

# TR29124 C++ Special Math Functions

## 2.0

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Mathematical Special Functions</b>	<b>1</b>
1.1	Introduction and History . . . . .	1
1.2	Contents . . . . .	1
1.3	General Features . . . . .	4
1.3.1	Argument Promotion . . . . .	4
1.3.2	NaN Arguments . . . . .	4
1.4	Implementation . . . . .	5
1.5	Testing . . . . .	5
1.6	General Bibliography . . . . .	5
<b>2</b>	<b>Todo List</b>	<b>7</b>
<b>3</b>	<b>Module Index</b>	<b>9</b>
3.1	Modules . . . . .	9
<b>4</b>	<b>Namespace Index</b>	<b>11</b>
4.1	Namespace List . . . . .	11
<b>5</b>	<b>Class Index</b>	<b>13</b>
5.1	Class List . . . . .	13
<b>6</b>	<b>File Index</b>	<b>15</b>
6.1	File List . . . . .	15

<b>7</b>	<b>Module Documentation</b>	<b>17</b>
7.1	C++ Mathematical Special Functions	17
7.1.1	Detailed Description	17
7.2	C++17/IS29124 Mathematical Special Functions	18
7.2.1	Detailed Description	20
7.2.2	Function Documentation	20
7.2.2.1	assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)	20
7.2.2.2	assoc_laguerref(unsigned int __n, unsigned int __m, float __x)	21
7.2.2.3	assoc_laguerrel(unsigned int __n, unsigned int __m, long double __x)	21
7.2.2.4	assoc_legendre(unsigned int __l, unsigned int __m, _Tp __x)	21
7.2.2.5	assoc_legendref(unsigned int __l, unsigned int __m, float __x)	22
7.2.2.6	assoc_legendrel(unsigned int __l, unsigned int __m, long double __x)	22
7.2.2.7	beta(_Tpa __a, _Tpb __b)	22
7.2.2.8	betaf(float __a, float __b)	23
7.2.2.9	betal(long double __a, long double __b)	23
7.2.2.10	comp_ellint_1(_Tp __k)	23
7.2.2.11	comp_ellint_1f(float __k)	24
7.2.2.12	comp_ellint_1l(long double __k)	24
7.2.2.13	comp_ellint_2(_Tp __k)	24
7.2.2.14	comp_ellint_2f(float __k)	25
7.2.2.15	comp_ellint_2l(long double __k)	25
7.2.2.16	comp_ellint_3(_Tp __k, _Tpn __nu)	25
7.2.2.17	comp_ellint_3f(float __k, float __nu)	26
7.2.2.18	comp_ellint_3l(long double __k, long double __nu)	26
7.2.2.19	cyl_bessel_i(_Tpn __nu, _Tp __x)	26
7.2.2.20	cyl_bessel_if(float __nu, float __x)	27
7.2.2.21	cyl_bessel_il(long double __nu, long double __x)	27
7.2.2.22	cyl_bessel_j(_Tpn __nu, _Tp __x)	27

7.2.2.23	<a href="#">cyl_bessel_jf(float __nu, float __x)</a>	28
7.2.2.24	<a href="#">cyl_bessel_jl(long double __nu, long double __x)</a>	28
7.2.2.25	<a href="#">cyl_bessel_k(_Tpnu __nu, _Tp __x)</a>	28
7.2.2.26	<a href="#">cyl_bessel_kf(float __nu, float __x)</a>	29
7.2.2.27	<a href="#">cyl_bessel_kl(long double __nu, long double __x)</a>	29
7.2.2.28	<a href="#">cyl_neumann(_Tpnu __nu, _Tp __x)</a>	29
7.2.2.29	<a href="#">cyl_neumannf(float __nu, float __x)</a>	30
7.2.2.30	<a href="#">cyl_neumannl(long double __nu, long double __x)</a>	30
7.2.2.31	<a href="#">ellint_1(_Tp __k, _Tpp __phi)</a>	30
7.2.2.32	<a href="#">ellint_1f(float __k, float __phi)</a>	31
7.2.2.33	<a href="#">ellint_1l(long double __k, long double __phi)</a>	31
7.2.2.34	<a href="#">ellint_2(_Tp __k, _Tpp __phi)</a>	31
7.2.2.35	<a href="#">ellint_2f(float __k, float __phi)</a>	32
7.2.2.36	<a href="#">ellint_2l(long double __k, long double __phi)</a>	32
7.2.2.37	<a href="#">ellint_3(_Tp __k, _Tpn __nu, _Tpp __phi)</a>	33
7.2.2.38	<a href="#">ellint_3f(float __k, float __nu, float __phi)</a>	33
7.2.2.39	<a href="#">ellint_3l(long double __k, long double __nu, long double __phi)</a>	34
7.2.2.40	<a href="#">expint(_Tp __x)</a>	34
7.2.2.41	<a href="#">expintf(float __x)</a>	34
7.2.2.42	<a href="#">expintl(long double __x)</a>	34
7.2.2.43	<a href="#">hermite(unsigned int __n, _Tp __x)</a>	35
7.2.2.44	<a href="#">hermitef(unsigned int __n, float __x)</a>	35
7.2.2.45	<a href="#">hermitel(unsigned int __n, long double __x)</a>	35
7.2.2.46	<a href="#">laguerre(unsigned int __n, _Tp __x)</a>	36
7.2.2.47	<a href="#">laguerref(unsigned int __n, float __x)</a>	36
7.2.2.48	<a href="#">laguerrel(unsigned int __n, long double __x)</a>	36
7.2.2.49	<a href="#">legendre(unsigned int __l, _Tp __x)</a>	37
7.2.2.50	<a href="#">legendref(unsigned int __l, float __x)</a>	37

7.2.2.51	<a href="#">legendrel(unsigned int __l, long double __x)</a>	37
7.2.2.52	<a href="#">riemann_zeta(_Tp __s)</a>	38
7.2.2.53	<a href="#">riemann_zetaf(float __s)</a>	38
7.2.2.54	<a href="#">riemann_zetal(long double __s)</a>	38
7.2.2.55	<a href="#">sph_bessel(unsigned int __n, _Tp __x)</a>	39
7.2.2.56	<a href="#">sph_besself(unsigned int __n, float __x)</a>	39
7.2.2.57	<a href="#">sph_bessell(unsigned int __n, long double __x)</a>	39
7.2.2.58	<a href="#">sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</a>	40
7.2.2.59	<a href="#">sph_legendref(unsigned int __l, unsigned int __m, float __theta)</a>	40
7.2.2.60	<a href="#">sph_legendrel(unsigned int __l, unsigned int __m, long double __theta)</a>	40
7.2.2.61	<a href="#">sph_neumann(unsigned int __n, _Tp __x)</a>	41
7.2.2.62	<a href="#">sph_neumannf(unsigned int __n, float __x)</a>	41
7.2.2.63	<a href="#">sph_neumannl(unsigned int __n, long double __x)</a>	41
7.3	<a href="#">GNU Extended Mathematical Special Functions</a>	42
7.3.1	<a href="#">Detailed Description</a>	50
7.3.2	<a href="#">Enumeration Type Documentation</a>	50
7.3.2.1	<a href="#">anonymous enum</a>	50
7.3.3	<a href="#">Function Documentation</a>	50
7.3.3.1	<a href="#">airy_ai(_Tp __x)</a>	50
7.3.3.2	<a href="#">airy_aif(float __x)</a>	51
7.3.3.3	<a href="#">airy_ail(long double __x)</a>	51
7.3.3.4	<a href="#">airy_bi(_Tp __x)</a>	51
7.3.3.5	<a href="#">airy_bif(float __x)</a>	52
7.3.3.6	<a href="#">airy_bil(long double __x)</a>	52
7.3.3.7	<a href="#">bernoulli(unsigned int __n)</a>	52
7.3.3.8	<a href="#">bernoullif(unsigned int __n)</a>	53
7.3.3.9	<a href="#">bernoullil(unsigned int __n)</a>	53
7.3.3.10	<a href="#">bincoef(unsigned int __n, unsigned int __k)</a>	53

7.3.3.11	<code>bincoeff(unsigned int __n, unsigned int __k)</code>	53
7.3.3.12	<code>bincoefl(unsigned int __n, unsigned int __k)</code>	53
7.3.3.13	<code>chebyshev_t(unsigned int __n, _Tp __x)</code>	53
7.3.3.14	<code>chebyshev_tf(unsigned int __n, float __x)</code>	54
7.3.3.15	<code>chebyshev_tl(unsigned int __n, long double __x)</code>	54
7.3.3.16	<code>chebyshev_u(unsigned int __n, _Tp __x)</code>	54
7.3.3.17	<code>chebyshev_uf(unsigned int __n, float __x)</code>	55
7.3.3.18	<code>chebyshev_ul(unsigned int __n, long double __x)</code>	55
7.3.3.19	<code>chebyshev_v(unsigned int __n, _Tp __x)</code>	55
7.3.3.20	<code>chebyshev_vf(unsigned int __n, float __x)</code>	56
7.3.3.21	<code>chebyshev_vl(unsigned int __n, long double __x)</code>	56
7.3.3.22	<code>chebyshev_w(unsigned int __n, _Tp __x)</code>	56
7.3.3.23	<code>chebyshev_wf(unsigned int __n, float __x)</code>	57
7.3.3.24	<code>chebyshev_wl(unsigned int __n, long double __x)</code>	57
7.3.3.25	<code>clausen(unsigned int __m, _Tp __w)</code>	57
7.3.3.26	<code>clausen(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	58
7.3.3.27	<code>clausen_c(unsigned int __m, _Tp __w)</code>	58
7.3.3.28	<code>clausen_cf(unsigned int __m, float __w)</code>	58
7.3.3.29	<code>clausen_cl(unsigned int __m, long double __w)</code>	59
7.3.3.30	<code>clausen_s(unsigned int __m, _Tp __w)</code>	59
7.3.3.31	<code>clausen_sf(unsigned int __m, float __w)</code>	59
7.3.3.32	<code>clausen_sl(unsigned int __m, long double __w)</code>	59
7.3.3.33	<code>clausenf(unsigned int __m, float __w)</code>	60
7.3.3.34	<code>clausenf(unsigned int __m, std::complex&lt; float &gt; __w)</code>	60
7.3.3.35	<code>clausenl(unsigned int __m, long double __w)</code>	60
7.3.3.36	<code>clausenl(unsigned int __m, std::complex&lt; long double &gt; __w)</code>	60
7.3.3.37	<code>comp_ellint_d(_Tk __k)</code>	60
7.3.3.38	<code>comp_ellint_df(float __k)</code>	61

7.3.3.39	<code>comp_ellint_dl(long double __k)</code>	61
7.3.3.40	<code>comp_ellint_rf(float __x, float __y)</code>	61
7.3.3.41	<code>comp_ellint_rf(long double __x, long double __y)</code>	61
7.3.3.42	<code>comp_ellint_rf(_Tx __x, _Ty __y)</code>	61
7.3.3.43	<code>comp_ellint_rg(float __x, float __y)</code>	62
7.3.3.44	<code>comp_ellint_rg(long double __x, long double __y)</code>	62
7.3.3.45	<code>comp_ellint_rg(_Tx __x, _Ty __y)</code>	62
7.3.3.46	<code>conf_hyperg(_Tpa __a, _Tpc __c, _Tp __x)</code>	63
7.3.3.47	<code>conf_hyperg_lim(_Tpc __c, _Tp __x)</code>	63
7.3.3.48	<code>conf_hyperg_limf(float __c, float __x)</code>	64
7.3.3.49	<code>conf_hyperg_liml(long double __c, long double __x)</code>	64
7.3.3.50	<code>conf_hypergf(float __a, float __c, float __x)</code>	64
7.3.3.51	<code>conf_hypergl(long double __a, long double __c, long double __x)</code>	64
7.3.3.52	<code>coshint(_Tp __x)</code>	64
7.3.3.53	<code>coshintf(float __x)</code>	65
7.3.3.54	<code>coshintl(long double __x)</code>	65
7.3.3.55	<code>cosint(_Tp __x)</code>	65
7.3.3.56	<code>cosintf(float __x)</code>	66
7.3.3.57	<code>cosintl(long double __x)</code>	66
7.3.3.58	<code>cyl_hankel_1(_Tpnu __nu, _Tp __z)</code>	66
7.3.3.59	<code>cyl_hankel_1(std::complex&lt; _Tpnu &gt; __nu, std::complex&lt; _Tp &gt; __x)</code>	67
7.3.3.60	<code>cyl_hankel_1f(float __nu, float __z)</code>	67
7.3.3.61	<code>cyl_hankel_1f(std::complex&lt; float &gt; __nu, std::complex&lt; float &gt; __x)</code>	68
7.3.3.62	<code>cyl_hankel_1l(long double __nu, long double __z)</code>	68
7.3.3.63	<code>cyl_hankel_1l(std::complex&lt; long double &gt; __nu, std::complex&lt; long double &gt; __x)</code>	68
7.3.3.64	<code>cyl_hankel_2(_Tpnu __nu, _Tp __z)</code>	68
7.3.3.65	<code>cyl_hankel_2(std::complex&lt; _Tpnu &gt; __nu, std::complex&lt; _Tp &gt; __x)</code>	69
7.3.3.66	<code>cyl_hankel_2f(float __nu, float __z)</code>	69



7.3.3.67	<code>cyl_hankel_2f(std::complex&lt; float &gt; __nu, std::complex&lt; float &gt; __x)</code>	70
7.3.3.68	<code>cyl_hankel_2l(long double __nu, long double __z)</code>	70
7.3.3.69	<code>cyl_hankel_2l(std::complex&lt; long double &gt; __nu, std::complex&lt; long double &gt; __x)</code>	70
7.3.3.70	<code>dawson(_Tp __x)</code>	70
7.3.3.71	<code>dawsonf(float __x)</code>	71
7.3.3.72	<code>dawsonl(long double __x)</code>	71
7.3.3.73	<code>digamma(_Tp __z)</code>	71
7.3.3.74	<code>digammaf(float __z)</code>	71
7.3.3.75	<code>digammal(long double __z)</code>	71
7.3.3.76	<code>dilog(_Tp __x)</code>	71
7.3.3.77	<code>dilogf(float __x)</code>	72
7.3.3.78	<code>dilogl(long double __x)</code>	72
7.3.3.79	<code>dirichlet_beta(_Tp __s)</code>	72
7.3.3.80	<code>dirichlet_betaf(float __s)</code>	73
7.3.3.81	<code>dirichlet_betall(long double __s)</code>	73
7.3.3.82	<code>dirichlet_eta(_Tp __s)</code>	73
7.3.3.83	<code>dirichlet_etaf(float __s)</code>	74
7.3.3.84	<code>dirichlet_etall(long double __s)</code>	74
7.3.3.85	<code>double_factorial(int __n)</code>	74
7.3.3.86	<code>double_factorialf(int __n)</code>	74
7.3.3.87	<code>double_factoriall(int __n)</code>	74
7.3.3.88	<code>ellint_cel(_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)</code>	74
7.3.3.89	<code>ellint_celf(float __k_c, float __p, float __a, float __b)</code>	75
7.3.3.90	<code>ellint_cell(long double __k_c, long double __p, long double __a, long double __b)</code>	75
7.3.3.91	<code>ellint_d(_Tk __k, _Tphi __phi)</code>	75
7.3.3.92	<code>ellint_df(float __k, float __phi)</code>	76
7.3.3.93	<code>ellint_dl(long double __k, long double __phi)</code>	76
7.3.3.94	<code>ellint_el1(_Tp __x, _Tk __k_c)</code>	76

7.3.3.95	<code>ellint_el1f(float __x, float __k_c)</code>	77
7.3.3.96	<code>ellint_el1l(long double __x, long double __k_c)</code>	77
7.3.3.97	<code>ellint_el2(_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)</code>	77
7.3.3.98	<code>ellint_el2f(float __x, float __k_c, float __a, float __b)</code>	78
7.3.3.99	<code>ellint_el2l(long double __x, long double __k_c, long double __a, long double __b)</code>	78
7.3.3.100	<code>ellint_el3(_Tx __x, _Tk __k_c, _Tp __p)</code>	78
7.3.3.101	<code>ellint_el3f(float __x, float __k_c, float __p)</code>	79
7.3.3.102	<code>ellint_el3l(long double __x, long double __k_c, long double __p)</code>	79
7.3.3.103	<code>ellint_rc(_Tp __x, _Up __y)</code>	79
7.3.3.104	<code>ellint_rcf(float __x, float __y)</code>	80
7.3.3.105	<code>ellint_rcl(long double __x, long double __y)</code>	80
7.3.3.106	<code>ellint_rd(_Tp __x, _Up __y, _Vp __z)</code>	80
7.3.3.107	<code>ellint_rdf(float __x, float __y, float __z)</code>	81
7.3.3.108	<code>ellint_rdl(long double __x, long double __y, long double __z)</code>	81
7.3.3.109	<code>ellint_rf(_Tp __x, _Up __y, _Vp __z)</code>	81
7.3.3.110	<code>ellint_rff(float __x, float __y, float __z)</code>	82
7.3.3.111	<code>ellint_rfl(long double __x, long double __y, long double __z)</code>	82
7.3.3.112	<code>ellint_rg(_Tp __x, _Up __y, _Vp __z)</code>	82
7.3.3.113	<code>ellint_rgf(float __x, float __y, float __z)</code>	83
7.3.3.114	<code>ellint_rgl(long double __x, long double __y, long double __z)</code>	83
7.3.3.115	<code>ellint_rj(_Tp __x, _Up __y, _Vp __z, _Wp __p)</code>	83
7.3.3.116	<code>ellint_rjf(float __x, float __y, float __z, float __p)</code>	84
7.3.3.117	<code>ellint_rjl(long double __x, long double __y, long double __z, long double __p)</code>	84
7.3.3.118	<code>ellnome(_Tp __k)</code>	84
7.3.3.119	<code>ellnomef(float __k)</code>	84
7.3.3.120	<code>ellnomel(long double __k)</code>	85
7.3.3.121	<code>expint(unsigned int __n, _Tp __x)</code>	85
7.3.3.122	<code>expintf(unsigned int __n, float __x)</code>	85

7.3.3.123 expintl(unsigned int __n, long double __x) . . . . .	86
7.3.3.124 factorial(unsigned int __n) . . . . .	86
7.3.3.125 factorialf(unsigned int __n) . . . . .	86
7.3.3.126 factoriall(unsigned int __n) . . . . .	86
7.3.3.127 fresnel_c(_Tp __x) . . . . .	86
7.3.3.128 fresnel_cf(float __x) . . . . .	86
7.3.3.129 fresnel_cl(long double __x) . . . . .	87
7.3.3.130 fresnel_s(_Tp __x) . . . . .	87
7.3.3.131 fresnel_sf(float __x) . . . . .	87
7.3.3.132 fresnel_sl(long double __x) . . . . .	87
7.3.3.133 gamma_l(_Tn __n, _Tp __x) . . . . .	87
7.3.3.134 gamma_lf(float __n, float __x) . . . . .	87
7.3.3.135 gamma_ll(long double __n, long double __x) . . . . .	87
7.3.3.136 gamma_u(_Tn __n, _Tp __x) . . . . .	88
7.3.3.137 gamma_uf(float __n, float __x) . . . . .	88
7.3.3.138 gamma_ul(long double __n, long double __x) . . . . .	88
7.3.3.139 gegenbauer(unsigned int __n, _Talpha __alpha, _Tp __x) . . . . .	88
7.3.3.140 gegenbauerf(unsigned int __n, float __alpha, float __x) . . . . .	88
7.3.3.141 gegenbauerl(unsigned int __n, long double __alpha, long double __x) . . . . .	89
7.3.3.142 heuman_lambda(_Tk __k, _Tphi __phi) . . . . .	89
7.3.3.143 heuman_lambdaf(float __k, float __phi) . . . . .	89
7.3.3.144 heuman_lambdal(long double __k, long double __phi) . . . . .	90
7.3.3.145 hurwitz_zeta(_Tp __s, _Up __a) . . . . .	90
7.3.3.146 hurwitz_zeta(_Tp __s, std::complex< _Up > __a) . . . . .	90
7.3.3.147 hurwitz_zetaf(float __s, float __a) . . . . .	90
7.3.3.148 hurwitz_zetal(long double __s, long double __a) . . . . .	91
7.3.3.149 hyperg(_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x) . . . . .	91
7.3.3.150 hypergf(float __a, float __b, float __c, float __x) . . . . .	91

7.3.3.151 hypergl(long double __a, long double __b, long double __c, long double __x) . . . . .	92
7.3.3.152 ibeta(_Ta __a, _Tb __b, _Tp __x) . . . . .	92
7.3.3.153 ibetac(_Ta __a, _Tb __b, _Tp __x) . . . . .	92
7.3.3.154 ibetacf(float __a, float __b, float __x) . . . . .	93
7.3.3.155 ibetacl(long double __a, long double __b, long double __x) . . . . .	93
7.3.3.156 ibetaf(float __a, float __b, float __x) . . . . .	93
7.3.3.157 ibetal(long double __a, long double __b, long double __x) . . . . .	93
7.3.3.158 jacobi(unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x) . . . . .	93
7.3.3.159 jacobi_cn(_Kp __k, _Up __u) . . . . .	94
7.3.3.160 jacobi_cnf(float __k, float __u) . . . . .	94
7.3.3.161 jacobi_cnl(long double __k, long double __u) . . . . .	95
7.3.3.162 jacobi_dn(_Kp __k, _Up __u) . . . . .	95
7.3.3.163 jacobi_dnf(float __k, float __u) . . . . .	95
7.3.3.164 jacobi_dnl(long double __k, long double __u) . . . . .	96
7.3.3.165 jacobi_sn(_Kp __k, _Up __u) . . . . .	96
7.3.3.166 jacobi_snf(float __k, float __u) . . . . .	96
7.3.3.167 jacobi_snl(long double __k, long double __u) . . . . .	97
7.3.3.168 jacobi_zeta(_Tk __k, _Tphi __phi) . . . . .	97
7.3.3.169 jacobi_zetaf(float __k, float __phi) . . . . .	97
7.3.3.170 jacobi_zetal(long double __k, long double __phi) . . . . .	98
7.3.3.171 jacobif(unsigned __n, float __alpha, float __beta, float __x) . . . . .	98
7.3.3.172 jacobil(unsigned __n, long double __alpha, long double __beta, long double __x) . . . . .	98
7.3.3.173 lbincoef(unsigned int __n, unsigned int __k) . . . . .	98
7.3.3.174 lbincoeff(unsigned int __n, unsigned int __k) . . . . .	98
7.3.3.175 lbincoefl(unsigned int __n, unsigned int __k) . . . . .	98
7.3.3.176 ldouble_factorial(int __n) . . . . .	99
7.3.3.177 ldouble_factorialf(int __n) . . . . .	99
7.3.3.178 ldouble_factoriall(int __n) . . . . .	99

7.3.3.179	legendre_q(unsigned int __n, _Tp __x)	99
7.3.3.180	legendre_qf(unsigned int __n, float __x)	99
7.3.3.181	legendre_ql(unsigned int __n, long double __x)	99
7.3.3.182	lfactorial(unsigned int __n)	99
7.3.3.183	lfactorialf(unsigned int __n)	100
7.3.3.184	lfactoriall(unsigned int __n)	100
7.3.3.185	logint(_Tp __x)	100
7.3.3.186	logintf(float __x)	100
7.3.3.187	logintl(long double __x)	100
7.3.3.188	lpochhammer_l(_Tp __a, _Tn __n)	101
7.3.3.189	lpochhammer_lf(float __a, float __n)	101
7.3.3.190	lpochhammer_ll(long double __a, long double __n)	101
7.3.3.191	lpochhammer_u(_Tp __a, _Tn __n)	101
7.3.3.192	lpochhammer_uf(float __a, float __n)	101
7.3.3.193	lpochhammer_ul(long double __a, long double __n)	101
7.3.3.194	owens_t(_Tph __h, _Tpa __a)	101
7.3.3.195	owens_tf(float __h, float __a)	102
7.3.3.196	owens_tl(long double __h, long double __a)	102
7.3.3.197	pgamma(_Ta __a, _Tp __x)	102
7.3.3.198	pgammaf(float __a, float __x)	102
7.3.3.199	pgammal(long double __a, long double __x)	102
7.3.3.200	pochhammer_l(_Tp __a, _Tn __n)	102
7.3.3.201	pochhammer_lf(float __a, float __n)	103
7.3.3.202	pochhammer_ll(long double __a, long double __n)	103
7.3.3.203	pochhammer_u(_Tp __a, _Tn __n)	103
7.3.3.204	pochhammer_uf(float __a, float __n)	103
7.3.3.205	pochhammer_ul(long double __a, long double __n)	103
7.3.3.206	polylog(_Tp __s, _Wp __w)	103

7.3.3.207 polylog(_Tp __s, std::complex< _Tp > __w)	103
7.3.3.208 polylogf(float __s, float __w)	104
7.3.3.209 polylogf(float __s, std::complex< float > __w)	104
7.3.3.210 polylogl(long double __s, long double __w)	104
7.3.3.211 polylogl(long double __s, std::complex< long double > __w)	105
7.3.3.212 psi(_Tp __x)	105
7.3.3.213 psif(float __x)	105
7.3.3.214 psil(long double __x)	105
7.3.3.215 qgamma(_Ta __a, _Tp __x)	105
7.3.3.216 qgammaf(float __a, float __x)	105
7.3.3.217 qgamma_l(long double __a, long double __x)	106
7.3.3.218 radpoly(unsigned int __n, unsigned int __m, _Tp __rho)	106
7.3.3.219 radpolyf(unsigned int __n, unsigned int __m, float __rho)	106
7.3.3.220 radpolyl(unsigned int __n, unsigned int __m, long double __rho)	107
7.3.3.221 sinc(_Tp __x)	107
7.3.3.222 sinc_pi(_Tp __x)	107
7.3.3.223 sinc_pif(float __x)	107
7.3.3.224 sinc_pil(long double __x)	107
7.3.3.225 sincf(float __x)	107
7.3.3.226 sincl(long double __x)	107
7.3.3.227 sinhc(_Tp __x)	107
7.3.3.228 sinhc_pi(_Tp __x)	108
7.3.3.229 sinhc_pif(float __x)	108
7.3.3.230 sinhc_pil(long double __x)	108
7.3.3.231 sinhcf(float __x)	108
7.3.3.232 sinhcl(long double __x)	108
7.3.3.233 sinhint(_Tp __x)	108
7.3.3.234 sinhintf(float __x)	109

7.3.3.235 <code>sinhintl(long double __x)</code> . . . . .	109
7.3.3.236 <code>sinint(_Tp __x)</code> . . . . .	109
7.3.3.237 <code>sinintf(float __x)</code> . . . . .	109
7.3.3.238 <code>sinintl(long double __x)</code> . . . . .	110
7.3.3.239 <code>sph_bessel_i(unsigned int __n, _Tp __x)</code> . . . . .	110
7.3.3.240 <code>sph_bessel_if(unsigned int __n, float __x)</code> . . . . .	110
7.3.3.241 <code>sph_bessel_il(unsigned int __n, long double __x)</code> . . . . .	111
7.3.3.242 <code>sph_bessel_k(unsigned int __n, _Tp __x)</code> . . . . .	111
7.3.3.243 <code>sph_bessel_kf(unsigned int __n, float __x)</code> . . . . .	111
7.3.3.244 <code>sph_bessel_kl(unsigned int __n, long double __x)</code> . . . . .	112
7.3.3.245 <code>sph_hankel_1(unsigned int __n, _Tp __z)</code> . . . . .	112
7.3.3.246 <code>sph_hankel_1(unsigned int __n, std::complex&lt; _Tp &gt; __x)</code> . . . . .	112
7.3.3.247 <code>sph_hankel_1f(unsigned int __n, float __z)</code> . . . . .	113
7.3.3.248 <code>sph_hankel_1f(unsigned int __n, std::complex&lt; float &gt; __x)</code> . . . . .	113
7.3.3.249 <code>sph_hankel_1l(unsigned int __n, long double __z)</code> . . . . .	113
7.3.3.250 <code>sph_hankel_1l(unsigned int __n, std::complex&lt; long double &gt; __x)</code> . . . . .	114
7.3.3.251 <code>sph_hankel_2(unsigned int __n, _Tp __z)</code> . . . . .	114
7.3.3.252 <code>sph_hankel_2(unsigned int __n, std::complex&lt; _Tp &gt; __x)</code> . . . . .	114
7.3.3.253 <code>sph_hankel_2f(unsigned int __n, float __z)</code> . . . . .	115
7.3.3.254 <code>sph_hankel_2f(unsigned int __n, std::complex&lt; float &gt; __x)</code> . . . . .	115
7.3.3.255 <code>sph_hankel_2l(unsigned int __n, long double __z)</code> . . . . .	115
7.3.3.256 <code>sph_hankel_2l(unsigned int __n, std::complex&lt; long double &gt; __x)</code> . . . . .	116
7.3.3.257 <code>sph_harmonic(unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)</code> . . . . .	116
7.3.3.258 <code>sph_harmonicf(unsigned int __l, int __m, float __theta, float __phi)</code> . . . . .	116
7.3.3.259 <code>sph_harmonicl(unsigned int __l, int __m, long double __theta, long double __phi)</code> . . . . .	117
7.3.3.260 <code>theta_1(_Tpnu __nu, _Tp __x)</code> . . . . .	117
7.3.3.261 <code>theta_1f(float __nu, float __x)</code> . . . . .	117
7.3.3.262 <code>theta_1l(long double __nu, long double __x)</code> . . . . .	117

7.3.3.263 theta_2(_Tpnu __nu, _Tp __x) . . . . .	118
7.3.3.264 theta_2f(float __nu, float __x) . . . . .	118
7.3.3.265 theta_2l(long double __nu, long double __x) . . . . .	118
7.3.3.266 theta_3(_Tpnu __nu, _Tp __x) . . . . .	118
7.3.3.267 theta_3f(float __nu, float __x) . . . . .	119
7.3.3.268 theta_3l(long double __nu, long double __x) . . . . .	119
7.3.3.269 theta_4(_Tpnu __nu, _Tp __x) . . . . .	119
7.3.3.270 theta_4f(float __nu, float __x) . . . . .	120
7.3.3.271 theta_4l(long double __nu, long double __x) . . . . .	120
7.3.3.272 theta_c(_Tp k, _Tp __x) . . . . .	120
7.3.3.273 theta_cf(float __k, float __x) . . . . .	120
7.3.3.274 theta_cl(long double __k, long double __x) . . . . .	121
7.3.3.275 theta_d(_Tp k, _Tp __x) . . . . .	121
7.3.3.276 theta_df(float __k, float __x) . . . . .	121
7.3.3.277 theta_dl(long double __k, long double __x) . . . . .	122
7.3.3.278 theta_n(_Tp k, _Tp __x) . . . . .	122
7.3.3.279 theta_nf(float __k, float __x) . . . . .	122
7.3.3.280 theta_nl(long double __k, long double __x) . . . . .	122
7.3.3.281 theta_s(_Tp k, _Tp __x) . . . . .	123
7.3.3.282 theta_sf(float __k, float __x) . . . . .	123
7.3.3.283 theta_sl(long double __k, long double __x) . . . . .	123
7.3.3.284 zernike(unsigned int __n, int __m, _Trho __rho, _Tphi __phi) . . . . .	123
7.3.3.285 zernikef(unsigned int __n, int __m, float __rho, float __phi) . . . . .	124
7.3.3.286 zernikel(unsigned int __n, int __m, long double __rho, long double __phi) . . . . .	124



