnome_k (
            _Tp __k )
```

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

$$q(k) = exp\left(-\pi\frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1-k^2}$ is the complementary elliptic argument and is the Legendre elliptic integral of the first kind.

Definition at line 365 of file sf_theta.tcc.

References __comp_ellint_1().

Referenced by __ellnome().

9.3.1.106 __ellnome_series()

```
template<typename _Tp >
_Tp std::__detail::__ellnome_series (
            _Tp __k )
```

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

$$q(k) = exp\left(-\pi\frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1-k^2}$ is the complementary elliptic argument and is the Legendre elliptic integral of the first kind.

Definition at line 344 of file sf_theta.tcc.

Referenced by __ellnome().

9.3.1.107 __euler() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__euler (
            unsigned int __n )  [inline]
```

This returns Euler number E_n.

Parameters

__n	the order n of the Euler number.

Returns

The Euler number of order n.

Definition at line 119 of file sf_euler.tcc.

9.3.1.108 __euler() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__euler (
            unsigned int __n,
            _Tp __x )
```

Return the Euler polynomial $E_n(x)$ of order n at argument x.

The derivative is proportional to the previous polynomial:

$$E_n'(x) = nE_{n-1}(x)$$

$$E_n(1/2) = \frac{E_n}{2^n}, \text{ where } E_n \text{ is the n-th Euler number.}$$

Definition at line 137 of file sf_euler.tcc.

References __bernoulli().

9.3.1.109 __euler_series()

```
template<typename _Tp >
_Tp std::__detail::__euler_series (
            unsigned int __n )
```

Return the Euler number from lookup or by series expansion.

The Euler numbers are given by the recursive sum:

$$E_n = B_n(1) = B_n$$

where $E_0 = 1$, $E_1 = 0$, $E_2 = -1$

Todo Find a way to predict the maximum Euler number for a type.

Definition at line 61 of file sf_euler.tcc.

9.3.1.110 __eulerian_1()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_1 (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle {n \atop m} \right\rangle = (n-m)\left\langle {n-1 \atop m-1} \right\rangle + (m+1)\left\langle {n-1 \atop m} \right\rangle \text{ for } n > 0$$

Note that $A(n, m)$ is a common older notation.

Definition at line 207 of file sf_euler.tcc.

9.3.1.111 __eulerian_1_recur()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_1_recur (
            unsigned int __n,
            unsigned int __m )
```

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle {n \atop m} \right\rangle = (n-m)\left\langle {n-1 \atop m-1} \right\rangle + (m+1)\left\langle {n-1 \atop m} \right\rangle \text{ for } n > 0$$

Note that $A(n, m)$ is a common older notation.

Definition at line 166 of file sf_euler.tcc.

9.3.1.112 __eulerian_2()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_2 (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the Eulerian number of the second kind. The Eulerian numbers of the second kind are defined by recursion:

$$A(n, m) = (2n - m - 1)A(n - 1, m - 1) + (m + 1)A(n - 1, m) \text{ for } n > 0$$

Definition at line 254 of file sf_euler.tcc.

9.3.1.113 __eulerian_2_recur()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_2_recur (
            unsigned int __n,
            unsigned int __m )
```

Return the Eulerian number of the second kind by recursion. The recursion is:

$$A(n, m) = (2n - m - 1)A(n - 1, m - 1) + (m + 1)A(n - 1, m) \text{ for } n > 0$$

Definition at line 219 of file sf_euler.tcc.

9.3.1.114 __expint() [1/2]

```
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

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

Parameters

_← _n	The order of the exponential integral function.
_← _x	The argument of the exponential integral function.

Returns

The exponential integral.

Todo Study arbitrary switch to large-n $E_n(x)$.

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

Definition at line 476 of file sf_expint.tcc.

References __expint_E1(), __expint_En_asymp(), __expint_En_cont_frac(), __expint_En_large_n(), and __expint_←
En_series().

Referenced by __logint().

9.3.1.115 __expint() [2/2]

```
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 517 of file sf_expint.tcc.

References __expint_Ei().

9.3.1.116 __expint_E1()

```
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 381 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().

9.3.1.117 __expint_E1_asymp()

```
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 114 of file sf_expint.tcc.

Referenced by __expint_E1().

9.3.1.118 __expint_E1_series()

```
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 76 of file sf_expint.tcc.

Referenced by __expint_E1().

9.3.1.119 __expint_Ei()

```
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 356 of file sf_expint.tcc.

References __expint_E1(), __expint_Ei_asymp(), and __expint_Ei_series().

Referenced by __coshint(), __expint(), __expint_E1(), and __sinhint().

9.3.1.120 __expint_Ei_asymp()

```
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 322 of file sf_expint.tcc.

Referenced by __expint_Ei().

9.3.1.121 __expint_Ei_series()

```
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 289 of file sf_expint.tcc.

Referenced by __expint_Ei().

9.3.1.122 __expint_En_asymp()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_asymp (
            unsigned int __n,
            _Tp __x )
```

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

The exponential integral is given by

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

Parameters

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

Returns

The exponential integral.

Definition at line 410 of file sf_expint.tcc.

Referenced by __expint().

9.3.1.123 __expint_En_cont_frac()

```
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 198 of file sf_expint.tcc.

Referenced by __expint(), and __expint_E1().

9.3.1.124 __expint_En_large_n()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_large_n (
            unsigned int __n,
            _Tp __x )
```

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

The exponential integral is given by

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

Parameters

_↩ _n	The order of the exponential integral function.
_↩ _x	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 442 of file sf_expint.tcc.

Referenced by __expint().

9.3.1.125 __expint_En_recursion()

```
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 244 of file sf_expint.tcc.

References __expint_E1().

9.3.1.126 __expint_En_series()

```
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 150 of file sf_expint.tcc.

References __psi().

Referenced by __expint().

9.3.1.127 __exponential_cdf()

```
template<typename _Tp >
_Tp std::__detail::__exponential_cdf (
            _Tp __lambda,
            _Tp __x )
```

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-\lambda x} \text{ for } x >= 0$$

Definition at line 328 of file sf_distributions.tcc.

9.3.1.128 __exponential_cdfc()

```
template<typename _Tp >
_Tp std::__detail::__exponential_cdfc (
            _Tp __lambda,
            _Tp __x )
```

Return the complement of the exponential cumulative probability density function.

The formula for the complement of the exponential cumulative probability density function is

$$F(x|\lambda) = e^{-\lambda x} \text{ for } x >= 0$$

Definition at line 350 of file sf_distributions.tcc.

9.3.1.129 __exponential_pdf()

```
template<typename _Tp >
_Tp std::__detail::__exponential_pdf (
            _Tp __lambda,
            _Tp __x )
```

Return the exponential probability density function.

The formula for the exponential probability density function is

$$f(x|\lambda) = \lambda e^{-\lambda x} \text{ for } x >= 0$$

Definition at line 308 of file sf_distributions.tcc.

9.3.1.130 __factorial()

```
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 1615 of file sf_gamma.tcc.

References std::__detail::_Factorial_table$<$ _Tp $>$::__n, and _S_factorial_table.

9.3.1.131 __falling_factorial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__falling_factorial (
            _Tp __a,
            int __n )
```

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

In particular, $n^{\underline{n}} = n!$.

Definition at line 2918 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __log_gamma(), __log_gamma_sign(), and std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __falling_factorial(), and __log_falling_factorial().

9.3.1.132 __falling_factorial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__falling_factorial (
            _Tp __a,
            _Tp __nu )
```

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν. The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a+1)/\Gamma(a-\nu+1)$$

.

Definition at line 2973 of file sf_gamma.tcc.

References __falling_factorial(), __gnu_cxx::__fp_is_integer(), __log_gamma(), and __log_gamma_sign().

9.3.1.133 __fermi_dirac()

```
template<typename _Sp , typename _Tp >
_Tp std::__detail::__fermi_dirac (
            _Sp __s,
            _Tp __x )
```

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

See also

https://en.wikipedia.org/wiki/Clausen_function
http://dlmf.nist.gov/25.12.16

$$F_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x}+1} dt = -Li_{s+1}(-e^x)$$

Parameters

_↩ _s	The order s > -1.
_↩ _x	The real argument.

Returns

The real Fermi-Dirac integral F_s(x),

Definition at line 1430 of file sf_polylog.tcc.

References __polylog_exp().

9.3.1.134 __fisher_f_cdf()

```
template<typename _Tp >
_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 523 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.135 __fisher_f_cdfc()

```
template<typename _Tp >
_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 552 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.136 __fisher_f_pdf()

```
template<typename _Tp >
_Tp std::__detail::__fisher_f_pdf (
            _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 493 of file sf_distributions.tcc.

References __beta().

9.3.1.137 __fock_airy()

```
template<typename _Tp >
__gnu_cxx::__fock_airy_t<_Tp, std::complex<_Tp> > std::__detail::__fock_airy (
            _Tp __x )
```

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.

Returns

A struct containing the Fock-type Airy functions of the first and second kinds and their derivatives.

Definition at line 551 of file sf_mod_bessel.tcc.

References __airy().

9.3.1.138 __fresnel()

```
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 170 of file sf_fresnel.tcc.

References __fresnel_cont_frac(), and __fresnel_series().

9.3.1.139 __fresnel_cont_frac()

```
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 109 of file sf_fresnel.tcc.

Referenced by __fresnel().

9.3.1.140 __fresnel_series()

```
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 51 of file sf_fresnel.tcc.

Referenced by __fresnel().

9.3.1.141 __gamma() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma (
            _Tp __a )
```

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^\infty e^{-t} t^{a-1} dt (a > 0)$$

.

Parameters

_↩ _a	The argument of the gamma function.

Returns

The gamma function.

Definition at line 2616 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_reciprocal_series(), __log_gamma(), __log_gamma_sign(), std↩::__detail::_Factorial_table< _Tp >::__n, and _S_factorial_table.

Referenced by __beta_gamma(), __binomial(), __dirichlet_eta(), __gamma_cdf(), __gamma_cdfc(), __gamma_pdf(), __gamma_reciprocal(), __gamma_reciprocal_series(), __hurwitz_zeta_polylog(), __polylog_exp_pos(), __riemann_↩zeta(), __riemann_zeta_glob(), __riemann_zeta_m_1(), __riemann_zeta_sum(), __student_t_pdf(), and std::__detail↩::_Airy_series< _Tp >::_S_Scorer2().

9.3.1.142 __gamma() [2/2]

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma (
            _Tp __a,
            _Tp __x )
```

Return the incomplete gamma functions.

Definition at line 2743 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

9.3.1.143 __gamma_cdf()

```
template<typename _Tp >
_Tp std::__detail::__gamma_cdf (
            _Tp __alpha,
            _Tp __beta,
            _Tp __x )
```

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)}(x/\beta)^{\alpha-1}e^{-x/\beta}$$

Definition at line 141 of file sf_distributions.tcc.

References __gamma(), and __tgamma_lower().

9.3.1.144 __gamma_cdfc()

```
template<typename _Tp >
_Tp std::__detail::__gamma_cdfc (
            _Tp __alpha,
            _Tp __beta,
            _Tp __x )
```

Return the gamma complementary cumulative propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)}(x/\beta)^{\alpha-1}e^{-x/\beta}$$

Definition at line 162 of file sf_distributions.tcc.

References __gamma(), and __tgamma().

9.3.1.145 __gamma_cont_frac()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (
            _Tp __a,
            _Tp __x )
```

Return the incomplete gamma function by continued fraction.

Definition at line 2698 of file sf_gamma.tcc.

References __log_gamma(), __log_gamma_sign(), and std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __gamma(), __pgamma(), __qgamma(), __tgamma(), and __tgamma_lower().

9.3.1.146 __gamma_pdf()

```
template<typename _Tp >
_Tp std::__detail::__gamma_pdf (
            _Tp __alpha,
            _Tp __beta,
            _Tp __x )
```

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha,\beta) = \frac{1}{\beta\Gamma(\alpha)}(x/\beta)^{\alpha-1}e^{-x/\beta}$$

Definition at line 121 of file sf_distributions.tcc.

References __gamma().

9.3.1.147 __gamma_reciprocal()

```
template<typename _Tp >
_Tp std::__detail::__gamma_reciprocal (
            _Tp __a )
```

Return the reciprocal of the Gamma function:

$$\frac{1}{\Gamma(a)}$$

Parameters

__a	The argument of the reciprocal of the gamma function.

Returns

The reciprocal of the gamma function.

Definition at line 2246 of file sf_gamma.tcc.

References std::__detail::_Factorial_table< _Tp >::__factorial, __gnu_cxx::__fp_is_integer(), __gamma(), __gamma_reciprocal_series(), std::__detail::_Factorial_table< _Tp >::__n, __sin_pi(), and _S_factorial_table.

Referenced by __polylog_exp_asymp().

9.3.1.148 __gamma_reciprocal_series()

```
template<typename _Tp >
_Tp std::__detail::__gamma_reciprocal_series (
            _Tp __a )
```

Return the reciprocal of the Gamma function by series. The reciprocal of the Gamma function is given by

$$\frac{1}{\Gamma(a)} = \sum_{k=1}^{\infty} c_k a^k$$

where the coefficients are defined by recursion:

$$c_{k+1} = \frac{1}{k}\left[\gamma_E c_k + (-1)^k \sum_{j=1}^{k-1} (-1)^j \zeta(j+1-k) c_j\right]$$

where $c_1 = 1$

Parameters

__a	The argument of the reciprocal of the gamma function.

Returns

The reciprocal of the gamma function.

Definition at line 2180 of file sf_gamma.tcc.

References __gamma().

Referenced by __gamma(), __gamma_reciprocal(), and __gamma_temme().

9.3.1.149 __gamma_series()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma_series (
            _Tp __a,
            _Tp __x )
```

Return the incomplete gamma function by series summation.

$$\gamma(a,x) = x^a e^{-z} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

.

Definition at line 2653 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __log_gamma(), __log_gamma_sign(), and std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __gamma(), __pgamma(), __qgamma(), __tgamma(), and __tgamma_lower().

9.3.1.150 __gamma_temme()

```
template<typename _Tp >
__gnu_cxx::__gamma_temme_t<_Tp> std::__detail::__gamma_temme (
            _Tp __mu )
```

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

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

and

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

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

The accuracy requirements on this are exquisite.

Parameters

__mu	The input parameter of the gamma functions.

Returns

An output structure containing four gamma functions.

Definition at line 158 of file sf_bessel.tcc.

References __gamma_reciprocal_series().

Referenced by __cyl_bessel_ik_steed(), and __cyl_bessel_jn_steed().

9.3.1.151 __gauss()

```
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.

9.3.1.152 __gegenbauer_poly()

```
template<typename _Tp >
_Tp std::__detail::__gegenbauer_poly (
            unsigned int __n,
            _Tp __alpha1,
            _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
__alpha1	The real order
__x	The real argument

Definition at line 63 of file sf_gegenbauer.tcc.

9.3.1.153 __gegenbauer_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__gegenbauer_zeros (
            unsigned int __n,
            _Tp __alpha1 )
```

Return a vector containing the zeros of the Gegenbauer or ultraspherical polynomial $C_n^{(\alpha)}$.

Definition at line 97 of file sf_gegenbauer.tcc.

References __gnu_cxx::lgamma().

9.3.1.154 __hankel()

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__↩
detail::__hankel (
            std::complex< _Tp > __nu,
            std::complex< _Tp > __z )
```

Parameters

in	*__nu*	The order for which the Hankel functions are evaluated.
in	*__z*	The argument at which the Hankel functions are evaluated.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 1081 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().

9.3.1.155 __hankel_debye()

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__↩
detail::__hankel_debye (
             std::complex< _Tp > __nu,
             std::complex< _Tp > __z,
             std::complex< _Tp > __alpha,
             int __indexr,
             char & __aorb,
             int & __morn )
```

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*	

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 914 of file sf_hankel.tcc.

References __sin_pi().

Referenced by __hankel().

9.3.1.156 __hankel_params()

```
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 109 of file sf_hankel.tcc.

Referenced by __hankel_uniform_outer().

9.3.1.157 __hankel_uniform()

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__↩
detail::__hankel_uniform (
            std::complex< _Tp > __nu,
            std::complex< _Tp > __z )
```

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.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 861 of file sf_hankel.tcc.

References __hankel_uniform_olver().

Referenced by __hankel().

9.3.1.158 __hankel_uniform_olver()

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__↩
detail::__hankel_uniform_olver (
            std::complex< _Tp > __nu,
            std::complex< _Tp > __z )
```

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.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 778 of file sf_hankel.tcc.

References __hankel_uniform_outer(), and __hankel_uniform_sum().

Referenced by __hankel_uniform().

9.3.1.159 __hankel_uniform_outer()

```
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 248 of file sf_hankel.tcc.

References __airy_arg(), and __hankel_params().

Referenced by __hankel_uniform_olver().

9.3.1.160 __hankel_uniform_sum()

```
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 325 of file sf_hankel.tcc.

Referenced by __hankel_uniform_olver().

9.3.1.161 __harmonic_number()

```
template<typename _Tp >
_Tp std::__detail::__harmonic_number (
            unsigned int __n )
```

Definition at line 3263 of file sf_gamma.tcc.

References std::__detail::_Factorial_table< _Tp >::__n, _S_harmonic_denom, _S_harmonic_numer, and _S_num_↩
harmonic_numer.

9.3.1.162 __hermite()

```
template<typename _Tp >
_Tp std::__detail::__hermite (
            unsigned int __n,
            _Tp __x )
```

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

The Hermite polynomial is defined by:

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

An explicit series formula is:

$$H_n(x) = \sum_{k=0}^{m} \frac{(-1)^k}{k!(n-2k)!} (2x)^{n-2k} \text{ where } m = \left\lfloor \frac{n}{2} \right\rfloor$$

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 185 of file sf_hermite.tcc.

References __hermite_asymp(), and __hermite_recur().

9.3.1.163 __hermite_asymp()

```
template<typename _Tp >
_Tp std::__detail::__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 also

"Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, ar↩ Xiv: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 116 of file sf_hermite.tcc.

References __airy().

Referenced by __hermite().

9.3.1.164 __hermite_recur()

```
template<typename _Tp >
__gnu_cxx::__hermite_t<_Tp> std::__detail::__hermite_recur (
              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 71 of file sf_hermite.tcc.

Referenced by __hermite().

9.3.1.165 __hermite_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__hermite_zeros (
            unsigned int __n,
            _Tp __proto = _Tp{} )
```

Build a vector of the Gauss-Hermite integration rule abscissae and weights.

Definition at line 247 of file sf_hermite.tcc.

9.3.1.166 __heuman_lambda()

```
template<typename _Tp >
_Tp std::__detail::__heuman_lambda (
            _Tp __k,
            _Tp __phi )
```

Return the Heuman lambda function.

Definition at line 986 of file sf_ellint.tcc.

References __comp_ellint_1(), __ellint_1(), __ellint_rf(), __ellint_rj(), and __jacobi_zeta().

9.3.1.167 __hurwitz_zeta()

```
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 775 of file sf_zeta.tcc.

References __hurwitz_zeta_euler_maclaurin(), and __riemann_zeta().

Referenced by __psi().

9.3.1.168 __hurwitz_zeta_euler_maclaurin()

```
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"0160tas

Parameters

_↩ _s	The argument $s! = 1$
_↩ _a	The scale parameter $a > -1$

Definition at line 727 of file sf_zeta.tcc.

References _S_Euler_Maclaurin_zeta.

Referenced by __hurwitz_zeta().

9.3.1.169 __hurwitz_zeta_polylog()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__hurwitz_zeta_polylog (
```

```
            _Tp __s,
            std::complex< _Tp > __a )
```

Return the Hurwitz Zeta function for real s and complex a. This uses Jonquiere's identity:

$$\frac{(i2\pi)^s}{\Gamma(s)}\zeta(a,1-s) = Li_s(e^{i2\pi a}) + (-1)^s Li_s(e^{-i2\pi a})$$

Parameters

__s	The real argument
__a	The complex parameter

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

Definition at line 1088 of file sf_polylog.tcc.

References __gamma(), and __polylog_exp().

9.3.1.170 __hydrogen()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__hydrogen (
            unsigned int __n,
            unsigned int __l,
            unsigned int __m,
            _Tp __Z,
            _Tp __r,
            _Tp __theta,
            _Tp __phi )
```

Return the bound-state Coulomb wave-function.

Definition at line 245 of file sf_coulomb.tcc.

References __assoc_laguerre(), __log_gamma(), __psi(), and __sph_legendre().

9.3.1.171 __hyperg()

```
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 814 of file sf_hyperg.tcc.

References __hyperg_luke(), __hyperg_reflect(), __hyperg_series(), __log_gamma(), and __log_gamma_sign().

9.3.1.172 __hyperg_luke()

```
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 405 of file sf_hyperg.tcc.

Referenced by __hyperg().

9.3.1.173 __hyperg_reflect()

```
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:

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

Definition at line 540 of file sf_hyperg.tcc.

References __hyperg_series(), __log_gamma(), __log_gamma_sign(), and __psi().

Referenced by __hyperg().

9.3.1.174 __hyperg_series()

```
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 374 of file sf_hyperg.tcc.

Referenced by __hyperg(), and __hyperg_reflect().

9.3.1.175 __ibeta_cont_frac()

```
template<typename _Tp >
_Tp std::__detail::__ibeta_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 239 of file sf_beta.tcc.

Referenced by __beta_inc().

9.3.1.176 __jacobi_ellint()

```
template<typename _Tp >
__gnu_cxx::__jacobi_ellint_t<_Tp> std::__detail::__jacobi_ellint (
            _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 974 of file sf_theta.tcc.

9.3.1.177 __jacobi_recur()

```
template<typename _Tp >
__gnu_cxx::__jacobi_t<_Tp> std::__detail::__jacobi_recur (
            unsigned int __n,
            _Tp __alpha1,
            _Tp __beta1,
            _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 59 of file sf_jacobi.tcc.

Referenced by __poly_radial_jacobi().

9.3.1.178 __jacobi_theta_0()

```
template<typename _Tp >
__jacobi_theta_0_t<_Tp> std::__detail::__jacobi_theta_0 (
            _Tp __q )
```

Return a struct of the Jacobi theta functions and up to three non-zero derivatives evaluated at zero argument.

Definition at line 533 of file sf_theta.tcc.

References std::__detail::__jacobi_theta_0_t< _Tp >::th1p.

9.3.1.179 __jacobi_theta_1() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_1 (
            const std::complex< _Tp > & __q,
            const std::complex< _Tp > & __x )
```

Return the Jacobi θ_1 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_1(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 621 of file sf_theta.tcc.

References __jacobi_theta_1_sum(), and __polar_pi().

Referenced by __jacobi_theta_1().

9.3.1.180 __jacobi_theta_1() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1 (
            _Tp __q,
            const _Tp __x )
```

Return the Jacobi θ_1 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_1(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 664 of file sf_theta.tcc.

References __jacobi_theta_1().

9.3.1.181 __jacobi_theta_1_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1_sum (
            _Tp __q,
            _Tp __x )
```

Return the Jacobi θ_1 function by summation of the series.

The Jacobi or elliptic theta-1 function is defined by

$$\theta_1(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{(n+\frac{1}{2})^2} \sin{(2n+1)x}$$

Definition at line 590 of file sf_theta.tcc.

Referenced by __jacobi_theta_1().

9.3.1.182 __jacobi_theta_2() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_2 (
            const std::complex< _Tp > & __q,
            const std::complex< _Tp > & __x )
```

Return the Jacobi θ_2 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_2(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2} \cos{2nx}$$

Definition at line 718 of file sf_theta.tcc.

References __jacobi_theta_2_sum(), and __polar_pi().

Referenced by __jacobi_theta_2().

9.3.1.183 __jacobi_theta_2() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2 (
            _Tp __q,
            const _Tp __x )
```

Return the Jacobi θ_2 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_2(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 761 of file sf_theta.tcc.

References __jacobi_theta_2().

9.3.1.184 __jacobi_theta_2_prod0()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_prod0 (
            _Tp __q )
```

Compute and return the Jacobi θ_2 at zero argument by product expansion.

Definition at line 160 of file sf_theta.tcc.

9.3.1.185 __jacobi_theta_2_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_sum (
            _Tp __q,
            _Tp __x )
```

Return the Jacobi θ_2 function by summation of the series.

The Jacobi or elliptic theta-2 function is defined by

$$\theta_2(q,x) = 2\sum_{n=1}^{\infty} q^{(n+\frac{1}{2})^2}\cos{(2n+1)x}$$

Definition at line 690 of file sf_theta.tcc.

Referenced by __jacobi_theta_2().

9.3.1.186 __jacobi_theta_3() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_3 (
            const std::complex< _Tp > & __q,
            const std::complex< _Tp > & __x )
```

Return the Jacobi θ_3 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_3(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 814 of file sf_theta.tcc.

References __jacobi_theta_3_sum().

Referenced by __jacobi_theta_3().

9.3.1.187 __jacobi_theta_3() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3 (
            _Tp __q,
            const _Tp __x )
```

Return the Jacobi θ_3 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_3(q,x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 856 of file sf_theta.tcc.

References __jacobi_theta_3().

9.3.1.188 __jacobi_theta_3_prod0()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_prod0 (
            _Tp __q )
```

Compute and return the Jacobi θ_3 at zero argument by product expansion.

Definition at line 183 of file sf_theta.tcc.

9.3.1.189 __jacobi_theta_3_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_sum (
            _Tp __q,
            _Tp __x )
```

Return the Jacobi θ_3 function by summation of the series.

The Jacobi or elliptic theta-3 function is defined by

$$\theta_3(q,x) = 2\sum_{n=1}^{\infty} q^{n^2} \cos 2nx$$

Definition at line 786 of file sf_theta.tcc.

Referenced by __jacobi_theta_3().