<b>8 Namespace Documentation</b>	<b>125</b>
8.1 <code>__gnu_cxx</code> Namespace Reference	125
8.2 <code>std</code> Namespace Reference	133
8.3 <code>std::__detail</code> Namespace Reference	135
8.3.1 Enumeration Type Documentation	152
8.3.1.1 anonymous enum	152
8.3.2 Function Documentation	153
8.3.2.1 <code>__airy(_Tp __z, _Tp &amp;_Ai, _Tp &amp;_Bi, _Tp &amp;_Aip, _Tp &amp;_Bip)</code>	153
8.3.2.2 <code>__airy(const std::complex&lt; _Tp &gt; &amp;__z, _Tp __eps, std::complex&lt; _Tp &gt; &amp;_Ai, std::complex&lt; _Tp &gt; &amp;_Aip, std::complex&lt; _Tp &gt; &amp;_Bi, std::complex&lt; _Tp &gt; &amp;_Bip)</code>	153
8.3.2.3 <code>__airy_ai(std::complex&lt; _Tp &gt; __z)</code>	155
8.3.2.4 <code>__airy_arg(std::complex&lt; _Tp &gt; __num2d3, std::complex&lt; _Tp &gt; __zeta, std::complex&lt; _Tp &gt; &amp;__argp, std::complex&lt; _Tp &gt; &amp;__argm)</code>	155
8.3.2.5 <code>__airy_asymp_absarg_ge_pio3(std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; &amp;__Ai, std::complex&lt; _Tp &gt; &amp;_Aip, int __sign=1)</code>	156
8.3.2.6 <code>__airy_asymp_absarg_lt_pio3(std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; &amp;__Ai, std::complex&lt; _Tp &gt; &amp;_Aip)</code>	156
8.3.2.7 <code>__airy_bessel_i(const std::complex&lt; _Tp &gt; &amp;__z, _Tp __eps, std::complex&lt; _Tp &gt; &amp;_lp1d3, std::complex&lt; _Tp &gt; &amp;_lm1d3, std::complex&lt; _Tp &gt; &amp;_lp2d3, std::complex&lt; _Tp &gt; &amp;_lm2d3)</code>	157
8.3.2.8 <code>__airy_bessel_k(const std::complex&lt; _Tp &gt; &amp;__z, _Tp __eps, std::complex&lt; _Tp &gt; &amp;_Kp1d3, std::complex&lt; _Tp &gt; &amp;_Kp2d3)</code>	158
8.3.2.9 <code>__airy_bi(std::complex&lt; _Tp &gt; __z)</code>	158
8.3.2.10 <code>__airy_hyperg_rational(const std::complex&lt; _Tp &gt; &amp;__z, std::complex&lt; _Tp &gt; &amp;__Ai, std::complex&lt; _Tp &gt; &amp;_Aip, std::complex&lt; _Tp &gt; &amp;_Bi, std::complex&lt; _Tp &gt; &amp;_Bip)</code>	159
8.3.2.11 <code>__assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)</code>	159
8.3.2.12 <code>__assoc_legendre_p(unsigned int __l, unsigned int __m, _Tp __x)</code>	160
8.3.2.13 <code>__bernoulli(int __n)</code>	160
8.3.2.14 <code>__bernoulli_2n(int __n)</code>	161
8.3.2.15 <code>__bernoulli_series(unsigned int __n)</code>	161
8.3.2.16 <code>__beta(_Tp __a, _Tp __b)</code>	162

8.3.2.17	<code>__beta_gamma(_Tp __a, _Tp __b)</code>	162
8.3.2.18	<code>__beta_inc(_Tp __a, _Tp __b, _Tp __x)</code>	163
8.3.2.19	<code>__beta_inc_cont_frac(_Tp __a, _Tp __b, _Tp __x)</code>	163
8.3.2.20	<code>__beta_lgamma(_Tp __a, _Tp __b)</code>	164
8.3.2.21	<code>__beta_product(_Tp __a, _Tp __b)</code>	164
8.3.2.22	<code>__bincoef(unsigned int __n, unsigned int __k)</code>	165
8.3.2.23	<code>__binomial_cdf(_Tp __p, unsigned int __n, unsigned int __k)</code>	165
8.3.2.24	<code>__binomial_cdfc(_Tp __p, unsigned int __n, unsigned int __k)</code>	166
8.3.2.25	<code>__bose_einstein(_Tp __s, _Tp __x)</code>	166
8.3.2.26	<code>__chebyshev_recur(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)</code>	167
8.3.2.27	<code>__chebyshev_t(unsigned int __n, _Tp __x)</code>	167
8.3.2.28	<code>__chebyshev_u(unsigned int __n, _Tp __x)</code>	168
8.3.2.29	<code>__chebyshev_v(unsigned int __n, _Tp __x)</code>	168
8.3.2.30	<code>__chebyshev_w(unsigned int __n, _Tp __x)</code>	169
8.3.2.31	<code>__chshint(_Tp __x, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	169
8.3.2.32	<code>__chshint_cont_frac(_Tp __t, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	170
8.3.2.33	<code>__chshint_series(_Tp __t, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	170
8.3.2.34	<code>__clamp_0_m2pi(std::complex&lt; _Tp &gt; __w)</code>	170
8.3.2.35	<code>__clamp_pi(std::complex&lt; _Tp &gt; __w)</code>	170
8.3.2.36	<code>__clausen(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	170
8.3.2.37	<code>__clausen(unsigned int __m, _Tp __w)</code>	171
8.3.2.38	<code>__clausen_c(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	171
8.3.2.39	<code>__clausen_c(unsigned int __m, _Tp __w)</code>	172
8.3.2.40	<code>__clausen_s(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	172
8.3.2.41	<code>__clausen_s(unsigned int __m, _Tp __w)</code>	173
8.3.2.42	<code>__comp_ellint_1(_Tp __k)</code>	173
8.3.2.43	<code>__comp_ellint_2(_Tp __k)</code>	174
8.3.2.44	<code>__comp_ellint_3(_Tp __k, _Tp __nu)</code>	174

8.3.2.45	<code>__comp_ellint_d(_Tp __k)</code>	175
8.3.2.46	<code>__comp_ellint_rf(_Tp __x, _Tp __y)</code>	175
8.3.2.47	<code>__comp_ellint_rg(_Tp __x, _Tp __y)</code>	175
8.3.2.48	<code>__conf_hyperg(_Tp __a, _Tp __c, _Tp __x)</code>	175
8.3.2.49	<code>__conf_hyperg_lim(_Tp __c, _Tp __x)</code>	176
8.3.2.50	<code>__conf_hyperg_lim_series(_Tp __c, _Tp __x)</code>	176
8.3.2.51	<code>__conf_hyperg_luke(_Tp __a, _Tp __c, _Tp __xin)</code>	177
8.3.2.52	<code>__conf_hyperg_series(_Tp __a, _Tp __c, _Tp __x)</code>	177
8.3.2.53	<code>__coshint(const _Tp __x)</code>	178
8.3.2.54	<code>__cyl_bessel(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	178
8.3.2.55	<code>__cyl_bessel_i(_Tp __nu, _Tp __x)</code>	178
8.3.2.56	<code>__cyl_bessel_ij_series(_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)</code>	179
8.3.2.57	<code>__cyl_bessel_ik(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	180
8.3.2.58	<code>__cyl_bessel_ik_asymp(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	181
8.3.2.59	<code>__cyl_bessel_ik_steel(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	181
8.3.2.60	<code>__cyl_bessel_j(_Tp __nu, _Tp __x)</code>	182
8.3.2.61	<code>__cyl_bessel_jn(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	182
8.3.2.62	<code>__cyl_bessel_jn_asymp(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	183
8.3.2.63	<code>__cyl_bessel_jn_steel(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	183
8.3.2.64	<code>__cyl_bessel_k(_Tp __nu, _Tp __x)</code>	183
8.3.2.65	<code>__cyl_hankel_1(_Tp __nu, _Tp __x)</code>	184
8.3.2.66	<code>__cyl_hankel_1(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	184
8.3.2.67	<code>__cyl_hankel_2(_Tp __nu, _Tp __x)</code>	185
8.3.2.68	<code>__cyl_hankel_2(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	185
8.3.2.69	<code>__cyl_neumann(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	186

8.3.2.70	<code>__cyl_neumann_n(_Tp __nu, _Tp __x)</code>	186
8.3.2.71	<code>__dawson(_Tp __x)</code>	187
8.3.2.72	<code>__dawson_cont_frac(_Tp __x)</code>	187
8.3.2.73	<code>__dawson_series(_Tp __x)</code>	187
8.3.2.74	<code>__debye_region(std::complex&lt; _Tp &gt; __alpha, int &amp;__indexr, char &amp;__aorb)</code>	187
8.3.2.75	<code>__dilog(_Tp __x)</code>	188
8.3.2.76	<code>__dirichlet_beta(std::complex&lt; _Tp &gt; __w)</code>	188
8.3.2.77	<code>__dirichlet_beta(_Tp __w)</code>	188
8.3.2.78	<code>__dirichlet_eta(std::complex&lt; _Tp &gt; __w)</code>	189
8.3.2.79	<code>__dirichlet_eta(_Tp __w)</code>	189
8.3.2.80	<code>__double_factorial(int __n)</code>	190
8.3.2.81	<code>__ellint_1(_Tp __k, _Tp __phi)</code>	190
8.3.2.82	<code>__ellint_2(_Tp __k, _Tp __phi)</code>	191
8.3.2.83	<code>__ellint_3(_Tp __k, _Tp __nu, _Tp __phi)</code>	191
8.3.2.84	<code>__ellint_cel(_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)</code>	192
8.3.2.85	<code>__ellint_d(_Tp __k, _Tp __phi)</code>	192
8.3.2.86	<code>__ellint_el1(_Tp __x, _Tp __k_c)</code>	192
8.3.2.87	<code>__ellint_el2(_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)</code>	192
8.3.2.88	<code>__ellint_el3(_Tp __x, _Tp __k_c, _Tp __p)</code>	192
8.3.2.89	<code>__ellint_rc(_Tp __x, _Tp __y)</code>	192
8.3.2.90	<code>__ellint_rd(_Tp __x, _Tp __y, _Tp __z)</code>	193
8.3.2.91	<code>__ellint_rf(_Tp __x, _Tp __y, _Tp __z)</code>	194
8.3.2.92	<code>__ellint_rg(_Tp __x, _Tp __y, _Tp __z)</code>	194
8.3.2.93	<code>__ellint_rj(_Tp __x, _Tp __y, _Tp __z, _Tp __p)</code>	195
8.3.2.94	<code>__ellnome(_Tp __k)</code>	196
8.3.2.95	<code>__ellnome_k(_Tp __k)</code>	196
8.3.2.96	<code>__ellnome_series(_Tp __k)</code>	196
8.3.2.97	<code>__expint(unsigned int __n, _Tp __x)</code>	196

8.3.2.98	<code>__expint(_Tp __x)</code>	197
8.3.2.99	<code>__expint_asyp(unsigned int __n, _Tp __x)</code>	197
8.3.2.100	<code>__expint_E1(_Tp __x)</code>	198
8.3.2.101	<code>__expint_E1_asyp(_Tp __x)</code>	198
8.3.2.102	<code>__expint_E1_series(_Tp __x)</code>	199
8.3.2.103	<code>__expint_Ei(_Tp __x)</code>	199
8.3.2.104	<code>__expint_Ei_asyp(_Tp __x)</code>	200
8.3.2.105	<code>__expint_Ei_series(_Tp __x)</code>	200
8.3.2.106	<code>__expint_En_cont_frac(unsigned int __n, _Tp __x)</code>	201
8.3.2.107	<code>__expint_En_recursion(unsigned int __n, _Tp __x)</code>	201
8.3.2.108	<code>__expint_En_series(unsigned int __n, _Tp __x)</code>	202
8.3.2.109	<code>__expint_large_n(unsigned int __n, _Tp __x)</code>	202
8.3.2.110	<code>__factorial(unsigned int __n)</code>	203
8.3.2.111	<code>__fermi_dirac(_Tp __s, _Tp __x)</code>	203
8.3.2.112	<code>__fisher_f_cdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	204
8.3.2.113	<code>__fisher_f_cdfc(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	204
8.3.2.114	<code>__fock_airy(_Tp __x, std::complex&lt;_Tp&gt; &amp;__w1, std::complex&lt;_Tp&gt; &amp;__w2, std::complex&lt;_Tp&gt; &amp;__w1p, std::complex&lt;_Tp&gt; &amp;__w2p)</code>	205
8.3.2.115	<code>__fpequal(const _Tp &amp;__a, const _Tp &amp;__b)</code>	205
8.3.2.116	<code>__fpimag(const std::complex&lt;_Tp&gt; &amp;__w)</code>	205
8.3.2.117	<code>__fpimag(const _Tp)</code>	206
8.3.2.118	<code>__fpreal(const std::complex&lt;_Tp&gt; &amp;__w)</code>	206
8.3.2.119	<code>__fpreal(const _Tp)</code>	206
8.3.2.120	<code>__fresnel(const _Tp __x)</code>	207
8.3.2.121	<code>__fresnel_cont_frac(const _Tp __ax, _Tp &amp;__Cf, _Tp &amp;__Sf)</code>	207
8.3.2.122	<code>__fresnel_series(const _Tp __ax, _Tp &amp;__Cf, _Tp &amp;__Sf)</code>	207
8.3.2.123	<code>__gamma(_Tp __x)</code>	207
8.3.2.124	<code>__gamma_cont_frac(_Tp __a, _Tp __x)</code>	208

8.3.2.125	<code>__gamma_l(_Tp __a, _Tp __x)</code>	208
8.3.2.126	<code>__gamma_series(_Tp __a, _Tp __x)</code>	208
8.3.2.127	<code>__gamma_temme(_Tp __mu, _Tp &amp;__gam1, _Tp &amp;__gam2, _Tp &amp;__gampl, _Tp &amp;__gammi)</code>	208
8.3.2.128	<code>__gamma_u(_Tp __a, _Tp __x)</code>	209
8.3.2.129	<code>__gauss(_Tp __x)</code>	209
8.3.2.130	<code>__gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)</code>	209
8.3.2.131	<code>__hankel(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; &amp;_H1, std::complex&lt; _Tp &gt; &amp;_H2, std::complex&lt; _Tp &gt; &amp;_H1p, std::complex&lt; _Tp &gt; &amp;_H2p)</code>	210
8.3.2.132	<code>__hankel_debye(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; __alpha, int __indexr, char &amp;__aorb, int &amp;__morn, std::complex&lt; _Tp &gt; &amp;_H1, std::complex&lt; _Tp &gt; &amp;_H2, std::complex&lt; _Tp &gt; &amp;_H1p, std::complex&lt; _Tp &gt; &amp;_H2p)</code>	210
8.3.2.133	<code>__hankel_params(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __zhat, std::complex&lt; _Tp &gt; &amp;__p, std::complex&lt; _Tp &gt; &amp;__p2, std::complex&lt; _Tp &gt; &amp;__nup2, std::complex&lt; _Tp &gt; &amp;__num2, std::complex&lt; _Tp &gt; &amp;__num1d3, std::complex&lt; _Tp &gt; &amp;__num2d3, std::complex&lt; _Tp &gt; &amp;__num4d3, std::complex&lt; _Tp &gt; &amp;__zeta, std::complex&lt; _Tp &gt; &amp;__zetaphf, std::complex&lt; _Tp &gt; &amp;__zetamhf, std::complex&lt; _Tp &gt; &amp;__zetam3hf, std::complex&lt; _Tp &gt; &amp;__zetat)</code>	211
8.3.2.134	<code>__hankel_uniform(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; &amp;_H1, std::complex&lt; _Tp &gt; &amp;_H2, std::complex&lt; _Tp &gt; &amp;_H1p, std::complex&lt; _Tp &gt; &amp;_H2p)</code>	211
8.3.2.135	<code>__hankel_uniform_olver(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z, std::complex&lt; _Tp &gt; &amp;_H1, std::complex&lt; _Tp &gt; &amp;_H2, std::complex&lt; _Tp &gt; &amp;_H1p, std::complex&lt; _Tp &gt; &amp;_H2p)</code>	211
8.3.2.136	<code>__hankel_uniform_outer(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z, __Tp __eps, std::complex&lt; _Tp &gt; &amp;__zhat, std::complex&lt; _Tp &gt; &amp;__1dnsq, std::complex&lt; _Tp &gt; &amp;__num1d3, std::complex&lt; _Tp &gt; &amp;__num2d3, std::complex&lt; _Tp &gt; &amp;__p, std::complex&lt; _Tp &gt; &amp;__p2, std::complex&lt; _Tp &gt; &amp;__etm3h, std::complex&lt; _Tp &gt; &amp;__etrat, std::complex&lt; _Tp &gt; &amp;__Aip, std::complex&lt; _Tp &gt; &amp;__o4dp, std::complex&lt; _Tp &gt; &amp;__Aim, std::complex&lt; _Tp &gt; &amp;__o4dm, std::complex&lt; _Tp &gt; &amp;__od2p, std::complex&lt; _Tp &gt; &amp;__od0dp, std::complex&lt; _Tp &gt; &amp;__od2m, std::complex&lt; _Tp &gt; &amp;__od0dm)</code>	212
8.3.2.137	<code>__hankel_uniform_sum(std::complex&lt; _Tp &gt; __p, std::complex&lt; _Tp &gt; __p2, std::complex&lt; _Tp &gt; __num2, std::complex&lt; _Tp &gt; __zetam3hf, std::complex&lt; _Tp &gt; __Aip, std::complex&lt; _Tp &gt; __o4dp, std::complex&lt; _Tp &gt; __Aim, std::complex&lt; _Tp &gt; __o4dm, std::complex&lt; _Tp &gt; __od2p, std::complex&lt; _Tp &gt; __od0dp, std::complex&lt; _Tp &gt; __od2m, std::complex&lt; _Tp &gt; __od0dm, __Tp __eps, std::complex&lt; _Tp &gt; &amp;_H1sum, std::complex&lt; _Tp &gt; &amp;_H1psum, std::complex&lt; _Tp &gt; &amp;_H2sum, std::complex&lt; _Tp &gt; &amp;_H2psum)</code>	212
8.3.2.138	<code>__heuman_lambda(_Tp __k, _Tp __phi)</code>	213

8.3.2.139	<code>__hurwitz_zeta(_Tp __s, _Tp __a)</code>	. 213
8.3.2.140	<code>__hurwitz_zeta(_Tp __s, std::complex&lt; _Tp &gt; __a)</code>	. 214
8.3.2.141	<code>__hurwitz_zeta_euler_maclaurin(_Tp __s, _Tp __a)</code>	. 214
8.3.2.142	<code>__hydrogen(const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)</code>	. 215
8.3.2.143	<code>__hyperg(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	. 215
8.3.2.144	<code>__hyperg_luke(_Tp __a, _Tp __b, _Tp __c, _Tp __xin)</code>	. 215
8.3.2.145	<code>__hyperg_reflect(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	. 216
8.3.2.146	<code>__hyperg_series(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	. 216
8.3.2.147	<code>__jacobi_sncndn(_Tp __k, _Tp __u)</code>	. 217
8.3.2.148	<code>__jacobi_zeta(_Tp __k, _Tp __phi)</code>	. 217
8.3.2.149	<code>__laguerre(unsigned int __n, _Tp __x)</code>	. 217
8.3.2.150	<code>__legendre_q(unsigned int __l, _Tp __x)</code>	. 218
8.3.2.151	<code>__log_bincoef(unsigned int __n, unsigned int __k)</code>	. 218
8.3.2.152	<code>__log_double_factorial(_Tp __x)</code>	. 218
8.3.2.153	<code>__log_double_factorial(int __n)</code>	. 219
8.3.2.154	<code>__log_factorial(unsigned int __n)</code>	. 219
8.3.2.155	<code>__log_gamma(_Tp __x)</code>	. 219
8.3.2.156	<code>__log_gamma_bernoulli(_Tp __x)</code>	. 220
8.3.2.157	<code>__log_gamma_lanczos(_Tp __x)</code>	. 220
8.3.2.158	<code>__log_gamma_sign(_Tp __x)</code>	. 220
8.3.2.159	<code>__log_gamma_spouge(_Tp __z)</code>	. 221
8.3.2.160	<code>__log_pochhammer_l(_Tp __a, _Tp __n)</code>	. 221
8.3.2.161	<code>__log_pochhammer_u(_Tp __a, _Tp __n)</code>	. 222
8.3.2.162	<code>__logint(const _Tp __x)</code>	. 222
8.3.2.163	<code>__owens_t(_Tp __h, _Tp __a)</code>	. 223
8.3.2.164	<code>__pgamma(_Tp __a, _Tp __x)</code>	. 223
8.3.2.165	<code>__pochhammer_l(_Tp __a, _Tp __n)</code>	. 224

8.3.2.166	<code>__pochhammer_u(Tp __a, Tp __n)</code>	224
8.3.2.167	<code>__poly_hermite(unsigned int __n, Tp __x)</code>	224
8.3.2.168	<code>__poly_hermite_asymp(unsigned int __n, Tp __x)</code>	225
8.3.2.169	<code>__poly_hermite_recursion(unsigned int __n, Tp __x)</code>	225
8.3.2.170	<code>__poly_jacobi(unsigned int __n, Tp __alpha, Tp __beta, Tp __x)</code>	226
8.3.2.171	<code>__poly_laguerre(unsigned int __n, Tpa __alpha1, Tp __x)</code>	226
8.3.2.172	<code>__poly_laguerre_hyperg(unsigned int __n, Tpa __alpha1, Tp __x)</code>	227
8.3.2.173	<code>__poly_laguerre_large_n(unsigned __n, Tpa __alpha1, Tp __x)</code>	228
8.3.2.174	<code>__poly_laguerre_recursion(unsigned int __n, Tpa __alpha1, Tp __x)</code>	228
8.3.2.175	<code>__poly_legendre_p(unsigned int __l, Tp __x)</code>	229
8.3.2.176	<code>__poly_radial_jacobi(unsigned int __n, unsigned int __m, Tp __rho)</code>	229
8.3.2.177	<code>__polylog(Tp __s, Tp __x)</code>	230
8.3.2.178	<code>__polylog(Tp __s, std::complex&lt;Tp&gt; __w)</code>	231
8.3.2.179	<code>__polylog_exp(Tp __s, ArgType __w)</code>	231
8.3.2.180	<code>__polylog_exp_asymp(Tp __s, std::complex&lt;Tp&gt; __w)</code>	232
8.3.2.181	<code>__polylog_exp_int_neg(int __s, std::complex&lt;Tp&gt; __w)</code>	232
8.3.2.182	<code>__polylog_exp_int_neg(const int __s, Tp __w)</code>	233
8.3.2.183	<code>__polylog_exp_int_pos(unsigned int __s, std::complex&lt;Tp&gt; __w)</code>	233
8.3.2.184	<code>__polylog_exp_int_pos(unsigned int __s, Tp __w)</code>	234
8.3.2.185	<code>__polylog_exp_neg(Tp __s, std::complex&lt;Tp&gt; __w)</code>	234
8.3.2.186	<code>__polylog_exp_neg(int __s, std::complex&lt;Tp&gt; __w)</code>	235
8.3.2.187	<code>__polylog_exp_neg_even(unsigned int __n, std::complex&lt;Tp&gt; __w)</code>	235
8.3.2.188	<code>__polylog_exp_neg_odd(unsigned int __n, std::complex&lt;Tp&gt; __w)</code>	236
8.3.2.189	<code>__polylog_exp_negative_real_part(PowTp __s, Tp __w)</code>	237
8.3.2.190	<code>__polylog_exp_pos(unsigned int __s, std::complex&lt;Tp&gt; __w)</code>	237
8.3.2.191	<code>__polylog_exp_pos(unsigned int __s, Tp __w)</code>	238
8.3.2.192	<code>__polylog_exp_pos(Tp __s, std::complex&lt;Tp&gt; __w)</code>	239
8.3.2.193	<code>__polylog_exp_real_neg(Tp __s, std::complex&lt;Tp&gt; __w)</code>	239



8.3.2.194	<code>__polylog_exp_real_neg(_Tp __s, _Tp __w)</code>	240
8.3.2.195	<code>__polylog_exp_real_pos(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	240
8.3.2.196	<code>__polylog_exp_real_pos(_Tp __s, _Tp __w)</code>	241
8.3.2.197	<code>__psi(_Tp __x)</code>	241
8.3.2.198	<code>__psi(unsigned int __n, _Tp __x)</code>	241
8.3.2.199	<code>__psi_asymp(_Tp __x)</code>	242
8.3.2.200	<code>__psi_series(_Tp __x)</code>	242
8.3.2.201	<code>__qgamma(_Tp __a, _Tp __x)</code>	242
8.3.2.202	<code>__riemann_zeta(_Tp __s)</code>	242
8.3.2.203	<code>__riemann_zeta_alt(_Tp __s)</code>	243
8.3.2.204	<code>__riemann_zeta_euler_maclaurin(_Tp __s)</code>	243
8.3.2.205	<code>__riemann_zeta_glob(_Tp __s)</code>	243
8.3.2.206	<code>__riemann_zeta_m_1(_Tp __s)</code>	244
8.3.2.207	<code>__riemann_zeta_m_1_sum(_Tp __s)</code>	244
8.3.2.208	<code>__riemann_zeta_product(_Tp __s)</code>	244
8.3.2.209	<code>__riemann_zeta_sum(_Tp __s)</code>	245
8.3.2.210	<code>__sinc(_Tp __a, _Tp __x)</code>	245
8.3.2.211	<code>__sinc(_Tp __x)</code>	245
8.3.2.212	<code>__sinc_pi(_Tp __x)</code>	246
8.3.2.213	<code>__sincosint(_Tp __x)</code>	246
8.3.2.214	<code>__sincosint_asymp(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code>	246
8.3.2.215	<code>__sincosint_cont_frac(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code>	246
8.3.2.216	<code>__sincosint_series(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code>	247
8.3.2.217	<code>__sinhc(_Tp __a, _Tp __x)</code>	247
8.3.2.218	<code>__sinhc(_Tp __x)</code>	247
8.3.2.219	<code>__sinhc_pi(_Tp __x)</code>	247
8.3.2.220	<code>__sinhint(const _Tp __x)</code>	247
8.3.2.221	<code>__sph_bessel(unsigned int __n, _Tp __x)</code>	248

8.3.2.222	<code>__sph_bessel(unsigned int __n, std::complex&lt;_Tp&gt; __z)</code>	248
8.3.2.223	<code>__sph_bessel_ik(unsigned int __n, _Tp __x, _Tp &amp;__i_n, _Tp &amp;__k_n, _Tp &amp;__ip_n, _Tp &amp;__kp_n)</code>	249
8.3.2.224	<code>__sph_bessel_jn(unsigned int __n, _Tp __x, _Tp &amp;__j_n, _Tp &amp;__n_n, _Tp &amp;__jp_n, _Tp &amp;__np_n)</code>	249
8.3.2.225	<code>__sph_hankel(unsigned int __n, std::complex&lt;_Tp&gt; __z, std::complex&lt;_Tp&gt; &amp;__H1, std::complex&lt;_Tp&gt; &amp;__H1p, std::complex&lt;_Tp&gt; &amp;__H2, std::complex&lt;_Tp&gt; &amp;__H2p)</code>	250
8.3.2.226	<code>__sph_hankel_1(unsigned int __n, _Tp __x)</code>	250
8.3.2.227	<code>__sph_hankel_1(unsigned int __n, std::complex&lt;_Tp&gt; __z)</code>	251
8.3.2.228	<code>__sph_hankel_2(unsigned int __n, _Tp __x)</code>	251
8.3.2.229	<code>__sph_hankel_2(unsigned int __n, std::complex&lt;_Tp&gt; __z)</code>	252
8.3.2.230	<code>__sph_harmonic(unsigned int __l, int __m, _Tp __theta, _Tp __phi)</code>	252
8.3.2.231	<code>__sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</code>	252
8.3.2.232	<code>__sph_neumann(unsigned int __n, _Tp __x)</code>	253
8.3.2.233	<code>__sph_neumann(unsigned int __n, std::complex&lt;_Tp&gt; __z)</code>	254
8.3.2.234	<code>__student_t_cdf(_Tp __t, unsigned int __nu)</code>	255
8.3.2.235	<code>__student_t_cdfc(_Tp __t, unsigned int __nu)</code>	255
8.3.2.236	<code>__theta_1(_Tp __nu, _Tp __x)</code>	256
8.3.2.237	<code>__theta_2(_Tp __nu, _Tp __x)</code>	256
8.3.2.238	<code>__theta_2_asyp(_Tp __nu, _Tp __x)</code>	257
8.3.2.239	<code>__theta_2_sum(_Tp __nu, _Tp __x)</code>	257
8.3.2.240	<code>__theta_3(_Tp __nu, _Tp __x)</code>	257
8.3.2.241	<code>__theta_3_asyp(_Tp __nu, _Tp __x)</code>	257
8.3.2.242	<code>__theta_3_sum(_Tp __nu, _Tp __x)</code>	258
8.3.2.243	<code>__theta_4(_Tp __nu, _Tp __x)</code>	258
8.3.2.244	<code>__theta_c(_Tp __k, _Tp __x)</code>	258
8.3.2.245	<code>__theta_d(_Tp __k, _Tp __x)</code>	258
8.3.2.246	<code>__theta_n(_Tp __k, _Tp __x)</code>	259
8.3.2.247	<code>__theta_s(_Tp __k, _Tp __x)</code>	259

8.3.2.248	<code>__zernike(unsigned int __n, int __m, _Tp __rho, _Tp __phi)</code>	259
8.3.2.249	<code>__znorm1(_Tp __x)</code>	260
8.3.2.250	<code>__znorm2(_Tp __x)</code>	260
8.3.2.251	<code>evenzeta(unsigned int __k)</code>	260
8.3.3	Variable Documentation	260
8.3.3.1	<code>_Num_Euler_Maclaurin_zeta</code>	260
8.3.3.2	<code>_S_double_factorial_table</code>	261
8.3.3.3	<code>_S_Euler_Maclaurin_zeta</code>	261
8.3.3.4	<code>_S_factorial_table</code>	261
8.3.3.5	<code>_S_neg_double_factorial_table</code>	261
8.3.3.6	<code>_S_num_double_factorials</code>	261
8.3.3.7	<code>_S_num_double_factorials&lt; double &gt;</code>	261
8.3.3.8	<code>_S_num_double_factorials&lt; float &gt;</code>	261
8.3.3.9	<code>_S_num_double_factorials&lt; long double &gt;</code>	261
8.3.3.10	<code>_S_num_factorials</code>	262
8.3.3.11	<code>_S_num_factorials&lt; double &gt;</code>	262
8.3.3.12	<code>_S_num_factorials&lt; float &gt;</code>	262
8.3.3.13	<code>_S_num_factorials&lt; long double &gt;</code>	262
8.3.3.14	<code>_S_num_neg_double_factorials</code>	262
8.3.3.15	<code>_S_num_neg_double_factorials&lt; double &gt;</code>	262
8.3.3.16	<code>_S_num_neg_double_factorials&lt; float &gt;</code>	262
8.3.3.17	<code>_S_num_neg_double_factorials&lt; long double &gt;</code>	262
8.3.3.18	<code>_S_num_zetam1</code>	262
8.3.3.19	<code>_S_zetam1</code>	262

<b>9</b>	<b>Class Documentation</b>	<b>263</b>
9.1	std::__detail::_Factorial_table<_Tp> Struct Template Reference . . . . .	263
9.1.1	Detailed Description . . . . .	263
9.1.2	Member Data Documentation . . . . .	263
9.1.2.1	__factorial . . . . .	263
9.1.2.2	__log_factorial . . . . .	263
9.1.2.3	__n . . . . .	264
<b>10</b>	<b>File Documentation</b>	<b>265</b>
10.1	bits/sf_airy.tcc File Reference . . . . .	265
10.1.1	Detailed Description . . . . .	267
10.1.2	Macro Definition Documentation . . . . .	267
10.1.2.1	_GLIBCXX_BITS_SF_AIRY_TCC . . . . .	267
10.2	bits/sf_bessel.tcc File Reference . . . . .	267
10.2.1	Detailed Description . . . . .	269
10.2.2	Macro Definition Documentation . . . . .	269
10.2.2.1	_GLIBCXX_BITS_SF_BESSEL_TCC . . . . .	269
10.3	bits/sf_beta.tcc File Reference . . . . .	269
10.3.1	Detailed Description . . . . .	271
10.3.2	Macro Definition Documentation . . . . .	271
10.3.2.1	_GLIBCXX_BITS_SF_BETA_TCC . . . . .	271
10.4	bits/sf_cardinal.tcc File Reference . . . . .	271
10.4.1	Macro Definition Documentation . . . . .	273
10.4.1.1	_GLIBCXX_BITS_SF_CARDINAL_TCC . . . . .	273
10.5	bits/sf_chebyshev.tcc File Reference . . . . .	273
10.5.1	Detailed Description . . . . .	274
10.5.2	Macro Definition Documentation . . . . .	275
10.5.2.1	_GLIBCXX_SF_CHEBYSHEV_TCC . . . . .	275

10.6 bits/sf_dawson.tcc File Reference . . . . .	275
10.6.1 Detailed Description . . . . .	276
10.6.2 Macro Definition Documentation . . . . .	276
10.6.2.1 _GLIBCXX_SF_DAWSON_TCC . . . . .	276
10.7 bits/sf_ellint.tcc File Reference . . . . .	276
10.7.1 Detailed Description . . . . .	278
10.7.2 Macro Definition Documentation . . . . .	278
10.7.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC . . . . .	278
10.8 bits/sf_expint.tcc File Reference . . . . .	279
10.8.1 Detailed Description . . . . .	281
10.8.2 Macro Definition Documentation . . . . .	281
10.8.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC . . . . .	281
10.9 bits/sf_fresnel.tcc File Reference . . . . .	281
10.9.1 Detailed Description . . . . .	282
10.9.2 Macro Definition Documentation . . . . .	282
10.9.2.1 _GLIBCXX_SF_FRESNEL_TCC . . . . .	282
10.10bits/sf_gamma.tcc File Reference . . . . .	283
10.10.1 Detailed Description . . . . .	288
10.10.2 Macro Definition Documentation . . . . .	288
10.10.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC . . . . .	288
10.11bits/sf_gegenbauer.tcc File Reference . . . . .	288
10.11.1 Detailed Description . . . . .	289
10.11.2 Macro Definition Documentation . . . . .	289
10.11.2.1 _GLIBCXX_SF_GEGENBAUER_TCC . . . . .	289
10.12bits/sf_hankel.tcc File Reference . . . . .	290
10.12.1 Detailed Description . . . . .	292
10.12.2 Macro Definition Documentation . . . . .	292
10.12.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC . . . . .	292

10.13bits/sf_hankel_new.tcc File Reference . . . . .	293
10.13.1 Macro Definition Documentation . . . . .	293
10.13.1.1 _GLIBCXX_BITS_SF_HANKEL_NEW_TCC . . . . .	293
10.14bits/sf_hermite.tcc File Reference . . . . .	293
10.14.1 Detailed Description . . . . .	294
10.14.2 Macro Definition Documentation . . . . .	294
10.14.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC . . . . .	294
10.15bits/sf_hydrogen.tcc File Reference . . . . .	295
10.15.1 Detailed Description . . . . .	295
10.15.2 Macro Definition Documentation . . . . .	295
10.15.2.1 _GLIBCXX_BITS_SF_HYDROGEN_TCC . . . . .	295
10.16bits/sf_hyperg.tcc File Reference . . . . .	296
10.16.1 Detailed Description . . . . .	297
10.16.2 Macro Definition Documentation . . . . .	297
10.16.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC . . . . .	297
10.17bits/sf_hypint.tcc File Reference . . . . .	298
10.17.1 Detailed Description . . . . .	299
10.17.2 Macro Definition Documentation . . . . .	299
10.17.2.1 _GLIBCXX_SF_HYPINT_TCC . . . . .	299
10.18bits/sf_jacobi.tcc File Reference . . . . .	299
10.18.1 Detailed Description . . . . .	300
10.18.2 Macro Definition Documentation . . . . .	300
10.18.2.1 _GLIBCXX_SF_JACOBI_TCC . . . . .	300
10.19bits/sf_laguerre.tcc File Reference . . . . .	301
10.19.1 Detailed Description . . . . .	302
10.19.2 Macro Definition Documentation . . . . .	302
10.19.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC . . . . .	302
10.20bits/sf_legendre.tcc File Reference . . . . .	303

10.20.1 Detailed Description	304
10.20.2 Macro Definition Documentation	304
10.20.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC	304
10.21bits/sf_mod_bessel.tcc File Reference	304
10.21.1 Detailed Description	306
10.21.2 Macro Definition Documentation	306
10.21.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC	306
10.22bits/sf_owens.t.tcc File Reference	306
10.22.1 Detailed Description	307
10.22.2 Macro Definition Documentation	307
10.22.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC	307
10.23bits/sf_polylog.tcc File Reference	307
10.23.1 Detailed Description	310
10.23.2 Macro Definition Documentation	310
10.23.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC	310
10.24bits/sf_theta.tcc File Reference	310
10.24.1 Detailed Description	312
10.24.2 Macro Definition Documentation	312
10.24.2.1 _GLIBCXX_SF_THETA_TCC	312
10.25bits/sf_trigint.tcc File Reference	312
10.25.1 Detailed Description	314
10.25.2 Macro Definition Documentation	314
10.25.2.1 _GLIBCXX_SF_TRIGINT_TCC	314
10.26bits/sf_zeta.tcc File Reference	314
10.26.1 Detailed Description	316
10.26.2 Macro Definition Documentation	316
10.26.2.1 _GLIBCXX_BITS_SF_ZETA_TCC	316
10.27bits/specfun.h File Reference	317
10.27.1 Detailed Description	327
10.27.2 Macro Definition Documentation	327
10.27.2.1 __cpp_lib_math_special_functions	327
10.27.2.2 __STDCPP_MATH_SPEC_FUNCS__	327





# Chapter 1

## Mathematical Special Functions

### 1.1 Introduction and History

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

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

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

### 1.2 Contents

The following functions are implemented in namespace `std`:

- [assoc\\_laguerre](#) - Associated Laguerre functions
- [assoc\\_legendre](#) - Associated Legendre functions
- [beta](#) - Beta functions
- [comp\\_ellint\\_1](#) - Complete elliptic functions of the first kind
- [comp\\_ellint\\_2](#) - Complete elliptic functions of the second kind
- [comp\\_ellint\\_3](#) - Complete elliptic functions of the third kind
- [cyl\\_bessel\\_i](#) - Regular modified cylindrical Bessel functions
- [cyl\\_bessel\\_j](#) - Cylindrical Bessel functions of the first kind
- [cyl\\_bessel\\_k](#) - Irregular modified cylindrical Bessel functions
- [cyl\\_neumann](#) - Cylindrical Neumann functions or Cylindrical Bessel functions of the second kind
- [ellint\\_1](#) - Incomplete elliptic functions of the first kind

- [ellint\\_2](#) - Incomplete elliptic functions of the second kind
- [ellint\\_3](#) - Incomplete elliptic functions of the third kind
- [expint](#) - The exponential integral
- [hermite](#) - Hermite polynomials
- [laguerre](#) - Laguerre functions
- [legendre](#) - Legendre polynomials
- [riemann\\_zeta](#) - The Riemann zeta function
- [sph\\_bessel](#) - Spherical Bessel functions
- [sph\\_legendre](#) - Spherical Legendre functions
- [sph\\_neumann](#) - Spherical Neumann functions

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

- [conf\\_hyperg](#) - Confluent hypergeometric functions
- [hyperg](#) - Hypergeometric functions

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

- [airy\\_ai](#) - Airy functions of the first kind
- [airy\\_bi](#) - Airy functions of the second kind
- [bincoef](#) - Binomial coefficients
- [chebyshev\\_t](#) - Chebyshev polynomials of the first kind
- [chebyshev\\_u](#) - Chebyshev polynomials of the second kind
- [chebyshev\\_v](#) - Chebyshev polynomials of the third kind
- [chebyshev\\_w](#) - Chebyshev polynomials of the fourth kind
- [clausen](#) - Clausen integrals
- [clausen\\_c](#) - Clausen cosine integrals
- [clausen\\_s](#) - Clausen sine integrals
- [comp\\_ellint\\_d](#) - Incomplete Legendre D elliptic integral
- [conf\\_hyperg\\_lim](#) - Confluent hypergeometric limit functions
- [coshint](#) - Hyperbolic cosine integral
- [cosint](#) - Cosine integral
- [cyl\\_hankel\\_1](#) - Cylindrical Hankel functions of the first kind
- [cyl\\_hankel\\_2](#) - Cylindrical Hankel functions of the second kind

- [dawson](#) - Dawson integrals
- [dilog](#) - Dilogarithm functions
- [dirichlet\\_beta](#) - Dirichlet beta function
- [dirichlet\\_eta](#) - Dirichlet beta function
- [dirichlet\\_lambda](#) - Dirichlet lambda function
- [double\\_factorial](#) -
- [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
- [expint](#) - Exponential integrals
- [factorial](#) - Factorials
- [fresnel\\_c](#) - Fresnel cosine integrals
- [fresnel\\_s](#) - Fresnel sine integrals
- [gamma\\_l](#) - Lower incomplete gamma functions
- [pgamma](#) - Regularized lower incomplete gamma functions
- [qgamma](#) - Regularized upper incomplete gamma functions
- [gamma\\_u](#) - upper incomplete gamma functions
- [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
- [lbincoef](#) - Log binomial coefficients
- [ldouble\\_factorial](#) - Log double factorials
- [legendre\\_q](#) - Legendre functions of the second kind
- [lfactorial](#) - Log factorials

- [lpochhammer\\_l](#) - Log lower Pochhammer functions
- [lpochhammer\\_u](#) - Log upper Pochhammer functions
- [owens\\_t](#) - Owens T functions
- [pochhammer\\_l](#) - Lower Pochhammer functions
- [pochhammer\\_u](#) - Upper Pochhammer functions
- [psi](#) - Psi of digamma function
- [radpoly](#) - Radial polynomials
- [sinhc](#) - Hyperbolic sinus cardinal function
- [sinhc\\_pi](#) -
- [sinc](#) - Sinus cardinal function
- [sinc\\_pi](#) -
- [sinhint](#) - Hyperbolic sine integral
- [sinint](#) - Sine integral
- [sph\\_bessel\\_i](#) - Spherical regular modified Bessel functions
- [sph\\_bessel\\_k](#) - Spherical irregular modified Bessel functions
- [sph\\_hankel\\_1](#) - Spherical Hankel functions of the first kind
- [sph\\_hankel\\_2](#) - Spherical Hankel functions of the second kind
- [sph\\_harmonic](#) - Spherical
- [zernike](#) - Zernike polynomials

## 1.3 General Features

### 1.3.1 Argument Promotion

The arguments supplied 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. Similarly, `long double` should give you more dynamic range and slightly more precision than `double` on many systems.

## 1.5 Testing

These functions have been tested against equivalent implementations from the [Gnu Scientific Library](http://www.gnu.org/software/gsl/), [GSL](http://www.boost.org/doc/libs/1_60_0/libs/math/doc/html/index.html) and [Boost](http://www.boost.org/doc/libs/1_60_0/libs/math/doc/html/index.html) and the ratio

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

is generally found to be within  $10^{-15}$  for 64-bit double on linux-x86\_64 systems over most of the ranges of validity.

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

## 1.6 General Bibliography

### See also

Abramowitz and Stegun: Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables Edited by Milton Abramowitz and Irene A. Stegun, National Bureau of Standards Applied Mathematics Series - 55 Issued June 1964, Tenth Printing, December 1972, with corrections Electronic versions of A&S abound including both pdf and navigable html.

for example <http://people.math.sfu.ca/~cbm/aands/>

The old A&S has been redone as the NIST Digital Library of Mathematical Functions: <http://dlmf.nist.gov/> This version is far more navigable and includes more recent work.

An Atlas of Functions: with Equator, the Atlas Function Calculator 2nd Edition, by Oldham, Keith B., Myland, Jan, Spanier, Jerome

Asymptotics and Special Functions by Frank W. J. Olver, Academic Press, 1974

Numerical Recipes in C, The Art of Scientific Computing, by William H. Press, Second Ed., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, Cambridge University Press, 1992

The Special Functions and Their Approximations: Volumes 1 and 2, by Yudell L. Luke, Academic Press, 1969



## Chapter 2

### Todo List

page [Mathematical Special Functions](#)

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

Member [std::\\_\\_detail::\\_\\_dawson\\_cont\\_frac](#) ([\\_Tp \\_\\_x](#))

this needs some compile-time construction!

Member [std::\\_\\_detail::\\_\\_expint\\_E1](#) ([\\_Tp \\_\\_x](#))

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

Member [std::\\_\\_detail::\\_\\_expint\\_En\\_recursion](#) ([unsigned int \\_\\_n](#), [\\_Tp \\_\\_x](#))

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

Member [std::\\_\\_detail::\\_\\_hurwitz\\_zeta](#) ([\\_Tp \\_\\_s](#), [std::complex<\\_Tp> \\_\\_a](#))

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





## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

C++ Mathematical Special Functions . . . . .	<a href="#">17</a>
C++17/IS29124 Mathematical Special Functions . . . . .	<a href="#">18</a>
GNU Extended Mathematical Special Functions . . . . .	<a href="#">42</a>



## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">__gnu_cxx</a>	125
<a href="#">std</a>	133
<a href="#">std::__detail</a>	135



## Chapter 5

# Class Index

### 5.1 Class List

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

[std::\\_\\_detail::\\_Factorial\\_table<\\_Tp>](#) . . . . . 263



## Chapter 6

# File Index

### 6.1 File List

Here is a list of all files with brief descriptions:

<a href="#">bits/sf_airy.tcc</a>	265
<a href="#">bits/sf_bessel.tcc</a>	267
<a href="#">bits/sf_beta.tcc</a>	269
<a href="#">bits/sf_cardinal.tcc</a>	271
<a href="#">bits/sf_chebyshev.tcc</a>	273
<a href="#">bits/sf_dawson.tcc</a>	275
<a href="#">bits/sf_ellint.tcc</a>	276
<a href="#">bits/sf_expint.tcc</a>	279
<a href="#">bits/sf_fresnel.tcc</a>	281
<a href="#">bits/sf_gamma.tcc</a>	283
<a href="#">bits/sf_gegenbauer.tcc</a>	288
<a href="#">bits/sf_hankel.tcc</a>	290
<a href="#">bits/sf_hankel_new.tcc</a>	293
<a href="#">bits/sf_hermite.tcc</a>	293
<a href="#">bits/sf_hydrogen.tcc</a>	295
<a href="#">bits/sf_hyperg.tcc</a>	296
<a href="#">bits/sf_hypint.tcc</a>	298
<a href="#">bits/sf_jacobi.tcc</a>	299
<a href="#">bits/sf_laguerre.tcc</a>	301
<a href="#">bits/sf_legendre.tcc</a>	303
<a href="#">bits/sf_mod_bessel.tcc</a>	304
<a href="#">bits/sf_owens_t.tcc</a>	306
<a href="#">bits/sf_polylog.tcc</a>	307
<a href="#">bits/sf_theta.tcc</a>	310
<a href="#">bits/sf_trigint.tcc</a>	312
<a href="#">bits/sf_zeta.tcc</a>	314
<a href="#">bits/specfun.h</a>	317



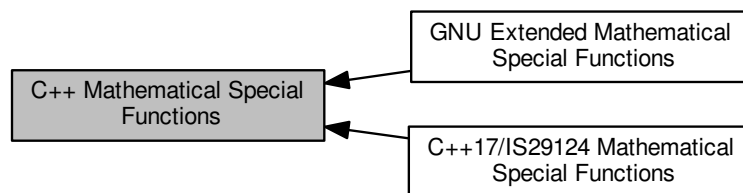


## Chapter 7

# Module Documentation

### 7.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



#### Modules

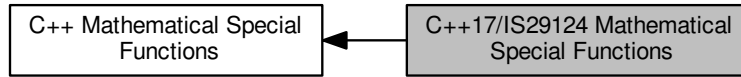
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

#### 7.1.1 Detailed Description

A collection of advanced mathematical special functions.

## 7.2 C++17/IS29124 Mathematical Special Functions

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



### Functions

- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::assoc_laguerre` (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_x)
- `float std::assoc_laguerref` (unsigned int \_\_n, unsigned int \_\_m, float \_\_x)
- `long double std::assoc_laguerrel` (unsigned int \_\_n, unsigned int \_\_m, long double \_\_x)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::assoc_legendre` (unsigned int \_\_l, unsigned int \_\_m, \_Tp \_\_x)
- `float std::assoc_legendref` (unsigned int \_\_l, unsigned int \_\_m, float \_\_x)
- `long double std::assoc_legendrel` (unsigned int \_\_l, unsigned int \_\_m, long double \_\_x)
- `template<typename _Tpa, typename _Tpb >`  
`__gnu_cxx::__promote_2< _Tpa, _Tpb >::__type std::beta` (\_Tpa \_\_a, \_Tpb \_\_b)
- `float std::betaf` (float \_\_a, float \_\_b)
- `long double std::betal` (long double \_\_a, long double \_\_b)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (\_Tp \_\_k)
- `float std::comp_ellint_1f` (float \_\_k)
- `long double std::comp_ellint_1l` (long double \_\_k)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (\_Tp \_\_k)
- `float std::comp_ellint_2f` (float \_\_k)
- `long double std::comp_ellint_2l` (long double \_\_k)
- `template<typename _Tp, typename _Tpn >`  
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (\_Tp \_\_k, \_Tpn \_\_nu)
- `float std::comp_ellint_3f` (float \_\_k, float \_\_nu)
- *Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for float modulus  $k$ .*
- `long double std::comp_ellint_3l` (long double \_\_k, long double \_\_nu)
- *Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for long double modulus  $k$ .*
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float std::cyl_bessel_if` (float \_\_nu, float \_\_x)
- `long double std::cyl_bessel_il` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float std::cyl_bessel_jf` (float \_\_nu, float \_\_x)
- `long double std::cyl_bessel_jl` (long double \_\_nu, long double \_\_x)

- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_k ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_kf (float __nu, float __x)`
- `long double std::cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_neumann ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_neumannf (float __nu, float __x)`
- `long double std::cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp, typename _Tpp >`  
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_1 ( _Tp __k, _Tpp __phi)`
- `float std::ellint_1f (float __k, float __phi)`
- `long double std::ellint_1l (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tpp >`  
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type std::ellint_2 ( _Tp __k, _Tpp __phi)`
- `float std::ellint_2f (float __k, float __phi)`  
*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$  for float argument.*
- `long double std::ellint_2l (long double __k, long double __phi)`  
*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$ .*
- `template<typename _Tp, typename _Tpn, typename _Tpp >`  
`__gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type std::ellint_3 ( _Tp __k, _Tpn __nu, _Tpp __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- `float std::ellint_3f (float __k, float __nu, float __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$  for float argument.*
- `long double std::ellint_3l (long double __k, long double __nu, long double __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::expint ( _Tp __x)`
- `float std::expintf (float __x)`
- `long double std::expintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)`
- `float std::hermitef (unsigned int __n, float __x)`
- `long double std::hermitel (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::laguerre (unsigned int __n, _Tp __x)`
- `float std::laguerref (unsigned int __n, float __x)`
- `long double std::laguerrel (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::legendre (unsigned int __l, _Tp __x)`
- `float std::legendref (unsigned int __l, float __x)`
- `long double std::legendrel (unsigned int __l, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::riemann_zeta ( _Tp __s)`
- `float std::riemann_zetaf (float __s)`
- `long double std::riemann_zetal (long double __s)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::sph_bessel (unsigned int __n, _Tp __x)`
- `float std::sph_besself (unsigned int __n, float __x)`
- `long double std::sph_bessell (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

- float [std::sph\\_legendref](#) (unsigned int \_\_l, unsigned int \_\_m, float \_\_theta)
- long double [std::sph\\_legendrel](#) (unsigned int \_\_l, unsigned int \_\_m, long double \_\_theta)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [std::sph\\_neumann](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [std::sph\\_neumannf](#) (unsigned int \_\_n, float \_\_x)
- long double [std::sph\\_neumannl](#) (unsigned int \_\_n, long double \_\_x)

## 7.2.1 Detailed Description

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

## 7.2.2 Function Documentation

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

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

The associated Laguerre function of real degree  $\alpha$ ,  $L_n^\alpha(x)$ , 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  $x \geq 0$ .

See also

[laguerre](#) for details of the Laguerre function of degree  $n$

### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

## Parameters

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

## Exceptions

<code>std::domain_error</code>	if __x < 0.
--------------------------------	-------------

Definition at line 372 of file specfun.h.

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

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

See also

[assoc\\_laguerre](#) for more details.

Definition at line 324 of file specfun.h.

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

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

See also

[assoc\\_laguerre](#) for more details.

Definition at line 335 of file specfun.h.

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

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

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

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

See also

[legendre](#) for details of the Legendre function of degree 1

## Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

## Parameters

<code>__l</code>	The degree <code>__l &gt;= 0</code> .
<code>__m</code>	The order <code>__m &lt;= 1</code> .
<code>__x</code>	The argument, <code>abs (__x) &lt;= 1</code> .

## Exceptions

<code>std::domain_error</code>	if <code>abs (__x) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 420 of file `specfun.h`.

**7.2.2.5** `float std::assoc_legendref ( unsigned int __l, unsigned int __m, float __x ) [inline]`

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

See also

[assoc\\_legendre](#) for more details.

Definition at line 387 of file `specfun.h`.

**7.2.2.6** `long double std::assoc_legendrel ( unsigned int __l, unsigned int __m, long double __x ) [inline]`

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

See also

[assoc\\_legendre](#) for more details.

Definition at line 398 of file `specfun.h`.

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

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

The beta function is defined by

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

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

## Template Parameters

<code>_Tpa</code>	The floating-point type of the parameter <code>__a</code> .
<code>_Tpb</code>	The floating-point type of the parameter <code>__b</code> .

## Parameters

<code>__a</code>	The first argument of the beta function, <code>__a &gt; 0</code> .
<code>__b</code>	The second argument of the beta function, <code>__b &gt; 0</code> .

## Exceptions

<code>std::domain_error</code>	if <code>__a &lt; 0</code> or <code>__b &lt; 0</code> .
--------------------------------	---

Definition at line 465 of file `specfun.h`.

7.2.2.8 `float std::betaf ( float __a, float __b ) [inline]`

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

See also

[beta](#) for more details.

Definition at line 434 of file `specfun.h`.

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

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

See also

[beta](#) for more details.

Definition at line 444 of file `specfun.h`.

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

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

The complete elliptic integral of the first kind is defined as

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

where  $F(k, \phi)$  is the incomplete elliptic integral of the first kind and the modulus  $|k| \leq 1$ .

See also

[ellint\\_1](#) for details of the incomplete elliptic function of the first kind.

## Template Parameters

<code>_Tp</code>	The floating-point type of the modulus <code>__k</code> .
------------------	---

## Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
------------------	---

## Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 513 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_1](#) for details.

Definition at line 480 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_1](#) for details.

Definition at line 490 of file `specfun.h`.

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

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

The complete elliptic integral of the second kind is defined as

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

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

See also

[ellint\\_2](#) for details of the incomplete elliptic function of the second kind.



## Template Parameters

<code>_Tp</code>	The floating-point type of the modulus <code>__k</code> .
------------------	---

## Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
------------------	---

## Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 560 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_2](#) for details.

Definition at line 528 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_2](#) for details.

Definition at line 538 of file `specfun.h`.

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

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

The complete elliptic integral of the third kind is defined as

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

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

See also

[ellint\\_3](#) for details of the incomplete elliptic function of the third kind.

## Template Parameters

<code>__Tp</code>	The floating-point type of the modulus <code>__k</code> .
<code>__Tpnu</code>	The floating-point type of the argument <code>__nu</code> .

## Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
<code>__nu</code>	The argument

## Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 611 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_3](#) for details.

Definition at line 575 of file `specfun.h`.

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

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

See also

[comp\\_ellint\\_3](#) for details.

Definition at line 585 of file `specfun.h`.

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

Return the regular modified Bessel function  $I_\nu(x)$  for real order  $\nu$  and argument  $x \geq 0$ .

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

<code>_Tpnu</code>	The floating-point type of the order <code>__nu</code> .
<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .

## Parameters

<code>__nu</code>	The order
<code>__x</code>	The argument, <code>__x &gt;= 0</code>

## Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 657 of file `specfun.h`.

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

Return the regular modified Bessel function  $I_\nu(x)$  for `float` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_bessel\\_i](#) for setails.

Definition at line 626 of file `specfun.h`.

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

Return the regular modified Bessel function  $I_\nu(x)$  for `long double` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_bessel\\_i](#) for setails.

Definition at line 636 of file `specfun.h`.

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

Return the Bessel function  $J_\nu(x)$  of real order  $\nu$  and argument  $x \geq 0$ .

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

<code>_Tpnu</code>	The floating-point type of the order <code>__nu</code> .
<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .

## Parameters

<code>__nu</code>	The order
<code>__x</code>	The argument, <code>__x &gt;= 0</code>

## Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 703 of file `specfun.h`.

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

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

See also

[cyl\\_bessel\\_j](#) for setails.

Definition at line 672 of file `specfun.h`.

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

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

See also

[cyl\\_bessel\\_j](#) for setails.

Definition at line 682 of file `specfun.h`.

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

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

The irregular modified Bessel function is defined by:

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

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

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

## Template Parameters

<code>_Tpnu</code>	The floating-point type of the order <code>__nu</code> .
<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .

## Parameters

<code>__nu</code>	The order
<code>__x</code>	The argument, <code>__x &gt;= 0</code>

## Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 755 of file `specfun.h`.

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

Return the irregular modified Bessel function  $K_\nu(x)$  for `float` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_bessel\\_k](#) for details.

Definition at line 718 of file `specfun.h`.

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

Return the irregular modified Bessel function  $K_\nu(x)$  for `long double` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_bessel\\_k](#) for details.

Definition at line 728 of file `specfun.h`.

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

Return the Neumann function  $N_\nu(x)$  of real order  $\nu$  and argument  $x \geq 0$ .

The Neumann function is defined by:

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

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

## Template Parameters

<code>_Tpnu</code>	The floating-point type of the order <code>__nu</code> .
<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .

## Parameters

<code>__nu</code>	The order
<code>__x</code>	The argument, <code>__x &gt;= 0</code>

## Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 803 of file `specfun.h`.

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

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

See also

[cyl\\_neumann](#) for setails.

Definition at line 770 of file `specfun.h`.

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

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

See also

[cyl\\_neumann](#) for setails.

Definition at line 780 of file `specfun.h`.

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

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

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

<code>_Tp</code>	The floating-point type of the modulus <code>__k</code> .
<code>_Tpp</code>	The floating-point type of the angle <code>__phi</code> .

## Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
<code>__phi</code>	The integral limit argument in radians

## Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 851 of file `specfun.h`.

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

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

See also

[ellint\\_1](#) for details.

Definition at line 818 of file `specfun.h`.

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

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

See also

[ellint\\_1](#) for details.

Definition at line 828 of file `specfun.h`.

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

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

The incomplete elliptic integral of the second kind is defined as

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

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

See also

[comp\\_ellint\\_2](#).

### Template Parameters

<code>__Tp</code>	The floating-point type of the modulus <code>__k</code> .
<code>__Tpp</code>	The floating-point type of the angle <code>__phi</code> .

### Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
<code>__phi</code>	The integral limit argument in radians

### Returns

The elliptic function of the second kind.

### Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 899 of file `specfun.h`.

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

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

### See also

[ellint\\_2](#) for details.

Definition at line 866 of file `specfun.h`.

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

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

### See also

[ellint\\_2](#) for details.

Definition at line 876 of file `specfun.h`.



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

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

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

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

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

See also

[comp\\_ellint\\_3](#).

#### Template Parameters

<code>_Tp</code>	The floating-point type of the modulus <code>__k</code> .
<code>_Tpn</code>	The floating-point type of the argument <code>__nu</code> .
<code>_Tpp</code>	The floating-point type of the angle <code>__phi</code> .

#### Parameters

<code>__k</code>	The modulus, <code>abs (__k) &lt;= 1</code>
<code>__nu</code>	The second argument
<code>__phi</code>	The integral limit argument in radians

#### Returns

The elliptic function of the third kind.

#### Exceptions

<code>std::domain_error</code>	if <code>abs (__k) &gt; 1</code> .
--------------------------------	------------------------------------

Definition at line 952 of file `specfun.h`.

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

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

See also

[ellint\\_3](#) for details.

Definition at line 914 of file `specfun.h`.

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

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

See also

[ellint\\_3](#) for details.

Definition at line 924 of file specfun.h.

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

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

The exponential integral is given by

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

Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

Parameters

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Definition at line 992 of file specfun.h.

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

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

See also

[expint](#) for details.

Definition at line 966 of file specfun.h.

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

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

See also

[expint](#) for details.

Definition at line 976 of file specfun.h.

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

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

The Hermite polynomial is defined by:

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

The Hermite polynomial obeys a reflection formula:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The order
<code>__x</code>	The argument

Definition at line 1040 of file `specfun.h`.

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

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

See also

[hermite](#) for details.

Definition at line 1007 of file `specfun.h`.

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

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

See also

[hermite](#) for details.

Definition at line 1017 of file `specfun.h`.

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

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

The Laguerre polynomial is defined by:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The nonnegative order
<code>__x</code>	The argument <code>__x</code> $\geq 0$

#### Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 1084 of file `specfun.h`.

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

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

#### See also

[laguerre](#) for more details.

Definition at line 1055 of file `specfun.h`.

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

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

#### See also

[laguerre](#) for more details.

Definition at line 1065 of file `specfun.h`.

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

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

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

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__l</code>	The degree $l \geq 0$
<code>__x</code>	The argument $\text{abs}(\text{__x}) \leq 1$

#### Exceptions

<code>std::domain_error</code>	if $\text{abs}(\text{__x}) > 1$
--------------------------------	---------------------------------

Definition at line 1129 of file `specfun.h`.

**7.2.2.50** `float std::legendrf ( unsigned int __l, float __x )` `[inline]`

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

See also

[legendre](#) for more details.

Definition at line 1099 of file `specfun.h`.

**7.2.2.51** `long double std::legendrl ( unsigned int __l, long double __x )` `[inline]`

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

See also

[legendre](#) for more details.

Definition at line 1109 of file `specfun.h`.

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

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

The Riemann zeta function is defined by:

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

and

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

For  $s < 1$  use the reflection formula:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__s</code> .
------------------	--

#### Parameters

<code>__s</code>	The argument $s \neq 1$
------------------	-------------------------

Definition at line 1180 of file `specfun.h`.

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

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

See also

[riemann\\_zeta](#) for more details.

Definition at line 1144 of file `specfun.h`.

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

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

See also

[riemann\\_zeta](#) for more details.

Definition at line 1154 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The integral order $n \geq 0$
<code>__x</code>	The real argument $x \geq 0$

#### Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 1224 of file `specfun.h`.

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

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

See also

[sph\\_bessel](#) for more details.

Definition at line 1195 of file `specfun.h`.

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

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

See also

[sph\\_bessel](#) for more details.

Definition at line 1205 of file `specfun.h`.

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

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

The spherical Legendre function is defined by

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the angle <code>__theta</code> .
------------------	---

#### Parameters

<code>__l</code>	The order <code>__l</code> $\geq 0$
<code>__m</code>	The degree <code>__m</code> $\geq 0$ and <code>__m</code> $\leq$ <code>__l</code>
<code>__theta</code>	The radian polar angle argument

Definition at line 1271 of file `specfun.h`.

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

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

See also

[sph\\_legendre](#) for details.

Definition at line 1239 of file `specfun.h`.

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

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

See also

[sph\\_legendre](#) for details.

Definition at line 1250 of file `specfun.h`.



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

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

The spherical Neumann function is defined by

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The integral order $n \geq 0$
<code>__x</code>	The real argument <code>__x</code> $\geq 0$

#### Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 1315 of file `specfun.h`.

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

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

See also

[sph\\_neumann](#) for details.

Definition at line 1286 of file `specfun.h`.