9.3.1.190 __jacobi_theta_4() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_4 (
            const std::complex< _Tp > & __q,
            const std::complex< _Tp > & __x )
```

Return the Jacobi θ_4 function by summation of the series.

The Jacobi or elliptic theta-4 function is defined by

$$\theta_4(q,x) = 2\sum_{n=1}^{\infty} (-1)^n q^{n^2} \cos 2nx$$

Definition at line 911 of file sf_theta.tcc.

References __jacobi_theta_4_sum().

Referenced by __jacobi_theta_4().

9.3.1.191 __jacobi_theta_4() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4 (
            _Tp __q,
            const _Tp __x )
```

Return the Jacobi θ_4 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_4(q, x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 953 of file sf_theta.tcc.

References __jacobi_theta_4().

9.3.1.192 __jacobi_theta_4_prod0()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_prod0 (
            _Tp __q )
```

Compute and return the Jacobi θ_4 at zero argument by product expansion.

Definition at line 206 of file sf_theta.tcc.

9.3.1.193 __jacobi_theta_4_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_sum (
            _Tp __q,
            _Tp __x )
```

Return the Jacobi θ_4 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_4(q, x) = 2\sum_{n=1}^{\infty}(-1)^n q^{n^2}\cos 2nx$$

Definition at line 881 of file sf_theta.tcc.

Referenced by __jacobi_theta_4().

9.3.1.194 __jacobi_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__jacobi_zeros (
            unsigned int __n,
            _Tp __alpha1,
            _Tp __beta1 )
```

Return a vector containing the zeros of the Jacobi polynomial $P_n^{(\alpha,\beta)}$.

Definition at line 127 of file sf_jacobi.tcc.

References __gnu_cxx::lgamma().

9.3.1.195 __jacobi_zeta()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_zeta (
            _Tp __k,
            _Tp __phi )
```

Return the Jacobi zeta function.

Definition at line 949 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rj().

Referenced by __heuman_lambda().

9.3.1.196 __laguerre() [1/2]

```
template<typename _Tpa , typename _Tp >
_Tp std::__detail::__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 316 of file sf_laguerre.tcc.

References __laguerre_hyperg(), __laguerre_large_n(), and __laguerre_recur().

9.3.1.197 __laguerre() [2/2]

```
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 386 of file sf_laguerre.tcc.

9.3.1.198 __laguerre_hyperg()

```
template<typename _Tpa , typename _Tp >
_Tp std::__detail::__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 131 of file sf_laguerre.tcc.

Referenced by __laguerre().

9.3.1.199 __laguerre_large_n()

```
template<typename _Tpa , typename _Tp >
_Tp std::__detail::__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 75 of file sf_laguerre.tcc.

References __log_gamma(), and __sin_pi().

Referenced by __laguerre().

9.3.1.200 __laguerre_recur()

```
template<typename _Tpa , typename _Tp >
__gnu_cxx::__laguerre_t<_Tpa, _Tp> std::__detail::__laguerre_recur (
            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 189 of file sf_laguerre.tcc.

Referenced by __laguerre().

9.3.1.201 __laguerre_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__laguerre_zeros (
            unsigned int __n,
            _Tp __alpha1 )
```

Return an array of abscissae and weights for the Gauss-Laguerre rule.

Definition at line 225 of file sf_laguerre.tcc.

References __gnu_cxx::lgamma().

9.3.1.202 __lanczos_binet1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (
            _Tp __z )
```

Return the Binet function $J(1 + z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $log(\Gamma^*(z))$ defined by

$$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

Parameters

__z	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Definition at line 2102 of file sf_gamma.tcc.

References std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __lanczos_log_gamma1p().

9.3.1.203 __lanczos_log_gamma1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_log_gamma1p (
            _Tp __z )
```

Return the logarithm of the gamma function $log(\Gamma(1 + z))$ by the Lanczos method.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to __log_gamma_sign.

For complex argument the fully complex log of the gamma function is returned.

Parameters

__z	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Definition at line 2136 of file sf_gamma.tcc.

References __lanczos_binet1p(), and __sin_pi().

9.3.1.204 __legendre_p()

```
template<typename _Tp >
__gnu_cxx::__legendre_p_t<_Tp> std::__detail::__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:

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

This can be expressed as a series:

$$P_l(x) = \frac{1}{2^l l!} \sum_{k=0}^{\lfloor l/2 \rfloor} \frac{(-1)^k (2l - 2k)!}{k!(l - k)!(l - 2k)!} x^{l-2k}$$

Parameters

__l	The order of the Legendre polynomial. $l >= 0$.
__x	The argument of the Legendre polynomial.

Definition at line 82 of file sf_legendre.tcc.

Referenced by __assoc_legendre_p(), and __sph_legendre().

9.3.1.205 __legendre_q()

```
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 141 of file sf_legendre.tcc.

9.3.1.206 __legendre_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__legendre_zeros (
            unsigned int __l,
            _Tp proto = _Tp{} )
```

Build a list of zeros and weights for the Gauss-Legendre integration rule for the Legendre polynomial of degree `l`.

Definition at line 385 of file sf_legendre.tcc.

9.3.1.207 __log_binomial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial (
            unsigned int __n,
            unsigned int __k )
```

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

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

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

.

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 2411 of file sf_gamma.tcc.

References __log_gamma(), and std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __binomial().

9.3.1.208 __log_binomial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial (
            _Tp __nu,
            unsigned int __k )
```

Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^\infty \binom{\nu}{k} t^k$$

.

Parameters

__nu	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 2448 of file sf_gamma.tcc.

References __log_gamma(), and std::__detail::_Factorial_table< _Tp >::__n.

9.3.1.209 __log_binomial_sign() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial_sign (
```

```
            _Tp __nu,
            unsigned int __k )
```

Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

.

Parameters

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

Returns

The sign of the gamma function.

Definition at line 2479 of file sf_gamma.tcc.

References __log_gamma_sign(), and std::__detail::_Factorial_table< _Tp >::__n.

Referenced by __binomial().

9.3.1.210 __log_binomial_sign() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__log_binomial_sign (
            std::complex< _Tp > __nu,
            unsigned int __k )
```

Definition at line 2494 of file sf_gamma.tcc.

9.3.1.211 __log_double_factorial() [1/2]

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (
            _Tp __x )
```

Definition at line 1643 of file sf_gamma.tcc.

References __cos_pi(), and __log_gamma().

Referenced by __double_factorial(), and __log_double_factorial().

9.3.1.212 __log_double_factorial() [2/2]

```
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 1709 of file sf_gamma.tcc.

References __log_double_factorial(), std::__detail::_Factorial_table< _Tp >::__log_factorial, std::__detail::_Factorial_table< _Tp >::__n, _S_double_factorial_table, and _S_neg_double_factorial_table.

9.3.1.213 __log_factorial()

```
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 1633 of file sf_gamma.tcc.

References __log_gamma(), std::__detail::_Factorial_table< _Tp >::__n, and _S_factorial_table.

9.3.1.214 __log_falling_factorial()

```
template<typename _Tp >
_Tp std::__detail::__log_falling_factorial (
            _Tp __a,
            _Tp __nu )
```

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

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

In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

Many notations exist for this function:

$$(a)_\nu$$

,

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

, and others.

Definition at line 3027 of file sf_gamma.tcc.

References __falling_factorial(), __gnu_cxx::__fp_is_integer(), and __log_gamma().

9.3.1.215 __log_gamma() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_gamma (
            _Tp __a )
```

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

Parameters

_↩ _a	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Definition at line 2302 of file sf_gamma.tcc.

References __sin_pi(), and __spouge_log_gamma1p().

Referenced by __beta_inc(), __beta_lgamma(), __cyl_bessel_ij_series(), __falling_factorial(), __gamma(), __↩
gamma_cont_frac(), __gamma_series(), __hydrogen(), __hyperg(), __hyperg_reflect(), __laguerre_large_n(), __log_↩
binomial(), __log_double_factorial(), __log_factorial(), __log_falling_factorial(), __log_gamma(), __log_rising_factorial(), __polylog_exp_neg(), __polylog_exp_pos(), __psi(), __riemann_zeta(), __rising_factorial(), and __sph_legendre().

9.3.1.216 __log_gamma() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__log_gamma (
            std::complex< _Tp > __a )
```

Return $log(\Gamma(a))$ for complex argument.

Parameters

_↩ _a	The complex argument of the log of the gamma function.

Returns

The complex logarithm of the gamma function.

Definition at line 2337 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), std::__detail::_Factorial_table< _Tp >::__log_factorial, __log_gamma(), std::__detail::_Factorial_table< _Tp >::__n, __sin_pi(), __spouge_log_gamma1p(), and _S_factorial_table.

9.3.1.217 __log_gamma_bernoulli()

```
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 1736 of file sf_gamma.tcc.

9.3.1.218 __log_gamma_sign() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_gamma_sign (
            _Tp __a )
```

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

Parameters

_← _a	The argument of the gamma function.

Returns

The sign of the gamma function.

Definition at line 2378 of file sf_gamma.tcc.

Referenced by __beta_inc(), __beta_lgamma(), __falling_factorial(), __gamma(), __gamma_cont_frac(), __gamma_← series(), __hyperg(), __hyperg_reflect(), __log_binomial_sign(), and __rising_factorial().

9.3.1.219 __log_gamma_sign() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__log_gamma_sign (
            std::complex< _Tp > __a )
```

Definition at line 2390 of file sf_gamma.tcc.

9.3.1.220 __log_rising_factorial()

```
template<typename _Tp >
_Tp std::__detail::__log_rising_factorial (
            _Tp __a,
            _Tp __nu )
```

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

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

Thus this function returns

$$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[(a)_0] = 0$$

Many notations exist for this function:

$$(a)_\nu$$

(especially in the literature of special functions),

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

, and others.

Definition at line 3176 of file sf_gamma.tcc.

References __log_gamma(), and __rising_factorial().

9.3.1.221 __log_stirling_1()

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_1 (
            unsigned int __n,
            unsigned int __m )
```

Return the logarithm of the absolute value of Stirling number of the first kind.

Definition at line 318 of file sf_stirling.tcc.

9.3.1.222 __log_stirling_1_sign()

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_1_sign (
            unsigned int __n,
            unsigned int __m )  [inline]
```

Return the sign of the exponent of the logarithm of the Stirling number of the first kind.

Definition at line 336 of file sf_stirling.tcc.

9.3.1.223 __log_stirling_2()

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_2 (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the second kind.

Todo Look into asymptotic solutions.

Definition at line 178 of file sf_stirling.tcc.

9.3.1.224 __logint()

```
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 538 of file sf_expint.tcc.

References __expint().

9.3.1.225 __logistic_cdf()

```
template<typename _Tp >
_Tp std::__detail::__logistic_cdf (
```

```
            _Tp __a,
            _Tp __b,
            _Tp __x )
```

Return the logistic cumulative distribution function.

The formula for the logistic probability function is

$$cdf(x|a,b) = \frac{e^{(x-a)/b}}{1 + e^{(x-a)/b}}$$

where $b > 0$.

Definition at line 688 of file sf_distributions.tcc.

9.3.1.226 __logistic_pdf()

```
template<typename _Tp >
_Tp std::__detail::__logistic_pdf (
            _Tp __a,
            _Tp __b,
            _Tp __x )
```

Return the logistic probability density function.

The formula for the logistic probability density function is

$$p(x|a,b) = \frac{e^{(x-a)/b}}{b[1 + e^{(x-a)/b}]^2}$$

where $b > 0$.

Definition at line 670 of file sf_distributions.tcc.

9.3.1.227 __lognormal_cdf()

```
template<typename _Tp >
_Tp std::__detail::__lognormal_cdf (
            _Tp __mu,
            _Tp __sigma,
            _Tp __x )
```

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

$$F(x|\mu,\sigma) = \frac{1}{2}\left[1 - erf(\frac{\ln x - \mu}{\sqrt{2}\sigma})\right]$$

Definition at line 287 of file sf_distributions.tcc.

9.3.1.228 __lognormal_pdf()

```
template<typename _Tp >
_Tp std::__detail::__lognormal_pdf (
            _Tp __nu,
            _Tp __sigma,
            _Tp __x )
```

Return the lognormal probability density function.

The formula for the lognormal probability density function is

$$f(x|\mu,\sigma) = \frac{e^{(\ln x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 259 of file sf_distributions.tcc.

9.3.1.229 __normal_cdf()

```
template<typename _Tp >
_Tp std::__detail::__normal_cdf (
            _Tp __mu,
            _Tp __sigma,
            _Tp __x )
```

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

$$F(x|\mu,\sigma) = \frac{1}{2}\left[1 - erf(\frac{x-\mu}{\sqrt{2}\sigma})\right]$$

Definition at line 238 of file sf_distributions.tcc.

9.3.1.230 __normal_pdf()

```
template<typename _Tp >
_Tp std::__detail::__normal_pdf (
            _Tp __mu,
            _Tp __sigma,
            _Tp __x )
```

Return the normal probability density function.

The formula for the normal probability density function is

$$f(x|\mu,\sigma) = \frac{e^{(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 210 of file sf_distributions.tcc.

9.3.1.231 __owens_t()

```
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().

9.3.1.232 __pgamma()

```
template<typename _Tp >
_Tp std::__detail::__pgamma (
            _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 2782 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __chi_squared_pdf().

9.3.1.233 __polar_pi() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polar_pi (
            _Tp __rho,
            _Tp __phi_pi )  [inline]
```

Reperiodized complex constructor.

Definition at line 397 of file sf_trig.tcc.

References __gnu_cxx::__sincos_t< _Tp >::__cos_v, __gnu_cxx::__sincos_t< _Tp >::__sin_v, and __sincos_pi().

Referenced by __cyl_bessel_jn_neg_arg(), __cyl_hankel_1(), __cyl_hankel_2(), __jacobi_theta_1(), __jacobi_theta_↩
2(), __polylog_exp_neg(), and __polylog_exp_pos().

9.3.1.234 __polar_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polar_pi (
            _Tp __rho,
            const std::complex< _Tp > & __phi_pi )  [inline]
```

Reperiodized complex constructor.

Definition at line 409 of file sf_trig.tcc.

References __gnu_cxx::__sincos_t< _Tp >::__cos_v, __gnu_cxx::__sincos_t< _Tp >::__sin_v, and __sincos_pi().

9.3.1.235 __poly_radial_jacobi()

```
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.
Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

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 264 of file sf_jacobi.tcc.

References __jacobi_recur().

Referenced by __zernike(), __gnu_cxx::radpolyf(), and __gnu_cxx::radpolyl().

9.3.1.236 __polylog() [1/2]

```
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 1025 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_equal(), __gnu_cxx::__fp_is_integer(), __gnu_cxx::__fp_is_zero(), and __polylog_exp().

Referenced by __dirichlet_beta(), __dirichlet_eta(), and __polylog().

9.3.1.237 __polylog() [2/2]

```
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 1066 of file sf_polylog.tcc.

References __polylog(), and __polylog_exp().

9.3.1.238 __polylog_exp()

```
template<typename _Tp , typename _ArgType >
__gnu_cxx::__promote_fp_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 989 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_integer(), __polylog_exp_neg_int(), __polylog_exp_neg_real(), __polylog_exp_pos_int(), __polylog_exp_pos_real(), and __polylog_exp_sum().

Referenced by __bose_einstein(), __clausen(), __clausen_cl(), __clausen_sl(), __fermi_dirac(), __hurwitz_zeta_polylog(), and __polylog().

9.3.1.239 __polylog_exp_asymp()

```
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 w the imaginary part of the polylog is given by $Im(Li_s(e^w)) = -\pi w^{s-1}/\Gamma(s)$. Check this relation for any benchmark that you use.

Parameters

__s	the real index s.
__w	the large complex argument w.

Returns

the value of the polylogarithm.

Definition at line 602 of file sf_polylog.tcc.

References __gamma_reciprocal().

Referenced by __polylog_exp_neg_int(), __polylog_exp_neg_real(), __polylog_exp_pos_int(), and __polylog_exp_pos_real().

9.3.1.240 __polylog_exp_neg() [1/2]

```
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

$$Li_s(e^w) = \Gamma(1-s)(-w)^{s-1} + \frac{(2\pi)^{-s}}{\pi}A_p(w)$$
$$A_p(w) = \sum_k \frac{\Gamma(1+k-s)}{k!}\sin\left(\frac{\pi}{2}(s-k)\right)\left(\frac{w}{2\pi}\right)^k\zeta(1+k-s)$$

Parameters

__s	The negative real index
__w	The complex argument

Returns

The value of the polylogarithm.

Definition at line 366 of file sf_polylog.tcc.

References __log_gamma(), __polar_pi(), and __riemann_zeta_m_1().

Referenced by __polylog_exp_neg_int(), and __polylog_exp_neg_real().

9.3.1.241 __polylog_exp_neg() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg (
            int __n,
            std::complex< _Tp > __w )
```

Compute the polylogarithm for negative integer order.

$$Li_{-p}(e^w) = p!(-w)^{-(p+1)} - \sum_{k=0}^{\infty} \frac{B_{p+2k+q+1}}{(p+2k+q+1)!} \frac{(p+2k+q)!}{(2k+q)!} w^{2k+q}$$

where $q = (p+1) mod 2$.

Parameters

_↩ _n	the negative integer index $n = -p$.
_↩ _w	the argument w.

Returns

the value of the polylogarithm.

Definition at line 452 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_equal(), __gnu_cxx::__fp_is_zero(), _Num_Euler_Maclaurin_zeta, and _S_Euler_↩ Maclaurin_zeta.

9.3.1.242 __polylog_exp_neg_int() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_int (
            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 784 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __gnu_cxx::__fp_is_equal(), __polylog_exp_asymp(), __polylog_exp_↩
neg(), and __polylog_exp_sum().

Referenced by __polylog_exp().

9.3.1.243 __polylog_exp_neg_int() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_int (
            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 828 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_zero(), __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_sum().

9.3.1.244 __polylog_exp_neg_real() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_real (
            _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 929 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_sum().

Referenced by __polylog_exp().

9.3.1.245 __polylog_exp_neg_real() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_real (
            _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 960 of file sf_polylog.tcc.

References __polylog_exp_asymp(), __polylog_exp_neg(), and __polylog_exp_sum().

9.3.1.246 __polylog_exp_pos() [1/3]

```
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 for complex argument w.

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

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: 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 positive integer index.
__w	the argument.

Returns

the value of the polylogarithm.

Definition at line 218 of file sf_polylog.tcc.

References __riemann_zeta().

Referenced by __polylog_exp_pos_int(), and __polylog_exp_pos_real().

9.3.1.247 __polylog_exp_pos() [2/3]

```
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 argument w.

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

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

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: 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 positive integer index.
__w	the argument.

Returns

the value of the polylogarithm.

Definition at line 294 of file sf_polylog.tcc.

References __riemann_zeta().

9.3.1.248 __polylog_exp_pos() [3/3]

```
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 515 of file sf_polylog.tcc.

References __gamma(), __log_gamma(), __polar_pi(), and __riemann_zeta().

9.3.1.249 __polylog_exp_pos_int() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_int (
            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 677 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __gnu_cxx::__fp_is_equal(), __gnu_cxx::__fp_is_zero(), __polylog_↩
exp_asymp(), __polylog_exp_pos(), and __polylog_exp_sum().

Referenced by __polylog_exp().

9.3.1.250 __polylog_exp_pos_int() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_int (
            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 736 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_zero(), __polylog_exp_asymp(), __polylog_exp_pos(), and __polylog_exp_sum().

9.3.1.251 __polylog_exp_pos_real() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_real (
            _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 855 of file sf_polylog.tcc.

References __clamp_0_m2pi(), __clamp_pi(), __gnu_cxx::__fp_is_equal(), __gnu_cxx::__fp_is_zero(), __polylog_↩ exp_asymp(), __polylog_exp_pos(), __polylog_exp_sum(), and __riemann_zeta().

Referenced by __polylog_exp().

9.3.1.252 __polylog_exp_pos_real() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_real (
            _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 895 of file sf_polylog.tcc.

References __gnu_cxx::__fp_is_equal(), __gnu_cxx::__fp_is_zero(), __polylog_exp_asymp(), __polylog_exp_pos(), ↩ __polylog_exp_sum(), and __riemann_zeta().

9.3.1.253 __polylog_exp_sum()

```
template<typename _PowTp , typename _Tp >
_Tp std::__detail::__polylog_exp_sum (
            _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:

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

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 646 of file sf_polylog.tcc.

Referenced by __polylog_exp(), __polylog_exp_neg_int(), __polylog_exp_neg_real(), __polylog_exp_pos_int(), and ↩
__polylog_exp_pos_real().

9.3.1.254 __prob_hermite_recursion()

```
template<typename _Tp >
__gnu_cxx::__hermite_he_t<_Tp> std::__detail::__prob_hermite_recursion (
            unsigned int __n,
            _Tp __x )
```

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

The Hermite polynomial is defined by:

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

or

$$He_n(x) = \frac{1}{2^{-n/2}} H_n\left(\frac{x}{\sqrt{2}}\right)$$

where $H_n(x)$ is the Physicists Hermite function.

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 218 of file sf_hermite.tcc.

9.3.1.255 __psi() [1/3]

```
template<typename _Tp >
_Tp std::__detail::__psi (
            unsigned int __n )
```

Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

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

The digamma series for integral argument is given by:

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

The latter sum is called the harmonic number, H_n.

Definition at line 3294 of file sf_gamma.tcc.

Referenced by __expint_En_series(), __hydrogen(), __hyperg_reflect(), and __psi().

9.3.1.256 __psi() [2/3]

```
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 3380 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_half_odd_integer(), __gnu_cxx::__fp_is_integer(), std::__detail::_Factorial_table< _Tp >::__n, __psi(), __psi_asymp(), and __tan_pi().

9.3.1.257 __psi() [3/3]

```
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 3436 of file sf_gamma.tcc.

References __hurwitz_zeta(), __log_gamma(), and __psi().

9.3.1.258 __psi_asymp()

```
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:

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

Definition at line 3349 of file sf_gamma.tcc.

Referenced by __psi().

9.3.1.259 __psi_series()

```
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 3318 of file sf_gamma.tcc.

9.3.1.260 __qgamma()

```
template<typename _Tp >
_Tp std::__detail::__qgamma (
            _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 2816 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __chi_squared_pdfc().

9.3.1.261 __rice_pdf()

```
template<typename _Tp >
_Tp std::__detail::__rice_pdf (
            _Tp __nu,
            _Tp __sigma,
            _Tp __x )
```

Return the Rice probability density function.

The formula for the Rice probability density function is

$$p(x|\nu,\sigma) = \frac{x}{\sigma^2}\exp\left(-\frac{x^2+\nu^2}{2\sigma^2}\right)I_0\left(\frac{x\nu}{\sigma^2}\right)$$

where $I_0(x)$ is the modified Bessel function of the first kind of order 0 and $\nu >= 0$ and $\sigma > 0$.

Definition at line 186 of file sf_distributions.tcc.

References __cyl_bessel_i().

9.3.1.262 __riemann_zeta()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta (
            _Tp __s )
```

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

The Riemann zeta function is defined by:

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

Parameters

__s	The argument

Todo Global double sum or MacLaurin series in riemann_zeta?

Definition at line 665 of file sf_zeta.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma(), __log_gamma(), __riemann_zeta_glob(), __riemann_zeta_m_1(), __riemann_zeta_product(), __riemann_zeta_sum(), and __sin_pi().

Referenced by __dirichlet_lambda(), __hurwitz_zeta(), __polylog_exp_pos(), and __polylog_exp_pos_real().

9.3.1.263 __riemann_zeta_euler_maclaurin()

```
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 300 of file sf_zeta.tcc.

References _S_Euler_Maclaurin_zeta.

9.3.1.264 __riemann_zeta_glob()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_glob (
            _Tp __s )
```

Definition at line 410 of file sf_zeta.tcc.

References __gnu_cxx::__fp_is_even_integer(), __gamma(), __riemann_zeta_m_1_glob(), and __sin_pi().

Referenced by __riemann_zeta().

9.3.1.265 __riemann_zeta_m_1()

```
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 630 of file sf_zeta.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma(), __riemann_zeta_m_1_glob(), __sin_pi(), _S_num_zetam1, and _S_zetam1.

Referenced by __polylog_exp_neg(), and __riemann_zeta().

9.3.1.266 __riemann_zeta_m_1_glob()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_m_1_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:

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

Definition at line 359 of file sf_zeta.tcc.

Referenced by __riemann_zeta_glob(), and __riemann_zeta_m_1().

9.3.1.267 __riemann_zeta_product()

```
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:

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

For (s) $<$ 1 use the reflection formula:

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

Parameters

__s	The argument

Definition at line 462 of file sf_zeta.tcc.

Referenced by __riemann_zeta().

9.3.1.268 __riemann_zeta_sum()

```
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:

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

Definition at line 257 of file sf_zeta.tcc.

References __gamma(), and __sin_pi().

Referenced by __riemann_zeta().

9.3.1.269 __rising_factorial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__rising_factorial (
            _Tp __a,
            int __n )
```

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

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

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

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

, and others.

Definition at line 3077 of file sf_gamma.tcc.

References __log_gamma(), __log_gamma_sign(), and std::__detail::_Factorial_table$<$ _Tp $>$::__n.

Referenced by __log_rising_factorial(), and __rising_factorial().

9.3.1.270 __rising_factorial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__rising_factorial (
            _Tp __a,
            _Tp __nu )
```

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

$$a^{\overline{\nu}} = \Gamma(a + \nu)/\Gamma(\nu)$$

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

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

, and others.

Definition at line 3132 of file sf_gamma.tcc.

References __log_gamma(), __log_gamma_sign(), std::__detail::_Factorial_table< _Tp >::__n, and __rising_factorial().

9.3.1.271 __sin_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sin_pi (
            _Tp __x )
```

Return the reperiodized sine of argument x:

$$\sin_\pi(x) = \sin(\pi x)$$

Definition at line 52 of file sf_trig.tcc.

Referenced by __cos_pi(), __cosh_pi(), __cyl_bessel_ik(), __cyl_bessel_jn(), __dirichlet_eta(), __gamma_reciprocal(), __hankel_debye(), __laguerre_large_n(), __lanczos_log_gamma1p(), __log_gamma(), __riemann_zeta(), __riemann_zeta_glob(), __riemann_zeta_m_1(), __riemann_zeta_sum(), __sin_pi(), __sinc_pi(), __sinh_pi(), and __spouge_log_gamma1p().

9.3.1.272 __sin_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sin_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized sine of complex argument z:

$$\sin_\pi(z) = \sin(\pi z) = \sin_\pi(x)\cosh_\pi(y) + i\cos_\pi(x)\sinh_\pi(y)$$

Definition at line 183 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.1.273 __sinc()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc (
            _Tp __x )
```

Return the sinus cardinal function

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

.

Definition at line 52 of file sf_cardinal.tcc.

9.3.1.274 __sinc_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc_pi (
            _Tp __x )
```

Return the reperiodized sinus cardinal function

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

.

Definition at line 72 of file sf_cardinal.tcc.

References __sin_pi().

9.3.1.275 __sincos() [1/4]

```
template<typename _Tp >
__gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos (
            _Tp __x ) [inline]
```

Definition at line 312 of file sf_trig.tcc.

Referenced by __sincos_pi().

9.3.1.276 __sincos() [2/4]

```
template<>
__gnu_cxx::__sincos_t<float> std::__detail::__sincos (
            float __x ) [inline]
```

Definition at line 320 of file sf_trig.tcc.

9.3.1.277 __sincos() [3/4]

```
template<>
__gnu_cxx::__sincos_t<double> std::__detail::__sincos (
            double __x ) [inline]
```

Definition at line 332 of file sf_trig.tcc.

9.3.1.278 __sincos() [4/4]

```
template<>
__gnu_cxx::__sincos_t<long double> std::__detail::__sincos (
            long double __x ) [inline]
```

Definition at line 344 of file sf_trig.tcc.

9.3.1.279 __sincos_pi()

```
template<typename _Tp >
__gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos_pi (
            _Tp __x )
```

Reperiodized sincos.

Definition at line 356 of file sf_trig.tcc.

References __gnu_cxx::__sincos_t< _Tp >::__cos_v, __gnu_cxx::__sincos_t< _Tp >::__sin_v, and __sincos().

Referenced by __polar_pi().

9.3.1.280 __sincosint()

```
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 226 of file sf_trigint.tcc.

References __sincosint_asymp(), __sincosint_cont_frac(), and __sincosint_series().

9.3.1.281 __sincosint_asymp()

```
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 159 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.1.282 __sincosint_cont_frac()

```
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 52 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.1.283 __sincosint_series()

```
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 95 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.1.284 __sinh_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sinh_pi (
            _Tp __x )
```

Return the reperiodized hyperbolic sine of argument x:

$$\sinh_\pi(x) = \sinh(\pi x)$$

Definition at line 83 of file sf_trig.tcc.

Referenced by __sinhc_pi().

9.3.1.285 __sinh_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sinh_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized hyperbolic sine of complex argument z:

$$\sinh_\pi(z) = \sinh(\pi z) = \sinh_\pi(x)\cos_\pi(y) + i\cosh_\pi(x)\sin_\pi(y)$$

Definition at line 205 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.1.286 __sinhc()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc (
             _Tp __x )
```

Return the hyperbolic sinus cardinal function

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

.

Definition at line 97 of file sf_cardinal.tcc.

9.3.1.287 __sinhc_pi()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc_pi (
             _Tp __x )
```

Return the reperiodized hyperbolic sinus cardinal function

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

.

Definition at line 115 of file sf_cardinal.tcc.

References __sinh_pi().

9.3.1.288 __sinhint()

```
template<typename _Tp >
_Tp std::__detail::__sinhint (
             const _Tp __x )
```

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

The hyperbolic sine integral is given by

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

Parameters

_↩ _x	The argument of the hyperbolic sine integral function.

Returns

The hyperbolic sine integral.

Definition at line 584 of file sf_expint.tcc.

References __expint_E1(), and __expint_Ei().

9.3.1.289 __sph_bessel() [1/2]

```
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 746 of file sf_bessel.tcc.

References __sph_bessel_jn().

9.3.1.290 __sph_bessel() [2/2]

```
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 1274 of file sf_hankel.tcc.

References __sph_hankel().

9.3.1.291 __sph_bessel_ik()

```
template<typename _Tp >
__gnu_cxx::__sph_mod_bessel_t<unsigned int, _Tp, _Tp> std::__detail::__sph_bessel_ik (
            unsigned int __n,
            _Tp __x )
```

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.

Returns

A struct containing the modified spherical Bessel functions of the first and second kinds and their derivatives.

Definition at line 419 of file sf_mod_bessel.tcc.

References __cyl_bessel_ik().

9.3.1.292 __sph_bessel_jn()

```
template<typename _Tp >
__gnu_cxx::__sph_bessel_t<unsigned int, _Tp, _Tp> std::__detail::__sph_bessel_jn (
            unsigned int __n,
            _Tp __x )
```

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.

Returns

The output derivative of the spherical Neumann function.

Definition at line 678 of file sf_bessel.tcc.

References __cyl_bessel_jn().

Referenced by __sph_bessel(), __sph_hankel_1(), __sph_hankel_2(), and __sph_neumann().

9.3.1.293 __sph_bessel_jn_neg_arg()

```
template<typename _Tp >
__gnu_cxx::__sph_bessel_t<unsigned int, _Tp, std::complex<_Tp> > std::__detail::__sph_bessel_jn_neg_arg (
            unsigned int __n,
            _Tp __x )
```

Return the spherical Bessel functions and their derivatives of order ν and argument $x < 0$.

Definition at line 702 of file sf_bessel.tcc.

References __cyl_bessel_jn_neg_arg().

Referenced by __sph_hankel_1(), and __sph_hankel_2().

9.3.1.294 __sph_hankel()

```
template<typename _Tp >
__gnu_cxx::__sph_hankel_t<unsigned int, std::complex<_Tp>, std::complex<_Tp> > std::__detail::↩
__sph_hankel (
            unsigned int __n,
            std::complex< _Tp > __z )
```

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.

Returns

A struct containing the spherical Hankel functions of the first and second kinds and their derivatives.

Definition at line 1210 of file sf_hankel.tcc.

References __hankel().

Referenced by __sph_bessel(), __sph_hankel_1(), __sph_hankel_2(), and __sph_neumann().

9.3.1.295 __sph_hankel_1() [1/2]

```
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 807 of file sf_bessel.tcc.

References __sph_bessel_jn(), and __sph_bessel_jn_neg_arg().

9.3.1.296 __sph_hankel_1() [2/2]

```
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 1240 of file sf_hankel.tcc.

References __sph_hankel().

9.3.1.297 __sph_hankel_2() [1/2]

```
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 842 of file sf_bessel.tcc.

References __sph_bessel_jn(), and __sph_bessel_jn_neg_arg().

9.3.1.298 __sph_hankel_2() [2/2]

```
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 1257 of file sf_hankel.tcc.

References __sph_hankel().

9.3.1.299 __sph_harmonic()

```
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 368 of file sf_legendre.tcc.

References __sph_legendre().

9.3.1.300 __sph_legendre()

```
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 271 of file sf_legendre.tcc.

References __legendre_p(), and __log_gamma().

Referenced by __hydrogen(), and __sph_harmonic().

9.3.1.301 __sph_neumann() [1/2]

```
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 779 of file sf_bessel.tcc.

References __sph_bessel_jn().

9.3.1.302 __sph_neumann() [2/2]

```
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 1291 of file sf_hankel.tcc.

References __sph_hankel().

9.3.1.303 __spouge_binet1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_binet1p (
            _Tp __z )
```

Return the Binet function $J(1+z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $log(\Gamma^*(z))$ defined by

$$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

Parameters

_↩ _z	The argument of the log of the gamma function.

Returns

The logarithm of the gamma function.

Definition at line 1918 of file sf_gamma.tcc.

Referenced by __spouge_log_gamma1p().

9.3.1.304 __spouge_log_gamma1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_log_gamma1p (
            _Tp __z )
```

Return the logarithm of the gamma function $log(\Gamma(1+z))$ 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

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

.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to __log_gamma_sign.

For complex argument the fully complex log of the gamma function is returned.

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 1962 of file sf_gamma.tcc.

References __sin_pi(), and __spouge_binet1p().

Referenced by __log_gamma().

9.3.1.305 __stirling_1()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1 (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the Pocchammer polynomials:

$$(x)_n = \sum_{k=0}^{n} S_n^{(k)} x^k$$

The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - nS_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0\rightarrow m)} = 1, 0, 0, ..., 0$$

and

$$S_{0\rightarrow n}^{(0)} = 1, 0, 0, ..., 0$$

Todo Find asymptotic solutions for the Stirling numbers of the first kind.

Develop an iterator model for Stirling numbers of the first kind.

Definition at line 300 of file sf_stirling.tcc.

9.3.1.306 __stirling_1_recur()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1_recur (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the first kind by recursion. The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - nS_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0\rightarrow m)} = 1, 0, 0, ..., 0$$

and

$$S_{0\rightarrow n}^{(0)} = 1, 0, 0, ..., 0$$

Definition at line 251 of file sf_stirling.tcc.

9.3.1.307 __stirling_1_series()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1_series (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the first kind by series expansion. N.B. This seems to be a total disaster.

Definition at line 196 of file sf_stirling.tcc.

References __gnu_cxx::__parity().

9.3.1.308 __stirling_2()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2 (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \sum_{k=0}^{m} \frac{(-1)^{m-k} k^n}{(m-k)!k!}$$

Todo Find asymptotic solutions for Stirling numbers of the second kind.

Develop an iterator model for Stirling numbers of the second kind.

Definition at line 159 of file sf_stirling.tcc.

9.3.1.309 __stirling_2_recur()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2_recur (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the second kind by recursion. The recursion is

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = m \left\{ \begin{matrix} n-1 \\ m \end{matrix} \right\} + \left\{ \begin{matrix} n-1 \\ m-1 \end{matrix} \right\}$$

with starting values

$$\left\{ \begin{matrix} 0 \\ 0 \rightarrow m \end{matrix} \right\} = 1, 0, 0, ..., 0$$

and

$$\left\{ \begin{matrix} 0 \rightarrow n \\ 0 \end{matrix} \right\} = 1, 0, 0, ..., 0$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Definition at line 122 of file sf_stirling.tcc.

9.3.1.310 __stirling_2_series()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2_series (
            unsigned int __n,
            unsigned int __m )
```

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^{m} \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Todo Find a way to predict the maximum Stirling number for a type.

Definition at line 67 of file sf_stirling.tcc.

9.3.1.311 __student_t_cdf()

```
template<typename _Tp >
_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 444 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.312 __student_t_cdfc()

```
template<typename _Tp >
_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 467 of file sf_distributions.tcc.

References __beta_inc().

9.3.1.313 __student_t_pdf()

```
template<typename _Tp >
_Tp std::__detail::__student_t_pdf (
            _Tp __t,
            unsigned int __nu )
```

Return the Students T probability density.

The students T propability density is:

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

Parameters

__t	
__nu	

Definition at line 419 of file sf_distributions.tcc.

References __gamma().

9.3.1.314 __tan_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__tan_pi (
            _Tp __x )
```

Return the reperiodized tangent of argument x:

$$\tan_p i(x) = \tan(\pi x)$$

Definition at line 149 of file sf_trig.tcc.

Referenced by __psi(), __tan_pi(), and __tanh_pi().

9.3.1.315 __tan_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__tan_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized tangent of complex argument z:

$$\tan_\pi(z) = \tan(\pi z) = \frac{\tan_\pi(x) + i\tanh_\pi(y)}{1 - i\tan_\pi(x)\tanh_\pi(y)}$$

Definition at line 271 of file sf_trig.tcc.

References __tan_pi().

9.3.1.316 __tanh_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__tanh_pi (
            _Tp __x )
```

Return the reperiodized hyperbolic tangent of argument x:

$$\tanh_\pi(x) = \tanh(\pi x)$$

Definition at line 165 of file sf_trig.tcc.

9.3.1.317 __tanh_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__tanh_pi (
            std::complex< _Tp > __z )
```

Return the reperiodized hyperbolic tangent of complex argument z:

$$\tanh_\pi(z) = \tanh(\pi z) = \frac{\tanh_\pi(x) + i\tan_\pi(y)}{1 + i\tanh_\pi(x)\tan_\pi(y)}$$

Definition at line 294 of file sf_trig.tcc.

References __tan_pi().

9.3.1.318 __tgamma()

```
template<typename _Tp >
_Tp std::__detail::__tgamma (
            _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 2880 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __gamma_cdfc().

9.3.1.319 __tgamma_lower()

```
template<typename _Tp >
_Tp std::__detail::__tgamma_lower (
            _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 2845 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __gamma_cdf().

9.3.1.320 __theta_1()

```
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 exponential 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 265 of file sf_theta.tcc.

References __gnu_cxx::__fp_is_zero(), and __theta_2().

Referenced by __theta_s().

9.3.1.321 __theta_2()

```
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 237 of file sf_theta.tcc.

References __theta_2_asymp(), and __theta_2_sum().

Referenced by __theta_1(), and __theta_c().

9.3.1.322 __theta_2_asymp()

```
template<typename _Tp >
_Tp std::__detail::__theta_2_asymp (
            _Tp __nu,
            _Tp __x )
```

Compute and return the exponential θ_2 function by asymptotic series expansion.

Definition at line 108 of file sf_theta.tcc.

Referenced by __theta_2().

9.3.1.323 __theta_2_sum()

```
template<typename _Tp >
_Tp std::__detail::__theta_2_sum (
            _Tp __nu,
            _Tp __x )
```

Compute and return the exponential θ_2 function by series expansion.

Definition at line 52 of file sf_theta.tcc.

Referenced by __theta_2().

9.3.1.324 __theta_3()

```
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 293 of file sf_theta.tcc.

References __theta_3_asymp(), and __theta_3_sum().

Referenced by __theta_4(), and __theta_d().

9.3.1.325 __theta_3_asymp()

```
template<typename _Tp >
_Tp std::__detail::__theta_3_asymp (
            _Tp __nu,
            _Tp __x )
```

Compute and return the exponential θ_3 function by asymptotic series expansion.

Definition at line 134 of file sf_theta.tcc.

Referenced by __theta_3().

9.3.1.326 __theta_3_sum()

```
template<typename _Tp >
_Tp std::__detail::__theta_3_sum (
            _Tp __nu,
            _Tp __x )
```

Compute and return the exponential θ_3 function by series expansion.

Definition at line 81 of file sf_theta.tcc.

Referenced by __theta_3().

9.3.1.327 __theta_4()

```
template<typename _Tp >
_Tp std::__detail::__theta_4 (
            _Tp __nu,
            _Tp __x )
```

Return the exponential theta-4 function of period `nu` and argument `x`.

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 321 of file sf_theta.tcc.

References __theta_3().

Referenced by __theta_n().

9.3.1.328 __theta_c()

```
template<typename _Tp >
_Tp std::__detail::__theta_c (
            _Tp __k,
            _Tp __x )
```

Return the Neville θ_c function

$$\theta_c(k,x) = \sqrt{\frac{\pi}{2kK(k)}}\theta_1\left(q(k),\frac{\pi x}{2K(k)}\right)$$

Definition at line 435 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_2().

9.3.1.329 __theta_d()

```
template<typename _Tp >
_Tp std::__detail::__theta_d (
            _Tp __k,
            _Tp __x )
```

Return the Neville θ_d function

$$\theta_d(k,x) = \sqrt{\frac{\pi}{2K(k)}}\theta_3\left(q(k),\frac{\pi x}{2K(k)}\right)$$

Definition at line 464 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_3().

9.3.1.330 __theta_n()

```
template<typename _Tp >
_Tp std::__detail::__theta_n (
            _Tp __k,
            _Tp __x )
```

Return the Neville θ_n function

The Neville theta-n function is defined by

$$\theta_n(k,x) = \sqrt{\frac{\pi}{2k'K(k)}}\theta_4\left(q(k), \frac{\pi x}{2K(k)}\right)$$

Definition at line 495 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_4().

9.3.1.331 __theta_s()

```
template<typename _Tp >
_Tp std::__detail::__theta_s (
            _Tp __k,
            _Tp __x )
```

Return the Neville θ_s function

$$\theta_s(k,x) = \sqrt{\frac{\pi}{2kk'K(k)}}\theta_1\left(q(k), \frac{\pi x}{2K(k)}\right)$$

Definition at line 405 of file sf_theta.tcc.

References __comp_ellint_1(), __ellnome(), and __theta_1().

9.3.1.332 __tricomi_u()

```
template<typename _Tp >
_Tp std::__detail::__tricomi_u (
            _Tp __a,
            _Tp __c,
            _Tp __x )
```

Return the Tricomi confluent hypergeometric function

$$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)}{}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)}x^{1-c}{}_1F_1(a-c+1;2-c;x)$$

.

Parameters

_↩ _a	The *numerator* parameter.
_↩ _c	The *denominator* parameter.
_↩ _x	The argument of the confluent hypergeometric function.

Returns

The Tricomi confluent hypergeometric function.

Definition at line 346 of file sf_hyperg.tcc.

References __tricomi_u_naive().

9.3.1.333 __tricomi_u_naive()

```
template<typename _Tp >
_Tp std::__detail::__tricomi_u_naive (
            _Tp __a,
            _Tp __c,
            _Tp __x )
```

Return the Tricomi confluent hypergeometric function

$$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)}{}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)}x^{1-c}{}_1F_1(a-c+1;2-c;x)$$

.

Parameters

_↩ _a	The *numerator* parameter.
_↩ _c	The *denominator* parameter.
_↩ _x	The argument of the confluent hypergeometric function.

Returns

The Tricomi confluent hypergeometric function.

Definition at line 312 of file sf_hyperg.tcc.

References __conf_hyperg(), __gnu_cxx::__fp_is_integer(), and __gnu_cxx::tgamma().

Referenced by __tricomi_u().

9.3.1.334 __weibull_cdf()

```
template<typename _Tp >
_Tp std::__detail::__weibull_cdf (
            _Tp __a,
            _Tp __b,
            _Tp __x )
```

Return the Weibull cumulative probability density function.

The formula for the Weibull cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-(x/b)^a} \text{ for } x >= 0$$

Definition at line 395 of file sf_distributions.tcc.

9.3.1.335 __weibull_pdf()

```
template<typename _Tp >
_Tp std::__detail::__weibull_pdf (
            _Tp __a,
            _Tp __b,
            _Tp __x )
```

Return the Weibull probability density function.

The formula for the Weibull probability density function is

$$f(x|a,b) = \frac{a}{b}\left(\frac{x}{b}\right)^{a-1}\exp-\left(\frac{x}{b}\right)^{a} \text{ for } x >= 0$$

Definition at line 374 of file sf_distributions.tcc.

9.3.1.336 __zernike()

```
template<typename _Tp >
__gnu_cxx::__promote_fp_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 integral degree n, signed integral 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).
Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

Template Parameters

_Tp	The real type of the radial coordinate and azimuthal angle

Parameters

__n	The non-negative integral degree.
__m	The integral azimuthal order
__rho	The radial coordinate
__phi	The azimuthal angle

Definition at line 313 of file sf_jacobi.tcc.

References __poly_radial_jacobi().

9.3.1.337 __znorm1()

```
template<typename _Tp >
_Tp std::__detail::__znorm1 (
            _Tp __x )
```

Definition at line 58 of file sf_owens_t.tcc.

Referenced by __owens_t().

9.3.1.338 __znorm2()

```
template<typename _Tp >
_Tp std::__detail::__znorm2 (
            _Tp __x )
```

Definition at line 47 of file sf_owens_t.tcc.

Referenced by __owens_t().

9.3.2 Variable Documentation

9.3.2.1 __max_FGH

```
template<typename _Tp >
constexpr int std::__detail::__max_FGH = _Airy_series<_Tp>::_N_FGH
```

Definition at line 179 of file sf_airy.tcc.

9.3.2.2 __max_FGH< double >

```
template<>
constexpr int std::__detail::__max_FGH< double > = 79
```

Definition at line 185 of file sf_airy.tcc.

9.3.2.3 __max_FGH< float >

```
template<>
constexpr int std::__detail::__max_FGH< float > = 15
```

Definition at line 182 of file sf_airy.tcc.

9.3.2.4 _Num_Euler_Maclaurin_zeta

```
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 67 of file sf_zeta.tcc.

Referenced by __polylog_exp_neg().

9.3.2.5 _S_double_factorial_table

```
constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]
```

Definition at line 278 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

9.3.2.6 _S_Euler_Maclaurin_zeta

```
constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]
```

Definition at line 70 of file sf_zeta.tcc.

Referenced by __hurwitz_zeta_euler_maclaurin(), __polylog_exp_neg(), and __riemann_zeta_euler_maclaurin().

9.3.2.7 _S_factorial_table

```
constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]
```

Definition at line 88 of file sf_gamma.tcc.

Referenced by __factorial(), __gamma(), __gamma_reciprocal(), __log_factorial(), and __log_gamma().

9.3.2.8 _S_harmonic_denom

```
constexpr unsigned long long std::__detail::_S_harmonic_denom[_S_num_harmonic_numer]
```

Definition at line 3229 of file sf_gamma.tcc.

Referenced by __harmonic_number().

9.3.2.9 _S_harmonic_numer

```
constexpr unsigned long long std::__detail::_S_harmonic_numer[_S_num_harmonic_numer]
```

Definition at line 3196 of file sf_gamma.tcc.

Referenced by __harmonic_number().

9.3.2.10 _S_neg_double_factorial_table

```
constexpr _Factorial_table<long double> std::__detail::_S_neg_double_factorial_table[999]
```

Definition at line 599 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

9.3.2.11 _S_num_double_factorials

```
template<typename _Tp >
constexpr std::size_t std::__detail::_S_num_double_factorials = 0
```

Definition at line 263 of file sf_gamma.tcc.

9.3.2.12 _S_num_double_factorials< double >

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301
```

Definition at line 268 of file sf_gamma.tcc.

9.3.2.13 _S_num_double_factorials< float >

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57
```

Definition at line 266 of file sf_gamma.tcc.

9.3.2.14 _S_num_double_factorials< long double >

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301
```

Definition at line 270 of file sf_gamma.tcc.

9.3.2.15 _S_num_factorials

```
template<typename _Tp >
constexpr std::size_t std::__detail::_S_num_factorials = 0
```

Definition at line 73 of file sf_gamma.tcc.

9.3.2.16 _S_num_factorials< double >

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< double > = 171
```

Definition at line 78 of file sf_gamma.tcc.

9.3.2.17 _S_num_factorials< float >

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< float > = 35
```

Definition at line 76 of file sf_gamma.tcc.

9.3.2.18 _S_num_factorials< long double >

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171
```

Definition at line 80 of file sf_gamma.tcc.

9.3.2.19 _S_num_harmonic_numer

```
constexpr unsigned long long std::__detail::_S_num_harmonic_numer = 29
```

Definition at line 3193 of file sf_gamma.tcc.

Referenced by __harmonic_number().

9.3.2.20 _S_num_neg_double_factorials

```
template<typename _Tp >
constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0
```

Definition at line 583 of file sf_gamma.tcc.

9.3.2.21 _S_num_neg_double_factorials< double >

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150
```

Definition at line 588 of file sf_gamma.tcc.

9.3.2.22 _S_num_neg_double_factorials< float >

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27
```

Definition at line 586 of file sf_gamma.tcc.

9.3.2.23 _S_num_neg_double_factorials< long double >

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999
```

Definition at line 590 of file sf_gamma.tcc.

9.3.2.24 _S_num_zetam1

```
constexpr size_t std::__detail::_S_num_zetam1 = 121
```

Table of zeta(n) - 1 from 0 - 120. MPFR @ 128 bits.

Definition at line 493 of file sf_zeta.tcc.

Referenced by __riemann_zeta_m_1().

9.3.2.25 _S_zetam1

```
constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]
```

Definition at line 497 of file sf_zeta.tcc.

Referenced by __riemann_zeta_m_1().

Chapter 10

Class Documentation

10.1 __gnu_cxx::__airy_t< _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

  Return the Wronskian of this Airy function state.

Public Attributes

- _Tp __Ai_deriv

  The derivative of the Airy function Ai.
- _Tp __Ai_value

  The value of the Airy function Ai.
- _Tp __Bi_deriv

  The derivative of the Airy function Bi.
- _Tp __Bi_value

  The value of the Airy function Bi.
- _Tx __x_arg

  The argument of the Airy fuctions.

10.1.1 Detailed Description

template<typename _Tx, typename _Tp>
struct __gnu_cxx::__airy_t< _Tx, _Tp >

Definition at line 211 of file specfun_state.h.

10.1.2 Member Function Documentation

10.1.2.1 __Wronskian()

template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Wronskian ( ) const [inline]

Return the Wronskian of this Airy function state.

Definition at line 229 of file specfun_state.h.

10.1.3 Member Data Documentation

10.1.3.1 __Ai_deriv

template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Ai_deriv

The derivative of the Airy function Ai.

Definition at line 220 of file specfun_state.h.

10.1.3.2 __Ai_value

template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Ai_value

The value of the Airy function Ai.

Definition at line 217 of file specfun_state.h.

10.1.3.3 __Bi_deriv

template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Bi_deriv

The derivative of the Airy function Bi.

Definition at line 226 of file specfun_state.h.