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

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

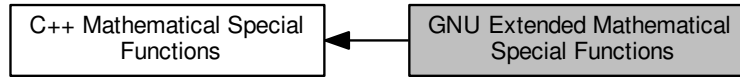
See also

[sph\\_neumann](#) for details.

Definition at line 1296 of file `specfun.h`.

## 7.3 GNU Extended Mathematical Special Functions

Collaboration diagram for GNU Extended Mathematical Special Functions:



### Enumerations

- enum { [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_SN](#), [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_CN](#), [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_DN](#) }

### Functions

- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::airy\\_ai](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_aif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_ail](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::airy\\_bi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_bif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_bil](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::bernoulli](#) (unsigned int \_\_n)
- float [\\_\\_gnu\\_cxx::bernoullif](#) (unsigned int \_\_n)
- long double [\\_\\_gnu\\_cxx::bernoullil](#) (unsigned int \_\_n)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::bincoef](#) (unsigned int \_\_n, unsigned int \_\_k)
- float [\\_\\_gnu\\_cxx::bincoeff](#) (unsigned int \_\_n, unsigned int \_\_k)
- long double [\\_\\_gnu\\_cxx::bincoefl](#) (unsigned int \_\_n, unsigned int \_\_k)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_t](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_tf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_tl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_u](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_uf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_ul](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_v](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_vf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_vl](#) (unsigned int \_\_n, long double \_\_x)

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

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

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

- float [\\_\\_gnu\\_cxx::fresnel\\_sf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::fresnel\\_sl](#) (long double \_\_x)
- template<typename \_Tn, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tn, \\_Tp>](#) [\\_\\_gnu\\_cxx::gamma\\_l](#) (\_Tn \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::gamma\\_lf](#) (float \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::gamma\\_ll](#) (long double \_\_n, long double \_\_x)
- template<typename \_Tn, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tn, \\_Tp>](#) [\\_\\_gnu\\_cxx::gamma\\_u](#) (\_Tn \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::gamma\\_uf](#) (float \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::gamma\\_ul](#) (long double \_\_n, long double \_\_x)
- template<typename \_Talpha, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Talpha, \\_Tp>](#) [\\_\\_gnu\\_cxx::gegenbauer](#) (unsigned int \_\_n, \_Talpha \_\_alpha, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::gegenbauerf](#) (unsigned int \_\_n, float \_\_alpha, float \_\_x)
- long double [\\_\\_gnu\\_cxx::gegenbauerl](#) (unsigned int \_\_n, long double \_\_alpha, long double \_\_x)
- template<typename \_Tk, typename \_Tphi >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tk, \\_Tphi>](#) [\\_\\_gnu\\_cxx::heuman\\_lambda](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::heuman\\_lambdaf](#) (float \_\_k, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::heuman\\_lambdal](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up>](#) [\\_\\_gnu\\_cxx::hurwitz\\_zeta](#) (\_Tp \_\_s, \_Up \_\_a)
- template<typename \_Tp, typename \_Up >  
std::complex<\_Tp> [\\_\\_gnu\\_cxx::hurwitz\\_zeta](#) (\_Tp \_\_s, std::complex<\_Up> \_\_a)
- float [\\_\\_gnu\\_cxx::hurwitz\\_zetaf](#) (float \_\_s, float \_\_a)
- long double [\\_\\_gnu\\_cxx::hurwitz\\_zetal](#) (long double \_\_s, long double \_\_a)
- template<typename \_Tpa, typename \_Tpb, typename \_Tpc, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_4<\\_Tpa, \\_Tpb, \\_Tpc, \\_Tp>::type](#) [\\_\\_gnu\\_cxx::hyperg](#) (\_Tpa \_\_a, \_Tpb \_\_b, \_Tpc \_\_c, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::hypergf](#) (float \_\_a, float \_\_b, float \_\_c, float \_\_x)
- long double [\\_\\_gnu\\_cxx::hypergl](#) (long double \_\_a, long double \_\_b, long double \_\_c, long double \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Ta, \\_Tb, \\_Tp>](#) [\\_\\_gnu\\_cxx::ibeta](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Ta, \\_Tb, \\_Tp>](#) [\\_\\_gnu\\_cxx::ibetac](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::ibetacf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [\\_\\_gnu\\_cxx::ibetacl](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- float [\\_\\_gnu\\_cxx::ibetaf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [\\_\\_gnu\\_cxx::ibetal](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- template<typename \_Talpha, typename \_Tbeta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Talpha, \\_Tbeta, \\_Tp>](#) [\\_\\_gnu\\_cxx::jacobi](#) (unsigned \_\_n, \_Talpha \_\_alpha, \_Tbeta \_\_beta, \_Tp \_\_x)
- template<typename \_Kp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Kp, \\_Up>](#) [\\_\\_gnu\\_cxx::jacobi\\_cn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [\\_\\_gnu\\_cxx::jacobi\\_cnf](#) (float \_\_k, float \_\_u)
- long double [\\_\\_gnu\\_cxx::jacobi\\_cnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Kp, \\_Up>](#) [\\_\\_gnu\\_cxx::jacobi\\_dn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [\\_\\_gnu\\_cxx::jacobi\\_dnf](#) (float \_\_k, float \_\_u)
- long double [\\_\\_gnu\\_cxx::jacobi\\_dnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Kp, \\_Up>](#) [\\_\\_gnu\\_cxx::jacobi\\_sn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [\\_\\_gnu\\_cxx::jacobi\\_snf](#) (float \_\_k, float \_\_u)

- long double [\\_\\_gnu\\_cxx::jacobi\\_snl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Tk, typename \_Tphi >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tk, \\_Tphi > \\_\\_gnu\\_cxx::jacobi\\_zeta](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::jacobi\\_zetaf](#) (float \_\_k, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::jacobi\\_zetal](#) (long double \_\_k, long double \_\_phi)
- float [\\_\\_gnu\\_cxx::jacobif](#) (unsigned \_\_n, float \_\_alpha, float \_\_beta, float \_\_x)
- long double [\\_\\_gnu\\_cxx::jacobil](#) (unsigned \_\_n, long double \_\_alpha, long double \_\_beta, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp > \\_\\_gnu\\_cxx::lbincoef](#) (unsigned int \_\_n, unsigned int \_\_k)
- float [\\_\\_gnu\\_cxx::lbincoeff](#) (unsigned int \_\_n, unsigned int \_\_k)
- long double [\\_\\_gnu\\_cxx::lbincoefl](#) (unsigned int \_\_n, unsigned int \_\_k)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp > \\_\\_gnu\\_cxx::ldouble\\_factorial](#) (int \_\_n)
- float [\\_\\_gnu\\_cxx::ldouble\\_factorialf](#) (int \_\_n)
- long double [\\_\\_gnu\\_cxx::ldouble\\_factoriall](#) (int \_\_n)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp > \\_\\_gnu\\_cxx::legendre\\_q](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::legendre\\_qf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::legendre\\_ql](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp > \\_\\_gnu\\_cxx::lfactorial](#) (unsigned int \_\_n)
- float [\\_\\_gnu\\_cxx::lfactorialf](#) (unsigned int \_\_n)
- long double [\\_\\_gnu\\_cxx::lfactoriall](#) (unsigned int \_\_n)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp > \\_\\_gnu\\_cxx::logint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::logintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::logintl](#) (long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn > \\_\\_gnu\\_cxx::lpochhammer\\_l](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::lpochhammer\\_lf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::lpochhammer\\_ll](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn > \\_\\_gnu\\_cxx::lpochhammer\\_u](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::lpochhammer\\_uf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::lpochhammer\\_ul](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tph, typename \_Tpa >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tph, \\_Tpa > \\_\\_gnu\\_cxx::owens\\_t](#) (\_Tph \_\_h, \_Tpa \_\_a)
- float [\\_\\_gnu\\_cxx::owens\\_tf](#) (float \_\_h, float \_\_a)
- long double [\\_\\_gnu\\_cxx::owens\\_tl](#) (long double \_\_h, long double \_\_a)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_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 \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn > \\_\\_gnu\\_cxx::pochhammer\\_l](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_lf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_ll](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn > \\_\\_gnu\\_cxx::pochhammer\\_u](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_uf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_ul](#) (long double \_\_a, long double \_\_n)

- `template<typename _Tp, typename _Wp >`  
`__gnu_cxx::__promote_num_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)`
- `template<typename _Tp, typename _Wp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > __gnu_cxx::polylog (_Tp __s, std::complex< _Tp > __w)`
- `float __gnu_cxx::polylogf (float __s, float __w)`
- `std::complex< float > __gnu_cxx::polylogf (float __s, std::complex< float > __w)`
- `long double __gnu_cxx::polylogl (long double __s, long double __w)`
- `std::complex< long double > __gnu_cxx::polylogl (long double __s, std::complex< long double > __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::psi (_Tp __x)`
- `float __gnu_cxx::psif (float __x)`
- `long double __gnu_cxx::psil (long double __x)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_num_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_num_t< _Tp > __gnu_cxx::radpoly (unsigned int __n, unsigned int __m, _Tp __rho)`
- `float __gnu_cxx::radpolyf (unsigned int __n, unsigned int __m, float __rho)`
- `long double __gnu_cxx::radpolyl (unsigned int __n, unsigned int __m, long double __rho)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc (_Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinc_pi (_Tp __x)`
- `float __gnu_cxx::sinc_pif (float __x)`
- `long double __gnu_cxx::sinc_pil (long double __x)`
- `float __gnu_cxx::sincf (float __x)`
- `long double __gnu_cxx::sincl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc (_Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhc_pi (_Tp __x)`
- `float __gnu_cxx::sinhc_pif (float __x)`
- `long double __gnu_cxx::sinhc_pil (long double __x)`
- `float __gnu_cxx::sinhcf (float __x)`
- `long double __gnu_cxx::sinhcl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhint (_Tp __x)`
- `float __gnu_cxx::sinhintf (float __x)`
- `long double __gnu_cxx::sinhintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinint (_Tp __x)`
- `float __gnu_cxx::sinintf (float __x)`
- `long double __gnu_cxx::sinintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)`



- long double [\\_\\_gnu\\_cxx::sph\\_bessel\\_kl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tp >](#) > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1](#) (unsigned int \_\_n, \_Tp \_\_z)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tp >](#) > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1](#) (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1f](#) (unsigned int \_\_n, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1f](#) (unsigned int \_\_n, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1l](#) (unsigned int \_\_n, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1l](#) (unsigned int \_\_n, std::complex< long double > \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tp >](#) > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2](#) (unsigned int \_\_n, \_Tp \_\_z)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tp >](#) > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2](#) (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2f](#) (unsigned int \_\_n, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2f](#) (unsigned int \_\_n, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2l](#) (unsigned int \_\_n, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2l](#) (unsigned int \_\_n, std::complex< long double > \_\_x)
- template<typename \_Ttheta, typename \_Tphi >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Ttheta, \\_Tphi >](#) > [\\_\\_gnu\\_cxx::sph\\_harmonic](#) (unsigned int \_\_l, int \_\_m, \_Ttheta \_\_theta, \_Tphi \_\_phi)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_harmonicf](#) (unsigned int \_\_l, int \_\_m, float \_\_theta, float \_\_phi)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_harmonicl](#) (unsigned int \_\_l, int \_\_m, long double \_\_theta, long double \_\_phi)
- template<typename \_Tpnu, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpnu, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_1](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_1f](#) (float \_\_nu, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_1l](#) (long double \_\_nu, long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpnu, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_2](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_2f](#) (float \_\_nu, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_2l](#) (long double \_\_nu, long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpnu, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_3](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_3f](#) (float \_\_nu, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_3l](#) (long double \_\_nu, long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpnu, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_4](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_4f](#) (float \_\_nu, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_4l](#) (long double \_\_nu, long double \_\_x)
- template<typename \_Tpk, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpk, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_c](#) (\_Tpk \_\_k, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_cf](#) (float \_\_k, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_cl](#) (long double \_\_k, long double \_\_x)
- template<typename \_Tpk, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpk, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_d](#) (\_Tpk \_\_k, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_df](#) (float \_\_k, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_dl](#) (long double \_\_k, long double \_\_x)
- template<typename \_Tpk, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tpk, \\_Tp >](#) [\\_\\_gnu\\_cxx::theta\\_n](#) (\_Tpk \_\_k, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_nf](#) (float \_\_k, float \_\_x)

- long double [\\_\\_gnu\\_cxx::theta\\_nl](#) (long double \_\_k, long double \_\_x)
- template<typename \_Tpk, typename \_Tp > [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tpk, \\_Tp> \\_\\_gnu\\_cxx::theta\\_s](#) (\_Tpk \_\_k, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::theta\\_sf](#) (float \_\_k, float \_\_x)
- long double [\\_\\_gnu\\_cxx::theta\\_sl](#) (long double \_\_k, long double \_\_x)
- template<typename \_Trho, typename \_Tphi > [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Trho, \\_Tphi> \\_\\_gnu\\_cxx::zernike](#) (unsigned int \_\_n, int \_\_m, \_Trho \_\_rho, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::zernikef](#) (unsigned int \_\_n, int \_\_m, float \_\_rho, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::zernikel](#) (unsigned int \_\_n, int \_\_m, long double \_\_rho, long double \_\_phi)

### 7.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

### 7.3.2 Enumeration Type Documentation

#### 7.3.2.1 anonymous enum

Enumerator

[\\_GLIBCXX\\_JACOBI\\_SN](#)  
[\\_GLIBCXX\\_JACOBI\\_CN](#)  
[\\_GLIBCXX\\_JACOBI\\_DN](#)

Definition at line 1735 of file specfun.h.

### 7.3.3 Function Documentation

#### 7.3.3.1 template<typename \_Tp> \_\_gnu\_cxx::\_\_promote\_num\_t<\_Tp> \_\_gnu\_cxx::airy\_ai( \_Tp \_\_x ) [inline]

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

The Airy function is defined by:

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

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>_↔</code>	The argument
<code>_x</code>	

Definition at line 2677 of file specfun.h.

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

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

## See also

[airy\\_ai](#) for details.

Definition at line 2642 of file specfun.h.

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

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

## See also

[airy\\_ai](#) for details.

Definition at line 2656 of file specfun.h.

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

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

The Airy function is defined by:

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

## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

$\leftrightarrow$ __x	The argument
--------------------------	--------------

Definition at line 2729 of file specfun.h.

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

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

## See also

[airy\\_bi](#) for details.

Definition at line 2693 of file specfun.h.

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

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

## See also

[airy\\_bi](#) for details.

Definition at line 2707 of file specfun.h.

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

Return the Bernoulli number of integer order  $n$ .

The Bernoulli numbers are defined by

## Parameters

$\leftrightarrow$ __n	The order.
--------------------------	------------

Definition at line 3716 of file specfun.h.

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

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

See also

[bernoulli](#) for details.

Definition at line 3691 of file `specfun.h`.

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

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

See also

[bernoulli](#) for details.

Definition at line 3701 of file `specfun.h`.

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

Definition at line 3656 of file `specfun.h`.

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

Definition at line 3644 of file `specfun.h`.

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

Definition at line 3648 of file `specfun.h`.

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

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

The Chebyshev polynomial of the first kind is defined by:

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

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

## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>↔ _n</code>	The non-negative integral order
<code>↔ _x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 1936 of file `specfun.h`.

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

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

See also

[chebyshev\\_t](#) for details.

Definition at line 1907 of file `specfun.h`.

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

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

See also

[chebyshev\\_t](#) for details.

Definition at line 1917 of file `specfun.h`.

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

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

The Chebyshev polynomial of the second kind is defined by:

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

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

## Template Parameters

<code>__Tp</code>	The real type of the argument
-------------------	-------------------------------

## Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 1980 of file `specfun.h`.

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

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

See also

[chebyshev\\_u](#) for details.

Definition at line 1951 of file `specfun.h`.

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

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

See also

[chebyshev\\_u](#) for details.

Definition at line 1961 of file `specfun.h`.

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

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

The Chebyshev polynomial of the third kind is defined by:

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

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

## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 2025 of file `specfun.h`.

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

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

See also

[chebyshev\\_v](#) for details.

Definition at line 1995 of file `specfun.h`.

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

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

See also

[chebyshev\\_v](#) for details.

Definition at line 2005 of file `specfun.h`.

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

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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



## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>↔ _n</code>	The non-negative integral order
<code>↔ _x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 2070 of file specfun.h.

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

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

See also

[chebyshev\\_w](#) for details.

Definition at line 2040 of file specfun.h.

**7.3.3.24** `long double __gnu_cxx::chebyshev_wl ( unsigned int __n, long double __x ) [inline]`

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

See also

[chebyshev\\_w](#) for details.

Definition at line 2050 of file specfun.h.

**7.3.3.25** `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::clausen ( unsigned int __m, _Tp __w ) [inline]`

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

The Clausen function is defined by

## Parameters

$\leftrightarrow$ __m	
$\leftrightarrow$ __w	The complex argument

Definition at line 4659 of file specfun.h.

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

Definition at line 4680 of file specfun.h.

```
7.3.3.27  template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::clausen_c ( unsigned int __m, _Tp __w )
    [inline]
```

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

The Clausen cosine function is defined by

## Parameters

$\leftrightarrow$ __m	
$\leftrightarrow$ __w	

Definition at line 4620 of file specfun.h.

```
7.3.3.28  float __gnu_cxx::clausen_cf ( unsigned int __m, float __w )  [inline]
```

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

## See also

[clausen\\_c](#) for details.

Definition at line 4595 of file specfun.h.

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

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

See also

[clausen\\_c](#) for details.

Definition at line 4604 of file `specfun.h`.

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

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

The Clausen sine function is defined by

#### Parameters

$\leftrightarrow$ $\_m$	
$\leftrightarrow$ $\_w$	

Definition at line 4581 of file `specfun.h`.

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

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

See also

[clausen\\_s](#) for details.

Definition at line 4556 of file `specfun.h`.

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

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

See also

[clausen\\_s](#) for details.

Definition at line 4565 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 4634 of file `specfun.h`.

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

Definition at line 4668 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 4643 of file `specfun.h`.

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

Definition at line 4672 of file `specfun.h`.

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

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

The complete Legendre elliptic integral D is defined by

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

Template Parameters

<code>_Tk</code>	The type of the modulus $k$
------------------	-----------------------------

## Parameters

$\leftrightarrow$ _k	The modulus $-1 \leq k \leq +1$
-------------------------	---------------------------------

Definition at line 3902 of file specfun.h.

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

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

See also

[comp\\_ellint\\_d](#) for details.

Definition at line 3875 of file specfun.h.

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

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

See also

[comp\\_ellint\\_d](#) for details.

Definition at line 3885 of file specfun.h.

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

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

See also

[comp\\_ellint\\_rf](#) for details.

Definition at line 2850 of file specfun.h.

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

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

See also

[comp\\_ellint\\_rf](#) for details.

Definition at line 2860 of file specfun.h.

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

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

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

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

## Parameters

$\leftrightarrow$ __x	The first argument.
$\leftrightarrow$ __y	The second argument.

Definition at line 2878 of file specfun.h.

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

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

## See also

[comp\\_ellint\\_rg](#) for details.

Definition at line 3083 of file specfun.h.

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

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

## See also

[comp\\_ellint\\_rg](#) for details.

Definition at line 3092 of file specfun.h.

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

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

The complete Carlson elliptic function is defined by:

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

## Parameters

$\leftrightarrow$ __x	The first argument.
$\leftrightarrow$ __y	The second argument.

Definition at line 3111 of file specfun.h.

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

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

The confluent hypergeometric function is defined by

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

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

#### Parameters

$\hookleftarrow$ _a	The numeratorial parameter
$\hookleftarrow$ _c	The denominatorial parameter
$\hookleftarrow$ _x	The argument

Definition at line 1378 of file specfun.h.

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

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

The confluent hypergeometric limit function is defined by

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

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

#### Parameters

$\hookleftarrow$ _c	The denominatorial parameter
$\hookleftarrow$ _x	The argument

Definition at line 1474 of file specfun.h.

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

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

See also

[conf\\_hyperg\\_lim](#) for details.

Definition at line 1445 of file `specfun.h`.

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

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

See also

[conf\\_hyperg\\_lim](#) for details.

Definition at line 1455 of file `specfun.h`.

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

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

See also

[conf\\_hyperg](#) for details.

Definition at line 1346 of file `specfun.h`.

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

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

See also

[conf\\_hyperg](#) for details.

Definition at line 1357 of file `specfun.h`.

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

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

The hyperbolic cosine integral is defined by

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



## Template Parameters

<code>_Tp</code>	The type of the real argument
------------------	-------------------------------

## Parameters

<code>_↵ _x</code>	The real argument
------------------------	-------------------

Definition at line 1728 of file `specfun.h`.

**7.3.3.53** `float __gnu_cxx::coshintf ( float _x ) [inline]`

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

See also

[coshint](#) for details.

Definition at line 1700 of file `specfun.h`.

**7.3.3.54** `long double __gnu_cxx::coshintl ( long double _x ) [inline]`

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

See also

[coshint](#) for details.

Definition at line 1710 of file `specfun.h`.

**7.3.3.55** `template<typename _Tp> __gnu_cxx::_promote_num_t<_Tp> __gnu_cxx::cosint ( _Tp _x ) [inline]`

Return the cosine integral  $Chi(x)$  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

<code>__x</code>	The real upper integration limit
------------------	----------------------------------

Definition at line 1645 of file specfun.h.

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

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

## See also

[cosint](#) for details.

Definition at line 1619 of file specfun.h.

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

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

## See also

[cosint](#) for details.

Definition at line 1629 of file specfun.h.

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

Return the cylindrical Hankel function of the first kind  $H_n^{(1)}(x)$  of real order  $\nu$  and argument  $x \geq 0$ .

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

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

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

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>__nu</code>	The real order
<code>__z</code>	The real argument

Definition at line 2379 of file `specfun.h`.

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

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

The cylindrical Hankel function of the first kind is defined by

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

## Template Parameters

<code>_Tpnu</code>	The complex type of the order
<code>_Tp</code>	The complex type of the argument

## Parameters

<code>__nu</code>	The complex order
<code>__x</code>	The complex argument

Definition at line 4179 of file `specfun.h`.

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

Return the cylindrical Hankel function of the first kind  $H_\nu^{(1)}(x)$  of `float` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_hankel\\_1](#) for details.

Definition at line 2346 of file `specfun.h`.

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

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

See also

[cyl\\_hankel\\_1](#) for more details.

Definition at line 4148 of file `specfun.h`.

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

Return the cylindrical Hankel function of the first kind  $H_\nu^{(1)}(x)$  of `long double` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_hankel\\_1](#) for details.

Definition at line 2357 of file `specfun.h`.

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

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

See also

[cyl\\_hankel\\_1](#) for more details.

Definition at line 4159 of file `specfun.h`.

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

Return the cylindrical Hankel function of the second kind  $H_n^{(2)}(x)$  of real order  $\nu$  and argument  $x \geq 0$ .

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

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

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

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>__nu</code>	The real order
<code>__z</code>	The real argument

Definition at line 2428 of file `specfun.h`.

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

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

The cylindrical Hankel function of the second kind is defined by

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

## Template Parameters

<code>_Tpnu</code>	The complex type of the order
<code>_Tp</code>	The complex type of the argument

## Parameters

<code>__nu</code>	The complex order
<code>__x</code>	The complex argument

Definition at line 4226 of file `specfun.h`.

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

Return the cylindrical Hankel function of the second kind  $H_\nu^{(2)}(x)$  of `float` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_hankel\\_2](#) for details.

Definition at line 2395 of file `specfun.h`.

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

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

See also

[cyl\\_hankel\\_2](#) for more details.

Definition at line 4195 of file `specfun.h`.

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

Return the cylindrical Hankel function of the second kind  $H_\nu^{(2)}(x)$  of `long double` order  $\nu$  and argument  $x \geq 0$ .

See also

[cyl\\_hankel\\_2](#) for details.

Definition at line 2406 of file `specfun.h`.

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

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

See also

[cyl\\_hankel\\_2](#) for more details.

Definition at line 4206 of file `specfun.h`.

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

## Parameters

<code>_↔ _x</code>	The argument $-inf < x < inf$ .
------------------------	---------------------------------

Definition at line 3421 of file specfun.h.

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

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

See also

[dawson](#) for details.

Definition at line 3393 of file specfun.h.

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

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

See also

[dawson](#) for details.

Definition at line 3402 of file specfun.h.

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

Definition at line 2794 of file specfun.h.

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

Definition at line 2782 of file specfun.h.

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

Definition at line 2786 of file specfun.h.

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

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

The dilogarithm is defined by:

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

## Parameters

$\leftrightarrow$	The argument.
$x$	

Definition at line 2835 of file specfun.h.

**7.3.3.77** `float __gnu_cxx::dilogf ( float  $x$  )` `[inline]`

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

## See also

[dilog](#) for details.

Definition at line 2809 of file specfun.h.

**7.3.3.78** `long double __gnu_cxx::dilogl ( long double  $x$  )` `[inline]`

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

## See also

[dilog](#) for details.

Definition at line 2819 of file specfun.h.

**7.3.3.79** `template<typename  $Tp$ >  $Tp$  __gnu_cxx::dirichlet_beta (  $Tp$   $s$  )` `[inline]`

Return the Dirichlet beta function of real argument  $s$ .

The Dirichlet beta function is defined by:

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

An important reflection formula is:

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

## Parameters

$\leftrightarrow$	
$s$	



Definition at line 4542 of file specfun.h.

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

Return the Dirichlet beta function of real argument  $s$ .

See also

[dirichlet\\_beta](#) for details.

Definition at line 4513 of file specfun.h.

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

Return the Dirichlet beta function of real argument  $s$ .

See also

[dirichlet\\_beta](#) for details.

Definition at line 4522 of file specfun.h.

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

Return the Dirichlet eta function of real argument  $s$ .

The Dirichlet eta function is defined by

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

An important reflection formula is:

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

#### Parameters

<code>__s</code>	
------------------	--

Definition at line 4499 of file specfun.h.

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

Return the Dirichlet eta function of real argument  $s$ .

See also

[dirichlet\\_eta](#) for details.

Definition at line 4469 of file specfun.h.

7.3.3.84 `long double __gnu_cxx::dirichlet_eta1 ( long double __s ) [inline]`

Return the Dirichlet eta function of real argument  $s$ .

See also

[dirichlet\\_eta](#) for details.

Definition at line 4478 of file specfun.h.

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

Definition at line 3593 of file specfun.h.

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

Definition at line 3581 of file specfun.h.

7.3.3.87 `long double __gnu_cxx::double_factorial1 ( int __n ) [inline]`

Definition at line 3585 of file specfun.h.

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

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

The Bulirsch complete elliptic integral is defined by

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

## Parameters

<code>__k↔ __c</code>	The complementary modulus $k_c = \sqrt{1 - k^2}$
<code>__p</code>	The parameter
<code>__a</code>	The parameter
<code>__b</code>	The parameter

Definition at line 4132 of file specfun.h.

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

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

See also

[ellint\\_cel](#) for details.

Definition at line 4100 of file specfun.h.

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

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

See also

[ellint\\_cel](#) for details.

Definition at line 4109 of file specfun.h.

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

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

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

<code>__k</code>	The modulus $-1 \leq \text{__k} \leq +1$
<code>__phi</code>	The angle

Definition at line 3945 of file specfun.h.

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

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

See also

[ellint\\_d](#) for details.

Definition at line 3917 of file specfun.h.

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

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

See also

[ellint\\_d](#) for details.

Definition at line 3927 of file specfun.h.

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

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

The Bulirsch elliptic integral of the first kind is defined by

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

## Parameters

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

Definition at line 3991 of file specfun.h.

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

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

See also

[ellint\\_el1](#) for details.

Definition at line 3961 of file specfun.h.

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

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

See also

[ellint\\_el1](#) for details.

Definition at line 3972 of file specfun.h.

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

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

The Bulirsch elliptic integral of the second kind is defined by

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

#### Parameters

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

Definition at line 4037 of file specfun.h.

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

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

See also

[ellint\\_el2](#) for details.

Definition at line 4006 of file specfun.h.

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

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

See also

[ellint\\_el2](#) for details.

Definition at line 4016 of file specfun.h.

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

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

The Bulirsch elliptic integral of the third kind is defined by

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

#### Parameters

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

Definition at line 4084 of file specfun.h.

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

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

See also

[ellint\\_el3](#) for details.

Definition at line 4053 of file specfun.h.

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

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

See also

[ellint\\_el3](#) for details.

Definition at line 4064 of file specfun.h.

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

#### Parameters

<code>__x</code>	The first argument.
<code>__y</code>	The second argument.

Definition at line 2970 of file specfun.h.

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

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

See also

[ellint\\_rc](#) for details.

Definition at line 2936 of file specfun.h.

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

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

See also

[ellint\\_rc](#) for details.

Definition at line 2945 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

#### Parameters

<code>__x</code>	The first of two symmetric arguments.
<code>__y</code>	The second of two symmetric arguments.
<code>__z</code>	The third argument.



Definition at line 3069 of file specfun.h.

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

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

See also

[ellint\\_rd](#) for details.

Definition at line 3033 of file specfun.h.

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

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

See also

[ellint\\_rd](#) for details.

Definition at line 3042 of file specfun.h.

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

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

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

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

#### Parameters

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.

Definition at line 2922 of file specfun.h.

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

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

See also

[ellint\\_rf](#) for details.

Definition at line 2893 of file `specfun.h`.

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

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

See also

[ellint\\_rf](#) for details.

Definition at line 2903 of file `specfun.h`.

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

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

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

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

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

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.

Definition at line 3160 of file specfun.h.

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

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

See also

[ellint\\_rg](#) for details.

Definition at line 3125 of file specfun.h.

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

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

See also

[ellint\\_rg](#) for details.

Definition at line 3134 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

#### Parameters

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.
<code>__p</code>	The fourth argument.

Definition at line 3019 of file specfun.h.

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

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

See also

[ellint\\_rj](#) for details.

Definition at line 2984 of file specfun.h.

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

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

See also

[ellint\\_rj](#) for details.

Definition at line 2993 of file specfun.h.

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

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

The elliptic nome function is defined by

$$q(k) =$$

Parameters

$\_k$	The modulus $-1 \leq k \leq +1$
-------	---------------------------------

Definition at line 4892 of file specfun.h.

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

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

See also

[ellnome](#) for details.

Definition at line 4867 of file specfun.h.

7.3.3.120 `long double __gnu_cxx::ellnome1 ( long double __k ) [inline]`

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

See also

[ellnome](#) for details.

Definition at line 4877 of file `specfun.h`.

7.3.3.121 `template<typename _Tp> __gnu_cxx::_promote_num_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

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>↵ _n</code>	The integral order
<code>↵ _x</code>	The real argument

Definition at line 3467 of file `specfun.h`.

7.3.3.122 `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 3436 of file `specfun.h`.

7.3.3.123 `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 3446 of file `specfun.h`.

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

Definition at line 3572 of file `specfun.h`.

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

Definition at line 3560 of file `specfun.h`.

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

Definition at line 3564 of file `specfun.h`.

7.3.3.127 `template<typename _Tp> __gnu_cxx::_promote_num_t<_Tp> __gnu_cxx::fresnel_c ( _Tp __x ) [inline]`

Return the Fresnel cosine integral of argument  $x$ .

The Fresnel cosine integral is defined by

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

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 3379 of file `specfun.h`.

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

Definition at line 3360 of file `specfun.h`.

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

Definition at line 3364 of file specfun.h.

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

Return the Fresnel sine integral of argument  $x$ .

The Fresnel sine integral is defined by

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

Parameters

$\longleftrightarrow$	The argument
<code>__x</code>	

Definition at line 3351 of file specfun.h.

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

Definition at line 3332 of file specfun.h.

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

Definition at line 3336 of file specfun.h.

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

Definition at line 2773 of file specfun.h.

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

Definition at line 2761 of file specfun.h.

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

Definition at line 2765 of file specfun.h.

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

Definition at line 2752 of file specfun.h.

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

Definition at line 2740 of file specfun.h.

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

Definition at line 2744 of file specfun.h.

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

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

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

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

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

#### Template Parameters

<code>_Talpha</code>	The real type of the order
<code>_Tp</code>	The real type of the argument

#### Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__x</code>	The real argument

Definition at line 2178 of file specfun.h.

7.3.3.140 `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 2145 of file specfun.h.

7.3.3.141 `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 2156 of file specfun.h.

7.3.3.142 `template<typename _Tk, typename _Tphi> __gnu_cxx::_promote_num_t<_Tk, _Tphi> __gnu_cxx::heuman_lambda ( _Tk __k, _Tphi __phi ) [inline]`

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

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

<code>_Tk</code>	the floating-point type of the modulus
<code>_Tphi</code>	the floating-point type of the angular limit argument

Parameters

<code>__k</code>	The modulus
<code>__phi</code>	The angle

Definition at line 3860 of file specfun.h.

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

Definition at line 3834 of file specfun.h.

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

Definition at line 3838 of file specfun.h.

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

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

The the Hurwitz zeta function is defined by

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

#### Parameters

<code>__s</code>	The argument
<code>__a</code>	The parameter

Definition at line 3201 of file specfun.h.

7.3.3.146 `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 3215 of file specfun.h.

7.3.3.147 `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 3175 of file specfun.h.

7.3.3.148 `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 3185 of file `specfun.h`.

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

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

The hypergeometric function is defined by

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

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

Parameters

<code>__a</code>	The first numeratorial parameter
<code>__b</code>	The second numeratorial parameter
<code>__c</code>	The denominatorial parameter
<code>__x</code>	The argument

Definition at line 1427 of file `specfun.h`.

7.3.3.150 `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 1394 of file `specfun.h`.

7.3.3.151 `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 1405 of file `specfun.h`.

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

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

The regularized incomplete beta function is defined by

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

where

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

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

#### Parameters

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 3292 of file `specfun.h`.

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

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

The regularized complementary incomplete beta function is defined by

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

## Parameters

$\leftrightarrow$ _a	The parameter
$\leftrightarrow$ _b	The parameter
$\leftrightarrow$ _x	The argument

Definition at line 3323 of file specfun.h.

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

Definition at line 3301 of file specfun.h.

References `__gnu_cxx::ibetacf()`.

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

Definition at line 3305 of file specfun.h.

References `__gnu_cxx::ibetal()`.

7.3.3.156 `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 3258 of file specfun.h.

Referenced by `__gnu_cxx::ibetacf()`.

7.3.3.157 `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 3268 of file specfun.h.

Referenced by `__gnu_cxx::ibetacl()`.

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

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

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

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

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

## Template Parameters

<code>_Talpha</code>	The real type of the order $\alpha$
<code>_Tbeta</code>	The real type of the order $\beta$
<code>_Tp</code>	The real type of the argument

## Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__beta</code>	The real order
<code>__x</code>	The real argument

Definition at line 2130 of file `specfun.h`.

References `std::__detail::__beta()`.

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

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

The Jacobi elliptic `cn` integral is defined by

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

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

## Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

## Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1839 of file `specfun.h`.

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

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

See also

[jacobi\\_cn](#) for details.

Definition at line 1803 of file specfun.h.

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

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

See also

[jacobi\\_cn](#) for details.

Definition at line 1816 of file specfun.h.

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

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

The Jacobi elliptic  $dn$  integral is defined by

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

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

#### Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

#### Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1891 of file specfun.h.

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

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

See also

[jacobi\\_dn](#) for details.

Definition at line 1855 of file specfun.h.

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

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

See also

[jacobi\\_dn](#) for details.

Definition at line 1868 of file specfun.h.

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

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

The Jacobi elliptic `sn` integral is defined by

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

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

#### Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

#### Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1787 of file specfun.h.

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

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



See also

[jacobi\\_sn](#) for details.

Definition at line 1751 of file specfun.h.

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

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

See also

[jacobi\\_sn](#) for details.

Definition at line 1764 of file specfun.h.

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

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

The Jacobi zeta function is defined by

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

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

#### Template Parameters

<code>_Tk</code>	the real type of the modulus
<code>_Tphi</code>	the real type of the angle limit

#### Parameters

<code>__k</code>	The modulus
<code>__phi</code>	The angle

Definition at line 3825 of file specfun.h.

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

Definition at line 3800 of file specfun.h.

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

Definition at line 3804 of file specfun.h.

7.3.3.171 `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 2086 of file specfun.h.

References `std::__detail::__beta()`.

7.3.3.172 `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 2097 of file specfun.h.

References `std::__detail::__beta()`.

7.3.3.173 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::lbincoef ( unsigned int __n, unsigned int __k )` `[inline]`

Definition at line 3677 of file specfun.h.

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

Definition at line 3665 of file specfun.h.

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

Definition at line 3669 of file specfun.h.

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

Definition at line 3635 of file `specfun.h`.

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

Definition at line 3623 of file `specfun.h`.

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

Definition at line 3627 of file `specfun.h`.

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

Definition at line 3749 of file `specfun.h`.

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

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

See also

[legendre\\_q](#) for details.

Definition at line 3731 of file `specfun.h`.

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

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

See also

[legendre\\_q](#) for details.

Definition at line 3741 of file `specfun.h`.

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

Definition at line 3614 of file `specfun.h`.

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

Definition at line 3602 of file specfun.h.

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

Definition at line 3606 of file specfun.h.

7.3.3.185 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::logint ( _Tp __x ) [inline]`

Return the logarithmic integral of argument  $x$ .

The logarithmic integral is defined by

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

Parameters

<code>__x</code>	The real upper integration limit
------------------	----------------------------------

Definition at line 1566 of file specfun.h.

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

Return the logarithmic integral of argument  $x$ .

See also

[logint](#) for details.

Definition at line 1542 of file specfun.h.

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

Return the logarithmic integral of argument  $x$ .

See also

[logint](#) for details.

Definition at line 1551 of file specfun.h.

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

Definition at line 3509 of file specfun.h.

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

Definition at line 3497 of file specfun.h.

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

Definition at line 3501 of file specfun.h.

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

Definition at line 3488 of file specfun.h.

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

Definition at line 3476 of file specfun.h.

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

Definition at line 3480 of file specfun.h.

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

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

The Owens T function is defined by

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

#### Parameters

$\leftrightarrow$ __h	The shape factor
$\leftrightarrow$ __a	The integration limit

Definition at line 5103 of file specfun.h.

7.3.3.195 `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 5075 of file specfun.h.

7.3.3.196 `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 5085 of file specfun.h.

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

Definition at line 3770 of file specfun.h.

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

Definition at line 3758 of file specfun.h.

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

Definition at line 3762 of file specfun.h.

7.3.3.200 `template<typename _Tp, typename _Tn> __gnu_cxx::_promote_num_t<_Tp, _Tn> __gnu_cxx::pochhammer_l ( _Tp __a, _Tn __n ) [inline]`

Definition at line 3551 of file specfun.h.

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

Definition at line 3539 of file specfun.h.

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

Definition at line 3543 of file specfun.h.

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

Definition at line 3530 of file specfun.h.

7.3.3.204 `float __gnu_cxx::pochhammer_uf ( float __a, float __n ) [inline]`

Definition at line 3518 of file specfun.h.

7.3.3.205 `long double __gnu_cxx::pochhammer_ul ( long double __a, long double __n ) [inline]`

Definition at line 3522 of file specfun.h.

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

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

The polylogarithm function is defined by

#### Parameters

$\leftarrow$ _s	
$\leftarrow$ _w	

Definition at line 4415 of file specfun.h.

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

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

The polylogarithm function is defined by

#### Parameters

$\_s$	
$\_w$	

Definition at line 4455 of file specfun.h.

**7.3.3.208** `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 4388 of file specfun.h.

**7.3.3.209** `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 4428 of file specfun.h.

**7.3.3.210** `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 4398 of file specfun.h.



7.3.3.211 `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 4438 of file `specfun.h`.

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

Return the psi or digamma function of argument  $x$ .

The the psi or digamma function is defined by

$$\psi(x) =$$

Parameters

$\leftarrow$ __x	The parameter
---------------------	---------------

Definition at line 3243 of file `specfun.h`.

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

Definition at line 3224 of file `specfun.h`.

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

Definition at line 3228 of file `specfun.h`.

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

Definition at line 3791 of file `specfun.h`.

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

Definition at line 3779 of file `specfun.h`.

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

Definition at line 3783 of file `specfun.h`.

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

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

The radial polynomials are defined by

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

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

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

#### Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The non-negative azimuthal order
<code>__rho</code>	The radial argument

Definition at line 2288 of file `specfun.h`.

7.3.3.219 `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  $m \leq n$ , and `float` radial argument  $\rho$ .

See also

[radpoly](#) for details.

Definition at line 2249 of file `specfun.h`.

References `std::__detail::__poly_radial_jacobi()`.

7.3.3.220 `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  $m \leq n$ , and long double radial argument  $\rho$ .

See also

[radpoly](#) for details.

Definition at line 2260 of file specfun.h.

References `std::__detail::__poly_radial_jacobi()`.

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

Definition at line 1528 of file specfun.h.

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

Definition at line 1501 of file specfun.h.

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

Definition at line 1486 of file specfun.h.

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

Definition at line 1493 of file specfun.h.

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

Definition at line 1513 of file specfun.h.

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

Definition at line 1520 of file specfun.h.

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

Definition at line 2330 of file specfun.h.

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

Definition at line 2309 of file specfun.h.

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

Definition at line 2297 of file specfun.h.

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

Definition at line 2301 of file specfun.h.

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

Definition at line 2318 of file specfun.h.

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

Definition at line 2322 of file specfun.h.

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

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

The hyperbolic sine integral is defined by

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

#### Template Parameters

<code>_Tp</code>	The type of the real argument
------------------	-------------------------------

#### Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 1686 of file specfun.h.

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

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

See also

[sinhint](#) for details.

Definition at line 1659 of file `specfun.h`.

7.3.3.235 `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 1669 of file `specfun.h`.

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

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

The sine integral is defined by

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

Parameters

<code>__x</code>	The real upper integration limit
------------------	----------------------------------

Definition at line 1605 of file `specfun.h`.

7.3.3.237 `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 1580 of file `specfun.h`.

7.3.3.238 `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 1590 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The integral order $n \geq 0$
<code>__x</code>	The real argument $x \geq 0$

#### Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 2568 of file `specfun.h`.

7.3.3.240 `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  $x \geq 0$ .

See also

[sph\\_bessel\\_i](#) for details.

Definition at line 2529 of file `specfun.h`.

7.3.3.241 `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  $x \geq 0$ .

See also

[sph\\_bessel\\_i](#) for details.

Definition at line 2544 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

#### Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

#### Parameters

<code>__n</code>	The integral order $n \geq 0$
<code>__x</code>	The real argument $x \geq 0$

#### Exceptions

<code>std::domain_error</code>	if <code>__x &lt; 0</code> .
--------------------------------	------------------------------

Definition at line 2625 of file `specfun.h`.

7.3.3.243 `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  $x \geq 0$ .

See also

[sph\\_bessel\\_k](#) for more details.

Definition at line 2586 of file `specfun.h`.

7.3.3.244 `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  $x \geq 0$ .

See also

[sph\\_bessel\\_k](#) for more details.

Definition at line 2601 of file `specfun.h`.

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

Return the spherical Hankel function of the first kind  $h_n^{(1)}(x)$  of nonnegative order  $n$  and real argument  $x \geq 0$ .

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

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

#### Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

#### Parameters

<code>__n</code>	The non-negative order
<code>__z</code>	The real argument

Definition at line 2471 of file `specfun.h`.

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

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

The spherical Hankel function of the first kind is defined by

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

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



## Parameters

$\leftrightarrow$ __n	The integral order $\geq 0$
$\leftrightarrow$ __x	The complex argument

Definition at line 4274 of file specfun.h.

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

Return the spherical Hankel function of the first kind  $h_n^{(1)}(x)$  of nonnegative order  $n$  and `float` argument  $x \geq 0$ .

## See also

[sph\\_hankel\\_1](#) for details.

Definition at line 2443 of file specfun.h.

7.3.3.248 `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  $h_n^{(1)}(x)$  of non-negative integral  $n$  and `std::complex<float>` argument  $x$ .

## See also

[sph\\_hankel\\_1](#) for more details.

Definition at line 4242 of file specfun.h.

7.3.3.249 `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  $h_n^{(1)}(x)$  of nonnegative order  $n$  and `long double` argument  $x \geq 0$ .

## See also

[sph\\_hankel\\_1](#) for details.

Definition at line 2453 of file specfun.h.

```
7.3.3.250 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  $h_n^{(1)}(x)$  of non-negative integral  $n$  and `std::complex<long double>` argument  $x$ .

See also

[sph\\_hankel\\_1](#) for more details.

Definition at line 4253 of file `specfun.h`.

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

Return the spherical Hankel function of the second kind  $h_n^{(2)}(x)$  of nonnegative order  $n$  and real argument  $x \geq 0$ .

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

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

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative order
<code>__z</code>	The real argument

Definition at line 2514 of file `specfun.h`.

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

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

The spherical Hankel function of the second kind is defined by

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

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

## Parameters

$\leftrightarrow$ __n	The integral order $\geq 0$
$\leftrightarrow$ __x	The complex argument

Definition at line 4322 of file specfun.h.

7.3.3.253 `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  $x \geq 0$ .

## See also

[sph\\_hankel\\_2](#) for details.

Definition at line 2486 of file specfun.h.

7.3.3.254 `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  $h_n^{(2)}(x)$  of non-negative integral  $n$  and `std::complex<float>` argument  $x$ .

## See also

[sph\\_hankel\\_2](#) for more details.

Definition at line 4290 of file specfun.h.

7.3.3.255 `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  $h_n^{(2)}(x)$  of nonnegative order  $n$  and `long double` argument  $x \geq 0$ .

## See also

[sph\\_hankel\\_2](#) for details.

Definition at line 2496 of file specfun.h.

7.3.3.256 `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  $h_n^{(2)}(x)$  of non-negative integral  $n$  and `std::complex<long double>` argument  $x$ .

See also

[sph\\_hankel\\_2](#) for more details.

Definition at line 4301 of file `specfun.h`.

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

Return the complex spherical harmonic function of degree  $l$ , order  $m$ , and real zenith angle  $\theta$ , and azimuth angle  $\phi$ .

The spherical harmonic function is defined by:

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

Parameters

<code>__l</code>	The order
<code>__m</code>	The degree
<code>__theta</code>	The zenith angle in radians
<code>__phi</code>	The azimuth angle in radians

Definition at line 4373 of file `specfun.h`.

7.3.3.258 `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  $\theta$ , and azimuth angle  $\phi$ .

See also

[sph\\_harmonic](#) for details.

Definition at line 4337 of file `specfun.h`.

7.3.3.259 `std::complex<long double> __gnu_cxx::sph_harmonic( 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  $\theta$ , and azimuth angle  $\phi$ .

See also

[sph\\_harmonic](#) for details.

Definition at line 4349 of file `specfun.h`.

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

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

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 4723 of file `specfun.h`.

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

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

See also

[theta\\_1](#) for details.

Definition at line 4695 of file `specfun.h`.

7.3.3.262 `long double __gnu_cxx::theta_1l( long double __nu, long double __x ) [inline]`

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

See also

[theta\\_1](#) for details.

Definition at line 4705 of file `specfun.h`.

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

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

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

#### Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 4766 of file `specfun.h`.

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

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

See also

[theta\\_2](#) for details.

Definition at line 4738 of file `specfun.h`.

7.3.3.265 `long double __gnu_cxx::theta_2l ( long double __nu, long double __x ) [inline]`

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

See also

[theta\\_2](#) for details.

Definition at line 4748 of file `specfun.h`.

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

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

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

## Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 4809 of file specfun.h.

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

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

See also

[theta\\_3](#) for details.

Definition at line 4781 of file specfun.h.

7.3.3.268 `long double __gnu_cxx::theta_3l( long double __nu, long double __x ) [inline]`

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

See also

[theta\\_3](#) for details.

Definition at line 4791 of file specfun.h.

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

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

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j + 1/2)^2}{x}\right)$$

## Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 4852 of file specfun.h.

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

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

See also

[theta\\_4](#) for details.

Definition at line 4824 of file specfun.h.

7.3.3.271 `long double __gnu_cxx::theta_4l ( long double __nu, long double __x ) [inline]`

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

See also

[theta\\_4](#) for details.

Definition at line 4834 of file specfun.h.

7.3.3.272 `template<typename _Tpk, typename _Tp> __gnu_cxx::_promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_c ( _Tpk __k, _Tp __x ) [inline]`

Return the Neville theta-c function  $\theta_c(k, x)$  of modulus  $k$  and argument  $x$ .

The Neville theta-c function is defined by

#### Parameters

$\leftrightarrow$ __k	The modulus $-1 \leq k \leq +1$
$\leftrightarrow$ __x	The argument

Definition at line 4976 of file specfun.h.

7.3.3.273 `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 4949 of file specfun.h.

7.3.3.274 `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 4959 of file specfun.h.

7.3.3.275 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_num_t<_Tp, _Tp> __gnu_cxx::theta_d ( _Tp __k, _Tp __x ) [inline]`

Return the Neville theta-d function  $\theta_d(k, x)$  of modulus  $k$  and argument  $x$ .

The Neville theta-d function is defined by

$$\theta_d(k, x) =$$

Parameters

$\hookleftarrow$ _k	The modulus $-1 \leq k \leq +1$
$\hookleftarrow$ _x	The argument

Definition at line 5018 of file specfun.h.

7.3.3.276 `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 4991 of file specfun.h.

7.3.3.277 `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 5001 of file `specfun.h`.

7.3.3.278 `template<typename _Tpk, typename _Tp> __gnu_cxx::_promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_n ( _Tpk __k, _Tp __x )` `[inline]`

Return the Neville theta-n function  $\theta_n(k, x)$  of modulus `k` and argument `x`.

The Neville theta-n function is defined by

$$\theta_n(k, x) =$$

Parameters

$\leftrightarrow$ <code>__k</code>	The modulus $-1 \leq k \leq +1$
$\leftrightarrow$ <code>__x</code>	The argument

Definition at line 5060 of file `specfun.h`.

7.3.3.279 `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 5033 of file `specfun.h`.

7.3.3.280 `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 5043 of file `specfun.h`.

```
7.3.3.281  template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_s ( _Tpk __k,
    _Tp __x )  [inline]
```

Return the Neville theta-s function  $\theta_s(k, x)$  of modulus  $k$  and argument  $x$ .

The Neville theta-s function is defined by

#### Parameters

$\leftrightarrow$ _k	The modulus $-1 \leq k \leq +1$
$\leftrightarrow$ _x	The argument

Definition at line 4934 of file specfun.h.

```
7.3.3.282  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 4907 of file specfun.h.

```
7.3.3.283  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 4917 of file specfun.h.

```
7.3.3.284  template<typename _Trho, typename _Tphi> __gnu_cxx::__promote_num_t<_Trho, _Tphi> __gnu_cxx::zernike (
    unsigned int __n, int __m, _Trho __rho, _Tphi __phi )  [inline]
```

Return the Zernicke polynomial  $Z_n^m(\rho, \phi)$  for non-negative degree  $n$ , signed order  $m$ , and real radial argument  $\rho$  and azimuthal angle  $\phi$ .

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  $m \leq n$  and where  $R_n^m(\rho)$  is the radial polynomial (

See also

[radpoly](#)).

## Template Parameters

<code>_Trho</code>	The real type of the radial coordinate
<code>_Tphi</code>	The real type of the azimuthal angle

## Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The (signed) azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 2233 of file `specfun.h`.

```
7.3.3.285 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  $\rho$  and azimuthal angle  $\phi$ .

See also

[zernike](#) for details.