10.1.3.4 __Bi_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Bi_value
```

The value of the Airy function Bi.

Definition at line 223 of file specfun_state.h.

10.1.3.5 __x_arg

```
template<typename _Tx , typename _Tp >
_Tx __gnu_cxx::__airy_t< _Tx, _Tp >::__x_arg
```

The argument of the Airy fuctions.

Definition at line 214 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.2 __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const
  Return the Wronskian of this cylindrical Bessel function state.

Public Attributes

- _Tp __J_deriv
  The derivative of the Bessel function of the first kind.
- _Tp __J_value
  The value of the Bessel function of the first kind.
- _Tp __N_deriv
  The derivative of the Bessel function of the second kind.
- _Tp __N_value
  The value of the Bessel function of the second kind.
- _Tnu __nu_arg
  The real order of the cylindrical Bessel functions.
- _Tx __x_arg
  The argument of the cylindrical Bessel functions.

10.2.1 Detailed Description

template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >

This struct captures the state of the cylindrical Bessel functions at a given order and argument.

Definition at line 264 of file specfun_state.h.

10.2.2 Member Function Documentation

10.2.2.1 __Wronskian()

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this cylindrical Bessel function state.

Definition at line 285 of file specfun_state.h.

10.2.3 Member Data Documentation

10.2.3.1 __J_deriv

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__J_deriv
```

The derivative of the Bessel function of the first kind.

Definition at line 276 of file specfun_state.h.

10.2.3.2 __J_value

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__J_value
```

The value of the Bessel function of the first kind.

Definition at line 273 of file specfun_state.h.

10.2.3.3 __N_deriv

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__N_deriv
```

The derivative of the Bessel function of the second kind.

Definition at line 282 of file specfun_state.h.

10.2.3.4 __N_value

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__N_value
```

The value of the Bessel function of the second kind.

Definition at line 279 of file specfun_state.h.

10.2.3.5 __nu_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tnu __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the cylindrical Bessel functions.

Definition at line 267 of file specfun_state.h.

10.2.3.6 __x_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the cylindrical Bessel functions.

Definition at line 270 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.3 __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

    Return the Wronskian of this Coulomb function state.

Public Attributes

- _Teta __eta_arg

    The real parameter of the Coulomb functions.
- _Tp __F_deriv

    The derivative of the regular Coulomb function.
- _Tp __F_value

    The value of the regular Coulomb function.
- _Tp __G_deriv

    The derivative of the irregular Coulomb function.
- _Tp __G_value

    The value of the irregular Coulomb function.
- unsigned int __l

    The nonnegative order of the Coulomb functions.
- _Trho __rho_arg

    The argument of the Coulomb functions.

10.3.1 Detailed Description

template<typename _Teta, typename _Trho, typename _Tp>
struct __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >

This struct captures the state of the Coulomb functions at a given order and argument.

Definition at line 294 of file specfun_state.h.

10.3.2 Member Function Documentation

10.3.2.1 __Wronskian()

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this Coulomb function state.

Definition at line 318 of file specfun_state.h.

10.3.3 Member Data Documentation

10.3.3.1 __eta_arg

```
template<typename _Teta , typename _Trho , typename _Tp >
_Teta __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__eta_arg
```

The real parameter of the Coulomb functions.

Definition at line 300 of file specfun_state.h.

10.3.3.2 __F_deriv

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__F_deriv
```

The derivative of the regular Coulomb function.

Definition at line 309 of file specfun_state.h.

10.3.3.3 __F_value

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__F_value
```

The value of the regular Coulomb function.

Definition at line 306 of file specfun_state.h.

10.3.3.4 __G_deriv

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__G_deriv
```

The derivative of the irregular Coulomb function.

Definition at line 315 of file specfun_state.h.

10.3.3.5 __G_value

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__G_value
```

The value of the irregular Coulomb function.

Definition at line 312 of file specfun_state.h.

10.3.3.6 __l

```
template<typename _Teta , typename _Trho , typename _Tp >
unsigned int __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__l
```

The nonnegative order of the Coulomb functions.

Definition at line 297 of file specfun_state.h.

10.3.3.7 __rho_arg

```
template<typename _Teta , typename _Trho , typename _Tp >
_Trho __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__rho_arg
```

The argument of the Coulomb functions.

Definition at line 303 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.4 __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

  Return the Wronskian of this cylindrical Hankel function state.

Public Attributes

- _Tp __H1_deriv

  The derivative of the cylindrical Hankel function of the first kind.
- _Tp __H1_value

  The value of the cylindrical Hankel function of the first kind.
- _Tp __H2_deriv

  The derivative of the cylindrical Hankel function of the second kind.
- _Tp __H2_value

  The value of the cylindrical Hankel function of the second kind.
- _Tnu __nu_arg

  The real order of the cylindrical Hankel functions.
- _Tx __x_arg

  The argument of the modified Hankel functions.

10.4.1 Detailed Description

template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >

_Tp pretty much has to be complex.

Definition at line 361 of file specfun_state.h.

10.4.2 Member Function Documentation

10.4.2.1 __Wronskian()

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this cylindrical Hankel function state.

Definition at line 382 of file specfun_state.h.

10.4.3 Member Data Documentation

10.4.3.1 __H1_deriv

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H1_deriv
```

The derivative of the cylindrical Hankel function of the first kind.

Definition at line 373 of file specfun_state.h.

10.4.3.2 __H1_value

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H1_value
```

The value of the cylindrical Hankel function of the first kind.

Definition at line 370 of file specfun_state.h.

10.4.3.3 __H2_deriv

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H2_deriv
```

The derivative of the cylindrical Hankel function of the second kind.

Definition at line 379 of file specfun_state.h.

10.4.3.4 __H2_value

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H2_value
```

The value of the cylindrical Hankel function of the second kind.

Definition at line 376 of file specfun_state.h.

10.4.3.5 __nu_arg

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tnu __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the cylindrical Hankel functions.

Definition at line 364 of file specfun_state.h.

10.4.3.6 __x_arg

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tx __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the modified Hankel functions.

Definition at line 367 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.5 __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

  Return the Wronskian of this modified cylindrical Bessel function state.

Public Attributes

- _Tp __I_deriv

  The derivative of the modified cylindrical Bessel function of the first kind.
- _Tp __I_value

  The value of the modified cylindrical Bessel function of the first kind.
- _Tp __K_deriv

  The derivative of the modified cylindrical Bessel function of the second kind.
- _Tp __K_value

  The value of the modified cylindrical Bessel function of the second kind.
- _Tnu __nu_arg

  The real order of the modified cylindrical Bessel functions.
- _Tx __x_arg

  The argument of the modified cylindrical Bessel functions.

10.5.1 Detailed Description

template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >

This struct captures the state of the modified cylindrical Bessel functions at a given order and argument.

Definition at line 327 of file specfun_state.h.

10.5.2 Member Function Documentation

10.5.2.1 __Wronskian()

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this modified cylindrical Bessel function state.

Definition at line 353 of file specfun_state.h.

10.5.3 Member Data Documentation

10.5.3.1 __I_deriv

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_deriv
```

The derivative of the modified cylindrical Bessel function of the first kind.

Definition at line 341 of file specfun_state.h.

10.5.3.2 __I_value

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_value
```

The value of the modified cylindrical Bessel function of the first kind.

Definition at line 337 of file specfun_state.h.

10.5.3.3 __K_deriv

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_deriv
```

The derivative of the modified cylindrical Bessel function of the second kind.

Definition at line 349 of file specfun_state.h.

10.5.3.4 __K_value

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_value
```

The value of the modified cylindrical Bessel function of the second kind.

Definition at line 345 of file specfun_state.h.

10.5.3.5 __nu_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tnu __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the modified cylindrical Bessel functions.

Definition at line 330 of file specfun_state.h.

10.5.3.6 __x_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the modified cylindrical Bessel functions.

Definition at line 333 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.6 __gnu_cxx::__fock_airy_t< _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

    Return the Wronskian of this Fock-type Airy function state.

Public Attributes

- _Tp __w1_deriv

    The derivative of the Fock-type Airy function w1.
- _Tp __w1_value

    The value of the Fock-type Airy function w1.
- _Tp __w2_deriv

    The derivative of the Fock-type Airy function w2.
- _Tp __w2_value

    The value of the Fock-type Airy function w2.
- _Tx __x_arg

    The argument of the Fock-type Airy fuctions.

10.6.1 Detailed Description

template<typename _Tx, typename _Tp>
struct __gnu_cxx::__fock_airy_t< _Tx, _Tp >

_Tp pretty much has to be complex.

Definition at line 237 of file specfun_state.h.

10.6.2 Member Function Documentation

10.6.2.1 __Wronskian()

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this Fock-type Airy function state.

Definition at line 255 of file specfun_state.h.

10.6.3 Member Data Documentation

10.6.3.1 __w1_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w1_deriv
```

The derivative of the Fock-type Airy function w1.

Definition at line 246 of file specfun_state.h.

10.6.3.2 __w1_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w1_value
```

The value of the Fock-type Airy function w1.

Definition at line 243 of file specfun_state.h.

10.6.3.3 __w2_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w2_deriv
```

The derivative of the Fock-type Airy function w2.

Definition at line 252 of file specfun_state.h.

10.6.3.4 __w2_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w2_value
```

The value of the Fock-type Airy function w2.

Definition at line 249 of file specfun_state.h.

10.6.3.5 __x_arg

```
template<typename _Tx , typename _Tp >
_Tx __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__x_arg
```

The argument of the Fock-type Airy fuctions.

Definition at line 240 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.7 __gnu_cxx::__fp_is_integer_t Struct Reference

```
#include <math_util.h>
```

Public Member Functions

- operator bool () const
- int operator() () const

Public Attributes

- bool __is_integral
- int __value

10.7.1 Detailed Description

A struct returned by floating point integer queries.

Definition at line 123 of file math_util.h.

10.7.2 Member Function Documentation

10.7.2.1 operator bool()

```
__gnu_cxx::__fp_is_integer_t::operator bool ( ) const [inline]
```

Definition at line 132 of file math_util.h.

References __is_integral.

10.7.2.2 operator()()

```
int __gnu_cxx::__fp_is_integer_t::operator() ( ) const [inline]
```

Definition at line 137 of file math_util.h.

References __value.

10.7.3 Member Data Documentation

10.7.3.1 __is_integral

```
bool __gnu_cxx::__fp_is_integer_t::__is_integral
```

Definition at line 126 of file math_util.h.

Referenced by operator bool().

10.7.3.2 __value

```
int __gnu_cxx::__fp_is_integer_t::__value
```

Definition at line 129 of file math_util.h.

Referenced by operator()().

The documentation for this struct was generated from the following file:

- ext/math_util.h

10.8 __gnu_cxx::__gamma_inc_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- _Tp __lgamma_value
  The value of the log of the incomplete gamma function.
- _Tp __tgamma_value
  The value of the total gamma function.

10.8.1 Detailed Description

template< typename _Tp>
struct __gnu_cxx::__gamma_inc_t< _Tp >

The sign of the exponentiated log(gamma) is appied to the tgamma value.

Definition at line 500 of file specfun_state.h.

10.8.2 Member Data Documentation

10.8.2.1 __lgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_inc_t< _Tp >::__lgamma_value
```

The value of the log of the incomplete gamma function.

Definition at line 505 of file specfun_state.h.

10.8.2.2 __tgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_inc_t< _Tp >::__tgamma_value
```

The value of the total gamma function.

Definition at line 503 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.9 __gnu_cxx::__gamma_temme_t< _Tp > Struct Template Reference

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

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

and

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

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.

```
#include <specfun_state.h>
```

Public Attributes

- _Tp __gamma_1_value

  The output function $\Gamma_1(\mu)$.
- _Tp __gamma_2_value

  The output function $\Gamma_2(\mu)$.
- _Tp __gamma_minus_value

  The output function $1/\Gamma(1-\mu)$.
- _Tp __gamma_plus_value

  The output function $1/\Gamma(1+\mu)$.
- _Tp __mu_arg

  The input parameter of the gamma functions.

10.9.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__gamma_temme_t< _Tp >

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

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

and

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

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

The accuracy requirements on this are high for $|\mu| < 0$.

Definition at line 528 of file specfun_state.h.

10.9.2 Member Data Documentation

10.9.2.1 __gamma_1_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_1_value
```

The output function $\Gamma_1(\mu)$.

Definition at line 540 of file specfun_state.h.

10.9.2.2 __gamma_2_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_2_value
```

The output function $\Gamma_2(\mu)$.

Definition at line 543 of file specfun_state.h.

10.9.2.3 __gamma_minus_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_minus_value
```

The output function $1/\Gamma(1-\mu)$.

Definition at line 537 of file specfun_state.h.

10.9.2.4 __gamma_plus_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_plus_value
```

The output function $1/\Gamma(1+\mu)$.

Definition at line 534 of file specfun_state.h.

10.9.2.5 __mu_arg

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__mu_arg
```

The input parameter of the gamma functions.

Definition at line 531 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.10 __gnu_cxx::__hermite_he_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp deriv () const

Public Attributes

- _Tp __He_n
- _Tp __He_nm1
- _Tp __He_nm2
- std::size_t __n
- _Tp __x

10.10.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__hermite_he_t< _Tp >

A struct to store the state of a probabilist Hermite polynomial.

Definition at line 80 of file specfun_state.h.

10.10.2 Member Function Documentation

10.10.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_he_t< _Tp >::deriv ( ) const  [inline]
```

Definition at line 89 of file specfun_state.h.

10.10.3 Member Data Documentation

10.10.3.1 __He_n

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_n
```

Definition at line 84 of file specfun_state.h.

10.10.3.2 __He_nm1

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_nm1
```

Definition at line 85 of file specfun_state.h.

10.10.3.3 __He_nm2

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_nm2
```

Definition at line 86 of file specfun_state.h.

10.10.3.4 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__hermite_he_t< _Tp >::__n
```

Definition at line 82 of file specfun_state.h.

10.10.3.5 __x

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__x
```

Definition at line 83 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.11 __gnu_cxx::__hermite_t< _Tp > Struct Template Reference

`#include <specfun_state.h>`

Public Member Functions

- _Tp deriv () const

Public Attributes

- _Tp __H_n
- _Tp __H_nm1
- _Tp __H_nm2
- std::size_t __n
- _Tp __x

10.11.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__hermite_t< _Tp >

A struct to store the state of a Hermite polynomial.

Definition at line 63 of file specfun_state.h.

10.11.2 Member Function Documentation

10.11.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::deriv ( ) const  [inline]
```

Definition at line 72 of file specfun_state.h.

10.11.3 Member Data Documentation

10.11.3.1 __H_n

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_n
```

Definition at line 67 of file specfun_state.h.

10.11.3.2 __H_nm1

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_nm1
```

Definition at line 68 of file specfun_state.h.

10.11.3.3 __H_nm2

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_nm2
```

Definition at line 69 of file specfun_state.h.

10.11.3.4 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__hermite_t< _Tp >::__n
```

Definition at line 65 of file specfun_state.h.

10.11.3.5 __x

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__x
```

Definition at line 66 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.12 __gnu_cxx::__jacobi_ellint_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __am () const
- _Tp __cd () const
- _Tp __cs () const
- _Tp __dc () const
- _Tp __ds () const
- _Tp __nc () const
- _Tp __nd () const
- _Tp __ns () const
- _Tp __sc () const
- _Tp __sd () const

Public Attributes

- _Tp __cn_value

  Jacobi cosine amplitude value.
- _Tp __dn_value

  Jacobi delta amplitude value.
- _Tp __sn_value

  Jacobi sine amplitude value.

10.12.1 Detailed Description

template< typename _Tp >
struct __gnu_cxx::__jacobi_ellint_t< _Tp >

Slots for Jacobi elliptic function tuple.

Definition at line 168 of file specfun_state.h.

10.12.2 Member Function Documentation

10.12.2.1 __am()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__am ( ) const  [inline]
```

Definition at line 179 of file specfun_state.h.

10.12.2.2 __cd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cd ( ) const [inline]
```

Definition at line 197 of file specfun_state.h.

10.12.2.3 __cs()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cs ( ) const [inline]
```

Definition at line 200 of file specfun_state.h.

10.12.2.4 __dc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__dc ( ) const [inline]
```

Definition at line 206 of file specfun_state.h.

10.12.2.5 __ds()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__ds ( ) const [inline]
```

Definition at line 203 of file specfun_state.h.

10.12.2.6 __nc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__nc ( ) const [inline]
```

Definition at line 185 of file specfun_state.h.

10.12.2.7 __nd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__nd ( ) const [inline]
```

Definition at line 188 of file specfun_state.h.

10.12.2.8 __ns()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__ns ( ) const [inline]
```

Definition at line 182 of file specfun_state.h.

10.12.2.9 __sc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sc ( ) const [inline]
```

Definition at line 191 of file specfun_state.h.

10.12.2.10 __sd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sd ( ) const [inline]
```

Definition at line 194 of file specfun_state.h.

10.12.3 Member Data Documentation

10.12.3.1 __cn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cn_value
```

Jacobi cosine amplitude value.

Definition at line 174 of file specfun_state.h.

10.12.3.2 __dn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__dn_value
```

Jacobi delta amplitude value.

Definition at line 177 of file specfun_state.h.

10.12.3.3 __sn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sn_value
```

Jacobi sine amplitude value.

Definition at line 171 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.13 __gnu_cxx::__jacobi_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp deriv () const

Public Attributes

- _Tp __alpha1
- _Tp __beta1
- std::size_t __n
- _Tp __P_n
- _Tp __P_nm1
- _Tp __P_nm2
- _Tp __x

10.13.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__jacobi_t< _Tp >

A struct to store the state of a Jacobi polynomial.

Definition at line 133 of file specfun_state.h.

10.13.2 Member Function Documentation

10.13.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::deriv ( ) const  [inline]
```

Definition at line 144 of file specfun_state.h.

10.13.3 Member Data Documentation

10.13.3.1 __alpha1

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__alpha1
```

Definition at line 136 of file specfun_state.h.

10.13.3.2 __beta1

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__beta1
```

Definition at line 137 of file specfun_state.h.

10.13.3.3 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__jacobi_t< _Tp >::__n
```

Definition at line 135 of file specfun_state.h.

10.13.3.4 __P_n

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__P_n
```

Definition at line 139 of file specfun_state.h.

10.13.3.5 __P_nm1

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__P_nm1
```

Definition at line 140 of file specfun_state.h.

10.13.3.6 __P_nm2

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__P_nm2
```

Definition at line 141 of file specfun_state.h.

10.13.3.7 __x

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t< _Tp >::__x
```

Definition at line 138 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.14 __gnu_cxx::__laguerre_t< _Tpa, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp deriv () const

Public Attributes

- _Tpa __alpha1
- _Tp __L_n
- _Tp __L_nm1
- _Tp __L_nm2
- std::size_t __n
- _Tp __x

10.14.1 Detailed Description

template<typename _Tpa, typename _Tp>
struct __gnu_cxx::__laguerre_t< _Tpa, _Tp >

A struct to store the state of a Laguerre polynomial.

Definition at line 115 of file specfun_state.h.

10.14.2 Member Function Documentation

10.14.2.1 deriv()

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::deriv ( ) const  [inline]
```

Definition at line 125 of file specfun_state.h.

10.14.3 Member Data Documentation

10.14.3.1 __alpha1

```
template<typename _Tpa , typename _Tp >
_Tpa __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__alpha1
```

Definition at line 118 of file specfun_state.h.

10.14.3.2 __L_n

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_n
```

Definition at line 120 of file specfun_state.h.

10.14.3.3 __L_nm1

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_nm1
```

Definition at line 121 of file specfun_state.h.

10.14.3.4 __L_nm2

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_nm2
```

Definition at line 122 of file specfun_state.h.

10.14.3.5 __n

```
template<typename _Tpa , typename _Tp >
std::size_t __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__n
```

Definition at line 117 of file specfun_state.h.

10.14.3.6 __x

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__x
```

Definition at line 119 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.15 __gnu_cxx::__legendre_p_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp deriv () const

Public Attributes

- std::size_t __l
- _Tp __P_l
- _Tp __P_lm1
- _Tp __P_lm2
- _Tp __z

10.15.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__legendre_p_t< _Tp >

A struct to store the state of a Legendre polynomial.

Definition at line 97 of file specfun_state.h.

10.15.2 Member Function Documentation

10.15.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 107 of file specfun_state.h.

10.15.3 Member Data Documentation

10.15.3.1 __l

```
template<typename _Tp >
std::size_t __gnu_cxx::__legendre_p_t< _Tp >::__l
```

Definition at line 99 of file specfun_state.h.

10.15.3.2 __P_l

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t< _Tp >::__P_l
```

Definition at line 101 of file specfun_state.h.

10.15.3.3 __P_lm1

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t< _Tp >::__P_lm1
```

Definition at line 102 of file specfun_state.h.

10.15.3.4 __P_lm2

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t< _Tp >::__P_lm2
```

Definition at line 103 of file specfun_state.h.

10.15.3.5 __z

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t< _Tp >::__z
```

Definition at line 100 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.16 __gnu_cxx::__lgamma_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- int __lgamma_sign
  The sign of the exponent of the log gamma value.
- _Tp __lgamma_value
  The value log gamma function.

10.16.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__lgamma_t< _Tp >

The log of the absolute value of the gamma function The sign of the exponentiated log(gamma) is stored in sign.

Definition at line 487 of file specfun_state.h.

10.16.2 Member Data Documentation

10.16.2.1 __lgamma_sign

```
template<typename _Tp >
int __gnu_cxx::__lgamma_t< _Tp >::__lgamma_sign
```

The sign of the exponent of the log gamma value.

Definition at line 493 of file specfun_state.h.

10.16.2.2 __lgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__lgamma_t< _Tp >::__lgamma_value
```

The value log gamma function.

Definition at line 490 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.17 __gnu_cxx::__pqgamma_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- _Tp __pgamma_value
- _Tp __qgamma_value

10.17.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__pqgamma_t< _Tp >

Definition at line 473 of file specfun_state.h.

10.17.2 Member Data Documentation

10.17.2.1 __pgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__pqgamma_t< _Tp >::__pgamma_value
```

Definition at line 476 of file specfun_state.h.

10.17.2.2 __qgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__pqgamma_t< _Tp >::__qgamma_value
```

Definition at line 479 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.18 __gnu_cxx::__quadrature_point_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- __quadrature_point_t ()=default
- __quadrature_point_t (_Tp __z, _Tp __w)

Public Attributes

- _Tp __weight
- _Tp __zero

10.18.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__quadrature_point_t< _Tp >

A struct to store a cosine and a sine value. A return for sincos-type functions.

Definition at line 46 of file specfun_state.h.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 __quadrature_point_t() [1/2]

```
template<typename _Tp >
__gnu_cxx::__quadrature_point_t< _Tp >::__quadrature_point_t ( ) [default]
```

10.18.2.2 __quadrature_point_t() [2/2]

```
template<typename _Tp >
__gnu_cxx::__quadrature_point_t< _Tp >::__quadrature_point_t (
            _Tp __z,
            _Tp __w ) [inline]
```

Definition at line 53 of file specfun_state.h.

10.18.3 Member Data Documentation

10.18.3.1 __weight

```
template<typename _Tp >
_Tp __gnu_cxx::__quadrature_point_t< _Tp >::__weight
```

Definition at line 49 of file specfun_state.h.

10.18.3.2 __zero

```
template<typename _Tp >
_Tp __gnu_cxx::__quadrature_point_t< _Tp >::__zero
```

Definition at line 48 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.19 __gnu_cxx::__sincos_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- _Tp __cos_v
- _Tp __sin_v

10.19.1 Detailed Description

template<typename _Tp>
struct __gnu_cxx::__sincos_t< _Tp >

A struct to store a cosine and a sine value. A return for sincos-type functions.

Definition at line 158 of file specfun_state.h.

10.19.2 Member Data Documentation

10.19.2.1 __cos_v

```
template<typename _Tp>
_Tp __gnu_cxx::__sincos_t< _Tp >::__cos_v
```

Definition at line 161 of file specfun_state.h.

Referenced by std::__detail::__polar_pi(), and std::__detail::__sincos_pi().

10.19.2.2 __sin_v

```
template<typename _Tp>
_Tp __gnu_cxx::__sincos_t< _Tp >::__sin_v
```

Definition at line 160 of file specfun_state.h.

Referenced by std::__detail::__polar_pi(), and std::__detail::__sincos_pi().

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.20 __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

    Return the Wronskian of this spherical Bessel function state.

Public Attributes

- _Tp __j_deriv

    The derivative of the spherical Bessel function of the first kind.
- _Tp __j_value

    The value of the spherical Bessel function of the first kind.
- _Tn __n_arg

    The integral order of the spherical Bessel functions.
- _Tp __n_deriv

    The derivative of the spherical Bessel function of the second kind.
- _Tp __n_value

    The value of the spherical Bessel function of the second kind.
- _Tx __x_arg

    The argument of the spherical Bessel functions.

10.20.1 Detailed Description

template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >

Definition at line 387 of file specfun_state.h.

10.20.2 Member Function Documentation

10.20.2.1 __Wronskian()

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this spherical Bessel function state.

Definition at line 408 of file specfun_state.h.

10.20.3 Member Data Documentation

10.20.3.1 __j_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__j_deriv
```

The derivative of the spherical Bessel function of the first kind.

Definition at line 399 of file specfun_state.h.

10.20.3.2 __j_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__j_value
```

The value of the spherical Bessel function of the first kind.

Definition at line 396 of file specfun_state.h.

10.20.3.3 __n_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tn __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_arg
```

The integral order of the spherical Bessel functions.

Definition at line 390 of file specfun_state.h.

10.20.3.4 __n_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_deriv
```

The derivative of the spherical Bessel function of the second kind.

Definition at line 405 of file specfun_state.h.

10.20.3.5 __n_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_value
```

The value of the spherical Bessel function of the second kind.

Definition at line 402 of file specfun_state.h.

10.20.3.6 __x_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the spherical Bessel functions.

Definition at line 393 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.21 __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const
  Return the Wronskian of this cylindrical Hankel function state.