Definition at line 2194 of file `specfun.h`.

```
7.3.3.286 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  $\rho$  and azimuthal angle  $\phi$ .

See also

[zernike](#) for details.

Definition at line 2205 of file `specfun.h`.

## Chapter 8

# Namespace Documentation

### 8.1 `__gnu_cxx` Namespace Reference

#### Enumerations

- enum { `_GLIBCXX_JACOBI_SN`, `_GLIBCXX_JACOBI_CN`, `_GLIBCXX_JACOBI_DN` }

#### Functions

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

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > chebyshev\_v (unsigned int __n, _Tp __x)`
- `float chebyshev\_vf (unsigned int __n, float __x)`
- `long double chebyshev\_vl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > chebyshev\_w (unsigned int __n, _Tp __x)`
- `float chebyshev\_wf (unsigned int __n, float __x)`
- `long double chebyshev\_wl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > clausen\_c (unsigned int __m, _Tp __w)`
- `float clausen\_cf (unsigned int __m, float __w)`
- `long double clausen\_cl (unsigned int __m, long double __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > clausen\_s (unsigned int __m, _Tp __w)`
- `float clausen\_sf (unsigned int __m, float __w)`
- `long double clausen\_sl (unsigned int __m, long double __w)`
- `float clausenf (unsigned int __m, float __w)`
- `std::complex< float > clausenf (unsigned int __m, std::complex< float > __w)`
- `long double clausenl (unsigned int __m, long double __w)`
- `std::complex< long double > clausenl (unsigned int __m, std::complex< long double > __w)`
- `template<typename _Tk >`  
`__gnu_cxx::__promote_num_t< _Tk > comp\_ellint\_d (_Tk __k)`
- `float comp\_ellint\_df (float __k)`
- `long double comp\_ellint\_dl (long double __k)`
- `float comp\_ellint\_rf (float __x, float __y)`
- `long double comp\_ellint\_rl (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_num_t< _Tx, _Ty > comp\_ellint\_rf (_Tx __x, _Ty __y)`
- `float comp\_ellint\_rg (float __x, float __y)`
- `long double comp\_ellint\_rl (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_num_t< _Tx, _Ty > comp\_ellint\_rg (_Tx __x, _Ty __y)`
- `template<typename _Tpa, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type conf\_hyperg (_Tpa __a, _Tpc __c, _Tp __x)`
- `template<typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpc, _Tp >::__type conf\_hyperg\_lim (_Tpc __c, _Tp __x)`
- `float conf\_hyperg\_limf (float __c, float __x)`
- `long double conf\_hyperg\_liml (long double __c, long double __x)`
- `float conf\_hypergf (float __a, float __c, float __x)`
- `long double conf\_hypergl (long double __a, long double __c, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > coshint (_Tp __x)`
- `float coshintf (float __x)`
- `long double coshintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > cosint (_Tp __x)`
- `float cosintf (float __x)`

- long double [cosintl](#) (long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_num\_t< \_Tpnu, \_Tp > > [cyl\\_hankel\\_1](#) (\_Tpnu \_\_nu, \_Tp \_\_z)
- template<typename \_Tpnu, typename \_Tp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_num\_t< \_Tpnu, \_Tp > > [cyl\\_hankel\\_1](#) (std::complex< \_Tpnu > \_\_nu, std::complex< \_Tp > \_\_x)
- std::complex< float > [cyl\\_hankel\\_1f](#) (float \_\_nu, float \_\_z)
- std::complex< float > [cyl\\_hankel\\_1f](#) (std::complex< float > \_\_nu, std::complex< float > \_\_x)
- std::complex< long double > [cyl\\_hankel\\_1l](#) (long double \_\_nu, long double \_\_z)
- std::complex< long double > [cyl\\_hankel\\_1l](#) (std::complex< long double > \_\_nu, std::complex< long double > \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_num\_t< \_Tpnu, \_Tp > > [cyl\\_hankel\\_2](#) (\_Tpnu \_\_nu, \_Tp \_\_z)
- template<typename \_Tpnu, typename \_Tp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_num\_t< \_Tpnu, \_Tp > > [cyl\\_hankel\\_2](#) (std::complex< \_Tpnu > \_\_nu, std::complex< \_Tp > \_\_x)
- std::complex< float > [cyl\\_hankel\\_2f](#) (float \_\_nu, float \_\_z)
- std::complex< float > [cyl\\_hankel\\_2f](#) (std::complex< float > \_\_nu, std::complex< float > \_\_x)
- std::complex< long double > [cyl\\_hankel\\_2l](#) (long double \_\_nu, long double \_\_z)
- std::complex< long double > [cyl\\_hankel\\_2l](#) (std::complex< long double > \_\_nu, std::complex< long double > \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [dawson](#) (\_Tp \_\_x)
- float [dawsonf](#) (float \_\_x)
- long double [dawsonl](#) (long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [digamma](#) (\_Tp \_\_z)
- float [digammaf](#) (float \_\_z)
- long double [digammal](#) (long double \_\_z)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [dilog](#) (\_Tp \_\_x)
- float [dilogf](#) (float \_\_x)
- long double [dilogl](#) (long double \_\_x)
- template<typename \_Tp >  
\_Tp [dirichlet\\_beta](#) (\_Tp \_\_s)
- float [dirichlet\\_betaf](#) (float \_\_s)
- long double [dirichlet\\_betalf](#) (long double \_\_s)
- template<typename \_Tp >  
\_Tp [dirichlet\\_eta](#) (\_Tp \_\_s)
- float [dirichlet\\_etaf](#) (float \_\_s)
- long double [dirichlet\\_etalf](#) (long double \_\_s)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [double\\_factorial](#) (int \_\_n)
- float [double\\_factorialf](#) (int \_\_n)
- long double [double\\_factoriall](#) (int \_\_n)
- template<typename \_Tk, typename \_Tp, typename \_Ta, typename \_Tb >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tk, \_Tp, \_Ta, \_Tb > [ellint\\_cel](#) (\_Tk \_\_k\_c, \_Tp \_\_p, \_Ta \_\_a, \_Tb \_\_b)
- float [ellint\\_celf](#) (float \_\_k\_c, float \_\_p, float \_\_a, float \_\_b)
- long double [ellint\\_cell](#) (long double \_\_k\_c, long double \_\_p, long double \_\_a, long double \_\_b)
- template<typename \_Tk, typename \_Tphi >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tk, \_Tphi > [ellint\\_d](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [ellint\\_dfl](#) (float \_\_k, float \_\_phi)

- long double [ellint\\_d1](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Tk >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Tk > [ellint\\_el1](#) (\_Tp \_\_x, \_Tk \_\_k\_c)
- float [ellint\\_el1f](#) (float \_\_x, float \_\_k\_c)
- long double [ellint\\_el1l](#) (long double \_\_x, long double \_\_k\_c)
- template<typename \_Tp, typename \_Tk, typename \_Ta, typename \_Tb >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Tk, \_Ta, \_Tb > [ellint\\_el2](#) (\_Tp \_\_x, \_Tk \_\_k\_c, \_Ta \_\_a, \_Tb \_\_b)
- float [ellint\\_el2f](#) (float \_\_x, float \_\_k\_c, float \_\_a, float \_\_b)
- long double [ellint\\_el2l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_a, long double \_\_b)
- template<typename \_Tx, typename \_Tk, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tx, \_Tk, \_Tp > [ellint\\_el3](#) (\_Tx \_\_x, \_Tk \_\_k\_c, \_Tp \_\_p)
- float [ellint\\_el3f](#) (float \_\_x, float \_\_k\_c, float \_\_p)
- long double [ellint\\_el3l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_p)
- template<typename \_Tp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up > [ellint\\_rc](#) (\_Tp \_\_x, \_Up \_\_y)
- float [ellint\\_rcf](#) (float \_\_x, float \_\_y)
- long double [ellint\\_rcl](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up, \_Vp > [ellint\\_rd](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rdf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rdl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up, \_Vp > [ellint\\_rf](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rff](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rfl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up, \_Vp > [ellint\\_rg](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rgf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rgl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp, typename \_Wp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up, \_Vp, \_Wp > [ellint\\_rj](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z, \_Wp \_\_p)
- float [ellint\\_rjf](#) (float \_\_x, float \_\_y, float \_\_z, float \_\_p)
- long double [ellint\\_rjl](#) (long double \_\_x, long double \_\_y, long double \_\_z, long double \_\_p)
- template<typename \_Tp >  
\_Tp [ellnome](#) (\_Tp \_\_k)
- float [ellnomef](#) (float \_\_k)
- long double [ellnomel](#) (long double \_\_k)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [expint](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [expintf](#) (unsigned int \_\_n, float \_\_x)
- long double [expintl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [factorial](#) (unsigned int \_\_n)
- float [factorialf](#) (unsigned int \_\_n)
- long double [factoriall](#) (unsigned int \_\_n)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [fresnel\\_c](#) (\_Tp \_\_x)
- float [fresnel\\_cf](#) (float \_\_x)
- long double [fresnel\\_cl](#) (long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp > [fresnel\\_s](#) (\_Tp \_\_x)



- float [fresnel\\_sf](#) (float \_\_x)
- long double [fresnel\\_sl](#) (long double \_\_x)
- template<typename \_Tn, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tn, \_Tp > [gamma\\_l](#) (\_Tn \_\_n, \_Tp \_\_x)
- float [gamma\\_lf](#) (float \_\_n, float \_\_x)
- long double [gamma\\_ll](#) (long double \_\_n, long double \_\_x)
- template<typename \_Tn, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tn, \_Tp > [gamma\\_u](#) (\_Tn \_\_n, \_Tp \_\_x)
- float [gamma\\_uf](#) (float \_\_n, float \_\_x)
- long double [gamma\\_ul](#) (long double \_\_n, long double \_\_x)
- template<typename \_Talpha, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Talpha, \_Tp > [gegenbauer](#) (unsigned int \_\_n, \_Talpha \_\_alpha, \_Tp \_\_x)
- float [gegenbauerf](#) (unsigned int \_\_n, float \_\_alpha, float \_\_x)
- long double [gegenbauerl](#) (unsigned int \_\_n, long double \_\_alpha, long double \_\_x)
- template<typename \_Tk, typename \_Tphi >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tk, \_Tphi > [heuman\\_lambda](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [heuman\\_lambdaf](#) (float \_\_k, float \_\_phi)
- long double [heuman\\_lambdal](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Tp, \_Up > [hurwitz\\_zeta](#) (\_Tp \_\_s, \_Up \_\_a)
- template<typename \_Tp, typename \_Up >  
std::complex< \_Tp > [hurwitz\\_zeta](#) (\_Tp \_\_s, std::complex< \_Up > \_\_a)
- float [hurwitz\\_zetaf](#) (float \_\_s, float \_\_a)
- long double [hurwitz\\_zetal](#) (long double \_\_s, long double \_\_a)
- template<typename \_Tpa, typename \_Tpb, typename \_Tpc, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_4< \_Tpa, \_Tpb, \_Tpc, \_Tp >::\_\_type [hyperg](#) (\_Tpa \_\_a, \_Tpb \_\_b, \_Tpc \_\_c, \_Tp \_\_x)
- float [hypergf](#) (float \_\_a, float \_\_b, float \_\_c, float \_\_x)
- long double [hypergl](#) (long double \_\_a, long double \_\_b, long double \_\_c, long double \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Ta, \_Tb, \_Tp > [ibeta](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Ta, \_Tb, \_Tp > [ibetac](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- float [ibetacf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [ibetacL](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- float [ibetaf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [ibetal](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- template<typename \_Talpha, typename \_Tbeta, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Talpha, \_Tbeta, \_Tp > [jacobi](#) (unsigned \_\_n, \_Talpha \_\_alpha, \_Tbeta \_\_beta, \_Tp \_\_x)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Kp, \_Up > [jacobi\\_cn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_cnf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_cnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Kp, \_Up > [jacobi\\_dn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_dnf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_dnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Kp, \_Up > [jacobi\\_sn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_snf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_snl](#) (long double \_\_k, long double \_\_u)

- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > jacobi\_zeta (_Tk __k, _Tphi __phi)`
- `float jacobi\_zetaf (float __k, float __phi)`
- `long double jacobi\_zetal (long double __k, long double __phi)`
- `float jacobi (unsigned __n, float __alpha, float __beta, float __x)`
- `long double jacobi (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > lbincoef (unsigned int __n, unsigned int __k)`
- `float lbincoef (unsigned int __n, unsigned int __k)`
- `long double lbincoef (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > ldouble\_factorial (int __n)`
- `float ldouble\_factorialf (int __n)`
- `long double ldouble\_factorial (int __n)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > legendre\_q (unsigned int __n, _Tp __x)`
- `float legendre\_qf (unsigned int __n, float __x)`
- `long double legendre\_q (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > lfactorial (unsigned int __n)`
- `float lfactorialf (unsigned int __n)`
- `long double lfactorial (unsigned int __n)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > logint (_Tp __x)`
- `float logintf (float __x)`
- `long double logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer\_l (_Tp __a, _Tn __n)`
- `float lpochhammer\_lf (float __a, float __n)`
- `long double lpochhammer\_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer\_u (_Tp __a, _Tn __n)`
- `float lpochhammer\_uf (float __a, float __n)`
- `long double lpochhammer\_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`  
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > owens\_t (_Tph __h, _Tpa __a)`
- `float owens\_tf (float __h, float __a)`
- `long double owens\_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_num_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 _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer\_l (_Tp __a, _Tn __n)`
- `float pochhammer\_lf (float __a, float __n)`
- `long double pochhammer\_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer\_u (_Tp __a, _Tn __n)`
- `float pochhammer\_uf (float __a, float __n)`
- `long double pochhammer\_ul (long double __a, long double __n)`
- `template<typename _Tp, typename _Wp >`  
`__gnu_cxx::__promote_num_t< _Tp, _Wp > polylog (_Tp __s, _Wp __w)`

- `template<typename _Tp, typename _Wp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > polylog (_Tp __s, std::complex< _Tp > __w)`
- `float polylogf (float __s, float __w)`
- `std::complex< float > polylogf (float __s, std::complex< float > __w)`
- `long double polylogl (long double __s, long double __w)`
- `std::complex< long double > polylogl (long double __s, std::complex< long double > __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > psi (_Tp __x)`
- `float psif (float __x)`
- `long double psil (long double __x)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_num_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_num_t< _Tp > radpoly (unsigned int __n, unsigned int __m, _Tp __rho)`
- `float radpolyf (unsigned int __n, unsigned int __m, float __rho)`
- `long double radpolyl (unsigned int __n, unsigned int __m, long double __rho)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinc (_Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinc_pi (_Tp __x)`
- `float sinc_pif (float __x)`
- `long double sinc_pil (long double __x)`
- `float sincf (float __x)`
- `long double sincl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinhc (_Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinhc_pi (_Tp __x)`
- `float sinhc_pif (float __x)`
- `long double sinhc_pil (long double __x)`
- `float sinhcf (float __x)`
- `long double sinhcl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinhnt (_Tp __x)`
- `float sinhntf (float __x)`
- `long double sinhntl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sinint (_Tp __x)`
- `float sinintf (float __x)`
- `long double sinintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sph_bessel_i (unsigned int __n, _Tp __x)`
- `float sph_bessel_if (unsigned int __n, float __x)`
- `long double sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > sph_bessel_k (unsigned int __n, _Tp __x)`
- `float sph_bessel_kf (unsigned int __n, float __x)`
- `long double sph_bessel_kl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)`

- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1` (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- `std::complex< float > sph_hankel_1f` (unsigned int \_\_n, float \_\_z)
- `std::complex< float > sph_hankel_1f` (unsigned int \_\_n, std::complex< float > \_\_x)
- `std::complex< long double > sph_hankel_1l` (unsigned int \_\_n, long double \_\_z)
- `std::complex< long double > sph_hankel_1l` (unsigned int \_\_n, std::complex< long double > \_\_x)
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2` (unsigned int \_\_n, \_Tp \_\_z)
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2` (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- `std::complex< float > sph_hankel_2f` (unsigned int \_\_n, float \_\_z)
- `std::complex< float > sph_hankel_2f` (unsigned int \_\_n, std::complex< float > \_\_x)
- `std::complex< long double > sph_hankel_2l` (unsigned int \_\_n, long double \_\_z)
- `std::complex< long double > sph_hankel_2l` (unsigned int \_\_n, std::complex< long double > \_\_x)
- `template<typename _Ttheta , typename _Tphi >`  
`std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > sph_harmonic` (unsigned int \_\_l, int \_\_m, \_Ttheta \_\_theta, \_Tphi \_\_phi)
- `std::complex< float > sph_harmonicf` (unsigned int \_\_l, int \_\_m, float \_\_theta, float \_\_phi)
- `std::complex< long double > sph_harmonicl` (unsigned int \_\_l, int \_\_m, long double \_\_theta, long double \_\_phi)
- `template<typename _Tpnu , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_1` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float theta_1f` (float \_\_nu, float \_\_x)
- `long double theta_1l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_2` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float theta_2f` (float \_\_nu, float \_\_x)
- `long double theta_2l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_3` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float theta_3f` (float \_\_nu, float \_\_x)
- `long double theta_3l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > theta_4` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float theta_4f` (float \_\_nu, float \_\_x)
- `long double theta_4l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tp_k , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > theta_c` (\_Tp\_k \_\_k, \_Tp \_\_x)
- `float theta_cf` (float \_\_k, float \_\_x)
- `long double theta_cl` (long double \_\_k, long double \_\_x)
- `template<typename _Tp_k , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > theta_d` (\_Tp\_k \_\_k, \_Tp \_\_x)
- `float theta_df` (float \_\_k, float \_\_x)
- `long double theta_dl` (long double \_\_k, long double \_\_x)
- `template<typename _Tp_k , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > theta_n` (\_Tp\_k \_\_k, \_Tp \_\_x)
- `float theta_nf` (float \_\_k, float \_\_x)
- `long double theta_nl` (long double \_\_k, long double \_\_x)
- `template<typename _Tp_k , typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > theta_s` (\_Tp\_k \_\_k, \_Tp \_\_x)
- `float theta_sf` (float \_\_k, float \_\_x)

- long double [theta\\_sl](#) (long double \_\_k, long double \_\_x)
- template<typename \_Trho, typename \_Tphi >  
\_\_gnu\_cxx::\_\_promote\_num\_t< \_Trho, \_Tphi > [zernike](#) (unsigned int \_\_n, int \_\_m, \_Trho \_\_rho, \_Tphi \_\_phi)
- float [zernikef](#) (unsigned int \_\_n, int \_\_m, float \_\_rho, float \_\_phi)
- long double [zernikel](#) (unsigned int \_\_n, int \_\_m, long double \_\_rho, long double \_\_phi)

## 8.2 std Namespace Reference

### Namespaces

- [\\_\\_detail](#)

### Functions

- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [assoc\\_laguerre](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_x)
- float [assoc\\_laguerref](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_x)
- long double [assoc\\_laguerrel](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [assoc\\_legendre](#) (unsigned int \_\_l, unsigned int \_\_m, \_Tp \_\_x)
- float [assoc\\_legendref](#) (unsigned int \_\_l, unsigned int \_\_m, float \_\_x)
- long double [assoc\\_legendrel](#) (unsigned int \_\_l, unsigned int \_\_m, long double \_\_x)
- template<typename \_Tpa, typename \_Tpb >  
\_\_gnu\_cxx::\_\_promote\_2< \_Tpa, \_Tpb >::\_\_type [beta](#) (\_Tpa \_\_a, \_Tpb \_\_b)
- float [betaf](#) (float \_\_a, float \_\_b)
- long double [betal](#) (long double \_\_a, long double \_\_b)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [comp\\_ellint\\_1](#) (\_Tp \_\_k)
- float [comp\\_ellint\\_1f](#) (float \_\_k)
- long double [comp\\_ellint\\_1l](#) (long double \_\_k)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [comp\\_ellint\\_2](#) (\_Tp \_\_k)
- float [comp\\_ellint\\_2f](#) (float \_\_k)
- long double [comp\\_ellint\\_2l](#) (long double \_\_k)
- template<typename \_Tp, typename \_Tpn >  
\_\_gnu\_cxx::\_\_promote\_2< \_Tp, \_Tpn >::\_\_type [comp\\_ellint\\_3](#) (\_Tp \_\_k, \_Tpn \_\_nu)
- float [comp\\_ellint\\_3f](#) (float \_\_k, float \_\_nu)  
*Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for float modulus k.*
- long double [comp\\_ellint\\_3l](#) (long double \_\_k, long double \_\_nu)  
*Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for long double modulus k.*
- template<typename \_Tpnu, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_2< \_Tpnu, \_Tp >::\_\_type [cyl\\_bessel\\_i](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [cyl\\_bessel\\_if](#) (float \_\_nu, float \_\_x)
- long double [cyl\\_bessel\\_il](#) (long double \_\_nu, long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_2< \_Tpnu, \_Tp >::\_\_type [cyl\\_bessel\\_j](#) (\_Tpnu \_\_nu, \_Tp \_\_x)
- float [cyl\\_bessel\\_jf](#) (float \_\_nu, float \_\_x)
- long double [cyl\\_bessel\\_jl](#) (long double \_\_nu, long double \_\_x)

- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl\_bessel\_k ( _Tpnu __nu, _Tp __x)`
- `float cyl\_bessel\_kf (float __nu, float __x)`
- `long double cyl\_bessel\_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl\_neumann ( _Tpnu __nu, _Tp __x)`
- `float cyl\_neumannf (float __nu, float __x)`
- `long double cyl\_neumannl (long double __nu, long double __x)`
- `template<typename _Tp, typename _Tpp >`  
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint\_1 ( _Tp __k, _Tpp __phi)`
- `float ellint\_1f (float __k, float __phi)`
- `long double ellint\_1l (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tpp >`  
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint\_2 ( _Tp __k, _Tpp __phi)`
- `float ellint\_2f (float __k, float __phi)`  
*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$  for float argument.*
- `long double ellint\_2l (long double __k, long double __phi)`  
*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$ .*
- `template<typename _Tp, typename _Tpn, typename _Tpp >`  
`__gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type ellint\_3 ( _Tp __k, _Tpn __nu, _Tpp __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- `float ellint\_3f (float __k, float __nu, float __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$  for float argument.*
- `long double ellint\_3l (long double __k, long double __nu, long double __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type expint ( _Tp __x)`
- `float expintf (float __x)`
- `long double expintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type hermite (unsigned int __n, _Tp __x)`
- `float hermitef (unsigned int __n, float __x)`
- `long double hermitel (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type laguerre (unsigned int __n, _Tp __x)`
- `float laguerref (unsigned int __n, float __x)`
- `long double laguerrel (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type legendre (unsigned int __l, _Tp __x)`
- `float legendref (unsigned int __l, float __x)`
- `long double legendrel (unsigned int __l, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type riemann\_zeta ( _Tp __s)`
- `float riemann\_zetaf (float __s)`
- `long double riemann\_zetal (long double __s)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type sph\_bessel (unsigned int __n, _Tp __x)`
- `float sph\_besself (unsigned int __n, float __x)`
- `long double sph\_bessell (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type sph\_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

- float [sph\\_legendref](#) (unsigned int \_\_l, unsigned int \_\_m, float \_\_theta)
- long double [sph\\_legendrel](#) (unsigned int \_\_l, unsigned int \_\_m, long double \_\_theta)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [sph\\_neumann](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [sph\\_neumannf](#) (unsigned int \_\_n, float \_\_x)
- long double [sph\\_neumannl](#) (unsigned int \_\_n, long double \_\_x)

## 8.3 std::\_\_detail Namespace Reference

### Classes

- struct [\\_Factorial\\_table](#)

### Enumerations

- enum { [SININT](#), [COSINT](#) }

### Functions

- template<typename \_Tp >  
void [\\_\\_airy](#) (\_Tp \_\_z, \_Tp &\_Ai, \_Tp &\_Bi, \_Tp &\_Aip, \_Tp &\_Bip)  
*Compute the Airy functions  $Ai(x)$  and  $Bi(x)$  and their first derivatives  $Ai'(x)$  and  $Bi'(x)$  respectively.*
- template<typename \_Tp >  
void [\\_\\_airy](#) (const std::complex< \_Tp > &\_\_z, \_Tp \_\_eps, std::complex< \_Tp > &\_Ai, std::complex< \_Tp > &\_Aip, std::complex< \_Tp > &\_Bi, std::complex< \_Tp > &\_Bip)  
*This function computes the Airy function  $Ai(z)$  and its first derivative in the complex z-plane.*
- template<typename \_Tp >  
std::complex< \_Tp > [\\_\\_airy\\_ai](#) (std::complex< \_Tp > \_\_z)  
*Return the complex Airy Ai function.*
- template<typename \_Tp >  
void [\\_\\_airy\\_arg](#) (std::complex< \_Tp > \_\_num2d3, std::complex< \_Tp > \_\_zeta, std::complex< \_Tp > &\_\_argp, std::complex< \_Tp > &\_\_argm)  
*Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.*
- template<typename \_Tp >  
void [\\_\\_airy\\_asymp\\_absarg\\_ge\\_pio3](#) (std::complex< \_Tp > \_\_z, std::complex< \_Tp > &\_Ai, std::complex< \_Tp > &\_Aip, int \_\_sign=-1)  
*This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|\arg(z)| < 2 * \pi/3$ . For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from  $\text{abs}(z)$ .*
- template<typename \_Tp >  
void [\\_\\_airy\\_asymp\\_absarg\\_lt\\_pio3](#) (std::complex< \_Tp > \_\_z, std::complex< \_Tp > &\_Ai, std::complex< \_Tp > &\_Aip)  
*This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|\arg(-z)| < \pi/3$ . For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from  $|z|$ .*
- template<typename \_Tp >  
void [\\_\\_airy\\_bessel\\_i](#) (const std::complex< \_Tp > &\_\_z, \_Tp \_\_eps, std::complex< \_Tp > &\_lp1d3, std::complex< \_Tp > &\_lm1d3, std::complex< \_Tp > &\_lp2d3, std::complex< \_Tp > &\_lm2d3)

- `template<typename _Tp >`  
`void __airy_bessel_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)`  
*Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.*
- `template<typename _Tp >`  
`std::complex< _Tp > __airy_bi (std::complex< _Tp > __z)`  
*Return the complex Airy Bi function.*
- `template<typename _Tp >`  
`void __airy_hyperg_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`  
*This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders  $\nu = + - 1/3$  and  $\nu + - 2/3$ . That is,  $A(z)/B(z)$ , Where  $A(z)$  and  $B(z)$  are cubic polynomials with real coefficients, approximates*

$$\frac{\Gamma(\nu + 1)}{(z/2)^n} I_\nu(z) = {}_0F_1(\nu + 1; z^2/4),$$
*where the function on the right is a confluent hypergeometric limit function. For  $|z| \leq 1/4$  and  $|\arg(z)| \leq \pi/2$ , the approximations are accurate to about 16 decimals.*
- `template<typename _Tp >`  
`_Tp __assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)`  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $m$ :  $L_n^m(x)$ .*
- `template<typename _Tp >`  
`_Tp __assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`  
*Return the associated Legendre function by recursion on  $l$  and downward recursion on  $m$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli (int __n)`  
*This returns Bernoulli number  $B_n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_2n (int __n)`  
*This returns Bernoulli number  $B_n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)`  
*This returns Bernoulli numbers from a table or by summation for larger values.*
- `template<typename _Tp >`  
`_Tp __beta (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp __beta_gamma (_Tp __a, _Tp __b)`  
*Return the beta function:  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp __beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __beta_lgamma (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$  using the log gamma functions.*
- `template<typename _Tp >`  
`_Tp __beta_product (_Tp __a, _Tp __b)`  
*Return the beta function  $B(x, y)$  using the product form.*
- `template<typename _Tp >`  
`_Tp __bincoef (unsigned int __n, unsigned int __k)`



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

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

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the binomial cumulative distribution function.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the complementary binomial cumulative distribution function.*
- `template<typename _Tp >`  
`_Tp __bose_einstein (_Tp __s, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`  
`_Tp __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_w (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`  
*This function returns the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals as a pair.*
- `template<typename _Tp >`  
`void __chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by continued fraction for positive argument.*
- `template<typename _Tp >`  
`void __chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by series summation for positive argument.*
- `template<typename _Tp >`  
`std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __clamp_pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __clausen_c (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen_c (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __clausen_s (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen_s (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __comp_ellint_1 (_Tp __k)`

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

- `template<typename _Tp >`  
`_Tp __comp_ellint_2 (_Tp __k)`

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

- `template<typename _Tp >`  
`_Tp __comp_ellint_3 (_Tp __k, _Tp __nu)`

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

- `template<typename _Tp >`  
`_Tp __comp_ellint_d (_Tp __k)`

- `template<typename _Tp >`  
`_Tp __comp_ellint_rf (_Tp __x, _Tp __y)`

- `template<typename _Tp >`  
`_Tp __comp_ellint_rg (_Tp __x, _Tp __y)`

- `template<typename _Tp >`  
`_Tp __conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __conf_hyperg_lim (_Tp __c, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __conf_hyperg_lim_series (_Tp __c, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`

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

- `template<typename _Tp >`  
`_Tp __conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __coshint (const _Tp __x)`

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

- `template<typename _Tp >`  
`std::complex< _Tp > __cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`

*Return the complex cylindrical Bessel function.*

- `template<typename _Tp >`  
`_Tp __cyl_bessel_i (_Tp __nu, _Tp __x)`

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

- `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  $\nu$ :  $J_\nu$  or  $I_\nu$  by series expansion.*

- `template<typename _Tp >`  
`void __cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

*Return the modified cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.*

- `template<typename _Tp >`  
`void __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

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

- `template<typename _Tp >`  
`void __cyl_bessel_ik_steel (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

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

- `template<typename _Tp >`  
`_Tp __cyl_bessel_j(_Tp __nu, _Tp __x)`  
 Return the Bessel function of order  $\nu$ :  $J_\nu(x)$ .
- `template<typename _Tp >`  
`void __cyl_bessel_jn(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
 Return the cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.
- `template<typename _Tp >`  
`void __cyl_bessel_jn_asymp(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
 This routine computes the asymptotic cylindrical Bessel and Neumann functions of order  $\nu$ :  $J_\nu(x)$ ,  $N_\nu(x)$ . Use this for  $x \gg \nu^2 + 1$ .
- `template<typename _Tp >`  
`void __cyl_bessel_jn_steud(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
 Compute the Bessel  $J_\nu(x)$  and Neumann  $N_\nu(x)$  functions and their first derivatives  $J'_\nu(x)$  and  $N'_\nu(x)$  respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`  
`_Tp __cyl_bessel_k(_Tp __nu, _Tp __x)`  
 Return the irregular modified Bessel function  $K_\nu(x)$  of order  $\nu$ .
- `template<typename _Tp >`  
`std::complex<_Tp> __cyl_hankel_1(_Tp __nu, _Tp __x)`  
 Return the cylindrical Hankel function of the first kind  $H_\nu^{(1)}(x)$ .
- `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  $H_n^{(2)}u(x)$ .
- `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  $\nu$ :  $N_\nu(x)$ .
- `template<typename _Tp >`  
`_Tp __dawson(_Tp __x)`  
 Return the Dawson integral,  $F(x)$ , for real argument  $x$ .
- `template<typename _Tp >`  
`_Tp __dawson_cont_frac(_Tp __x)`  
 Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`  
`_Tp __dawson_series(_Tp __x)`  
 Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`  
`void __debye_region(std::complex<_Tp> __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`  
`_Tp __dilog(_Tp __x)`

Compute the dilogarithm function  $Li_2(x)$  by summation for  $x \leq 1$ .

- `template<typename _Tp >`  
`_Tp __dirichlet_beta (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)`

Return the double factorial of the integer  $n$ .

- `template<typename _Tp >`  
`_Tp __ellint_1 (_Tp __k, _Tp __phi)`

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

- `template<typename _Tp >`  
`_Tp __ellint_2 (_Tp __k, _Tp __phi)`

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

- `template<typename _Tp >`  
`_Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`

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

- `template<typename _Tp >`  
`_Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`

- `template<typename _Tp >`  
`_Tp __ellint_d (_Tp __k, _Tp __phi)`

- `template<typename _Tp >`  
`_Tp __ellint_el1 (_Tp __x, _Tp __k_c)`

- `template<typename _Tp >`  
`_Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`

- `template<typename _Tp >`  
`_Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`

- `template<typename _Tp >`  
`_Tp __ellint_rc (_Tp __x, _Tp __y)`

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

- `template<typename _Tp >`  
`_Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)`

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

- `template<typename _Tp >`  
`_Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

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

- `template<typename _Tp >`  
`_Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

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

- `template<typename _Tp >`  
`_Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`

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

- `template<typename _Tp >`  
`_Tp __ellnome (_Tp __k)`

- `template<typename _Tp >`  
`_Tp __ellnome_k (_Tp __k)`
- `template<typename _Tp >`  
`_Tp __ellnome_series (_Tp __k)`
- `template<typename _Tp >`  
`_Tp __expint (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$ .*
- `template<typename _Tp >`  
`_Tp __expint (_Tp __x)`  
*Return the exponential integral  $Ei(x)$ .*
- `template<typename _Tp >`  
`_Tp __expint_asyp (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  for large argument.*
- `template<typename _Tp >`  
`_Tp __expint_E1 (_Tp __x)`  
*Return the exponential integral  $E_1(x)$ .*
- `template<typename _Tp >`  
`_Tp __expint_E1_asyp (_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_asyp (_Tp __x)`  
*Return the exponential integral  $Ei(x)$  by asymptotic expansion.*
- `template<typename _Tp >`  
`_Tp __expint_Ei_series (_Tp __x)`  
*Return the exponential integral  $Ei(x)$  by series summation.*
- `template<typename _Tp >`  
`_Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by continued fractions.*
- `template<typename _Tp >`  
`_Tp __expint_En_recursion (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by recursion. Use upward recursion for  $x < n$  and downward recursion (Miller's algorithm) otherwise.*
- `template<typename _Tp >`  
`_Tp __expint_En_series (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by series summation.*
- `template<typename _Tp >`  
`_Tp __expint_large_n (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  for large order.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)`  
*Return the factorial of the integer  $n$ .*
- `template<typename _Tp >`  
`_Tp __fermi_dirac (_Tp __s, _Tp __x)`

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*
- `template<typename _Tp >`  
`void __fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`  
*Compute the Fock-type Airy functions  $w_1(x)$  and  $w_2(x)$  and their first derivatives  $w_1'(x)$  and  $w_2'(x)$  respectively.*

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

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

- `template<typename _Tp >`  
`bool __fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`  
`bool __fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`  
`bool __fpimag (const _Tp)`
- `template<typename _Tp >`  
`bool __fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`  
`bool __fpreal (const _Tp)`
- `template<typename _Tp >`  
`std::complex< _Tp > __fresnel (const _Tp __x)`  
*Return the Fresnel cosine and sine integrals as a complex number  $\$f[ C(x) + iS(x) \$f]$ .*
- `template<typename _Tp >`  
`void __fresnel_cont_frac (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`  
*This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.*
- `template<typename _Tp >`  
`void __fresnel_series (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`  
*This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.*
- `template<typename _Tp >`  
`_Tp __gamma (_Tp __x)`  
*Return  $\Gamma(x)$ .*
- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __gamma_l (_Tp __a, _Tp __x)`  
*Return the lower incomplete gamma function. The lower incomplete gamma function is defined by*

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

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)`
- `template<typename _Tp >`  
`void __gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`

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

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

and

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

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

- `template<typename _Tp >`  
`_Tp __gamma_u(_Tp __a, _Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __gauss(_Tp __x)`
- `template<typename _Tp >`  
`_Tp __gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)`
- `template<typename _Tp >`  
`void __hankel(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> &_H1, std::complex<_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p)`
- `template<typename _Tp >`  
`void __hankel_debye(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> __alpha, int __indexr, char &__aorb, int &__morn, std::complex<_Tp> &_H1, std::complex<_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p)`
- `template<typename _Tp >`  
`void __hankel_params(std::complex<_Tp> __nu, std::complex<_Tp> __zhat, std::complex<_Tp> &__p, std::complex<_Tp> &__p2, std::complex<_Tp> &__nup2, std::complex<_Tp> &__num2, std::complex<_Tp> &__num1d3, std::complex<_Tp> &__num2d3, std::complex<_Tp> &__num4d3, std::complex<_Tp> &__zeta, std::complex<_Tp> &__zetaphf, std::complex<_Tp> &__zetamhf, std::complex<_Tp> &__zetam3hf, std::complex<_Tp> &__zetrat)`

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

- `template<typename _Tp >`  
`void __hankel_uniform(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> &_H1, std::complex<_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p)`

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

- `template<typename _Tp >`  
`void __hankel_uniform_olver(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> &_H1, std::complex<_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p)`

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

- `template<typename _Tp >`  
`void __hankel_uniform_outer(std::complex<_Tp> __nu, std::complex<_Tp> __z, _Tp __eps, std::complex<_Tp> &__zhat, std::complex<_Tp> &__1dnsq, std::complex<_Tp> &__num1d3, std::complex<_Tp> &__num2d3, std::complex<_Tp> &__p, std::complex<_Tp> &__p2, std::complex<_Tp> &__etm3h, std::complex<_Tp> &__etrat, std::complex<_Tp> &__Aip, std::complex<_Tp> &__o4dp, std::complex<_Tp> &__Aim, std::complex<_Tp> &__o4dm, std::complex<_Tp> &__od2p, std::complex<_Tp> &__od0dp, std::complex<_Tp> &__od2m, std::complex<_Tp> &__od0dm)`

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

- `template<typename _Tp >`  
`void __hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)`

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

- `template<typename _Tp >`  
`_Tp __heuman_lambda (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp __hurwitz_zeta (_Tp __s, _Tp __a)`  
Return the Hurwitz zeta function  $\zeta(s, a)$  for all  $s \neq 1$  and  $a > -1$ .
- `template<typename _Tp >`  
`std::complex< _Tp > __hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`  
`_Tp __hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)`  
Return the Hurwitz zeta function  $\zeta(s, a)$  for all  $s \neq 1$  and  $a > -1$ .
- `template<typename _Tp >`  
`std::complex< _Tp > __hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)`
- `template<typename _Tp >`  
`_Tp __hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
Return the hypergeometric function  ${}_2F_1(a, b; c; x)$ .
- `template<typename _Tp >`  
`_Tp __hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`  
Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by an iterative procedure described in Luke, *Algorithms for the Computation of Mathematical Functions*.
- `template<typename _Tp >`  
`_Tp __hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for  $d = c - a - b$  not integral and formula 15.3.11 for  $d = c - a - b$  integral. This assumes  $a, b, c \neq$  negative integer.
- `template<typename _Tp >`  
`_Tp __hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by series expansion.
- `template<typename _Tp >`  
`std::tuple< _Tp, _Tp, _Tp > __jacobi_sncndn (_Tp __k, _Tp __u)`
- `template<typename _Tp >`  
`_Tp __jacobi_zeta (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp __laguerre (unsigned int __n, _Tp __x)`  
This routine returns the Laguerre polynomial of order  $n$ :  $L_n(x)$ .
- `template<typename _Tp >`  
`_Tp __legendre_q (unsigned int __l, _Tp __x)`  
Return the Legendre function of the second kind by upward recursion on order  $l$ .
- `template<typename _Tp >`  
`_Tp __log_bincoef (unsigned int __n, unsigned int __k)`



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

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

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)`  
Return the logarithm of the double factorial of the integer  $n$ .
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)`  
Return the logarithm of the factorial of the integer  $n$ .
- `template<typename _Tp >`  
`_Tp __log_gamma (_Tp __x)`  
Return  $\log(|\Gamma(x)|)$ . This will return values even for  $x < 0$ . To recover the sign of  $\Gamma(x)$  for any argument use `__log_gamma_sign`.
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)`  
Return  $\log(\Gamma(x))$  by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_lanczos (_Tp __x)`  
Return  $\log(\Gamma(x))$  by the Lanczos method. This method dominates all others on the positive axis I think.
- `template<typename _Tp >`  
`_Tp __log_gamma_sign (_Tp __x)`  
Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned.
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_spouge (_Tp __z)`  
Return  $\Gamma(z)$  by the Spouge algorithm:

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

where

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

and the error is bounded by

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

- `template<typename _Tp >`  
`_Tp __log_pochhammer_l (_Tp __a, _Tp __n)`  
Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

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

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

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

Many notations exist:

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

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`

`_Tp __log_pochhammer_u (_Tp __a, _Tp __n)`

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

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

Thus this function returns

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

Many notations exist:

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

,

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

- `template<typename _Tp >`

`_Tp __logint (const _Tp __x)`

Return the logarithmic integral  $li(x)$ .

- `template<typename _Tp >`

`_Tp __owens_t (_Tp __h, _Tp __a)`

- `template<typename _Tp >`

`_Tp __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 __pochhammer_l (_Tp __a, _Tp __n)`

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

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

In particular,  $(n)_n = n!$ .

- `template<typename _Tp >`

`_Tp __pochhammer_u (_Tp __a, _Tp __n)`

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

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

Many notations exist:

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

,

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

- `template<typename _Tp >`

`_Tp __poly_hermite (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __poly_hermite_asymp (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __poly_hermite_recursion (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

- `template<typename _Tpa, typename _Tp >`  
`_Tp __poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`

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

- `template<typename _Tpa, typename _Tp >`  
`_Tp __poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)`

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

- `template<typename _Tpa, typename _Tp >`  
`_Tp __poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)`

*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha > -1$  for large  $n$ . Abramowitz & Stegun, 13.5.21.*

- `template<typename _Tpa, typename _Tp >`  
`_Tp __poly_laguerre_recursion (unsigned int __n, _Tpa __alpha1, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp __poly_legendre_p (unsigned int __l, _Tp __x)`

*Return the Legendre polynomial by upward recursion on order  $l$ .*

- `template<typename _Tp >`  
`_Tp __poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`

- `template<typename _Tp >`  
`_Tp __polylog (_Tp __s, _Tp __x)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog (_Tp __s, std::complex< _Tp > __w)`

- `template<typename _Tp, typename ArgType >`  
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > __polylog_exp (_Tp __s, ArgType __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_int_neg (int __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_int_neg (const int __s, _Tp __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, _Tp __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_neg (int __s, std::complex< _Tp > __w)`

- `template<typename _Tp, int __sigma >`  
`std::complex< _Tp > __polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`

- `template<typename _Tp, int __sigma >`  
`std::complex< _Tp > __polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`

- `template<typename _PowTp, typename _Tp >`  
`_Tp __polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __psi (_Tp __x)`

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

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

*For negative argument the reflection formula is used:*

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

- `template<typename _Tp >`  
`_Tp __psi (unsigned int __n, _Tp __x)`  
*Return the polygamma function  $\psi^{(n)}(x)$ .*
- `template<typename _Tp >`  
`_Tp __psi_asymp (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __psi_series (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __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 [\\_\\_riemann\\_zeta](#) (\_Tp \_\_s)  
*Return the Riemann zeta function  $\zeta(s)$ .*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_alt](#) (\_Tp \_\_s)  
*Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_euler\\_maclaurin](#) (\_Tp \_\_s)  
*Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_glob](#) (\_Tp \_\_s)  
*Evaluate the Riemann zeta function by series for all  $s \neq 1$ . Convergence is great until largish negative numbers. Then the convergence of the  $> 0$  sum gets better.*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_m\\_1](#) (\_Tp \_\_s)  
*Return the Riemann zeta function  $\zeta(s) - 1$ .*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_m\\_1\\_sum](#) (\_Tp \_\_s)  
*Return the Riemann zeta function  $\zeta(s) - 1$  by summation for  $s > 1$ . This is a small remainder for large  $s$ .*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_product](#) (\_Tp \_\_s)  
*Compute the Riemann zeta function  $\zeta(s)$  using the product over prime factors.*
- template<typename \_Tp >  
\_Tp [\\_\\_riemann\\_zeta\\_sum](#) (\_Tp \_\_s)  
*Compute the Riemann zeta function  $\zeta(s)$  by summation for  $s > 1$ .*
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t](#)<\_Tp> [\\_\\_sinc](#) (\_Tp \_\_a, \_Tp \_\_x)  
*Return the generalized sinus cardinal function*  
$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t](#)<\_Tp> [\\_\\_sinc](#) (\_Tp \_\_x)  
*Return the normalized sinus cardinal function*  
$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t](#)<\_Tp> [\\_\\_sinc\\_pi](#) (\_Tp \_\_x)  
*Return the unnormalized sinus cardinal function*  
$$\text{sinc}_\pi(x) = \frac{\sin(x)}{x}$$
- template<typename \_Tp >  
std::pair<\_Tp, \_Tp> [\\_\\_sincosint](#) (\_Tp \_\_x)  
*This function returns the sine  $Si(x)$  and cosine  $Ci(x)$  integrals as a pair.*
- template<typename \_Tp >  
void [\\_\\_sincosint\\_asymp](#) (\_Tp \_\_t, \_Tp &\_Si, \_Tp &\_Ci)  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by asymptotic series summation for positive argument.*

- `template<typename _Tp >`  
`void __sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by continued fraction for positive argument.*

- `template<typename _Tp >`  
`void __sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by series summation for positive argument.*

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t<_Tp > __sinhc (_Tp __a, _Tp __x)`  
*Return the generalized hyperbolic sinus cardinal function*

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t<_Tp > __sinhc (_Tp __x)`  
*Return the normalized hyperbolic sinus cardinal function*

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

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t<_Tp > __sinhc_pi (_Tp __x)`  
*Return the unnormalized hyperbolic sinus cardinal function*

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

- `template<typename _Tp >`  
`_Tp __sinhint (const _Tp __x)`  
*Return the hyperbolic sine integral  $li(x)$ .*

- `template<typename _Tp >`  
`_Tp __sph_bessel (unsigned int __n, _Tp __x)`  
*Return the spherical Bessel function  $j_n(x)$  of order  $n$  and non-negative real argument  $x$ .*

- `template<typename _Tp >`  
`std::complex<_Tp > __sph_bessel (unsigned int __n, std::complex<_Tp > __z)`  
*Return the complex spherical Bessel function.*

- `template<typename _Tp >`  
`void __sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`  
*Compute the spherical modified Bessel functions  $i_n(x)$  and  $k_n(x)$  and their first derivatives  $i'_n(x)$  and  $k'_n(x)$  respectively.*

- `template<typename _Tp >`  
`void __sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`  
*Compute the spherical Bessel  $j_n(x)$  and Neumann  $n_n(x)$  functions and their first derivatives  $j'_n(x)$  and  $n'_n(x)$  respectively.*

- `template<typename _Tp >`  
`void __sph_hankel (unsigned int __n, std::complex<_Tp > __z, std::complex<_Tp > &_H1, std::complex<_Tp > &_H1p, std::complex<_Tp > &_H2, std::complex<_Tp > &_H2p)`  
*Helper to compute complex spherical Hankel functions and their derivatives.*

- `template<typename _Tp >`  
`std::complex<_Tp > __sph_hankel_1 (unsigned int __n, _Tp __x)`  
*Return the spherical Hankel function of the first kind  $h_n^{(1)}(x)$ .*

- `template<typename _Tp >`  
`std::complex<_Tp > __sph_hankel_1 (unsigned int __n, std::complex<_Tp > __z)`

*Return the complex spherical Hankel function of the first kind.*

- `template<typename _Tp >`  
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, _Tp __x)`

*Return the spherical Hankel function of the second kind  $h_n^{(2)}(x)$ .*

- `template<typename _Tp >`  
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`

*Return the complex spherical Hankel function of the second kind.*

- `template<typename _Tp >`  
`std::complex< _Tp > __sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`

*Return the spherical harmonic function.*

- `template<typename _Tp >`  
`_Tp __sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

*Return the spherical associated Legendre function.*

- `template<typename _Tp >`  
`_Tp __sph_neumann (unsigned int __n, _Tp __x)`

*Return the spherical Neumann function  $n_n(x)$  of order  $n$  and non-negative real argument  $x$ .*

- `template<typename _Tp >`  
`std::complex< _Tp > __sph_neumann (unsigned int __n, std::complex< _Tp > __z)`

*Return the complex spherical Neumann function.*

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __student_t_cdf (_Tp __t, unsigned int __nu)`

*Return the Students T probability function.*

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __student_t_cdfc (_Tp __t, unsigned int __nu)`

*Return the complement of the Students T probability function.*

- `template<typename _Tp >`  
`_Tp __theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_2_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_3_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __theta_s (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`

- `template<typename _Tp >`  
`_Tp __znorm1 (_Tp __x)`
- `template<typename _Tp >`  
`_Tp __znorm2 (_Tp __x)`
- `template<typename _Tp = double>`  
`_Tp evenzeta (unsigned int __k)`

## Variables

- `constexpr size_t _Num_Euler_Maclaurin_zeta = 100`
- `constexpr _Factorial_table< long double > _S_double_factorial_table [301]`
- `constexpr long double _S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]`
- `constexpr _Factorial_table< long double > _S_factorial_table [171]`
- `constexpr _Factorial_table< long double > _S_neg_double_factorial_table [999]`
- `template<typename _Tp >`  
`constexpr std::size_t _S_num_double_factorials = 0`
- `template<>`  
`constexpr std::size_t _S_num_double_factorials< double > = 301`
- `template<>`  
`constexpr std::size_t _S_num_double_factorials< float > = 57`
- `template<>`  
`constexpr std::size_t _S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`  
`constexpr std::size_t _S_num_factorials = 0`
- `template<>`  
`constexpr std::size_t _S_num_factorials< double > = 171`
- `template<>`  
`constexpr std::size_t _S_num_factorials< float > = 35`
- `template<>`  
`constexpr std::size_t _S_num_factorials< long double > = 171`
- `template<typename _Tp >`  
`constexpr std::size_t _S_num_neg_double_factorials = 0`
- `template<>`  
`constexpr std::size_t _S_num_neg_double_factorials< double > = 150`
- `template<>`  
`constexpr std::size_t _S_num_neg_double_factorials< float > = 27`
- `template<>`  
`constexpr std::size_t _S_num_neg_double_factorials< long double > = 999`
- `constexpr size_t _S_num_zetam1 = 33`
- `constexpr long double _S_zetam1 [_S_num_zetam1]`

## 8.3.1 Enumeration Type Documentation

### 8.3.1.1 anonymous enum

Enumerator

***SININT***

***COSINT***

Definition at line 42 of file `sf_trigint.tcc`.



### 8.3.2 Function Documentation

8.3.2.1 `template<typename _Tp> void std::__detail::__airy ( _Tp _z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip )`

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

Parameters

<code>_z</code>	The argument of the Airy functions.
<code>_Ai</code>	The output Airy function of the first kind.
<code>_Bi</code>	The output Airy function of the second kind.
<code>_Aip</code>	The output derivative of the Airy function of the first kind.
<code>_Bip</code>	The output derivative of the Airy function of the second kind.

Definition at line 497 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

8.3.2.2 `template<typename _Tp> void std::__detail::__airy ( const std::complex<_Tp> &_z, _Tp __eps, std::complex<_Tp> &_Ai, std::complex<_Tp> &_Aip, std::complex<_Tp> &_Bi, std::complex<_Tp> &_Bip )`

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

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

$$\begin{aligned}
 (1a) \quad Ai(z) &= \frac{\sqrt{z}}{3} [I_{-1/3}(\zeta) - I_{1/3}(\zeta)] \\
 (1b) \quad Bi(z) &= \sqrt{\frac{z}{3}} [I_{-1/3}(\zeta) + I_{1/3}(\zeta)] \\
 (2) \quad Ai(z) &= \frac{\sqrt{z/3}}{\pi} K_{1/3}(\zeta) = \frac{2^{2/3} 3^{-5/6}}{\sqrt{\pi}} z \exp(-\zeta) U(5/6; 5/3; 2\zeta) \\
 (3a) \quad Ai(-z) &= \frac{\sqrt{z}}{3} [J_{-1/3}(\zeta) + J_{1/3}(\zeta)] \\
 (3b) \quad Bi(-z) &= \sqrt{\frac{z}{3}} [J_{-1/3}(\zeta) - J_{1/3}(\zeta)] \\
 (4a) \quad Ai'(z) &= \frac{z}{3} [I_{2/3}(\zeta) - I_{-2/3}(\zeta)] \\
 (4b) \quad Bi'(z) &= \frac{z}{\sqrt{3}} [I_{-2/3}(\zeta) + I_{2/3}(\zeta)] \\
 (5a) \quad Ai'(z) &= -\frac{z}{\pi\sqrt{3}} K_{2/3}(\zeta) = -\frac{4^{2/3} 3^{-7/6}}{\sqrt{\pi}} z^2 \exp(-\zeta) U(7/6; 7/3; 2\zeta) \\
 (6a) \quad Ai'(-z) &= \frac{z}{3} [J_{2/3}(\zeta) - J_{-2/3}(\zeta)] \\
 (6b) \quad Bi'(-z) &= \frac{z}{\sqrt{3}} [J_{-2/3}(\zeta) + J_{2/3}(\zeta)]
 \end{aligned}$$

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

**See also**

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

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

**See also**

Olver, F. W. J., Error Bounds for Asymptotic Expansions, with an Application to Cylinder Functions of Large Argument, in Asymptotic Solutions of Differential Equations (Wilcox, Ed.), Wiley and Sons, pp 163-183, 1964  
 Olver, F. W. J., Asymptotics and Special Functions, Academic Press, pp 266-268, 1974.

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

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

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

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

$$J_\nu(z) = e^{i\nu\pi/2} I_\nu(ze^{-i\pi/2}), 0 \leq \arg(z) \leq \pi/2$$

and

$$J_\nu(z) = e^{-i\nu\pi/2} I_\nu(ze^{i\pi/2}), -\pi/2 \leq \arg(z) < 0.$$

The particular backward recursion algorithm used is discussed in

**See also**

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

The following paper was also useful

**See also**

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

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

**See also**

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

## Parameters

in	<code>__z</code>	The argument at which the Airy function and its derivative are computed.
in	<code>__eps</code>	Relative error required. Currently, eps is used only in the backward recursion algorithms.
out	<code>__Ai</code>	The value computed for $Ai(z)$ .
out	<code>__Aip</code>	The value computed for $Ai'(z)$ .
out	<code>__Bi</code>	The value computed for $Bi(z)$ .
out	<code>__Bip</code>	The value computed for $Bi'(z)$ .

Definition at line 1004 of file sf\_airy.tcc.

References `__airy_asymp_absarg_ge_pio3()`, `__airy_asymp_absarg_lt_pio3()`, `__airy_bessel_i()`, `__airy_bessel_k()`, and `__airy_hyperg_rational()`.

Referenced by `__airy_ai()`, `__airy_bi()`, `__hankel_uniform_outer()`, and `__poly_hermite_asymp()`.

**8.3.2.3** `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_ai ( std::complex<_Tp> __z )`

Return the complex Airy Ai function.

Definition at line 1141 of file sf\_airy.tcc.

References `__airy()`.

**8.3.2.4** `template<typename _Tp> void std::__detail::__airy_arg ( std::complex<_Tp> __num2d3, std::complex<_Tp> __zeta, std::complex<_Tp> &__argp, std::complex<_Tp> &__argm )`

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

## Parameters

in	<code>__num2d3</code>	$\nu^{-2/3}$ - output from <code>hankel_params</code>
in	<code>__zeta</code>	zeta in the uniform asymptotic expansions - output from <code>hankel_params</code>
out	<code>__argp</code>	$e^{+i2\pi/3}\nu^{2/3}\zeta$
out	<code>__argm</code>	$e^{-i2\pi/3}\nu^{2/3}\zeta$

## Exceptions

<code>std::runtime_error</code>	if unable to compute Airy function arguments
---------------------------------	--

Definition at line 241 of file sf\_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

8.3.2.5 `template<typename _Tp> void std::__detail::__airy_asymp_absarg_ge_pio3 ( std::complex< _Tp> __z, std::complex< _Tp> & _Ai, std::complex< _Tp> & _Aip, int __sign = -1 )`

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

Note that for speed and since this function is called by another, checks for valid arguments are not made.

#### See also

Digital Library of Mathematical Functions Sec. 9.7 Asymptotic Expansions <http://dlmf.nist.gov/9.7>

#### Parameters

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

Definition at line 71 of file `sf_airy.tcc`.

Referenced by `__airy()`.

8.3.2.6 `template<typename _Tp> void std::__detail::__airy_asymp_absarg_lt_pio3 ( std::complex< _Tp> __z, std::complex< _Tp> & _Ai, std::complex< _Tp> & _Aip )`

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

Note that for speed and since this function is called by another, checks for valid arguments are not made. This function assumes  $|z| > 15$  and  $|arg(-z)| < \pi/3$ .

#### Parameters

in	<code>__z</code>	The value at which the Airy function and its derivative are evaluated.
out	<code>_Ai</code>	The computed value of the Airy function $Ai(z)$ .
out	<code>_Aip</code>	The computed value of the Airy function derivative $Ai'(z)$ .

Definition at line 186 of file `sf_airy.tcc`.

Referenced by `__airy()`.

8.3.2.7 `template<typename _Tp> void std::__detail::__airy_bessel_i ( const std::complex< _Tp> & __z, _Tp __eps, std::complex< _Tp> & __lp1d3, std::complex< _Tp> & __lm1d3, std::complex< _Tp> & __lp2d3, std::complex< _Tp> & __lm2d3 )`

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

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

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

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

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

This modification of the algorithm is given in part in

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

And further elaborated for the Bessel functions in

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

Insight was also gained from

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

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

$$|x| + |y| \leq \sqrt{2} \sqrt{x^2 + y^2}$$

#### Parameters

in	<code>__z</code>	The argument of the modified Bessel functions.
in	<code>__eps</code>	The maximum relative error required in the results.
out	<code>__lp1d3</code>	The value of $I_{(1/3)}(z)$ .
out	<code>__lm1d3</code>	The value of $I_{(-1/3)}(z)$ .
out	<code>__lp2d3</code>	The value of $I_{(2/3)}(z)$ .
out	<code>__lm2d3</code>	The value of $I_{(-2/3)}(z)$ .

Definition at line 390 of file sf\_airy.tcc.

Referenced by \_\_airy().

**8.3.2.8** `template<typename _Tp> void std::__detail::__airy_bessel_k ( const std::complex<_Tp> & __z, _Tp __eps, std::complex<_Tp> & _Kp1d3, std::complex<_Tp> & _Kp2d3 )`

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

This routine computes

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

using a rational approximation given in

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

Though the approximation converges in  $|\arg(z)| \leq \pi$ , The convergence weakens as  $\text{abs}(\arg(z))$  increases. Also, in the case of real order between 0 and 1, convergence weakens as the order approaches 1. For these reasons, we only use this code for  $|\arg(z)| \leq 3\pi/4$  and the convergence test is performed only for order 2/3.

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

#### Parameters

in	<code>__z</code>	The value for which the quantity $E_\nu$ is to be computed. It is recommended that $\text{abs}(z)$ not be too small and that $ \arg(z)  \leq 3\pi/4$ .
in	<code>__eps</code>	The maximum relative error allowable in the computed results. The relative error test is based on the comparison of successive iterates.
out	<code>_Kp1d3</code>	The value computed for $E_{+1/3}(z)$ .
out	<code>_Kp2d3</code>	The value computed for $E_{+2/3}(z)$ .

#### Note

In the worst case, say,  $z = 2$  and  $\arg(z) = 3\pi/4$ , 20 iterations should give 7 or 8 decimals of accuracy.

Definition at line 604 of file sf\_airy.tcc.

Referenced by \_\_airy().

**8.3.2.9** `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_bi ( std::complex<_Tp> __z )`

Return the complex Airy Bi function.

Definition at line 1154 of file sf\_airy.tcc.

References \_\_airy().

8.3.2.10 `template<typename _Tp> void std::__detail::__airy_hyperg_rational ( const std::complex<_Tp> & __z, std::complex<_Tp> & _Ai, std::complex<_Tp> & _Aip, std::complex<_Tp> & _Bi, std::complex<_Tp> & _Bip )`

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

$$\frac{\Gamma(\nu + 1)}{(z/2)^{\nu}} I_{\nu}(z) = {}_0F_1(; \nu + 1; z^2/4),$$

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

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

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

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

Note also that for speed and since this function is called by another, checks that are not absolutely necessary are not made.

#### Parameters

in	<code>__z</code>	The argument at which the hypergeometric given above is to be evaluated. Since the approximation is of fixed order, $ z $ must be small to ensure sufficient accuracy of the computed results.
out	<code>_Ai</code>	The Airy function $Ai(z)$ .
out	<code>_Aip</code>	The Airy function derivative $Ai'(z)$ .
out	<code>_Bi</code>	The Airy function $Bi(z)$ .
out	<code>_Bip</code>	The Airy function derivative $Bi'(z)$ .

Definition at line 787 of file sf\_airy.tcc.

Referenced by `__airy()`.

8.3.2.11 `template<typename _Tp> _Tp std::__detail::__assoc_laguerre ( unsigned int __n, unsigned int __m, _Tp __x )`

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

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

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

where the Laguerre polynomial is defined by:

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

## Template Parameters

<code>_Tp</code>	The type of the parameter
------------------	---------------------------

## Parameters

<code>↵ _n</code>	The order
<code>↵ _m</code>	The degree
<code>↵ _x</code>	The argument

## Returns

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

Definition at line 301 of file `sf_laguerre.tcc`.

Referenced by `__hydrogen()`.

**8.3.2.12** `template<typename _Tp> _Tp std::__detail::__assoc_legendre_p ( unsigned int __l, unsigned int __m, _Tp __x )`

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

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

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

## Parameters

<code>↵ _l</code>	The order of the associated Legendre function. $l \geq 0$ .
<code>↵ _m</code>	The order of the associated Legendre function. $m \leq l$ .
<code>↵ _x</code>	The argument of the associated Legendre function. $ x  \leq 1$ .

Definition at line 175 of file `sf_legendre.tcc`.

References `__poly_legendre_p()`.

**8.3.2.13** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli ( int __n )`

This returns Bernoulli number  $B_n$ .



## Parameters

$\longleftrightarrow$	the order $n$ of the Bernoulli number.
$n$	

## Returns

The Bernoulli number of order  $n$ .

Definition at line 1673 of file sf\_gamma.tcc.

References std::\_\_detail::\_Factorial\_table<\_Tp>::\_\_n.

8.3.2.14 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n ( int __n )`

This returns Bernoulli number  $B_n$ .

## Parameters

$\longleftrightarrow$	the order $n$ of the Bernoulli number.
$n$	

## Returns

The Bernoulli number of order  $n$ .

Definition at line 1685 of file sf\_gamma.tcc.

References std::\_\_detail::\_Factorial\_table<\_Tp>::\_\_n.

8.3.2.15 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series ( unsigned int __n )`

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

Upward recursion is unstable.

## Parameters

$\longleftrightarrow$	the order $n$ of the Bernoulli number.
$n$	

## Returns

The Bernoulli number of order  $n$ .

Definition at line 1608 of file sf\_gamma.tcc.

References std::\_\_detail::\_Factorial\_table<\_Tp>::\_\_n.

**8.3.2.16** `template<typename _Tp> _Tp std::__detail::__beta ( _Tp __a, _Tp __b )`

Return the beta function  $B(a, b)$ .

The beta function is defined by

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

#### Parameters

$\longleftrightarrow$ __a	The first argument of the beta function.
$\longleftrightarrow$ __b	The second argument of the beta function.

#### Returns

The beta function.

Definition at line 173 of file sf\_beta.tcc.

References \_\_beta\_lgamma().

Referenced by \_\_poly\_jacobi(), \_\_gnu\_cxx::jacobi(), \_\_gnu\_cxx::jacobif(), and \_\_gnu\_cxx::jacobil().

**8.3.2.17** `template<typename _Tp> _Tp std::__detail::__beta_gamma ( _Tp __a, _Tp __b )`

Return the beta function:  $B(a, b)$ .

The beta function is defined by

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

#### Parameters

$\longleftrightarrow$ __a	The first argument of the beta function.
$\longleftrightarrow$ __b	The second argument of the beta function.

**Returns**

The beta function.

Definition at line 75 of file sf\_beta.tcc.

References `__gamma()`.

**8.3.2.18** `template<typename _Tp> _Tp std::__detail::__beta_inc ( _Tp __a, _Tp __b, _Tp __x )`

Return the regularized incomplete beta function,  $I_x(a, b)$ , of arguments  $a$ ,  $b$ , and  $x$ .

The regularized incomplete beta function is defined by:

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

where

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

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

**Parameters**

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 262 of file sf\_beta.tcc.

References `__beta_inc_cont_frac()`.

Referenced by `__binomial_cdf()`, `__binomial_cdfc()`, `__fisher_f_cdf()`, `__fisher_f_cdfc()`, `__student_t_cdf()`, and `__student_t_cdfc()`.

**8.3.2.19** `template<typename _Tp> _Tp std::__detail::__beta_inc_cont_frac ( _Tp __a, _Tp __b, _Tp __x )`

Return the regularized incomplete beta function,  $I_x(a, b)$ , of arguments  $a$ ,  $b$ , and  $x$ .

**Parameters**

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 193 of file sf\_beta.tcc.

Referenced by \_\_beta\_inc().

**8.3.2.20** `template<typename _Tp> _Tp std::__detail::__beta_lgamma ( _Tp __a, _Tp __b )`

Return the beta function  $B(a, b)$  using the log gamma functions.

The beta function is defined by

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

#### Parameters

$\leftrightarrow$ __a	The first argument of the beta function.
$\leftrightarrow$ __b	The second argument of the beta function.

#### Returns

The beta function.

Definition at line 109 of file sf\_beta.tcc.

References \_\_log\_gamma().

Referenced by \_\_beta().

**8.3.2.21** `template<typename _Tp> _Tp std::__detail::__beta_product ( _Tp __a, _Tp __b )`

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

The beta function is defined by

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

Here, we employ the product form:

$$B(a, b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1 + (a+b)/k}{(1 + a/k)(1 + b/k)}$$

#### Parameters

$\leftrightarrow$ __a	The first argument of the beta function.
$\leftrightarrow$ __b	The second argument of the beta function.

**Returns**

The beta function.

Definition at line 140 of file sf\_beta.tcc.

**8.3.2.22** `template<typename _Tp> _Tp std::__detail::__bincoef ( unsigned int __n, unsigned int __k )`

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

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

**Parameters**

$\leftrightarrow$ __n	The first argument of the binomial coefficient.
$\leftrightarrow$ __k	The second argument of the binomial coefficient.

**Returns**

The binomial coefficient.

Definition at line 1888 of file sf\_gamma.tcc.

References std::\_\_detail::\_Factorial\_table< \_Tp >::\_\_n.

**8.3.2.23** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf ( _Tp __p, unsigned int __n, unsigned int __k )`

Return the binomial cumulative distribution function.

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

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

**Parameters**

$\leftrightarrow$ __p	
$\leftrightarrow$ __n	
$\leftrightarrow$ __k	

Definition at line 405 of file sf\_beta.tcc.

References `__beta_inc()`.

**8.3.2.24** `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdfc ( _Tp __p, unsigned int __n, unsigned int __k )`

Return the complementary binomial cumulative distribution function.

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

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

#### Parameters

<code>__p</code>	
<code>__n</code>	
<code>__k</code>	

Definition at line 435 of file sf\_beta.tcc.

References `__beta_inc()`.

**8.3.2.25** `template<typename _Tp > _Tp std::__detail::__bose_einstein ( _Tp __s, _Tp __x )`

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

#### See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)  
<http://dlmf.nist.gov/25.12#iii>

#### Parameters

<code>__s</code>	The order $s \geq 0$ .
<code>__x</code>	The real argument.

#### Returns

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

Definition at line 1401 of file sf\_polylog.tcc.

References `__polylog_exp()`.

**8.3.2.26** `template<typename _Tp> _Tp std::__detail::__chebyshev_recur ( unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1 )`

Return a Chebyshev polynomial of non-negative order  $n$  and real argument  $x$  by the recursion

$$C_n(x) = 2xC_{n-1} - C_{n-2}$$

#### Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

#### Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$
<code>__C0</code>	The value of the zeroth-order Chebyshev polynomial at $x$
<code>__C1</code>	The value of the first-order Chebyshev polynomial at $x$

Definition at line 57 of file sf\_chebyshev.tcc.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

**8.3.2.27** `template<typename _Tp> _Tp std::__detail::__chebyshev_t ( unsigned int __n, _Tp __x )`

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

The Chebyshev polynomial of the first kind is defined by:

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

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

#### Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

#### Parameters

<code>__n</code>	The non-negative integral order
------------------	---------------------------------

## Parameters

$\leftrightarrow$ _x	The real argument $-1 \leq x \leq +1$
-------------------------	---------------------------------------

Definition at line 85 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**8.3.2.28** `template<typename _Tp> _Tp std::__detail::__chebyshev_u ( unsigned int __n, _Tp __x )`

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

The Chebyshev polynomial of the second kind is defined by:

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

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

## Template Parameters

_Tp	The real type of the argument
-----	-------------------------------

## Parameters

$\leftrightarrow$ _n	The non-negative integral order
$\leftrightarrow$ _x	The real argument $-1 \leq x \leq +1$

Definition at line 114 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**8.3.2.29** `template<typename _Tp> _Tp std::__detail::__chebyshev_v ( unsigned int __n, _Tp __x )`

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

The Chebyshev polynomial of the third kind is defined by:

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

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



## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>↔ _n</code>	The non-negative integral order
<code>↔ _x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 144 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**8.3.2.30** `template<typename _Tp> _Tp std::__detail::__chebyshev_w( unsigned int _n, _Tp _x )`

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

## Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

## Parameters

<code>↔ _n</code>	The non-negative integral order
<code>↔ _x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 174 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**8.3.2.31** `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__chshint( _Tp _x, _Tp &_Chi, _Tp &_Shi )`

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

The hyperbolic cosine integral is defined by:

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

The hyperbolic sine integral is defined by:

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

Definition at line 162 of file sf\_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

**8.3.2.32** `template<typename _Tp> void std::__detail::__chshint_cont_frac ( _Tp __t, _Tp & _Chi, _Tp & _Shi )`

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

Definition at line 50 of file sf\_hypint.tcc.

Referenced by `__chshint()`.

**8.3.2.33** `template<typename _Tp> void std::__detail::__chshint_series ( _Tp __t, _Tp & _Chi, _Tp & _Shi )`

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

Definition at line 93 of file sf\_hypint.tcc.

Referenced by `__chshint()`.

**8.3.2.34** `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi ( std::complex<_Tp> __w )`

Definition at line 136 of file sf\_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↔real_pos()`.

**8.3.2.35** `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi ( std::complex<_Tp> __w )`

Definition at line 123 of file sf\_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↔real_pos()`.

**8.3.2.36** `template<typename _Tp> std::complex<_Tp> std::__detail::__clausen ( unsigned int __m, std::complex<_Tp> __w )`

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

## Parameters

$\leftrightarrow$ _m	The non-negative integral order.
$\leftrightarrow$ _w	The complex argument.

## Returns

The complex Clausen function.

Definition at line 1230 of file sf\_polylog.tcc.

References `__polylog_exp()`.

**8.3.2.37** `template<typename _Tp> _Tp std::__detail::__clausen ( unsigned int __m, _Tp __w )`

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

## Parameters

$\leftrightarrow$ _m	The integer order $m \geq 1$ .
$\leftrightarrow$ _w	The real argument.

## Returns

The Clausen function.

Definition at line 1254 of file sf\_polylog.tcc.

References `__polylog_exp()`.

**8.3.2.38** `template<typename _Tp> _Tp std::__detail::__clausen_c ( unsigned int __m, std::complex<_Tp> __w )`

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

## See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)

**Parameters**

$\leftrightarrow$ _m	The integer order $m \geq 1$ .
$\leftrightarrow$ _w	The real argument.

**Returns**

The Clausen cosine sum  $Cl_m(w)$ ,

Definition at line 1329 of file sf\_polylog.tcc.

References `__polylog_exp()`.

8.3.2.39 `template<typename _Tp> _Tp std::__detail::__clausen_c( unsigned int __m, _Tp __w )`

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

**See also**

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)

**Parameters**

$\leftrightarrow$ _m	The integer order $m \geq 1$ .
$\leftrightarrow$ _w	The real argument.

**Returns**

The real Clausen cosine sum  $Cl_m(w)$ ,

Definition at line 1354 of file sf\_polylog.tcc.

References `__polylog_exp()`.

8.3.2.40 `template<typename _Tp> _Tp std::__detail::__clausen_s( unsigned int __m, std::complex<_Tp> __w )`

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

**See also**

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)

## Parameters

$\leftrightarrow$ _m	The integer order m >= 1.
$\leftrightarrow$ _w	The complex argument.

## Returns

The Clausen sine sum  $Sl_m(w)$ ,

Definition at line 1279 of file sf\_polylog.tcc.

References `__polylog_exp()`.

8.3.2.41 `template<typename _Tp> _Tp std::__detail::__clausen_s ( unsigned int _m, _Tp _w )`

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

## See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)

## Parameters

$\leftrightarrow$ _m	The integer order m >= 1.
$\leftrightarrow$ _w	The complex argument.

## Returns

The Clausen sine sum  $Sl_m(w)$ ,

Definition at line 1304 of file sf\_polylog.tcc.

References `__polylog_exp()`.

8.3.2.42 `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

$\leftrightarrow$ _k	The modulus of the complete elliptic function.
-------------------------	--

## Returns

The complete elliptic function of the first kind.

Definition at line 565 of file sf\_ellint.tcc.

References \_\_comp\_ellint\_rf().

Referenced by \_\_ellint\_1(), \_\_ellnome\_k(), \_\_jacobi\_zeta(), \_\_theta\_c(), \_\_theta\_d(), \_\_theta\_n(), and \_\_theta\_s().

**8.3.2.43** `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

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

## Parameters

$\leftrightarrow$ _k	The modulus of the complete elliptic function.
-------------------------	--

## Returns

The complete elliptic function of the second kind.

Definition at line 638 of file sf\_ellint.tcc.

References \_\_ellint\_rd(), and \_\_ellint\_rf().

Referenced by \_\_ellint\_2().

**8.3.2.44** `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

<code>__k</code>	The argument of the elliptic function.
<code>__nu</code>	The second argument of the elliptic function.

## Returns

The complete elliptic function of the third kind.

Definition at line 727 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

**8.3.2.45** `template<typename _Tp> _Tp std::__detail::__comp_ellint_d ( _Tp __k )`

Return the complete Legendre elliptic integral D.

Definition at line 832 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

**8.3.2.46** `template<typename _Tp> _Tp std::__detail::__comp_ellint_rf ( _Tp __x, _Tp __y )`

Definition at line 235 of file `sf_ellint.tcc`.

Referenced by `__comp_ellint_1()`, and `__ellint_rf()`.

**8.3.2.47** `template<typename _Tp> _Tp std::__detail::__comp_ellint_rg ( _Tp __x, _Tp __y )`

Definition at line 346 of file `sf_ellint.tcc`.

Referenced by `__ellint_rg()`.

**8.3.2.48** `template<typename _Tp> _Tp std::__detail::__conf_hyperg ( _Tp __a, _Tp __c, _Tp __x )`

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

## Parameters

<code>__a</code>	The <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

**Returns**

The confluent hypergeometric function.

Definition at line 281 of file sf\_hyperg.tcc.

References `__conf_hyperg_luke()`, and `__conf_hyperg_series()`.

**8.3.2.49** `template<typename _Tp> _Tp std::__detail::__conf_hyperg_lim ( _Tp __c, _Tp __x )`

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

**Parameters**

<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.

**Returns**

The confluent limit hypergeometric function.

Definition at line 109 of file sf\_hyperg.tcc.

References `__conf_hyperg_lim_series()`.

**8.3.2.50** `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**

<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.



**Returns**

The confluent hypergeometric limit function.

Definition at line 76 of file sf\_hyperg.tcc.

Referenced by \_\_conf\_hyperg\_lim().

### 8.3.2.51 `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  ${}_2F_1$  rational approximations, these are probably guaranteed to converge for  $x < 0$ , barring gross numerical instability in the pre-asymptotic regime.

Definition at line 176 of file sf\_hyperg.tcc.

Referenced by \_\_conf\_hyperg().

### 8.3.2.52 `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**

<code>__a</code>	The "numerator" parameter.
<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

**Returns**

The confluent hypergeometric function.

Definition at line 141 of file sf\_hyperg.tcc.

Referenced by \_\_conf\_hyperg().

**8.3.2.53** `template<typename _Tp> _Tp std::__detail::__coshint ( const _Tp __x )`

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

The hyperbolic cosine integral is given by

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

#### Parameters

<code>__x</code>	The argument of the hyperbolic cosine integral function.
------------------	--

#### Returns

The hyperbolic cosine integral.

Definition at line 554 of file `sf_expint.tcc`.

References `__expint_E1()`, and `__expint_Ei()`.

**8.3.2.54** `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	<code>__nu</code>	The order for which the cylindrical Bessel function is evaluated.
in	<code>__z</code>	The argument at which the cylindrical Bessel function is evaluated.

#### Returns

The complex cylindrical Bessel function.

Definition at line 1222 of file `sf_hankel.tcc`.

References `__hankel()`.

**8.3.2.55** `template<typename _Tp> _Tp std::__detail::__cyl_bessel_i ( _Tp __nu, _Tp __x )`

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

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

<code>__nu</code>	The order of the regular modified Bessel function.
<code>__x</code>	The argument of the regular modified Bessel function.

## Returns

The output regular modified Bessel function.

Definition at line 386 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

**8.3.2.56** `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  $\nu$ :  $J_\nu$  or  $I_\nu$  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

<code>__nu</code>	The order of the Bessel function.
<code>__x</code>	The argument of the Bessel function.
<code>__sgn</code>	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.
<code>__max_iter</code>	The maximum number of iterations for sum.

## Returns

The output Bessel function.

Definition at line 413 of file `sf_bessel.tcc`.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

8.3.2.57 `template<typename _Tp > void std::__detail::__cyl_bessel_ik ( _Tp __nu, _Tp __x, _Tp & _Inu, _Tp & _Knu, _Tp & _Ipnu, _Tp & _Kpnu )`

Return the modified cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.

## Parameters

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

Definition at line 316 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__cyl_bessel_ik_steel()`.

Referenced by `__airy()`, `__cyl_bessel_i()`, `__cyl_bessel_k()`, and `__sph_bessel_ik()`.

**8.3.2.58** `template<typename _Tp> void std::__detail::__cyl_bessel_ik_asymp ( _Tp __nu, _Tp __x, _Tp & _Inu, _Tp & _Knu, _Tp & _Ipnu, _Tp & _Kpnu )`

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

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

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

Definition at line 81 of file `sf_mod_bessel.tcc`.

Referenced by `__cyl_bessel_ik()`, and `__cyl_bessel_ik_steel()`.

**8.3.2.59** `template<typename _Tp> void std::__detail::__cyl_bessel_ik_steel ( _Tp __nu, _Tp __x, _Tp & _Inu, _Tp & _Knu, _Tp & _Ipnu, _Tp & _Kpnu )`

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

## Parameters

<code>__nu</code>	The order of the Bessel functions.
-------------------	------------------------------------

## Parameters

<code>__x</code>	The argument of the Bessel functions.
<code>_Inu</code>	The output regular modified Bessel function.
<code>_Knu</code>	The output irregular modified Bessel function.
<code>_Ipnu</code>	The output derivative of the regular modified Bessel function.
<code>_Kpnu</code>	The output derivative of the irregular modified Bessel function.

Definition at line 152 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

8.3.2.60 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_j( _Tp __nu, _Tp __x )`

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

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

<code>__nu</code>	The order of the Bessel function.
<code>__x</code>	The argument of the Bessel function.

## Returns

The output Bessel function.

Definition at line 533 of file `sf_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

8.3.2.61 `template<typename _Tp> void std::__detail::__cyl_bessel_jn( _Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu )`

Return the cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.

Definition at line 452 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__cyl_bessel_jn_stepped()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

**8.3.2.62** `template<typename _Tp > void std::__detail::__cyl_bessel_jn_asymp ( _Tp __nu, _Tp __x, _Tp & _Jnu, _Tp & _Nnu, _Tp & _Jpnu, _Tp & _Npnu )`

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

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

	<code>__nu</code>	The order of the Bessel functions.
	<code>__x</code>	The argument of the Bessel functions.
out	<code>_Jnu</code>	The Bessel function of the first kind.
out	<code>_Nnu</code>	The Neumann function (Bessel function of the second kind).
out	<code>_Jpnu</code>	The Bessel function of the first kind.
out	<code>_Npnu</code>	The Neumann function (Bessel function of the second kind).

Definition at line 79 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_stepped()`.

**8.3.2.63** `template<typename _Tp > void std::__detail::__cyl_bessel_jn_stepped ( _Tp __nu, _Tp __x, _Tp & _Jnu, _Tp & _Nnu, _Tp & _Jpnu, _Tp & _Npnu )`

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

#### Parameters

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

Definition at line 197 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

**8.3.2.64** `template<typename _Tp > _Tp std::__detail::__cyl_bessel_k ( _Tp __nu, _Tp __x )`

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

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:  $\lim_{\nu \rightarrow n}$ . For negative argument we have simply:

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

#### Parameters

<code>__nu</code>	The order of the irregular modified Bessel function.
<code>__x</code>	The argument of the irregular modified Bessel function.

#### Returns

The output irregular modified Bessel function.

Definition at line 424 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

**8.3.2.65** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 ( _Tp __nu, _Tp __x )`

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

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

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

#### Parameters

<code>__nu</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

#### Returns

The output spherical Neumann function.

Definition at line 598 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

**8.3.2.66** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 ( std::complex<_Tp> __nu, std::complex<_Tp> __z )`

Return the complex cylindrical Hankel function of the first kind.



## Parameters

in	__nu	The order for which the cylindrical Hankel function of the first kind is evaluated.
in	__z	The argument at which the cylindrical Hankel function of the first kind is evaluated.

## Returns

The complex cylindrical Hankel function of the first kind.

Definition at line 1190 of file sf\_hankel.tcc.

References `__hankel()`.

**8.3.2.67** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 ( _Tp __nu, _Tp __x )`

Return the cylindrical Hankel function of the second kind  $H_n^{(2)}u(x)$ .

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

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

## 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 633 of file sf\_bessel.tcc.

References `__cyl_bessel_jn()`.

**8.3.2.68** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 ( std::complex<_Tp> __nu, std::complex<_Tp> __z )`

Return the complex cylindrical Hankel function of the second kind.

## Parameters

in	__nu	The order for which the cylindrical Hankel function of the second kind is evaluated.
in	__z	The argument at which the cylindrical Hankel function of the second kind is evaluated.

**Returns**

The complex cylindrical Hankel function of the second kind.

Definition at line 1206 of file sf\_hankel.tcc.

References `__hankel()`.

**8.3.2.69** `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	<code>__nu</code>	The order for which the cylindrical Neumann function is evaluated.
in	<code>__z</code>	The argument at which the cylindrical Neumann function is evaluated.

**Returns**

The complex cylindrical Neumann function.

Definition at line 1238 of file sf\_hankel.tcc.

References `__hankel()`.

**8.3.2.70** `template<typename _Tp > _Tp std::__detail::__cyl_neumann_n ( _Tp __nu, _Tp __x )`

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

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:  $\lim_{\nu \rightarrow n}$ .

**Parameters**

<code>__nu</code>	The order of the Neumann function.
<code>__x</code>	The argument of the Neumann function.

**Returns**

The output Neumann function.

Definition at line 568 of file sf\_bessel.tcc.

References `__cyl_bessel_jn()`.

8.3.2.71 `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

<code>__x</code>	The argument $-\infty < x < \infty$ .
------------------	---------------------------------------

Definition at line 233 of file `sf_dawson.tcc`.

References `__dawson_cont_frac()`, and `__dawson_series()`.

8.3.2.72 `template<typename _Tp> _Tp std::__detail::__dawson_cont_frac ( _Tp __x )`

Compute the Dawson integral using a sampling theorem representation.

**Todo** this needs some compile-time construction!

Definition at line 71 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

8.3.2.73 `template<typename _Tp> _Tp std::__detail::__dawson_series ( _Tp __x )`

Compute the Dawson integral using the series expansion.

Definition at line 47 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

8.3.2.74 `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()`.

### 8.3.2.75 `template<typename _Tp> _Tp std::__detail::__dilog ( _Tp __x )`

Compute the dilogarithm function  $Li_2(x)$  by summation for  $x \leq 1$ .

The Riemann zeta 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:

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

Definition at line 194 of file `sf_zeta.tcc`.

### 8.3.2.76 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta ( std::complex<_Tp> __w )`

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

#### Parameters

<code>__w</code>	The complex (but on-real-axis) argument.
------------------	--

#### Returns

The Dirichlet Beta function of real argument.

#### Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1192 of file `sf_polylog.tcc`.

References `__fpequal()`, and `__polylog()`.

### 8.3.2.77 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta ( _Tp __w )`

Return the Dirichlet beta function for real argument.

## Parameters

$\_w$	The real argument.
-------	--------------------

## Returns

The Dirichlet Beta function of real argument.

Definition at line 1211 of file sf\_polylog.tcc.

References `__polylog()`.

**8.3.2.78** `template<typename _Tp> std::complex<_Tp> std::__detail::__dirichlet_eta ( std::complex<_Tp> __w )`

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

## Parameters

$\_w$	The complex (but on-real-axis) argument.
-------	--

## Returns

The complex Dirichlet eta function.

## Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1155 of file sf\_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

**8.3.2.79** `template<typename _Tp> _Tp std::__detail::__dirichlet_eta ( _Tp __w )`

Return the Dirichlet eta function for real argument.

## Parameters

$\_w$	The real argument.
-------	--------------------

**Returns**

The Dirichlet eta function.

Definition at line 1173 of file sf\_polylog.tcc.

References `__polylog()`.

### 8.3.2.80 `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:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } n-1!! = 10!! = 1$$

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

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

for  $n = -2m - 1$ .

Definition at line 2480 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`.

### 8.3.2.81 `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**

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

**Returns**

The elliptic function of the first kind.

Definition at line 594 of file sf\_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rf()`.

8.3.2.82 `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

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

#### Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

#### Returns

The elliptic function of the second kind.

Definition at line 673 of file `sf_ellint.tcc`.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

8.3.2.83 `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

<code>__k</code>	The argument of the elliptic function.
<code>__nu</code>	The second argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

#### Returns

The elliptic function of the third kind.

Definition at line 768 of file `sf_ellint.tcc`.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

**8.3.2.84** `template<typename _Tp> _Tp std::__detail::__ellint_cel ( _Tp __k_c, _Tp __p, _Tp __a, _Tp __b )`

Return the Bulirsch complete elliptic integrals.

Definition at line 920 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

**8.3.2.85** `template<typename _Tp> _Tp std::__detail::__ellint_d ( _Tp __k, _Tp __phi )`

Return the Legendre elliptic integral D.

Definition at line 809 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

**8.3.2.86** `template<typename _Tp> _Tp std::__detail::__ellint_el1 ( _Tp __x, _Tp __k_c )`

Return the Bulirsch elliptic integrals of the first kind.

Definition at line 848 of file `sf_ellint.tcc`.

References `__ellint_rf()`.

**8.3.2.87** `template<typename _Tp> _Tp std::__detail::__ellint_el2 ( _Tp __x, _Tp __k_c, _Tp __a, _Tp __b )`

Return the Bulirsch elliptic integrals of the second kind.

Definition at line 869 of file `sf_ellint.tcc`.

References `__ellint_rd()`, and `__ellint_rf()`.

**8.3.2.88** `template<typename _Tp> _Tp std::__detail::__ellint_el3 ( _Tp __x, _Tp __k_c, _Tp __p )`

Return the Bulirsch elliptic integrals of the third kind.

Definition at line 894 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

**8.3.2.89** `template<typename _Tp> _Tp std::__detail::__ellint_rc ( _Tp __x, _Tp __y )`

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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



## Parameters

$\_x$	The first argument.
$\_y$	The second argument.

## Returns

The Carlson elliptic function.

Definition at line 81 of file sf\_ellint.tcc.

Referenced by `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.90 `template<typename _Tp> _Tp std::__detail::__ellint_rd ( _Tp __x, _Tp __y, _Tp __z )`

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

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

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

Based on Carlson's algorithms:

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

## Parameters

$\_x$	The first of two symmetric arguments.
$\_y$	The second of two symmetric arguments.
$\_z$	The third argument.

## Returns

The Carlson elliptic function of the second kind.

Definition at line 163 of file sf\_ellint.tcc.

Referenced by `__comp_ellint_2()`, `__comp_ellint_d()`, `__ellint_2()`, `__ellint_d()`, `__ellint_el2()`, `__ellint_rg()`, and `__ellint_rj()`.

**8.3.2.91** `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

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.

#### Returns

The Carlson elliptic function of the first kind.

Definition at line 277 of file `sf_ellint.tcc`.

References `__comp_ellint_rf()`, and `__ellint_rc()`.

Referenced by `__comp_ellint_2()`, `__comp_ellint_3()`, `__ellint_1()`, `__ellint_2()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el1()`, `__ellint_el2()`, `__ellint_el3()`, and `__heuman_lambda()`.

**8.3.2.92** `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:

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

Based on Carlson's algorithms:

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

## Parameters

$\_x$	The first of three symmetric arguments.
$\_y$	The second of three symmetric arguments.
$\_z$	The third of three symmetric arguments.

## Returns

The Carlson symmetric elliptic function of the second kind.

Definition at line 408 of file sf\_ellint.tcc.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

**8.3.2.93** `template<typename _Tp> _Tp std::__detail::__ellint_rj( _Tp __x, _Tp __y, _Tp __z, _Tp __p )`

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

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

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

Based on Carlson's algorithms:

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

## Parameters

$\_x$	The first of three symmetric arguments.
$\_y$	The second of three symmetric arguments.
$\_z$	The third of three symmetric arguments.
$\_p$	The fourth argument.

**Returns**

The Carlson elliptic function of the fourth kind.

Definition at line 456 of file sf\_ellint.tcc.

References `__ellint_rc()`, and `__ellint_rd()`.

Referenced by `__comp_ellint_3()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el3()`, `__heuman_lambda()`, and `__jacobi_zeta()`.

**8.3.2.94 `template<typename _Tp> _Tp std::__detail::__ellnome ( _Tp __k )`**

Return the elliptic nome given the modulus  $k$ .

Definition at line 292 of file sf\_theta.tcc.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

**8.3.2.95 `template<typename _Tp> _Tp std::__detail::__ellnome_k ( _Tp __k )`**

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

Definition at line 278 of file sf\_theta.tcc.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

**8.3.2.96 `template<typename _Tp> _Tp std::__detail::__ellnome_series ( _Tp __k )`**

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

Definition at line 262 of file sf\_theta.tcc.

Referenced by `__ellnome()`.

**8.3.2.97 `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

$\leftrightarrow$ _n	The order of the exponential integral function.
$\leftrightarrow$ _x	The argument of the exponential integral function.

## Returns

The exponential integral.

Definition at line 470 of file sf\_expint.tcc.

References `__expint_E1()`, and `__expint_En_recursion()`.

Referenced by `__logint()`.

**8.3.2.98** `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

$\leftrightarrow$ _x	The argument of the exponential integral function.
-------------------------	--

## Returns

The exponential integral.

Definition at line 510 of file sf\_expint.tcc.

References `__expint_Ei()`.

**8.3.2.99** `template<typename _Tp> _Tp std::__detail::__expint_asymp ( unsigned int __n, _Tp __x )`

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

The exponential integral is given by

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

This is something of an extension.

## Parameters

$\leftrightarrow$ _n	The order of the exponential integral function.
$\leftrightarrow$ _x	The argument of the exponential integral function.

## Returns

The exponential integral.

Definition at line 403 of file sf\_expint.tcc.

8.3.2.100 `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

$\leftrightarrow$ _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 372 of file sf\_expint.tcc.

References `__expint_E1_asymp()`, `__expint_E1_series()`, `__expint_Ei()`, and `__expint_En_cont_frac()`.

Referenced by `__coshint()`, `__expint()`, `__expint_Ei()`, `__expint_En_recursion()`, and `__sinhint()`.

8.3.2.101 `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

$\leftrightarrow$	The argument of the exponential integral function.
$x$	

## Returns

The exponential integral.

Definition at line 111 of file sf\_expint.tcc.

Referenced by \_\_expint\_E1().

8.3.2.102 `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

$\leftrightarrow$	The argument of the exponential integral function.
$x$	

## Returns

The exponential integral.

Definition at line 74 of file sf\_expint.tcc.

Referenced by \_\_expint\_E1().

8.3.2.103 `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**

<code>_↔</code>	The argument of the exponential integral function.
<code>_x</code>	

**Returns**

The exponential integral.

Definition at line 348 of file sf\_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asymp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

**8.3.2.104** `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**

<code>_↔</code>	The argument of the exponential integral function.
<code>_x</code>	

**Returns**

The exponential integral.

Definition at line 315 of file sf\_expint.tcc.

Referenced by `__expint_Ei()`.

**8.3.2.105** `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

$\leftrightarrow$ _x	The argument of the exponential integral function.
-------------------------	--

## Returns

The exponential integral.

Definition at line 283 of file sf\_expint.tcc.

Referenced by \_\_expint\_Ei().

8.3.2.106 `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

$\leftrightarrow$ _n	The order of the exponential integral function.
$\leftrightarrow$ _x	The argument of the exponential integral function.

## Returns

The exponential integral.

Definition at line 193 of file sf\_expint.tcc.

Referenced by \_\_expint\_E1().

8.3.2.107 `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**

$\leftrightarrow$ __n	The order of the exponential integral function.
$\leftrightarrow$ __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 238 of file sf\_expint.tcc.

References \_\_expint\_E1().

Referenced by \_\_expint().

8.3.2.108 `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**

$\leftrightarrow$ __n	The order of the exponential integral function.
$\leftrightarrow$ __x	The argument of the exponential integral function.

**Returns**

The exponential integral.

Definition at line 147 of file sf\_expint.tcc.

References \_\_psi().

8.3.2.109 `template<typename _Tp > _Tp std::__detail::__expint_large_n ( unsigned int __n, _Tp __x )`

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

The exponential integral is given by

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

This is something of an extension.

#### Parameters

$\longleftrightarrow$ _n	The order of the exponential integral function.
$\longleftrightarrow$ _x	The argument of the exponential integral function.

#### Returns

The exponential integral.

Definition at line 437 of file sf\_expint.tcc.

**8.3.2.110** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial ( unsigned int __n )`

Return the factorial of the integer n.

The factorial is:

$$n! = 12\dots(n-1)n, 0! = 1$$

Definition at line 2422 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

**8.3.2.111** `template<typename _Tp> _Tp std::__detail::__fermi_dirac ( _Tp __s, _Tp __x )`

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

#### See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)  
<http://dlmf.nist.gov/25.12#iii>

#### Parameters

$\longleftrightarrow$ _s	The order $s \geq 0$ .
$\longleftrightarrow$ _x	The real argument.

**Returns**

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

Definition at line 1379 of file sf\_polylog.tcc.

References `__polylog_exp()`.

**8.3.2.112** `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf ( _Tp __F, unsigned int __nu1, unsigned int __nu2 )`

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

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

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

**Parameters**

<code>__nu1</code>	The number of degrees of freedom of sample 1
<code>__nu2</code>	The number of degrees of freedom of sample 2
<code>__F</code>	The F statistic

Definition at line 350 of file sf\_beta.tcc.

References `__beta_inc()`.

**8.3.2.113** `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc ( _Tp __F, unsigned int __nu1, unsigned int __nu2 )`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .

The f-distribution propability function is related to the incomplete beta function:

$$P(F|\nu_1, \nu_2) = 1 - I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right) = 1 - Q(F|\nu_1, \nu_2)$$

**Parameters**

<code>__F</code>	
<code>__nu1</code>	
<code>__nu2</code>	

Definition at line 379 of file sf\_beta.tcc.

References `__beta_inc()`.

**8.3.2.114** `template<typename _Tp> void std::__detail::__fock_airy ( _Tp __x, std::complex< _Tp> & __w1, std::complex< _Tp> & __w2, std::complex< _Tp> & __w1p, std::complex< _Tp> & __w2p )`

Compute the Fock-type Airy functions  $w_1(x)$  and  $w_2(x)$  and their first derivatives  $w'_1(x)$  and  $w'_2(x)$  respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

#### Parameters

<code>__x</code>	The argument of the Airy functions.
<code>__w1</code>	The output Fock-type Airy function of the first kind.
<code>__w2</code>	The output Fock-type Airy function of the second kind.
<code>__w1p</code>	The output derivative of the Fock-type Airy function of the first kind.
<code>__w2p</code>	The output derivative of the Fock-type Airy function of the second kind.

Definition at line 580 of file `sf_mod_bessel.tcc`.

**8.3.2.115** `template<typename _Tp> bool std::__detail::__fpequal ( const _Tp & __a, const _Tp & __b )`

A function to reliably compare two floating point numbers.

#### Parameters

<code>__a</code>	the left hand side.
<code>__b</code>	the right hand side

#### Returns

returns true if a and b are equal to zero or differ only by  $\max(a, b) * 5 * eps$

Definition at line 62 of file `sf_polylog.tcc`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, `__fpimag()`, `__fpreal()`, `__polylog()`, `__polylog_exp_asymp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_neg_even()`, `__polylog_exp_neg_odd()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__polylog_exp_real_pos()`.

**8.3.2.116** `template<typename _Tp> bool std::__detail::__fpimag ( const std::complex< _Tp> & __w )`

A function to reliably test a complex number for imaginarity [?].

## Parameters

$\_w$	The complex argument.
-------	-----------------------

## Returns

`true` if  $Re(w)$  is zero within  $5 * \epsilon$ , `false` otherwise.

Definition at line 107 of file `sf_polylog.tcc`.

References `__fpequal()`.

8.3.2.117 `template<typename _Tp> bool std::__detail::__fpimag ( const _Tp )`

Definition at line 117 of file `sf_polylog.tcc`.

8.3.2.118 `template<typename _Tp> bool std::__detail::__fpreal ( const std::complex< _Tp > & _w )`

A function to reliably test a complex number for realness.

## Parameters

$\_w$	The complex argument.
-------	-----------------------

## Returns

`true` if  $Im(w)$  is zero within  $5 * \epsilon$ , `false` otherwise.

Definition at line 84 of file `sf_polylog.tcc`.

References `__fpequal()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

8.3.2.119 `template<typename _Tp> bool std::__detail::__fpreal ( const _Tp )`

Definition at line 94 of file `sf_polylog.tcc`.

8.3.2.120 `template<typename _Tp> std::complex<_Tp> std::__detail::__fresnel ( const _Tp __x )`

Return the Fresnel cosine and sine integrals as a complex number  $C(x) + iS(x)$ .

The Fresnel cosine integral is defined by:

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right)dt$$

The Fresnel sine integral is defined by:

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$$

#### Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 166 of file `sf_fresnel.tcc`.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

8.3.2.121 `template<typename _Tp> void std::__detail::__fresnel_cont_frac ( const _Tp __ax, _Tp &_Cf, _Tp &_Sf )`

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

Definition at line 105 of file `sf_fresnel.tcc`.

Referenced by `__fresnel()`.

8.3.2.122 `template<typename _Tp> void std::__detail::__fresnel_series ( const _Tp __ax, _Tp &_Cf, _Tp &_Sf )`

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

Definition at line 48 of file `sf_fresnel.tcc`.

Referenced by `__fresnel()`.

8.3.2.123 `template<typename _Tp> _Tp std::__detail::__gamma ( _Tp __x )`

Return  $\Gamma(x)$ .

#### Parameters

<code>__x</code>	The argument of the gamma function.
------------------	-------------------------------------

**Returns**

The gamma function.

Definition at line 1918 of file sf\_gamma.tcc.

References `__log_gamma()`.

Referenced by `__beta_gamma()`, and `__riemann_zeta()`.

**8.3.2.124** `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac ( _Tp __a, _Tp __x )`

Definition at line 1964 of file sf\_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma_l()`, `__gamma_u()`, `__pgamma()`, and `__qgamma()`.

**8.3.2.125** `template<typename _Tp> _Tp std::__detail::__gamma_l ( _Tp __a, _Tp __x )`

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2070 of file sf\_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

**8.3.2.126** `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_series ( _Tp __a, _Tp __x )`

Definition at line 1930 of file sf\_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma_l()`, `__gamma_u()`, `__pgamma()`, and `__qgamma()`.

**8.3.2.127** `template<typename _Tp> void std::__detail::__gamma_temme ( _Tp __mu, _Tp & __gam1, _Tp & __gam2, _Tp & __gampl, _Tp & __gammi )`

Compute the gamma functions required by the Temme series expansions of  $N_\nu(x)$  and  $K_\nu(x)$ .

$$\Gamma_1 = \frac{1}{2\mu} \left[ \frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[ \frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where  $-1/2 \leq \mu \leq 1/2$  is  $\mu = \nu - N$  and  $N$  is the nearest integer to  $\nu$ . The values of  $\Gamma(1+\mu)$  and  $\Gamma(1-\mu)$  are returned as well.

The accuracy requirements on this are exquisite.



## Parameters

	<code>__mu</code>	The input parameter of the gamma functions.
out	<code>__gam1</code>	The output function $\Gamma_1(\mu)$
out	<code>__gam2</code>	The output function $\Gamma_2(\mu)$
out	<code>__gampl</code>	The output function $\Gamma(1 + \mu)$
out	<code>__gammi</code>	The output function $\Gamma(1 - \mu)$

Definition at line 163 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

### 8.3.2.128 `template<typename _Tp> _Tp std::__detail::__gamma_u ( _Tp __a, _Tp __x )`

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

Definition at line 2102 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

### 8.3.2.129 `template<typename _Tp> _Tp std::__detail::__gauss ( _Tp __x )`

The CDF of the normal distribution. i.e. the integrated lower tail of the normal PDF.

Definition at line 70 of file `sf_owens_t.tcc`.

### 8.3.2.130 `template<typename _Tp> _Tp std::__detail::__gegenbauer_poly ( unsigned int __n, _Tp __alpha, _Tp __x )`

Return the Gegenbauer polynomial  $C_n^\alpha(x)$  of degree  $n$  and real order  $\alpha$  and argument  $x$ .

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^\alpha(x) = \frac{1}{n} [2x(n + \alpha - 1)C_{n-1}^\alpha(x) - (n + 2\alpha - 2)C_{n-2}^\alpha(x)]$$

and  $C_0^\alpha(x) = 1$ ,  $C_1^\alpha(x) = 2\alpha x$ .

## Template Parameters

<code>_Talpha</code>	The real type of the order
<code>_Tp</code>	The real type of the argument

## Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__x</code>	The real argument

Definition at line 61 of file `sf_gegenbauer.tcc`.