Public Attributes

- _Tp __h1_deriv
  The derivative of the spherical Hankel function of the first kind.
- _Tp __h1_value
  The velue of the spherical Hankel function of the first kind.
- _Tp __h2_deriv
  The derivative of the spherical Hankel function of the second kind.
- _Tp __h2_value
  The velue of the spherical Hankel function of the second kind.
- _Tn __n_arg
  The integral order of the spherical Hankel functions.
- _Tx __x_arg
  The argument of the spherical Hankel functions.

10.21.1 Detailed Description

template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >

_Tp pretty much has to be complex.

Definition at line 447 of file specfun_state.h.

10.21.2 Member Function Documentation

10.21.2.1 __Wronskian()

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this cylindrical Hankel function state.

Definition at line 468 of file specfun_state.h.

10.21.3 Member Data Documentation

10.21.3.1 __h1_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h1_deriv
```

The derivative of the spherical Hankel function of the first kind.

Definition at line 459 of file specfun_state.h.

10.21.3.2 __h1_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h1_value
```

The velue of the spherical Hankel function of the first kind.

Definition at line 456 of file specfun_state.h.

10.21.3.3 __h2_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h2_deriv
```

The derivative of the spherical Hankel function of the second kind.

Definition at line 465 of file specfun_state.h.

10.21.3.4 __h2_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h2_value
```

The velue of the spherical Hankel function of the second kind.

Definition at line 462 of file specfun_state.h.

10.21.3.5 __n_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tn __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__n_arg
```

The integral order of the spherical Hankel functions.

Definition at line 450 of file specfun_state.h.

10.21.3.6 __x_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the spherical Hankel functions.

Definition at line 453 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.22 __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- _Tp __Wronskian () const

  Return the Wronskian of this modified cylindrical Bessel function state.

Public Attributes

- _Tp __i_deriv

  The derivative of the modified spherical Bessel function of the first kind.
- _Tp __i_value

  The value of the modified spherical Bessel function of the first kind.
- _Tp __k_deriv

  The derivative of the modified spherical Bessel function of the second kind.
- _Tp __k_value

  The value of the modified spherical Bessel function of the second kind.
- _Tx __x_arg

  The argument of the modified spherical Bessel functions.
- _Tn n_arg

  The integral order of the modified spherical Bessel functions.

10.22.1 Detailed Description

template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >

Definition at line 413 of file specfun_state.h.

10.22.2 Member Function Documentation

10.22.2.1 __Wronskian()

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__Wronskian ( ) const  [inline]
```

Return the Wronskian of this modified cylindrical Bessel function state.

Definition at line 439 of file specfun_state.h.

10.22.3 Member Data Documentation

10.22.3.1 __i_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_deriv
```

The derivative of the modified spherical Bessel function of the first kind.

Definition at line 427 of file specfun_state.h.

10.22.3.2 __i_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_value
```

The value of the modified spherical Bessel function of the first kind.

Definition at line 423 of file specfun_state.h.

10.22.3.3 __k_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_deriv
```

The derivative of the modified spherical Bessel function of the second kind.

Definition at line 435 of file specfun_state.h.

10.22.3.4 __k_value

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_value
```

The value of the modified spherical Bessel function of the second kind.

Definition at line 431 of file specfun_state.h.

10.22.3.5 __x_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the modified spherical Bessel functions.

Definition at line 416 of file specfun_state.h.

10.22.3.6 n_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tn __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::n_arg
```

The integral order of the modified spherical Bessel functions.

Definition at line 419 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- bits/specfun_state.h

10.23 std::__detail::__gamma_lanczos_data< _Tp > Struct Template Reference

10.23.1 Detailed Description

template<typename _Tp>
struct std::__detail::__gamma_lanczos_data< _Tp >

A struct for Lanczos algorithm Chebyshev arrays of coefficients.

Definition at line 1995 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.24 std::__detail::__gamma_lanczos_data< double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< double, 10 > _S_cheby
- static constexpr double _S_g = 9.5

10.24.1 Detailed Description

template<>
struct std::__detail::__gamma_lanczos_data< double >

Definition at line 2017 of file sf_gamma.tcc.

10.24.2 Member Data Documentation

10.24.2.1 _S_cheby

`constexpr std::array<double, 10> std::__detail::__gamma_lanczos_data< double >::_S_cheby [static]`

Initial value:

```
{
      5.557569219204146e+03,
     -4.248114953727554e+03,
      1.881719608233706e+03,
     -4.705537221412237e+02,
      6.325224688788239e+01,
     -4.206901076213398e+00,
      1.202512485324405e-01,
     -1.141081476816908e-03,
      2.055079676210880e-06,
      1.280568540096283e-09,
       }
```

Definition at line 2022 of file sf_gamma.tcc.

10.24.2.2 _S_g

`constexpr double std::__detail::__gamma_lanczos_data< double >::_S_g = 9.5 [static]`

Definition at line 2019 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.25 std::__detail::__gamma_lanczos_data< float > Struct Template Reference

Static Public Attributes

- static constexpr std::array< float, 7 > _S_cheby
- static constexpr float _S_g = 6.5F

10.25.1 Detailed Description

template<>
struct std::__detail::__gamma_lanczos_data< float >

Definition at line 2000 of file sf_gamma.tcc.

10.25.2 Member Data Documentation

10.25.2.1 _S_cheby

```
constexpr std::array<float, 7> std::__detail::__gamma_lanczos_data< float >::_S_cheby [static]
```

Initial value:

```
{
     3.307139e+02F,
    -2.255998e+02F,
     6.989520e+01F,
    -9.058929e+00F,
     4.110107e-01F,
    -4.150391e-03F,
    -3.417969e-03F,
      }
```

Definition at line 2005 of file sf_gamma.tcc.

10.25.2.2 _S_g

```
constexpr float std::__detail::__gamma_lanczos_data< float >::_S_g = 6.5F [static]
```

Definition at line 2002 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.26 std::__detail::__gamma_lanczos_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< long double, 11 > _S_cheby
- static constexpr long double _S_g = 10.5L

10.26.1 Detailed Description

template<>
struct std::__detail::__gamma_lanczos_data< long double >

Definition at line 2037 of file sf_gamma.tcc.

10.26.2 Member Data Documentation

10.26.2.1 _S_cheby

```
constexpr std::array<long double, 11> std::__detail::__gamma_lanczos_data< long double >::_S_←
cheby [static]
```

Initial value:

```
{
     1.440399692024250728e+04L,
    -1.128006201837065341e+04L,
     5.384108670160999829e+03L,
    -1.536234184127325861e+03L,
     2.528551924697309561e+02L,
    -2.265389090278717887e+01L,
     1.006663776178612579e+00L,
    -1.900805731354182626e-02L,
     1.150508317664389324e-04L,
    -1.208915136885480024e-07L,
    -1.518856151960790157e-10L,
      }
```

Definition at line 2042 of file sf_gamma.tcc.

10.26.2.2 _S_g

```
constexpr long double std::__detail::__gamma_lanczos_data< long double >::_S_g = 10.5L [static]
```

Definition at line 2039 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.27 std::__detail::__gamma_spouge_data< _Tp > Struct Template Reference

10.27.1 Detailed Description

template<typename _Tp>
struct std::__detail::__gamma_spouge_data< _Tp >

A struct for Spouge algorithm Chebyshev arrays of coefficients.

Definition at line 1769 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.28 std::__detail::__gamma_spouge_data< double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< double, 18 > _S_cheby

10.28.1 Detailed Description

template<>
struct std::__detail::__gamma_spouge_data< double >

Definition at line 1790 of file sf_gamma.tcc.

10.28.2 Member Data Documentation

10.28.2.1 _S_cheby

constexpr std::array<double, 18> std::__detail::__gamma_spouge_data< double >::_S_cheby [static]

Initial value:

```
{
     2.785716565770350e+08,
    -1.693088166941517e+09,
     4.549688586500031e+09,
    -7.121728036151557e+09,
     7.202572947273274e+09,
    -4.935548868770376e+09,
     2.338187776097503e+09,
    -7.678102458920741e+08,
     1.727524819329867e+08,
    -2.595321377008346e+07,
     2.494811203993971e+06,
    -1.437252641338402e+05,
     4.490767356961276e+03,
    -6.505596924745029e+01,
     3.362323142416327e-01,
    -3.817361443986454e-04,
     3.273137866873352e-08,
    -7.642333165976788e-15,
      }
```

Definition at line 1794 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.29 std::__detail::__gamma_spouge_data< float > Struct Template Reference

Static Public Attributes

- static constexpr std::array< float, 7 > _S_cheby

10.29.1 Detailed Description

template<>
struct std::__detail::__gamma_spouge_data< float >

Definition at line 1774 of file sf_gamma.tcc.

10.29.2 Member Data Documentation

10.29.2.1 _S_cheby

```
constexpr std::array<float, 7> std::__detail::__gamma_spouge_data< float >::_S_cheby  [static]
```

Initial value:

```
{
      2.901419e+03F,
     -5.929168e+03F,
      4.148274e+03F,
     -1.164761e+03F,
      1.174135e+02F,
     -2.786588e+00F,
      3.775392e-03F,
       }
```

Definition at line 1778 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.30 std::__detail::__gamma_spouge_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< long double, 22 > _S_cheby

10.30.1 Detailed Description

template<>
struct std::__detail::__gamma_spouge_data< long double >

Definition at line 1817 of file sf_gamma.tcc.

10.30.2 Member Data Documentation

10.30.2.1 _S_cheby

```
constexpr std::array<long double, 22> std::__detail::__gamma_spouge_data< long double >::_S_↩
cheby  [static]
```

Initial value:

```
{
     1.681473171108908244e+10L,
    -1.269150315503303974e+11L,
     4.339449429013039995e+11L,
    -8.893680202692714895e+11L,
     1.218472425867950986e+12L,
    -1.178403473259353616e+12L,
     8.282455311246278274e+11L,
    -4.292112878930625978e+11L,
     1.646988347276488710e+11L,
    -4.661514921989111004e+10L,
     9.619972564515443397e+09L,
    -1.419382551781042824e+09L,
     1.454145470816386107e+08L,
    -9.923020719435758179e+06L,
     4.253557563919127284e+05L,
    -1.053371059784341875e+04L,
     1.332425479537961437e+02L,
    -7.118343974029489132e-01L,
     1.172051640057979518e-03L,
    -3.323940885824119041e-07L,
     4.503801674404338524e-12L,
    -5.320477002211632680e-20L,
      }
```

Definition at line 1821 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.31 std::__detail::__jacobi_theta_0_t< _Tp > Struct Template Reference

Public Attributes

- _Tp th1p
- _Tp th1ppp
- _Tp th2
- _Tp th2pp
- _Tp th3
- _Tp th3pp
- _Tp th4
- _Tp th4pp

10.31.1 Detailed Description

template<typename _Tp>
struct std::__detail::__jacobi_theta_0_t< _Tp >

Definition at line 517 of file sf_theta.tcc.

10.31.2 Member Data Documentation

10.31.2.1 th1p

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th1p
```

Definition at line 519 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_0().

10.31.2.2 th1ppp

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th1ppp
```

Definition at line 519 of file sf_theta.tcc.

10.31.2.3 th2

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th2
```

Definition at line 520 of file sf_theta.tcc.

10.31.2.4 th2pp

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th2pp
```

Definition at line 520 of file sf_theta.tcc.

10.31.2.5 th3

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th3
```

Definition at line 521 of file sf_theta.tcc.

10.31.2.6 th3pp

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th3pp
```

Definition at line 521 of file sf_theta.tcc.

10.31.2.7 th4

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th4
```

Definition at line 522 of file sf_theta.tcc.

10.31.2.8 th4pp

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_0_t< _Tp >::th4pp
```

Definition at line 522 of file sf_theta.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_theta.tcc

10.32 std::__detail::_Airy< _Tp > Class Template Reference

Public Types

- using scalar_type = std::__detail::__num_traits_t< value_type >
- using value_type = _Tp

Public Member Functions

- constexpr _Airy ()=default
- _Airy (const _Airy &)=default
- _Airy (_Airy &&)=default
- constexpr _AiryState< value_type > operator() (value_type __y) const

Public Attributes

- scalar_type inner_radius {_Airy_default_radii<scalar_type>::inner_radius}
- scalar_type outer_radius {_Airy_default_radii<scalar_type>::outer_radius}

10.32.1 Detailed Description

template<typename _Tp>
class std::__detail::_Airy< _Tp >

Class to manage the asymptotic expansions for Airy functions. The parameters describing the various regions are adjustable.

Definition at line 2504 of file sf_airy.tcc.

10.32.2 Member Typedef Documentation

10.32.2.1 scalar_type

```
template<typename _Tp>
using std::__detail::_Airy< _Tp >::scalar_type = std::__detail::__num_traits_t<value_type>
```

Definition at line 2509 of file sf_airy.tcc.

10.32.2.2 value_type

```
template<typename _Tp>
using std::__detail::_Airy< _Tp >::value_type = _Tp
```

Definition at line 2508 of file sf_airy.tcc.

10.32.3 Constructor & Destructor Documentation

10.32.3.1 _Airy() [1/3]

```
template<typename _Tp>
constexpr std::__detail::_Airy< _Tp >::_Airy ( )  [default]
```

10.32.3.2 _Airy() [2/3]

```
template<typename _Tp>
std::__detail::_Airy< _Tp >::_Airy (
             const _Airy< _Tp > & )  [default]
```

10.32.3.3 _Airy() [3/3]

```
template<typename _Tp>
std::__detail::_Airy< _Tp >::_Airy (
             _Airy< _Tp > && )  [default]
```

10.32.4 Member Function Documentation

10.32.4.1 operator()()

```
template<typename _Tp>
constexpr _AiryState< _Tp > std::__detail::_Airy< _Tp >::operator() (
             value_type __y ) const
```

Return the Airy functions for complex argument.

Definition at line 2527 of file sf_airy.tcc.

References std::__detail::__beta(), std::__detail::_Airy_series< _Tp >::_S_Ai(), and std::__detail::_Airy_series< _Tp >::_S_Bi().

10.32.5 Member Data Documentation

10.32.5.1 inner_radius

```
template<typename _Tp>
scalar_type std::__detail::_Airy< _Tp >::inner_radius {_Airy_default_radii<scalar_type>::inner←
_radius}
```

Definition at line 2518 of file sf_airy.tcc.

10.32.5.2 outer_radius

```
template<typename _Tp>
scalar_type std::__detail::_Airy< _Tp >::outer_radius {_Airy_default_radii<scalar_type>::outer←
_radius}
```

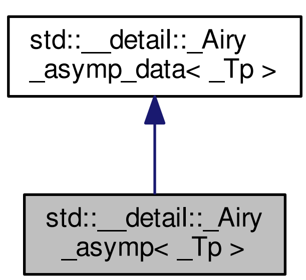
Definition at line 2519 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

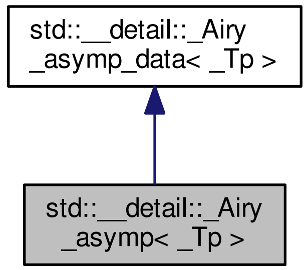
- bits/sf_airy.tcc

10.33 std::__detail::_Airy_asymp< _Tp > Class Template Reference

Inheritance diagram for std::__detail::_Airy_asymp< _Tp >:



Collaboration diagram for std::__detail::_Airy_asymp< _Tp >:



Public Types

- using _Cmplx = std::complex< _Tp >

Public Member Functions

- constexpr _Airy_asymp ()=default
- _AiryState< _Cmplx > _S_absarg_ge_pio3 (_Cmplx __z) const

  This function evaluates $Ai(z), Ai'(z)$ *and* $Bi(z), Bi'(z)$ *from their asymptotic expansions for* $|arg(z)| < 2 * \pi/3$ *i.e. roughly along the negative real axis.*
- _AiryState< _Cmplx > _S_absarg_lt_pio3 (_Cmplx __z) const

  This function evaluates $Ai(z)$ *and* $Ai'(z)$ *from their asymptotic expansions for* $|arg(-z)| < \pi/3$ *i.e. roughly along the negative real axis.*
- _AiryState< _Cmplx > operator() (_Cmplx __t, bool __return_fock_airy=false) const

10.33.1 Detailed Description

template<typename _Tp>
class std::__detail::_Airy_asymp< _Tp >

A class encapsulating the asymptotic expansions of Airy functions and their derivatives.

Template Parameters

_Tp	A real type

Definition at line 1998 of file sf_airy.tcc.

10.33.2 Member Typedef Documentation

10.33.2.1 _Cmplx

```
template<typename _Tp >
using std::__detail::_Airy_asymp< _Tp >::_Cmplx = std::complex<_Tp>
```

Definition at line 2003 of file sf_airy.tcc.

10.33.3 Constructor & Destructor Documentation

10.33.3.1 _Airy_asymp()

```
template<typename _Tp >
constexpr std::__detail::_Airy_asymp< _Tp >::_Airy_asymp ( ) [default]
```

10.33.4 Member Function Documentation

10.33.4.1 _S_absarg_ge_pio3()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::_S_absarg_ge_pio3 (
            _Cmplx __z ) const
```

This function evaluates $Ai(z), Ai'(z)$ and $Bi(z), Bi'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$ i.e. roughly along the negative real axis.

Template Parameters

_Tp	A real type

Parameters

in	__z	Complex argument at which Ai(z) and Bi(z) and their derivative are evaluated. This function assumes $\|z\| > 15$ and $\|(arg(z)\| < 2\pi/3$.

Returns

A struct containing $z, Ai(z), Ai'(z), Bi(z), Bi'(z)$.

Definition at line 2271 of file sf_airy.tcc.

References std::__detail::_AiryState$<$ _Tp $>$::__z.

10.33.4.2 _S_absarg_lt_pio3()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::_S_absarg_lt_pio3 (
            _Cmplx __z ) const
```

This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$ i.e. roughly along the negative real axis.

For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined for $|z|$. This function assumes $|z| > 15$ and $|arg(-z)| < \pi/3$.

Note that for speed and since this function is called by another, checks for valid arguments are not made. Hence, an error return is not needed.

Template Parameters

_Tp	A real type

Parameters

in	*__z*	The value at which the Airy function and their derivatives are evaluated.

Returns

A struct containing $z, Ai(z), Ai'(z), Bi(z), Bi'(z)$.

Todo Revisit these numbers of terms for the Airy asymptotic expansions.

Definition at line 2301 of file sf_airy.tcc.

References std::__detail::_AiryState$<$ _Tp $>$::__z.

10.33.4.3 operator()()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::operator() (
            _Cmplx __t,
            bool __return_fock_airy = false ) const
```

Return the Airy functions for a given argument using asymptotic series.

Template Parameters

_Tp	A real type

Definition at line 2029 of file sf_airy.tcc.

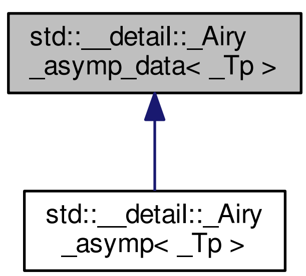
References std::__detail::_AiryState< _Tp >::__z.

The documentation for this class was generated from the following file:

- bits/sf_airy.tcc

10.34 std::__detail::_Airy_asymp_data< _Tp > Struct Template Reference

Inheritance diagram for std::__detail::_Airy_asymp_data< _Tp >:



10.34.1 Detailed Description

template<typename _Tp>
struct std::__detail::_Airy_asymp_data< _Tp >

A class encapsulating data for the asymptotic expansions of Airy functions and their derivatives.

Template Parameters

_Tp	A real type

Definition at line 632 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.35 std::__detail::_Airy_asymp_data< double > Struct Template Reference

Static Public Attributes

- static constexpr double _S_c [_S_max_cd]
- static constexpr double _S_d [_S_max_cd]
- static constexpr int _S_max_cd = 198

10.35.1 Detailed Description

template<>
struct std::__detail::_Airy_asymp_data< double >

Definition at line 739 of file sf_airy.tcc.

10.35.2 Member Data Documentation

10.35.2.1 _S_c

```
constexpr double std::__detail::_Airy_asymp_data< double >::_S_c[_S_max_cd]  [static]
```

Definition at line 745 of file sf_airy.tcc.

10.35.2.2 _S_d

```
constexpr double std::__detail::_Airy_asymp_data< double >::_S_d[_S_max_cd]  [static]
```

Definition at line 948 of file sf_airy.tcc.

10.35.2.3 _S_max_cd

```
constexpr int std::__detail::_Airy_asymp_data< double >::_S_max_cd = 198 [static]
```

Definition at line 741 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.36 std::__detail::_Airy_asymp_data< float > Struct Template Reference

Static Public Attributes

- static constexpr float _S_c [_S_max_cd]
- static constexpr float _S_d [_S_max_cd]
- static constexpr int _S_max_cd = 43

10.36.1 Detailed Description

template<>
struct std::__detail::_Airy_asymp_data< float >

Definition at line 636 of file sf_airy.tcc.

10.36.2 Member Data Documentation

10.36.2.1 _S_c

```
constexpr float std::__detail::_Airy_asymp_data< float >::_S_c[_S_max_cd] [static]
```

Definition at line 642 of file sf_airy.tcc.

10.36.2.2 _S_d

```
constexpr float std::__detail::_Airy_asymp_data< float >::_S_d[_S_max_cd] [static]
```

Definition at line 690 of file sf_airy.tcc.

10.36.2.3 _S_max_cd

```
constexpr int std::__detail::_Airy_asymp_data< float >::_S_max_cd = 43  [static]
```

Definition at line 638 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.37 std::__detail::_Airy_asymp_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr long double _S_c [_S_max_cd]
- static constexpr long double _S_d [_S_max_cd]
- static constexpr int _S_max_cd = 201

10.37.1 Detailed Description

template<>
struct std::__detail::_Airy_asymp_data< long double >

Definition at line 1152 of file sf_airy.tcc.

10.37.2 Member Data Documentation

10.37.2.1 _S_c

```
constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_c[_S_max_cd]  [static]
```

Definition at line 1158 of file sf_airy.tcc.

10.37.2.2 _S_d

```
constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_d[_S_max_cd]  [static]
```

Definition at line 1364 of file sf_airy.tcc.

10.37.2.3 _S_max_cd

```
constexpr int std::__detail::_Airy_asymp_data< long double >::_S_max_cd = 201 [static]
```

Definition at line 1154 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.38 std::__detail::_Airy_asymp_series< _Sum > Class Template Reference

Public Types

- using scalar_type = std::__detail::__num_traits_t< value_type >
- using value_type = typename _Sum::value_type

Public Member Functions

- _Airy_asymp_series (_Sum __proto)
- _Airy_asymp_series (const _Airy_asymp_series &)=default
- _Airy_asymp_series (_Airy_asymp_series &&)=default
- _AiryState< value_type > operator() (value_type __y)

Static Public Attributes

- static constexpr scalar_type _S_sqrt_pi = __gnu_cxx::__const_root_pi(scalar_type{})

10.38.1 Detailed Description

template<typename _Sum>
class std::__detail::_Airy_asymp_series< _Sum >

Class to manage the asymptotic series for Airy functions.

Template Parameters

_Sum	A sum type

Definition at line 2364 of file sf_airy.tcc.

10.38.2 Member Typedef Documentation

10.38.2.1 scalar_type

```
template<typename _Sum>
using std::__detail::_Airy_asymp_series< _Sum >::scalar_type = std::__detail::__num_traits_←
t<value_type>
```

Definition at line 2369 of file sf_airy.tcc.

10.38.2.2 value_type

```
template<typename _Sum>
using std::__detail::_Airy_asymp_series< _Sum >::value_type = typename _Sum::value_type
```

Definition at line 2368 of file sf_airy.tcc.

10.38.3 Constructor & Destructor Documentation

10.38.3.1 _Airy_asymp_series() [1/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (
            _Sum __proto )  [inline]
```

Definition at line 2373 of file sf_airy.tcc.

10.38.3.2 _Airy_asymp_series() [2/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (
            const _Airy_asymp_series< _Sum > &  )  [default]
```

10.38.3.3 _Airy_asymp_series() [3/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (
            _Airy_asymp_series< _Sum > && ) [default]
```

10.38.4 Member Function Documentation

10.38.4.1 operator()()

```
template<typename _Sum>
_AiryState< typename _Airy_asymp_series< _Sum >::value_type > std::__detail::_Airy_asymp_series<
_Sum >::operator() (
            value_type __y )
```

Return an _AiryState containing, not actual Airy functions, but four asymptotic Airy components:

Template Parameters

_Sum	A sum type

Definition at line 2418 of file sf_airy.tcc.

10.38.5 Member Data Documentation

10.38.5.1 _S_sqrt_pi

```
template<typename _Sum>
constexpr _Airy_asymp_series< _Sum >::scalar_type std::__detail::_Airy_asymp_series< _Sum >::_↩
S_sqrt_pi = __gnu_cxx::__const_root_pi(scalar_type{}) [static]
```

Definition at line 2371 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- bits/sf_airy.tcc

10.39 std::__detail::_Airy_default_radii< _Tp > Struct Template Reference

10.39.1 Detailed Description

template<typename _Tp>
struct std::__detail::_Airy_default_radii< _Tp >

Definition at line 2475 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.40 std::__detail::_Airy_default_radii< double > Struct Template Reference

Static Public Attributes

- static constexpr double inner_radius {4.0}
- static constexpr double outer_radius {12.0}

10.40.1 Detailed Description

template<>
struct std::__detail::_Airy_default_radii< double >

Definition at line 2486 of file sf_airy.tcc.

10.40.2 Member Data Documentation

10.40.2.1 inner_radius

```
constexpr double std::__detail::_Airy_default_radii< double >::inner_radius {4.0} [static]
```

Definition at line 2488 of file sf_airy.tcc.

10.40.2.2 outer_radius

```
constexpr double std::__detail::_Airy_default_radii< double >::outer_radius {12.0}  [static]
```

Definition at line 2489 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.41 std::__detail::_Airy_default_radii< float > Struct Template Reference

Static Public Attributes

- static constexpr float inner_radius {2.0F}
- static constexpr float outer_radius {6.0F}

10.41.1 Detailed Description

template<>
struct std::__detail::_Airy_default_radii< float >

Definition at line 2479 of file sf_airy.tcc.

10.41.2 Member Data Documentation

10.41.2.1 inner_radius

```
constexpr float std::__detail::_Airy_default_radii< float >::inner_radius {2.0F}  [static]
```

Definition at line 2481 of file sf_airy.tcc.

10.41.2.2 outer_radius

```
constexpr float std::__detail::_Airy_default_radii< float >::outer_radius {6.0F}  [static]
```

Definition at line 2482 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.42 std::__detail::_Airy_default_radii< long double > Struct Template Reference

Static Public Attributes

- static constexpr long double inner_radius {5.0L}
- static constexpr long double outer_radius {15.0L}

10.42.1 Detailed Description

template<>
struct std::__detail::_Airy_default_radii< long double >

Definition at line 2493 of file sf_airy.tcc.

10.42.2 Member Data Documentation

10.42.2.1 inner_radius

```
constexpr long double std::__detail::_Airy_default_radii< long double >::inner_radius {5.0L}
[static]
```

Definition at line 2495 of file sf_airy.tcc.

10.42.2.2 outer_radius

```
constexpr long double std::__detail::_Airy_default_radii< long double >::outer_radius {15.0L}
[static]
```

Definition at line 2496 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.43 std::__detail::_Airy_series< _Tp > Class Template Reference

Public Types

- using _Cmplx = std::complex< _Tp >

Static Public Member Functions

- static std::pair< _Cmplx, _Cmplx > _S_Ai (_Cmplx __t)
- static _AiryState< _Cmplx > _S_Airy (_Cmplx __t)
- static std::pair< _Cmplx, _Cmplx > _S_Bi (_Cmplx __t)
- static _AiryAuxilliaryState< _Cmplx > _S_FGH (_Cmplx __t)
- static _AiryState< _Cmplx > _S_Fock (_Cmplx __t)
- static _AiryState< _Cmplx > _S_Scorer (_Cmplx __t)
- static _AiryState< _Cmplx > _S_Scorer2 (_Cmplx __t)

Static Public Attributes

- static constexpr int _N_FGH = 200
- static constexpr _Tp _S_Ai0 = _Tp{3.550280538878172392600631860041831763980e-1L}
- static constexpr _Tp _S_Aip0 = _Tp{-2.588194037928067984051835601892039634793e-1L}
- static constexpr _Tp _S_Bi0 = _Tp{6.149266274460007351509223690936135535960e-1L}
- static constexpr _Tp _S_Bip0 = _Tp{4.482883573538263579148237103988283908668e-1L}
- static constexpr _Tp _S_eps = __gnu_cxx::__epsilon(_Tp{})
- static constexpr _Tp _S_Gi0 = _Tp{2.049755424820002450503074563645378511979e-1L}
- static constexpr _Tp _S_Gip0 = _Tp{1.494294524512754526382745701329427969551e-1L}
- static constexpr _Tp _S_Hi0 = _Tp{4.099510849640004901006149127290757023959e-1L}
- static constexpr _Tp _S_Hip0 = _Tp{2.988589049025509052765491402658855939102e-1L}
- static constexpr _Cmplx _S_i {_Tp{0}, _Tp{1}}
- static constexpr _Tp _S_pi = __gnu_cxx::__const_pi(_Tp{})
- static constexpr _Tp _S_sqrt_pi = __gnu_cxx::__const_root_pi(_Tp{})

10.43.1 Detailed Description

template<typename _Tp>
class std::__detail::_Airy_series< _Tp >

This class orgianizes series solutions of the Airy function.

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

This class contains tabulations of the factors appearing in the sums above.

Definition at line 108 of file sf_airy.tcc.

10.43.2 Member Typedef Documentation

10.43.2.1 _Cmplx

```
template<typename _Tp >
using std::__detail::_Airy_series< _Tp >::_Cmplx = std::complex<_Tp>
```

Definition at line 112 of file sf_airy.tcc.

10.43.3 Member Function Documentation

10.43.3.1 _S_Ai()

```
template<typename _Tp >
std::pair< std::complex< _Tp >, std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Ai
(
            _Cmplx __t )  [static]
```

Return the Airy function of the first kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the first kind is then defined by:

$$Ai(x) = Ai(0)fai(x) + Ai'(0)gai(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3 - 1/2Bi'(0)$ and $Bi(0) = 3^{1/2}Ai(0)$, $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

Template Parameters

_Tp	A real type

Definition at line 341 of file sf_airy.tcc.

Referenced by std::__detail::_Airy< _Tp >::operator()().

10.43.3.2 _S_Airy()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Airy (
            _Cmplx __t )  [static]
```

Return the Fock-type Airy functions $Ai(t)$, and $Bi(t)$ and their derivatives of complex argument.

Template Parameters

_Tp	A real type

Parameters

↩ _↩ ↩ _↩ *t*	The complex argument

Definition at line 609 of file sf_airy.tcc.

10.43.3.3 _S_Bi()

```
template<typename _Tp >
std::pair< std::complex< _Tp >, std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Bi
(
            _Cmplx __t ) [static]
```

Return the Airy function of the second kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the second kind is then defined by:

$$Bi(x) = Bi(0)fai(x) + Bi'(0)gai(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3{-}1/2Bi'(0)$ and $Bi(0) = 3^{1/2}Ai(0)$, $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

Template Parameters

_Tp	A real type

Definition at line 364 of file sf_airy.tcc.

Referenced by std::__detail::_Airy< _Tp >::operator()().

10.43.3.4 _S_FGH()

```
template<typename _Tp >
_AiryAuxilliaryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_FGH (
            _Cmplx __t ) [static]
```

Return the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

Template Parameters

_Tp	A real type

Definition at line 383 of file sf_airy.tcc.

10.43.3.5 _S_Fock()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Fock (
            _Cmplx __t ) [static]
```

Return the Fock-type Airy functions $w_1(t)$, and $w_2(t)$ and their derivatives of complex argument.

Template Parameters

_Tp	A real type

Parameters

↩ _↩ ↩ _↩ *t*	The complex argument

Definition at line 621 of file sf_airy.tcc.

10.43.3.6 _S_Scorer()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Scorer (
            _Cmplx __t ) [static]
```

Return the Scorer functions by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

The Scorer function is then defined by:

$$Hi(x) = Hi(0)\left(fai(x) + gai(x) + hai(x)\right)$$

where $Hi(0) = 2/(3^{7/6}\Gamma(2/3))$ and $Hi'(0) = 2/(3^{5/6}\Gamma(1/3))$. The other Scorer function is found from the identity

$$Gi(x) + Hi(x) = Bi(x)$$

Todo Find out what is wrong with the Hi = fai + gai + hai scorer function.

Template Parameters

_Tp	A real type

Definition at line 464 of file sf_airy.tcc.

10.43.3.7 _S_Scorer2()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Scorer2 (
            _Cmplx __t ) [static]
```

Return the Scorer functions by using the series expansions:

$$Hi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Hi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k-1}{3}\pi\right) \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k+1}{3}\pi\right) \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

Definition at line 501 of file sf_airy.tcc.

References std::__detail::__gamma().

10.43.4 Member Data Documentation

10.43.4.1 _N_FGH

```
template<typename _Tp >
constexpr int std::__detail::_Airy_series< _Tp >::_N_FGH = 200  [static]
```

Definition at line 114 of file sf_airy.tcc.

10.43.4.2 _S_Ai0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Ai0 = _Tp{3.550280538878172392600631860041831763980e-1↩
L}  [static]
```

Definition at line 130 of file sf_airy.tcc.

10.43.4.3 _S_Aip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Aip0 = _Tp{-2.588194037928067984051835601892039634793e-1↩
L}  [static]
```

Definition at line 132 of file sf_airy.tcc.

10.43.4.4 _S_Bi0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Bi0 = _Tp{6.149266274460007351509223690936135535960e-1↩
L} [static]
```

Definition at line 134 of file sf_airy.tcc.

10.43.4.5 _S_Bip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Bip0 = _Tp{4.482883573538263579148237103988283908668e-1↩
L} [static]
```

Definition at line 136 of file sf_airy.tcc.

10.43.4.6 _S_eps

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_eps = __gnu_cxx::__epsilon(_Tp{}) [static]
```

Definition at line 125 of file sf_airy.tcc.

10.43.4.7 _S_Gi0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Gi0 = _Tp{2.049755424820002450503074563645378511979e-1↩
L} [static]
```

Definition at line 142 of file sf_airy.tcc.

10.43.4.8 _S_Gip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Gip0 = _Tp{1.494294524512754526382745701329427969551e-1↩
L} [static]
```

Definition at line 144 of file sf_airy.tcc.

10.43.4.9 _S_Hi0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hi0 = _Tp{4.099510849640004901006149127290757023959e-1↩
L} [static]
```

Definition at line 138 of file sf_airy.tcc.

10.43.4.10 _S_Hip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hip0 = _Tp{2.988589049025509052765491402658855939102e-1↩
L} [static]
```

Definition at line 140 of file sf_airy.tcc.

10.43.4.11 _S_i

```
template<typename _Tp >
constexpr std::complex< _Tp > std::__detail::_Airy_series< _Tp >::_S_i {_Tp{0}, _Tp{1}} [static]
```

Definition at line 145 of file sf_airy.tcc.

10.43.4.12 _S_pi

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_pi = __gnu_cxx::__const_pi(_Tp{}) [static]
```

Definition at line 126 of file sf_airy.tcc.

10.43.4.13 _S_sqrt_pi

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_sqrt_pi = __gnu_cxx::__const_root_pi(_Tp{})
[static]
```

Definition at line 128 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- bits/sf_airy.tcc

10.44 std::__detail::_AiryAuxilliaryState< _Tp > Struct Template Reference

Public Types

- using _Val = std::__detail::__num_traits_t< _Tp >

Public Attributes

- _Tp __fai_deriv
- _Tp __fai_value
- _Tp __gai_deriv
- _Tp __gai_value
- _Tp __hai_deriv
- _Tp __hai_value
- _Tp __z

10.44.1 Detailed Description

template<typename _Tp>
struct std::__detail::_AiryAuxilliaryState< _Tp >

A structure containing three auxilliary Airy functions and their derivatives.

Definition at line 80 of file sf_airy.tcc.

10.44.2 Member Typedef Documentation

10.44.2.1 _Val

```
template<typename _Tp>
using std::__detail::_AiryAuxilliaryState< _Tp >::_Val = std::__detail::__num_traits_t<_Tp>
```

Definition at line 82 of file sf_airy.tcc.

10.44.3 Member Data Documentation

10.44.3.1 __fai_deriv

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__fai_deriv
```

Definition at line 86 of file sf_airy.tcc.

10.44.3.2 __fai_value

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__fai_value
```

Definition at line 85 of file sf_airy.tcc.

10.44.3.3 __gai_deriv

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__gai_deriv
```

Definition at line 88 of file sf_airy.tcc.

10.44.3.4 __gai_value

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__gai_value
```

Definition at line 87 of file sf_airy.tcc.

10.44.3.5 __hai_deriv

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__hai_deriv
```

Definition at line 90 of file sf_airy.tcc.

10.44.3.6 __hai_value

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__hai_value
```

Definition at line 89 of file sf_airy.tcc.

10.44.3.7 __z

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__z
```

Definition at line 84 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.45 std::__detail::_AiryState< _Tp > Struct Template Reference

Public Types

- using _Real = std::__detail::__num_traits_t< _Tp >

Public Member Functions

- _Real true_Wronskian ()
- _Tp Wronskian () const

Public Attributes

- _Tp __Ai_deriv
- _Tp __Ai_value
- _Tp __Bi_deriv
- _Tp __Bi_value
- _Tp __z

10.45.1 Detailed Description

template<typename _Tp>
struct std::__detail::_AiryState< _Tp >

This struct defines the Airy function state with two presumably numerically useful Airy functions and their derivatives. The data mambers are directly accessible. The lone method computes the Wronskian from the stored functions. A static method returns the correct Wronskian.

Definition at line 55 of file sf_airy.tcc.

10.45.2 Member Typedef Documentation

10.45.2.1 _Real

```
template<typename _Tp>
using std::__detail::_AiryState< _Tp >::_Real = std::__detail::__num_traits_t<_Tp>
```

Definition at line 57 of file sf_airy.tcc.

10.45.3 Member Function Documentation

10.45.3.1 true_Wronskian()

```
template<typename _Tp>
_Real std::__detail::_AiryState< _Tp >::true_Wronskian ( ) [inline]
```

Definition at line 70 of file sf_airy.tcc.

10.45.3.2 Wronskian()

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::Wronskian ( ) const [inline]
```

Definition at line 66 of file sf_airy.tcc.

References std::__detail::_AiryState< _Tp >::__Ai_deriv.

10.45.4 Member Data Documentation

10.45.4.1 __Ai_deriv

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__Ai_deriv
```

Definition at line 61 of file sf_airy.tcc.

Referenced by std::__detail::_AiryState< _Tp >::Wronskian().

10.45.4.2 __Ai_value

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__Ai_value
```

Definition at line 60 of file sf_airy.tcc.

10.45.4.3 __Bi_deriv

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__Bi_deriv
```

Definition at line 63 of file sf_airy.tcc.

10.45.4.4 __Bi_value

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__Bi_value
```

Definition at line 62 of file sf_airy.tcc.

10.45.4.5 __z

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__z
```

Definition at line 59 of file sf_airy.tcc.

Referenced by std::__detail::_Airy_asymp< _Tp >::_S_absarg_ge_pio3(), std::__detail::_Airy_asymp< _Tp >::_S_absarg_lt_pio3(), and std::__detail::_Airy_asymp< _Tp >::operator()().

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.46 std::__detail::_AsympTerminator< _Tp > Class Template Reference

Public Member Functions

- _AsympTerminator (std::size_t __max_iter, _Real __mul=_Real{1})
- std::size_t num_terms () const

  Return the current number of terms summed.
- bool operator() (_Tp __term, _Tp __sum)

  Detect if the sum should terminate either because the incoming term is small enough or because the terms are starting to grow or.
- _Tp operator<< (_Tp __term)

  Filter a term before applying it to the sum.

10.46.1 Detailed Description

template<typename _Tp>
class std::__detail::_AsympTerminator< _Tp >

This class manages the termination of asymptotic series. In particular, this termination watches for the growth of the sequence of terms to stop the series.

Termination conditions involve both a maximum iteration count and a relative precision.

Definition at line 108 of file sf_polylog.tcc.

10.46.2 Constructor & Destructor Documentation

10.46.2.1 _AsympTerminator()

```
template<typename _Tp>
std::__detail::_AsympTerminator< _Tp >::_AsympTerminator (
            std::size_t __max_iter,
            _Real __mul = _Real{1} ) [inline]
```

Definition at line 121 of file sf_polylog.tcc.

10.46.3 Member Function Documentation

10.46.3.1 num_terms()

```
template<typename _Tp>
std::size_t std::__detail::_AsympTerminator< _Tp >::num_terms ( ) const [inline]
```

Return the current number of terms summed.

Definition at line 141 of file sf_polylog.tcc.

10.46.3.2 operator()()

```
template<typename _Tp>
bool std::__detail::_AsympTerminator< _Tp >::operator() (
            _Tp __term,
            _Tp __sum ) [inline]
```

Detect if the sum should terminate either because the incoming term is small enough or because the terms are starting to grow or.

Definition at line 148 of file sf_polylog.tcc.

10.46.3.3 operator<<()

```
template<typename _Tp>
_Tp std::__detail::_AsympTerminator< _Tp >::operator<< (
            _Tp __term ) [inline]
```

Filter a term before applying it to the sum.

Definition at line 128 of file sf_polylog.tcc.

The documentation for this class was generated from the following file:

- bits/sf_polylog.tcc

10.47 std::__detail::_Factorial_table< _Tp > Struct Template Reference

Public Attributes

- _Tp __factorial
- _Tp __log_factorial
- int __n

10.47.1 Detailed Description

template<typename _Tp>
struct std::__detail::_Factorial_table< _Tp >

Definition at line 65 of file sf_gamma.tcc.

10.47.2 Member Data Documentation

10.47.2.1 __factorial

```
template<typename _Tp >
_Tp std::__detail::_Factorial_table< _Tp >::__factorial
```

Definition at line 68 of file sf_gamma.tcc.

Referenced by std::__detail::__double_factorial(), and std::__detail::__gamma_reciprocal().

10.47.2.2 __log_factorial

```
template<typename _Tp >
_Tp std::__detail::_Factorial_table< _Tp >::__log_factorial
```

Definition at line 69 of file sf_gamma.tcc.

Referenced by std::__detail::__log_double_factorial(), and std::__detail::__log_gamma().

10.47.2.3 __n

```
template<typename _Tp >
int std::__detail::_Factorial_table< _Tp >::__n
```

Definition at line 67 of file sf_gamma.tcc.

Referenced by std::__detail::__binomial(), std::__detail::__double_factorial(), std::__detail::__factorial(), std::__detail::__falling_factorial(), std::__detail::__gamma(), std::__detail::__gamma_cont_frac(), std::__detail::__gamma_reciprocal(), std::__detail::__gamma_series(), std::__detail::__harmonic_number(), std::__detail::__lanczos_binet1p(), std::__detail::__log_binomial(), std::__detail::__log_binomial_sign(), std::__detail::__log_double_factorial(), std::__detail::__log_factorial(), std::__detail::__log_gamma(), std::__detail::__psi(), and std::__detail::__rising_factorial().

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.48 std::__detail::_Terminator< _Tp > Class Template Reference

Public Member Functions

- _Terminator (std::size_t __max_iter, _Real __mul=_Real{1})
- std::size_t num_terms () const

  Return the current number of terms summed.
- bool operator() (_Tp __term, _Tp __sum)

  Detect if the sum should terminate either because the incoming term is small enough or the maximum number of terms has been reached.

10.48.1 Detailed Description

template<typename _Tp>
class std::__detail::_Terminator< _Tp >

This class manages the termination of series. Termination conditions involve both a maximum iteration count and a relative precision.

Definition at line 63 of file sf_polylog.tcc.

10.48.2 Constructor & Destructor Documentation

10.48.2.1 _Terminator()

```
template<typename _Tp>
std::__detail::_Terminator< _Tp >::_Terminator (
            std::size_t __max_iter,
            _Real __mul = _Real{1} )  [inline]
```

Definition at line 74 of file sf_polylog.tcc.

10.48.3 Member Function Documentation

10.48.3.1 num_terms()

```
template<typename _Tp>
std::size_t std::__detail::_Terminator< _Tp >::num_terms ( ) const  [inline]
```

Return the current number of terms summed.

Definition at line 81 of file sf_polylog.tcc.

10.48.3.2 operator()()

```
template<typename _Tp>
bool std::__detail::_Terminator< _Tp >::operator() (
            _Tp __term,
            _Tp __sum )  [inline]
```

Detect if the sum should terminate either because the incoming term is small enough or the maximum number of terms has been reached.

Definition at line 87 of file sf_polylog.tcc.

The documentation for this class was generated from the following file:

- bits/sf_polylog.tcc

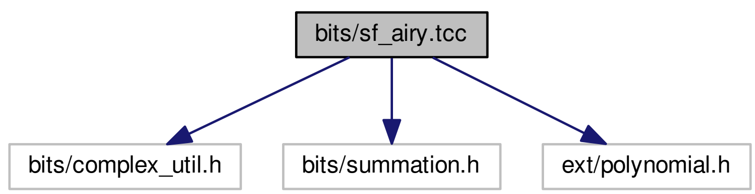
Chapter 11

File Documentation

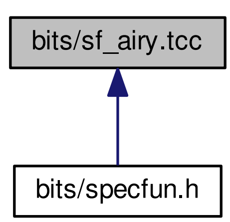
11.1 bits/sf_airy.tcc File Reference

```
#include <bits/complex_util.h>
#include <bits/summation.h>
#include <ext/polynomial.h>
```

Include dependency graph for sf_airy.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- class std::__detail::_Airy< _Tp >
- class std::__detail::_Airy_asymp< _Tp >
- struct std::__detail::_Airy_asymp_data< _Tp >
- struct std::__detail::_Airy_asymp_data< double >
- struct std::__detail::_Airy_asymp_data< float >
- struct std::__detail::_Airy_asymp_data< long double >
- class std::__detail::_Airy_asymp_series< _Sum >
- struct std::__detail::_Airy_default_radii< _Tp >
- struct std::__detail::_Airy_default_radii< double >
- struct std::__detail::_Airy_default_radii< float >
- struct std::__detail::_Airy_default_radii< long double >
- class std::__detail::_Airy_series< _Tp >
- struct std::__detail::_AiryAuxilliaryState< _Tp >
- struct std::__detail::_AiryState< _Tp >

Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_AIRY_TCC 1

Functions

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__airy_ai (std::complex< _Tp > __z)

  Return the complex Airy Ai function.
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__airy_bi (std::complex< _Tp > __z)

  Return the complex Airy Bi function.

Variables

- template<typename _Tp >
  constexpr int std::__detail::__max_FGH = _Airy_series<_Tp>::_N_FGH
- template<>
  constexpr int std::__detail::__max_FGH< double > = 79
- template<>
  constexpr int std::__detail::__max_FGH< float > = 15

11.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.1.2 Macro Definition Documentation

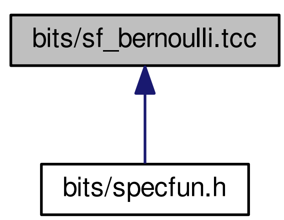
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC

```
#define _GLIBCXX_BITS_SF_AIRY_TCC 1
```

Definition at line 31 of file sf_airy.tcc.

11.2 bits/sf_bernoulli.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1

Functions

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (unsigned int __n)

  This returns Bernoulli number B_n.
- template<typename _Tp >
  _Tp std::__detail::__bernoulli (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (unsigned int __n)

  This returns Bernoulli number B_2n at even integer arguments $2n$.
- 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.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

.

11.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.2.2 Macro Definition Documentation

11.2.2.1 _GLIBCXX_BITS_SF_BERNOULLI_TCC

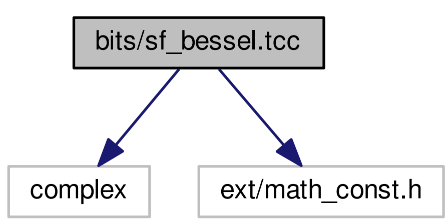
```
#define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1
```

Definition at line 35 of file sf_bernoulli.tcc.

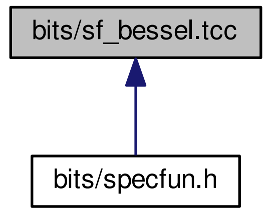
11.3 bits/sf_bessel.tcc File Reference

```
#include <complex>
#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 >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x)

  Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- template<typename _Tp >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, std::complex< _Tp > > std::__detail::__cyl_bessel_jn_neg_arg (_Tp ↩ __nu, _Tp __x)

  Return the cylindrical Bessel functions and their derivatives of order ν and argument $x < 0$.
- template<typename _Tp >
  __gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_jn_steed (_Tp __nu, _Tp __x)

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 >
  __gnu_cxx::__gamma_temme_t< _Tp > std::__detail::__gamma_temme (_Tp __mu)

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

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

  and

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

  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 >
  __gnu_cxx::__sph_bessel_t< unsigned int, _Tp, _Tp > std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x)

  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 >
  __gnu_cxx::__sph_bessel_t< unsigned int, _Tp, std::complex< _Tp > > std::__detail::__sph_bessel_jn_neg↩_arg (unsigned int __n, _Tp __x)
- 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.

11.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.3.2 Macro Definition Documentation

11.3.2.1 _GLIBCXX_BITS_SF_BESSEL_TCC

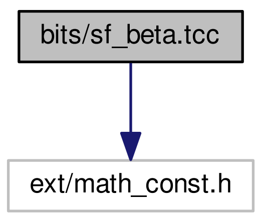
```
#define _GLIBCXX_BITS_SF_BESSEL_TCC 1
```

Definition at line 47 of file sf_bessel.tcc.

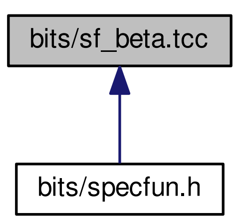
11.4 bits/sf_beta.tcc File Reference

```
#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_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 >
  _Tp std::__detail::__ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)

11.4.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.4.2 Macro Definition Documentation

11.4.2.1 _GLIBCXX_BITS_SF_BETA_TCC

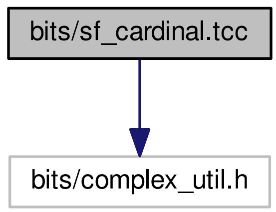
```
#define _GLIBCXX_BITS_SF_BETA_TCC 1
```

Definition at line 49 of file sf_beta.tcc.

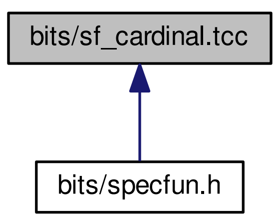
11.5 bits/sf_cardinal.tcc File Reference

`#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_fp_t< _Tp > std::__detail::__sinc (_Tp __x)

  Return the sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinc_pi (_Tp __x)

  Return the reperiodized sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc (_Tp __x)

  Return the hyperbolic sinus cardinal function

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

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)

  Return the reperiodized hyperbolic sinus cardinal function

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

  .

11.5.1 Macro Definition Documentation

11.5.1.1 _GLIBCXX_BITS_SF_CARDINAL_TCC

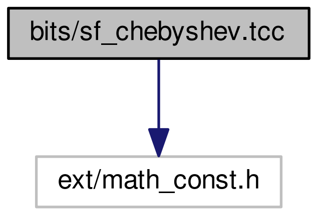
```
#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1
```

Definition at line 31 of file sf_cardinal.tcc.

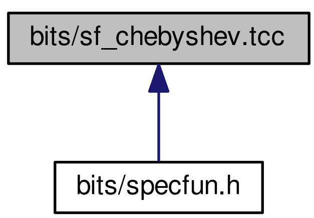
11.6 bits/sf_chebyshev.tcc File Reference

`#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_BITS_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)

11.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.6.2 Macro Definition Documentation

11.6.2.1 _GLIBCXX_BITS_SF_CHEBYSHEV_TCC

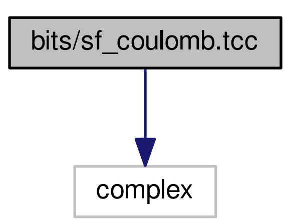
```
#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1
```

Definition at line 31 of file sf_chebyshev.tcc.

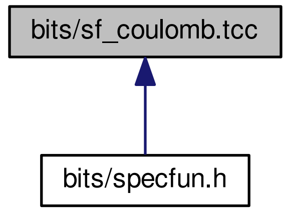
11.7 bits/sf_coulomb.tcc File Reference

```
#include <complex>
```

Include dependency graph for sf_coulomb.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_COULOMB_TCC 1

Functions

- template< typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__coulomb_CF1 (unsigned int __l, _Tp __eta, _Tp __x)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__coulomb_CF2 (unsigned int __l, _Tp __eta, _Tp __x)
- template< typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__coulomb_f_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _F_l_max, _Tp _Fp_l_max)
- template< typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__coulomb_g_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _G_l_min, _Tp _Gp_l_min)
- template< typename _Tp >
  _Tp std::__detail::__coulomb_norm (unsigned int __l, _Tp __eta)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__hydrogen (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi)

11.7.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.7.2 Macro Definition Documentation

11.7.2.1 _GLIBCXX_BITS_SF_COULOMB_TCC

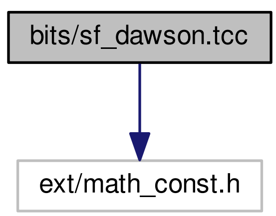
```
#define _GLIBCXX_BITS_SF_COULOMB_TCC 1
```

Definition at line 31 of file sf_coulomb.tcc.

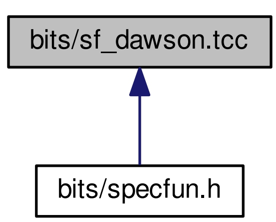
11.8 bits/sf_dawson.tcc File Reference

```
#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_BITS_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.

11.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.8.2 Macro Definition Documentation

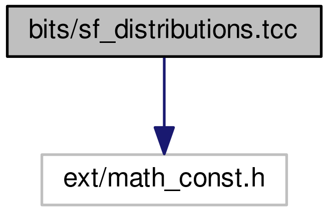
11.8.2.1 _GLIBCXX_BITS_SF_DAWSON_TCC

```
#define _GLIBCXX_BITS_SF_DAWSON_TCC 1
```

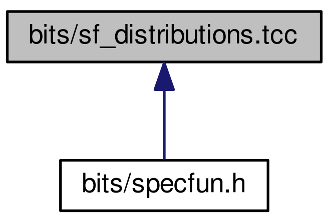
Definition at line 31 of file sf_dawson.tcc.

11.9 bits/sf_distributions.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.
- template<typename _Tp >
  _Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)

  Return the complementary binomial cumulative distribution function.
- template<typename _Tp >
  _Tp std::__detail::__binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial probability mass function.
- template<typename _Tp >
  _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 >
  _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 >
  _Tp std::__detail::__exponential_cdf (_Tp __lambda, _Tp __x)

  Return the exponential cumulative probability density function.
- template<typename _Tp >
  _Tp std::__detail::__exponential_cdfc (_Tp __lambda, _Tp __x)

  Return the complement of the exponential cumulative probability density function.
- template<typename _Tp >
  _Tp std::__detail::__exponential_pdf (_Tp __lambda, _Tp __x)

  Return the exponential probability density function.
- template<typename _Tp >
  _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 >
  _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 >
  _Tp std::__detail::__fisher_f_pdf (_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 >
  _Tp std::__detail::__gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma cumulative propability distribution function.
- template<typename _Tp >
  _Tp std::__detail::__gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma complementary cumulative propability distribution function.
- template<typename _Tp >
  _Tp std::__detail::__gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)

  Return the gamma propability distribution function.

- template<typename _Tp >
  _Tp std::__detail::__logistic_cdf (_Tp __a, _Tp __b, _Tp __x)

  Return the logistic cumulative distribution function.
- template<typename _Tp >
  _Tp std::__detail::__logistic_pdf (_Tp __a, _Tp __b, _Tp __x)

  Return the logistic probability density function.
- template<typename _Tp >
  _Tp std::__detail::__lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the lognormal cumulative probability density function.
- template<typename _Tp >
  _Tp std::__detail::__lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)

  Return the lognormal probability density function.
- template<typename _Tp >
  _Tp std::__detail::__normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the normal cumulative probability density function.
- template<typename _Tp >
  _Tp std::__detail::__normal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)

  Return the normal probability density function.
- template<typename _Tp >
  _Tp std::__detail::__rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)

  Return the Rice probability density function.
- template<typename _Tp >
  _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)

  Return the Students T probability function.
- template<typename _Tp >
  _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template<typename _Tp >
  _Tp std::__detail::__student_t_pdf (_Tp __t, unsigned int __nu)

  Return the Students T probability density.
- template<typename _Tp >
  _Tp std::__detail::__weibull_cdf (_Tp __a, _Tp __b, _Tp __x)

  Return the Weibull cumulative probability density function.
- template<typename _Tp >
  _Tp std::__detail::__weibull_pdf (_Tp __a, _Tp __b, _Tp __x)

  Return the Weibull probability density function.

11.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.9.2 Macro Definition Documentation

11.9.2.1 _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC

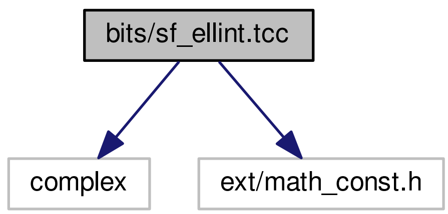
```
#define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1
```

Definition at line 49 of file sf_distributions.tcc.

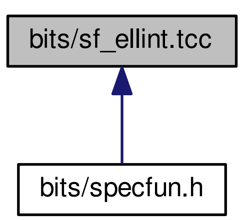
11.10 bits/sf_ellint.tcc File Reference

```
#include <complex>
#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)

11.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.10.2 Macro Definition Documentation

11.10.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC

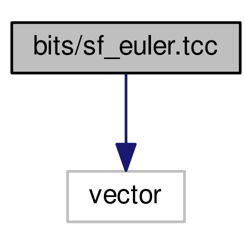
```
#define _GLIBCXX_BITS_SF_ELLINT_TCC 1
```

Definition at line 47 of file sf_ellint.tcc.

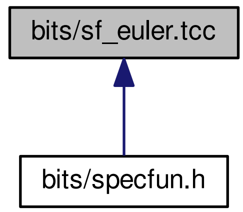
11.11 bits/sf_euler.tcc File Reference

```
#include <vector>
```

Include dependency graph for sf_euler.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_EULER_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__euler (unsigned int __n)

  This returns Euler number E_n.
- template<typename _Tp >
  _Tp std::__detail::__euler (unsigned int __n, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__euler_series (unsigned int __n)
- template<typename _Tp >
  _Tp std::__detail::__eulerian_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__eulerian_1_recur (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__eulerian_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__eulerian_2_recur (unsigned int __n, unsigned int __m)

11.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.11.2 Macro Definition Documentation

11.11.2.1 _GLIBCXX_BITS_SF_EULER_TCC

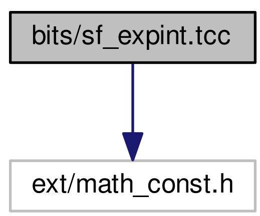
```
#define _GLIBCXX_BITS_SF_EULER_TCC 1
```

Definition at line 35 of file sf_euler.tcc.

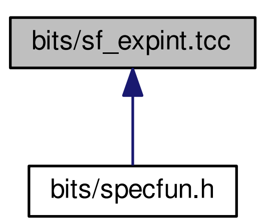
11.12 bits/sf_expint.tcc File Reference

```
#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 $Chi(x)$.
- 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_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_asymp (unsigned int __n, _Tp __x)

  Return the exponential integral $E_n(x)$ for large argument.
- 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_large_n (unsigned int __n, _Tp __x)

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

- 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::__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 $Shi(x)$.

11.12.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.12.2 Macro Definition Documentation

11.12.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC

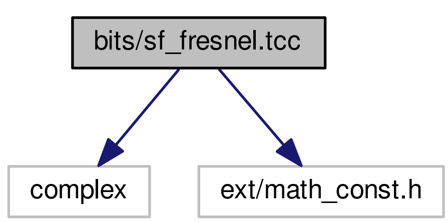
```
#define _GLIBCXX_BITS_SF_EXPINT_TCC 1
```

Definition at line 47 of file sf_expint.tcc.

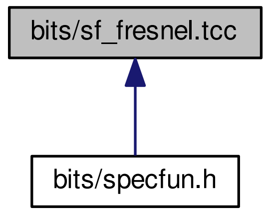
11.13 bits/sf_fresnel.tcc File Reference

```
#include <complex>
#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_BITS_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.

11.13.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.13.2 Macro Definition Documentation

11.13.2.1 _GLIBCXX_BITS_SF_FRESNEL_TCC

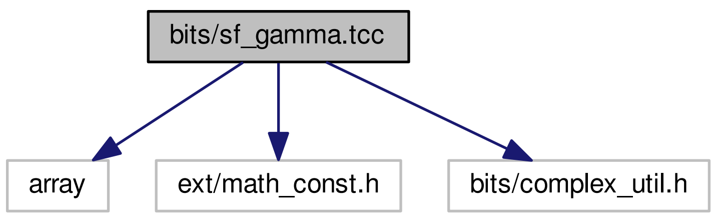
```
#define _GLIBCXX_BITS_SF_FRESNEL_TCC 1
```

Definition at line 31 of file sf_fresnel.tcc.

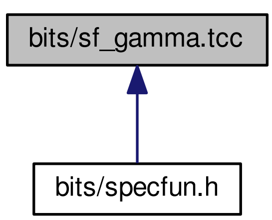
11.14 bits/sf_gamma.tcc File Reference

```
#include <array>
#include <ext/math_const.h>
#include <bits/complex_util.h>
```

Include dependency graph for sf_gamma.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- struct std::__detail::__gamma_lanczos_data< _Tp >
- struct std::__detail::__gamma_lanczos_data< double >
- struct std::__detail::__gamma_lanczos_data< float >
- struct std::__detail::__gamma_lanczos_data< long double >
- struct std::__detail::__gamma_spouge_data< _Tp >
- struct std::__detail::__gamma_spouge_data< double >
- struct std::__detail::__gamma_spouge_data< float >
- struct std::__detail::__gamma_spouge_data< long double >
- struct std::__detail::_Factorial_table< _Tp >

Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_GAMMA_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__binomial (unsigned int __n, unsigned int __k)

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

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

  The binomial coefficients are generated by:

  $$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  _Tp std::__detail::__binomial (_Tp __nu, unsigned int __k)

  Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

  $$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

  $$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- 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::__falling_factorial (_Tp __a, int __n)

  Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

  In particular, $n^{\underline{n}} = n!$.

- template<typename _Tp >
  _Tp std::__detail::__falling_factorial (_Tp __a, _Tp __nu)

  Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν. The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a+1)/\Gamma(a-\nu+1)$$

  .

- template<typename _Tp >
  _Tp std::__detail::__gamma (_Tp __a)

  Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^\infty e^{-t}t^{a-1}dt(a > 0)$$

  .

- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__gamma (_Tp __a, _Tp __x)

  Return the incomplete gamma functions.

- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)

  Return the incomplete gamma function by continued fraction.

- template<typename _Tp >
  _Tp std::__detail::__gamma_reciprocal (_Tp __a)
- template<typename _Tp >
  _Tp std::__detail::__gamma_reciprocal_series (_Tp __a)
- template<typename _Tp >
  std::pair< _Tp, _Tp > std::__detail::__gamma_series (_Tp __a, _Tp __x)

  Return the incomplete gamma function by series summation.

$$\gamma(a,x) = x^a e^{-z} \sum_{k=1}^\infty \frac{x^k}{(a)_k}$$

  .

- template<typename _Tp >
  _Tp std::__detail::__harmonic_number (unsigned int __n)
- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (_Tp __z)

  Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $log(\Gamma^*(z))$ defined by

$$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

  or

$$\Gamma(z) = \sqrt{2\pi}z^{z-\frac{1}{2}}e^{-z}e^{J(z)}$$

  where $\Gamma(z)$ is the gamma function.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_log_gamma1p (_Tp __z)

  Return the logarithm of the gamma function $log(\Gamma(1+z))$ *by the Lanczos method.*
- template<typename _Tp >
  _Tp std::__detail::__log_binomial (unsigned int __n, unsigned int __k)

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

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

  The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  _Tp std::__detail::__log_binomial (_Tp __nu, unsigned int __k)

  Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- template<typename _Tp >
  _Tp std::__detail::__log_binomial_sign (_Tp __nu, unsigned int __k)

  Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

  The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

  .
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__log_binomial_sign (std::complex< _Tp > __nu, unsigned int __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_falling_factorial (_Tp __a, _Tp __nu)

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

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

  In particular, $n^{\underline{n}} = n!$*. Thus this function returns*

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

Many notations exist for this function:

$$(a)_\nu$$

,

$$\{ \begin{array}{c} a \\ \nu \end{array} \}$$

, and others.

- template<typename _Tp >
  _Tp std::__detail::__log_gamma (_Tp __a)

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

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__log_gamma (std::complex< _Tp > __a)

  Return $log(\Gamma(a))$ for complex argument.

- 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 >
  _Tp std::__detail::__log_gamma_sign (_Tp __a)

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

- template<typename _Tp >
  std::complex< _Tp > std::__detail::__log_gamma_sign (std::complex< _Tp > __a)

- template<typename _Tp >
  _Tp std::__detail::__log_rising_factorial (_Tp __a, _Tp __nu)

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

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

  Thus this function returns

$$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[(a)_0] = 0$$

  Many notations exist for this function:

$$(a)_\nu$$

  (especially in the literature of special functions),

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

  , and others.

- template<typename _Tp >
  _Tp std::__detail::__pgamma (_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::__psi (unsigned int __n)

Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

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

The digamma series for integral argument is given by:

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

The latter sum is called the harmonic number, H_n.

- 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

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

  .
- template<typename _Tp >
  _Tp std::__detail::__qgamma (_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 >
  _Tp std::__detail::__rising_factorial (_Tp __a, int __n)

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

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

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

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

, and others.

- template<typename _Tp >
  _Tp std::__detail::__rising_factorial (_Tp __a, _Tp __nu)

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

  $$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

  Many notations exist for this function:

  $$(a)_\nu$$

  , (especially in the literature of special functions),

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

  , and others.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_binet1p (_Tp __z)

  Return the Binet function $J(1+z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $log(\Gamma^*(z))$ defined by

  $$J(z) = log(\Gamma^*(z)) = log\left(\Gamma(z)\right) + z - \left(z - \frac{1}{2}\right) log(z) - log(2\pi)$$

  or

  $$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

  where $\Gamma(z)$ is the gamma function.

- template<typename _Tp >
  _GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_log_gamma1p (_Tp __z)

  Return the logarithm of the gamma function $log(\Gamma(1+z))$ 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::__tgamma (_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::__tgamma_lower (_Tp __a, _Tp __x)

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

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

  .

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 unsigned long long std::__detail::_S_harmonic_denom [_S_num_harmonic_numer]
- constexpr unsigned long long std::__detail::_S_harmonic_numer [_S_num_harmonic_numer]
- 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
- constexpr unsigned long long std::__detail::_S_num_harmonic_numer = 29
- 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

11.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.14.2 Macro Definition Documentation

11.14.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC

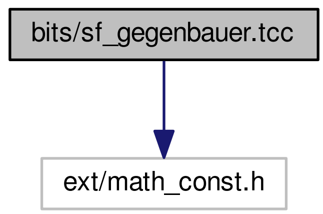
```
#define _GLIBCXX_BITS_SF_GAMMA_TCC 1
```

Definition at line 49 of file sf_gamma.tcc.

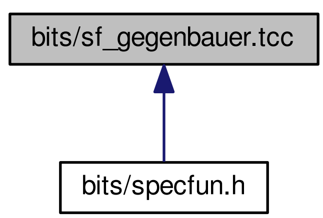
11.15 bits/sf_gegenbauer.tcc File Reference

`#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_BITS_SF_GEGENBAUER_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha1, _Tp __x)
- template<typename _Tp >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__gegenbauer_zeros (unsigned int __n, _Tp __alpha1)

11.15.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.15.2 Macro Definition Documentation

11.15.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC

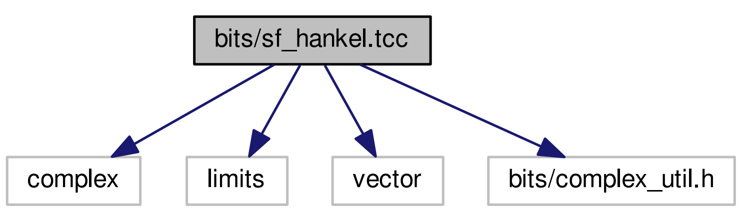
```
#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1
```

Definition at line 31 of file sf_gegenbauer.tcc.

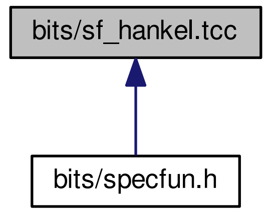
11.16 bits/sf_hankel.tcc File Reference

```
#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 >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn)
- 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 >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  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 >
  __gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z)

  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 >
  __gnu_cxx::__sph_hankel_t< unsigned int, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z)

  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.

11.16.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.16.2 Macro Definition Documentation

11.16.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC

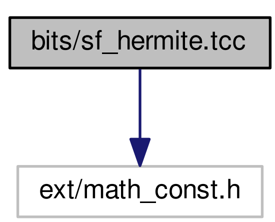
```
#define _GLIBCXX_BITS_SF_HANKEL_TCC 1
```

Definition at line 31 of file sf_hankel.tcc.

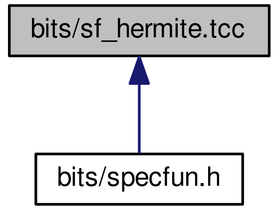
11.17 bits/sf_hermite.tcc File Reference

```
#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::__hermite (unsigned int __n, _Tp __x)

  This routine returns the Hermite polynomial of order n: $H_n(x)$.
- template< typename _Tp >
  _Tp std::__detail::__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 >
  __gnu_cxx::__hermite_t< _Tp > std::__detail::__hermite_recur (unsigned int __n, _Tp __x)

  This routine returns the Hermite polynomial of order n: $H_n(x)$ by recursion on n.
- template< typename _Tp >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__hermite_zeros (unsigned int __n, _Tp __proto=_Tp{})
- template< typename _Tp >
  __gnu_cxx::__hermite_he_t< _Tp > std::__detail::__prob_hermite_recursion (unsigned int __n, _Tp __x)

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

11.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.17.2 Macro Definition Documentation

11.17.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC

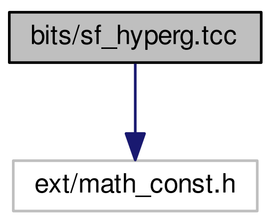
```
#define _GLIBCXX_BITS_SF_HERMITE_TCC 1
```

Definition at line 42 of file sf_hermite.tcc.

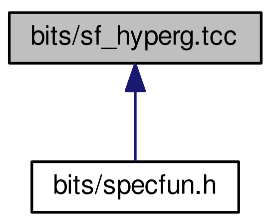
11.18 bits/sf_hyperg.tcc File Reference

```
#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 ${}_1F_1(a;c;x) = M(a,c,x)$.
- 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.*
- template<typename _Tp >
  _Tp std::__detail::__tricomi_u (_Tp __a, _Tp __c, _Tp __x)

  Return the Tricomi confluent hypergeometric function

  $$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1;2-c;x)$$

  .
- template<typename _Tp >
  _Tp std::__detail::__tricomi_u_naive (_Tp __a, _Tp __c, _Tp __x)

  Return the Tricomi confluent hypergeometric function

  $$U(a,c,x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a;c;x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1;2-c;x)$$

  .

11.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.18.2 Macro Definition Documentation

11.18.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC

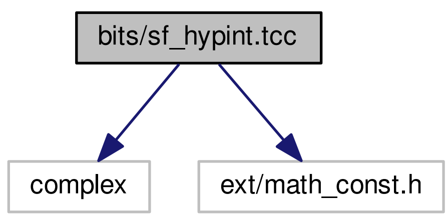
```
#define _GLIBCXX_BITS_SF_HYPERG_TCC 1
```

Definition at line 44 of file sf_hyperg.tcc.

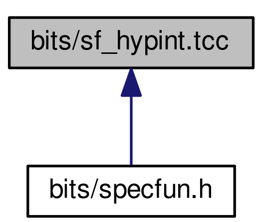
11.19 bits/sf_hypint.tcc File Reference

```
#include <complex>
#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_BITS_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.

11.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.19.2 Macro Definition Documentation

11.19.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC

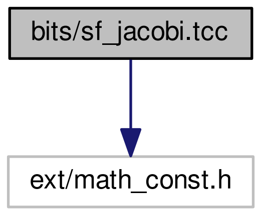
```
#define _GLIBCXX_BITS_SF_HYPINT_TCC 1
```

Definition at line 31 of file sf_hypint.tcc.

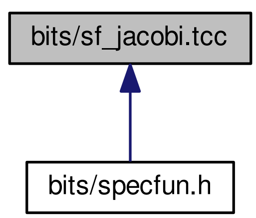
11.20 bits/sf_jacobi.tcc File Reference

```
#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_BITS_SF_JACOBI_TCC 1

Functions

- template<typename _Tp >
  __gnu_cxx::__jacobi_t< _Tp > std::__detail::__jacobi_recur (unsigned int __n, _Tp __alpha1, _Tp __beta1, _Tp __x)
- template<typename _Tp >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__jacobi_zeros (unsigned int __n, _Tp __alpha1, _Tp __beta1)
- template<typename _Tp >
  _Tp std::__detail::__poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)

11.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.20.2 Macro Definition Documentation

11.20.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC

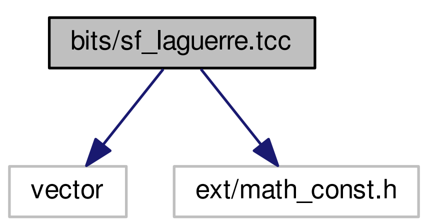
```
#define _GLIBCXX_BITS_SF_JACOBI_TCC 1
```

Definition at line 31 of file sf_jacobi.tcc.

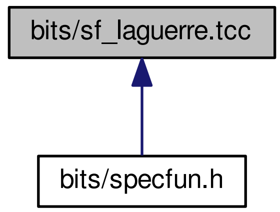
11.21 bits/sf_laguerre.tcc File Reference

```
#include <vector>
#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 _Tpa , typename _Tp >
  _Tp std::__detail::__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 _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::__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::__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 >
  __gnu_cxx::__laguerre_t< _Tpa, _Tp > std::__detail::__laguerre_recur (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 >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__laguerre_zeros (unsigned int __n, _Tp __alpha1)

11.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.21.2 Macro Definition Documentation

11.21.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC

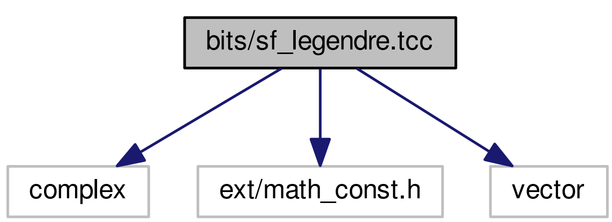
```
#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1
```

Definition at line 44 of file sf_laguerre.tcc.

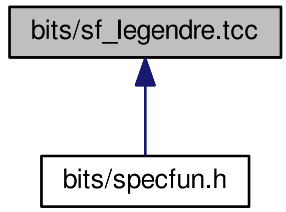
11.22 bits/sf_legendre.tcc File Reference

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

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 >
  __gnu_cxx::__legendre_p_t< _Tp > std::__detail::__legendre_p (unsigned int __l, _Tp __x)

  Return the Legendre polynomial by upward recursion on order l.
- 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 >
  std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__legendre_zeros (unsigned int __l, _Tp proto=_Tp{})
- 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.

11.22.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.22.2 Macro Definition Documentation

11.22.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC

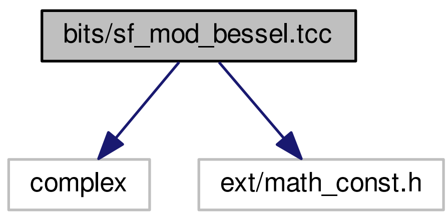
```
#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1
```

Definition at line 47 of file sf_legendre.tcc.

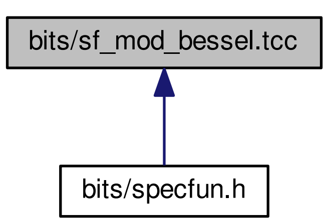
11.23 bits/sf_mod_bessel.tcc File Reference

```
#include <complex>
#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 >
  __gnu_cxx::__airy_t< _Tp, _Tp > std::__detail::__airy (_Tp __z)

  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 >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x)

  Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- template<typename _Tp >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik_steed (_Tp __nu, _Tp __x)

  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 >
  __gnu_cxx::__fock_airy_t< _Tp, std::complex< _Tp > > std::__detail::__fock_airy (_Tp __x)

  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 >
  __gnu_cxx::__sph_mod_bessel_t< unsigned int, _Tp, _Tp > std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x)

  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.

11.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.23.2 Macro Definition Documentation

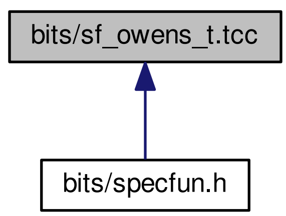
11.23.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC

```
#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1
```

Definition at line 47 of file sf_mod_bessel.tcc.

11.24 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)

11.24.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.24.2 Macro Definition Documentation

11.24.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC

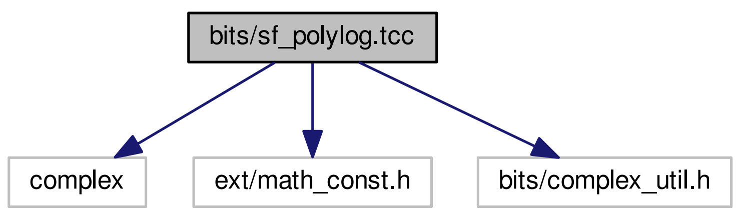
```
#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1
```

Definition at line 31 of file sf_owens_t.tcc.

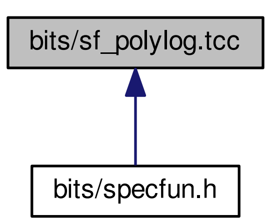
11.25 bits/sf_polylog.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
#include <bits/complex_util.h>
```

Include dependency graph for sf_polylog.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- class std::__detail::_AsympTerminator< _Tp >
- class std::__detail::_Terminator< _Tp >

Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_POLYLOG_TCC 1

Functions

- template<typename _Sp , typename _Tp >
  _Tp std::__detail::__bose_einstein (_Sp __s, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clamp_0_m2pi (std::complex< _Tp > __z)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clamp_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__clausen (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__clausen (unsigned int __m, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__clausen_cl (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__clausen_cl (unsigned int __m, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__clausen_sl (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__clausen_sl (unsigned int __m, _Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_beta (std::complex< _Tp > __s)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_beta (_Tp __s)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __s)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_eta (_Tp __s)
- template<typename _Tp >
  _Tp std::__detail::__dirichlet_lambda (_Tp __s)
- template<typename _Sp , typename _Tp >
  _Tp std::__detail::__fermi_dirac (_Sp __s, _Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__hurwitz_zeta_polylog (_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_fp_t< std::complex< _Tp >, _ArgType > std::__detail::__polylog_exp (_Tp __s, _ArgType __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_neg (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_neg (int __n, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __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_pos_int (unsigned int __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_pos_int (unsigned int __s, _Tp __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, std::complex< _Tp > __w)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, _Tp __w)
- template<typename _PowTp , typename _Tp >
  _Tp std::__detail::__polylog_exp_sum (_PowTp __s, _Tp __w)

11.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.25.2 Macro Definition Documentation

11.25.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC

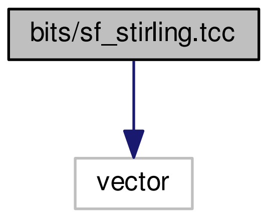
```
#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1
```

Definition at line 41 of file sf_polylog.tcc.

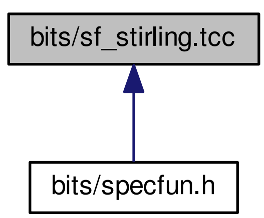
11.26 bits/sf_stirling.tcc File Reference

```
#include <vector>
```

Include dependency graph for sf_stirling.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_STIRLING_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__log_stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__log_stirling_1_sign (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__log_stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_1_recur (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_1_series (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_2_recur (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp std::__detail::__stirling_2_series (unsigned int __n, unsigned int __m)

11.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.26.2 Macro Definition Documentation

11.26.2.1 _GLIBCXX_BITS_SF_STIRLING_TCC

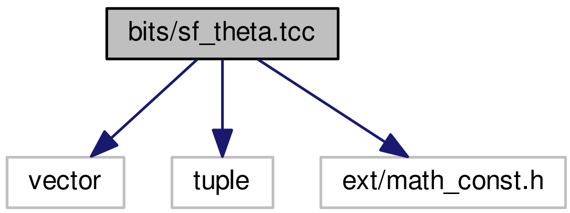
```
#define _GLIBCXX_BITS_SF_STIRLING_TCC 1
```

Definition at line 35 of file sf_stirling.tcc.

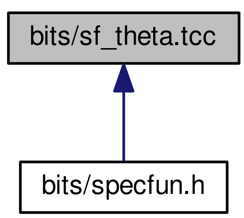
11.27 bits/sf_theta.tcc File Reference

```
#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:



Classes

- struct std::__detail::__jacobi_theta_0_t< _Tp >

Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_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 >
  __gnu_cxx::__jacobi_ellint_t< _Tp > std::__detail::__jacobi_ellint (_Tp __k, _Tp __u)
- template< typename _Tp >
  __jacobi_theta_0_t< _Tp > std::__detail::__jacobi_theta_0 (_Tp __q)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__jacobi_theta_1 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_1 (_Tp __q, const _Tp __x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_1_sum (_Tp __q, _Tp __x)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__jacobi_theta_2 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_2 (_Tp __q, const _Tp __x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_2_prod0 (_Tp __q)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_2_sum (_Tp __q, _Tp __x)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__jacobi_theta_3 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_3 (_Tp __q, const _Tp __x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_3_prod0 (_Tp __q)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_3_sum (_Tp __q, _Tp __x)
- template< typename _Tp >
  std::complex< _Tp > std::__detail::__jacobi_theta_4 (const std::complex< _Tp > &__q, const std::complex< _Tp > &__x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_4 (_Tp __q, const _Tp __x)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_4_prod0 (_Tp __q)
- template< typename _Tp >
  _Tp std::__detail::__jacobi_theta_4_sum (_Tp __q, _Tp __x)
- 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)

11.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.27.2 Macro Definition Documentation

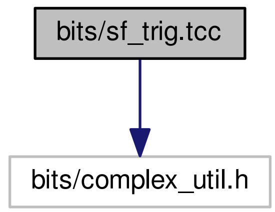
11.27.2.1 _GLIBCXX_BITS_SF_THETA_TCC

```
#define _GLIBCXX_BITS_SF_THETA_TCC 1
```

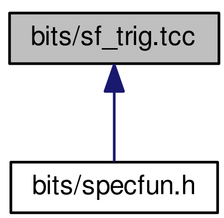
Definition at line 31 of file sf_theta.tcc.

11.28 bits/sf_trig.tcc File Reference

`#include <bits/complex_util.h>`
Include dependency graph for sf_trig.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- std
- std::__detail

Macros

- #define _GLIBCXX_BITS_SF_TRIG_TCC 1

Functions

- template<typename _Tp >
  _Tp std::__detail::__cos_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cos_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__cosh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__cosh_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polar_pi (_Tp __rho, _Tp __phi_pi)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__polar_pi (_Tp __rho, const std::complex< _Tp > &__phi_pi)
- template<typename _Tp >
  _Tp std::__detail::__sin_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sin_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos (_Tp __x)
- template<>
  __gnu_cxx::__sincos_t< float > std::__detail::__sincos (float __x)
- template<>
  __gnu_cxx::__sincos_t< double > std::__detail::__sincos (double __x)
- template<>
  __gnu_cxx::__sincos_t< long double > std::__detail::__sincos (long double __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos_pi (_Tp __x)
- template<typename _Tp >
  _Tp std::__detail::__sinh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__sinh_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__tan_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__tan_pi (std::complex< _Tp > __z)
- template<typename _Tp >
  _Tp std::__detail::__tanh_pi (_Tp __x)
- template<typename _Tp >
  std::complex< _Tp > std::__detail::__tanh_pi (std::complex< _Tp > __z)

11.28.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.28.2 Macro Definition Documentation

11.28.2.1 _GLIBCXX_BITS_SF_TRIG_TCC

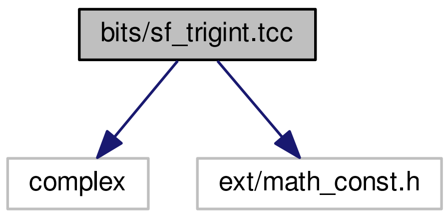
```
#define _GLIBCXX_BITS_SF_TRIG_TCC 1
```

Definition at line 31 of file sf_trig.tcc.

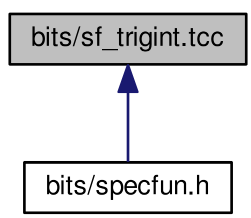
11.29 bits/sf_trigint.tcc File Reference

```
#include <complex>
#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_BITS_SF_TRIGINT_TCC 1

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.

11.29.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.29.2 Macro Definition Documentation

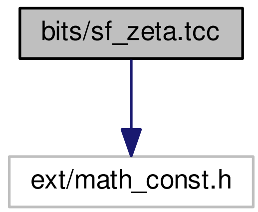
11.29.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC

```
#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1
```

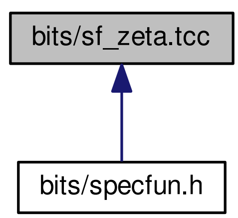
Definition at line 31 of file sf_trigint.tcc.

11.30 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::__debye (unsigned int __n, _Tp __x)
- 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_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)
- 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_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_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 = 121
- constexpr long double std::__detail::_S_zetam1 [_S_num_zetam1]

11.30.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.30.2 Macro Definition Documentation

11.30.2.1 _GLIBCXX_BITS_SF_ZETA_TCC

```
#define _GLIBCXX_BITS_SF_ZETA_TCC 1
```

Definition at line 46 of file sf_zeta.tcc.

11.31 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_state.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_trig.tcc>
#include <bits/sf_bernoulli.tcc>
#include <bits/sf_gamma.tcc>
#include <bits/sf_euler.tcc>
#include <bits/sf_stirling.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_coulomb.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_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 <bits/sf_distributions.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

Functions

- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::airy_ai (std::complex< _Tp > __x)
- float __gnu_cxx::airy_aif (float __x)
- long double __gnu_cxx::airy_ail (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)
- template< typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::airy_bi (std::complex< _Tp > __x)
- float __gnu_cxx::airy_bif (float __x)
- long double __gnu_cxx::airy_bil (long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp > __gnu_cxx::bernoulli (unsigned int __n)
- template< typename _Tp >
  _Tp __gnu_cxx::bernoulli (unsigned int __n, _Tp __x)
- float __gnu_cxx::bernoullif (unsigned int __n)
- long double __gnu_cxx::bernoullil (unsigned int __n)
- template< typename _Tpa , typename _Tpb >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpb > 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_fp_t< _Tp > __gnu_cxx::binomial (unsigned int __n, unsigned int __k)

  Return the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

  $$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial cumulative distribution function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)

  Return the binomial probability mass function.
- float __gnu_cxx::binomialf (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::binomiall (unsigned int __n, unsigned int __k)
- template<typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::bose_einstein (_Tps __s, _Tp __x)
- float __gnu_cxx::bose_einsteinf (float __s, float __x)
- long double __gnu_cxx::bose_einsteinl (long double __s, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __x)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_cl (unsigned int __m, _Tp __x)
- float __gnu_cxx::clausen_clf (unsigned int __m, float __x)
- long double __gnu_cxx::clausen_cll (unsigned int __m, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_sl (unsigned int __m, _Tp __x)
- float __gnu_cxx::clausen_slf (unsigned int __m, float __x)
- long double __gnu_cxx::clausen_sll (unsigned int __m, long double __x)

- float __gnu_cxx::clausenf (unsigned int __m, float __x)
- std::complex< float > __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __z)
- long double __gnu_cxx::clausenl (unsigned int __m, long double __x)
- std::complex< long double > __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __z)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_t< _Tp > 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_fp_t< _Tp, _Tpn > 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_fp_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_fp_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_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)
- template<typename _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __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_fp_t< _Tp > __gnu_cxx::cos_pi (_Tp __x)
- float __gnu_cxx::cos_pif (float __x)
- long double __gnu_cxx::cos_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::cosh_pi (_Tp __x)
- float __gnu_cxx::cosh_pif (float __x)
- long double __gnu_cxx::cosh_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > 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_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)
- template< typename _Tpnu , typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tpnu, _Tp > 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_fp_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_fp_t< _Tp > __gnu_cxx::debye (unsigned int __n, _Tp __x)
- float __gnu_cxx::debyef (unsigned int __n, float __x)
- long double __gnu_cxx::debyel (unsigned int __n, long double __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_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 >
  _Tp __gnu_cxx::dirichlet_lambda (_Tp __s)
- float __gnu_cxx::dirichlet_lambdaf (float __s)
- long double __gnu_cxx::dirichlet_lambdal (long double __s)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::double_factorial (int __n)

  Return the double factorial $n!!$ of the argument as a real number.

  $$n!! = n(n-2)...(2), 0!! = 1$$

  for even n and

  $$n!! = n(n-2)...(1), (-1)!! = 1$$

  for odd n.
- float __gnu_cxx::double_factorialf (int __n)
- long double __gnu_cxx::double_factoriall (int __n)
- template<typename _Tp , typename _Tpp >
  __gnu_cxx::__promote_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpp > 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_fp_t< _Tp, _Tpn, _Tpp > 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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 >
  _Tp __gnu_cxx::euler (unsigned int __n)

  This returns Euler number E_n.
- template< typename _Tp >
  _Tp __gnu_cxx::eulerian_1 (unsigned int __n, unsigned int __m)
- template< typename _Tp >
  _Tp __gnu_cxx::eulerian_2 (unsigned int __n, unsigned int __m)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::expint (_Tp __x)
- template< typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::expint (unsigned int __n, _Tp __x)
- float std::expintf (float __x)
- float __gnu_cxx::expintf (unsigned int __n, float __x)
- long double std::expintl (long double __x)
- long double __gnu_cxx::expintl (unsigned int __n, long double __x)
- template< typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_cdf (_Tlam __lambda, _Tp __x)

Return the exponential cumulative probability density function.

- template<typename _Tlam , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_pdf (_Tlam __lambda, _Tp __x)

  Return the exponential probability density function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::factorial (unsigned int __n)

  Return the factorial $n!$ of the argument as a real number.

$$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float __gnu_cxx::factorialf (unsigned int __n)
- long double __gnu_cxx::factoriall (unsigned int __n)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::falling_factorial (_Tp __a, _Tnu __nu)

  Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n. The falling factorial function is defined by

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

  In particular, $n^{\underline{n}} = n!$.
- float __gnu_cxx::falling_factorialf (float __a, float __nu)
- long double __gnu_cxx::falling_factoriall (long double __a, long double __nu)
- template<typename _Tps , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::fermi_dirac (_Tps __s, _Tp __x)
- float __gnu_cxx::fermi_diracf (float __s, float __x)
- long double __gnu_cxx::fermi_diracl (long double __s, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::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 >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::fisher_f_pdf (_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 >
  __gnu_cxx::__promote_fp_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_fp_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 _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::gamma_cdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma cumulative propability distribution function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::gamma_pdf (_Ta __alpha, _Tb __beta, _Tp __x)

  Return the gamma propability distribution function.
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::gamma_reciprocal (_Ta __a)
- float __gnu_cxx::gamma_reciprocalf (float __a)

- long double __gnu_cxx::gamma_reciprocall (long double __a)
- template<typename _Talpha , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Tp > __gnu_cxx::harmonic (unsigned int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > 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_fp_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_fp_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_fp_t< _Tpa, _Tpb, _Tpc, _Tp > __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_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _↩
  Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Tp > 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_fp_t< _Tp > __gnu_cxx::lbinomial (unsigned int __n, unsigned int __k)

  Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

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

  The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^{n} \binom{n}{k} t^k$$

  .
- float __gnu_cxx::lbinomialf (unsigned int __n, unsigned int __k)
- long double __gnu_cxx::lbinomiall (unsigned int __n, unsigned int __k)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)

  Return the logarithm of the double factorial $ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)...(2), 0!! = 1$$

  for even n and

$$n!! = n(n-2)...(1), (-1)!! = 1$$

  for odd n.
- float __gnu_cxx::ldouble_factorialf (int __n)
- long double __gnu_cxx::ldouble_factoriall (int __n)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > std::legendre (unsigned int __l, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::legendre_q (unsigned int __l, _Tp __x)
- float __gnu_cxx::legendre_qf (unsigned int __l, float __x)
- long double __gnu_cxx::legendre_ql (unsigned int __l, 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_fp_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)

  Return the logarithm of the factorial $ln(n!)$ of the argument as a real number.

$$n! = 1 \times 2 \times ... \times n, 0! = 1$$

  .
- float __gnu_cxx::lfactorialf (unsigned int __n)
- long double __gnu_cxx::lfactoriall (unsigned int __n)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lfalling_factorial (_Tp __a, _Tnu __nu)

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

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

In particular, $n^{\underline{n}} = n!$. *Thus this function returns*

$$ln[a^{\underline{n}}] = ln[\Gamma(a+1)] - ln[\Gamma(a-\nu+1)], ln[a^{\underline{0}}] = 0$$

Many notations exist for this function: $(a)_\nu$,

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

, and others.

- float __gnu_cxx::lfalling_factorialf (float __a, float __nu)
- long double __gnu_cxx::lfalling_factoriall (long double __a, long double __nu)
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::lgamma (_Ta __a)
- template<typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::lgamma (std::complex< _Ta > __a)
- float __gnu_cxx::lgammaf (float __a)
- std::complex< float > __gnu_cxx::lgammaf (std::complex< float > __a)
- long double __gnu_cxx::lgammal (long double __a)
- std::complex< long double > __gnu_cxx::lgammal (std::complex< long double > __a)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::logint (_Tp __x)
- float __gnu_cxx::logintf (float __x)
- long double __gnu_cxx::logintl (long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic cumulative distribution function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the logistic probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal cumulative probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the lognormal probability density function.
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lrising_factorial (_Tp __a, _Tnu __nu)

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

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

  Thus this function returns

  $$ln[a^{\overline{\nu}}] = ln[\Gamma(a+\nu)] - ln[\Gamma(\nu)], ln[a^{\overline{0}}] = 0$$

  Many notations exist for this function: $(a)_\nu$ *(especially in the literature of special functions),*

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

  , and others.

- float __gnu_cxx::lrising_factorialf (float __a, float __nu)
- long double __gnu_cxx::lrising_factoriall (long double __a, long double __nu)
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal cumulative probability density function.
- template<typename _Tmu , typename _Tsig , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)

  Return the normal probability density function.
- template<typename _Tph , typename _Tpa >
  __gnu_cxx::__promote_fp_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 _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::pgamma (_Ta __a, _Tp __x)
- float __gnu_cxx::pgammaf (float __a, float __x)
- long double __gnu_cxx::pgammal (long double __a, long double __x)
- template<typename _Tp , typename _Wp >
  __gnu_cxx::__promote_fp_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)
- template<typename _Tp , typename _Wp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > __gnu_cxx::psi (_Tp __x)
- float __gnu_cxx::psif (float __x)
- long double __gnu_cxx::psil (long double __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::qgamma (_Ta __a, _Tp __x)
- float __gnu_cxx::qgammaf (float __a, float __x)
- long double __gnu_cxx::qgammal (long double __a, long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_t< _Tp > std::riemann_zeta (_Tp __s)
- float std::riemann_zetaf (float __s)
- long double std::riemann_zetal (long double __s)
- template<typename _Tp , typename _Tnu >
  __gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::rising_factorial (_Tp __a, _Tnu __nu)

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

  $$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(\nu)$$

  Many notations exist for this function: $(a)_\nu$*, (especially in the literature of special functions),*

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

  , and others.

- float __gnu_cxx::rising_factorialf (float __a, float __nu)
- long double __gnu_cxx::rising_factoriall (long double __a, long double __nu)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sin_pi (_Tp __x)
- float __gnu_cxx::sin_pif (float __x)
- long double __gnu_cxx::sin_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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)
- __gnu_cxx::__sincos_t< double > __gnu_cxx::sincos (double __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos_pi (_Tp __x)
- __gnu_cxx::__sincos_t< float > __gnu_cxx::sincos_pif (float __x)
- __gnu_cxx::__sincos_t< long double > __gnu_cxx::sincos_pil (long double __x)
- __gnu_cxx::__sincos_t< float > __gnu_cxx::sincosf (float __x)
- __gnu_cxx::__sincos_t< long double > __gnu_cxx::sincosl (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinh_pi (_Tp __x)
- float __gnu_cxx::sinh_pif (float __x)
- long double __gnu_cxx::sinh_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinhc (_Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_t< _Tp > std::sph_bessel (unsigned int __n, _Tp __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)
- template<typename _Tp >
  std::complex< __gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tp > 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_fp_t< _Tp > 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 _Tp >
  _Tp __gnu_cxx::stirling_1 (unsigned int __n, unsigned int __m)
- template<typename _Tp >
  _Tp __gnu_cxx::stirling_2 (unsigned int __n, unsigned int __m)
- template<typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_cdf (_Tt __t, unsigned int __nu)

  Return the Students T probability function.
- template<typename _Tt , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_pdf (_Tt __t, unsigned int __nu)

  Return the complement of the Students T probability function.
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tan_pi (_Tp __x)
- float __gnu_cxx::tan_pif (float __x)
- long double __gnu_cxx::tan_pil (long double __x)
- template<typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tanh_pi (_Tp __x)

- float __gnu_cxx::tanh_pif (float __x)
- long double __gnu_cxx::tanh_pil (long double __x)
- template<typename _Ta >
  __gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::tgamma (_Ta __a)
- template<typename _Ta >
  std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma (std::complex< _Ta > __a)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma (_Ta __a, _Tp __x)
- template<typename _Ta , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower (_Ta __a, _Tp __x)
- float __gnu_cxx::tgamma_lowerf (float __a, float __x)
- long double __gnu_cxx::tgamma_lowerl (long double __a, long double __x)
- float __gnu_cxx::tgammaf (float __a)
- std::complex< float > __gnu_cxx::tgammaf (std::complex< float > __a)
- float __gnu_cxx::tgammaf (float __a, float __x)
- long double __gnu_cxx::tgammal (long double __a)
- std::complex< long double > __gnu_cxx::tgammal (std::complex< long double > __a)
- long double __gnu_cxx::tgammal (long double __a, long double __x)
- template<typename _Tpnu , typename _Tp >
  __gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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_fp_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 _Tpa , typename _Tpc , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __gnu_cxx::tricomi_u (_Tpa __a, _Tpc __c, _Tp __x)

- float __gnu_cxx::tricomi_uf (float __a, float __c, float __x)
- long double __gnu_cxx::tricomi_ul (long double __a, long double __c, long double __x)
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_cdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull cumulative probability density function.
- template<typename _Ta , typename _Tb , typename _Tp >
  __gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_pdf (_Ta __a, _Tb __b, _Tp __x)

  Return the Weibull probability density function.
- template<typename _Trho , typename _Tphi >
  __gnu_cxx::__promote_fp_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)

11.31.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.31.2 Macro Definition Documentation

11.31.2.1 __cpp_lib_math_special_functions

```
#define __cpp_lib_math_special_functions 201603L
```

Definition at line 39 of file specfun.h.

11.31.2.2 __STDCPP_MATH_SPEC_FUNCS__

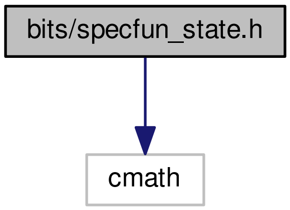
```
#define __STDCPP_MATH_SPEC_FUNCS__ 201003L
```

Definition at line 37 of file specfun.h.

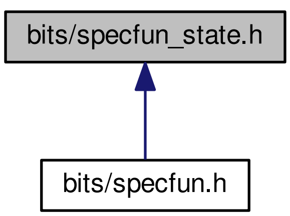
11.32 bits/specfun_state.h File Reference

```
#include <cmath>
```

Include dependency graph for specfun_state.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct __gnu_cxx::__airy_t< _Tx, _Tp >
- struct __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >
- struct __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >
- struct __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >
- struct __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >
- struct __gnu_cxx::__fock_airy_t< _Tx, _Tp >
- struct __gnu_cxx::__gamma_inc_t< _Tp >
- struct __gnu_cxx::__gamma_temme_t< _Tp >

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

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

and

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

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.

- struct __gnu_cxx::__hermite_he_t< _Tp >
- struct __gnu_cxx::__hermite_t< _Tp >
- struct __gnu_cxx::__jacobi_ellint_t< _Tp >
- struct __gnu_cxx::__jacobi_t< _Tp >
- struct __gnu_cxx::__laguerre_t< _Tpa, _Tp >
- struct __gnu_cxx::__legendre_p_t< _Tp >
- struct __gnu_cxx::__lgamma_t< _Tp >
- struct __gnu_cxx::__pqgamma_t< _Tp >
- struct __gnu_cxx::__quadrature_point_t< _Tp >
- struct __gnu_cxx::__sincos_t< _Tp >
- struct __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >
- struct __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >
- struct __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >

Namespaces

- __gnu_cxx

11.32.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include <cmath>.

11.33 ext/math_util.h File Reference

Classes

- struct __gnu_cxx::__fp_is_integer_t

Namespaces

- __gnu_cxx

Functions

- template<typename _Tp >
  bool __gnu_cxx::__fp_is_equal (_Tp __a, _Tp __b, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __gnu_cxx::__fp_is_even_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __gnu_cxx::__fp_is_half_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __gnu_cxx::__fp_is_half_odd_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __gnu_cxx::__fp_is_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  __fp_is_integer_t __gnu_cxx::__fp_is_odd_integer (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  bool __gnu_cxx::__fp_is_zero (_Tp __a, _Tp __mul=_Tp{1})
- template<typename _Tp >
  _Tp __gnu_cxx::__fp_max_abs (_Tp __a, _Tp __b)
- template<typename _Tp , typename _IntTp >
  _Tp __gnu_cxx::__parity (_IntTp __k)

11.33.1 Detailed Description

This file is a GNU extension to the Standard C++ Library.

Index