```
8.3.2.131  template<typename _Tp> void std::__detail::__hankel ( std::complex< _Tp> __nu, std::complex< _Tp> __z,
std::complex< _Tp> & _H1, std::complex< _Tp> & _H2, std::complex< _Tp> & _H1p, std::complex< _Tp> & _H2p
)
```

## Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 1127 of file `sf_hankel.tcc`.

References `__debye_region()`, `__hankel_debye()`, and `__hankel_uniform()`.

Referenced by `__cyl_bessel()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann()`, and `__sph_hankel()`.

```
8.3.2.132  template<typename _Tp> void std::__detail::__hankel_debye ( std::complex< _Tp> __nu, std::complex< _Tp> __z,
std::complex< _Tp> __alpha, int __indexr, char & __aorb, int & __morn, std::complex< _Tp> & _H1, std::complex<
_Tp> & _H2, std::complex< _Tp> & _H1p, std::complex< _Tp> & _H2p )
```

## Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
in	<code>__alpha</code>	
in	<code>__indexr</code>	
out	<code>__aorb</code>	
out	<code>__morn</code>	
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 959 of file sf\_hankel.tcc.

Referenced by `__hankel()`.

```
8.3.2.133 template<typename _Tp> void std::__detail::__hankel_params ( std::complex<_Tp> __nu, std::complex<_Tp>
    __zhat, std::complex<_Tp> & __p, std::complex<_Tp> & __p2, std::complex<_Tp> & __nup2, std::complex<
    _Tp> & __num2, std::complex<_Tp> & __num1d3, std::complex<_Tp> & __num2d3, std::complex<_Tp> &
    __num4d3, std::complex<_Tp> & __zeta, std::complex<_Tp> & __zetaphf, std::complex<_Tp> & __zetamhf,
    std::complex<_Tp> & __zetam3hf, std::complex<_Tp> & __zetrat )
```

Compute parameters depending on `z` and `nu` that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Definition at line 110 of file sf\_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

```
8.3.2.134 template<typename _Tp> void std::__detail::__hankel_uniform ( std::complex<_Tp> __nu, std::complex<_Tp>
    __z, std::complex<_Tp> & __H1, std::complex<_Tp> & __H2, std::complex<_Tp> & __H1p, std::complex<_Tp> &
    __H2p )
```

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

#### Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>__H1</code>	The Hankel function of the first kind.
out	<code>__H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>__H2</code>	The Hankel function of the second kind.
out	<code>__H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 904 of file sf\_hankel.tcc.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

```
8.3.2.135 template<typename _Tp> void std::__detail::__hankel_uniform_olver ( std::complex<_Tp> __nu, std::complex<_Tp>
    > __z, std::complex<_Tp> & __H1, std::complex<_Tp> & __H2, std::complex<_Tp> & __H1p, std::complex<_Tp>
    > & __H2p )
```

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

## Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 818 of file `sf_hankel.tcc`.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

```
8.3.2.136 template<typename _Tp> void std::__detail::__hankel_uniform_outer ( std::complex<_Tp> __nu, std::complex<_Tp> __z, _Tp __eps, std::complex<_Tp> & __zhat, std::complex<_Tp> & __1dnsq, std::complex<_Tp> & __num1d3, std::complex<_Tp> & __num2d3, std::complex<_Tp> & __p, std::complex<_Tp> & __p2, std::complex<_Tp> & __etm3h, std::complex<_Tp> & __etrat, std::complex<_Tp> & __Aip, std::complex<_Tp> & __o4dp, std::complex<_Tp> & __Aim, std::complex<_Tp> & __o4dm, std::complex<_Tp> & __od2p, std::complex<_Tp> & __od0dp, std::complex<_Tp> & __od2m, std::complex<_Tp> & __od0dm )
```

Compute outer factors and associated functions of `z` and `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of `z` and `nu` returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Definition at line 273 of file `sf_hankel.tcc`.

References `__airy()`, `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

```
8.3.2.137 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	<code>__p</code>	
in	<code>__p2</code>	
in	<code>__num2</code>	

## Parameters

in	<code>__zetam3hf</code>	
in	<code>_Aip</code>	The Airy function value $Ai()$ .
in	<code>__o4dp</code>	
in	<code>_Aim</code>	The Airy function value $Ai()$ .
in	<code>__o4dm</code>	
in	<code>__od2p</code>	
in	<code>__od0dp</code>	
in	<code>__od2m</code>	
in	<code>__od0dm</code>	
in	<code>__eps</code>	The error tolerance
out	<code>_H1sum</code>	The Hankel function of the first kind.
out	<code>_H1psum</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2sum</code>	The Hankel function of the second kind.
out	<code>_H2psum</code>	The derivative of the Hankel function of the second kind.

Definition at line 351 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_olver()`.

### 8.3.2.138 `template<typename _Tp> _Tp std::__detail::__heuman_lambda ( _Tp __k, _Tp __phi )`

Return the Heuman lambda function.

Definition at line 941 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

### 8.3.2.139 `template<typename _Tp> _Tp std::__detail::__hurwitz_zeta ( _Tp __s, _Tp __a )`

Return the Hurwitz zeta function  $\zeta(s, a)$  for all  $s \neq 1$  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

<code>__s</code>	The argument $s \neq 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 702 of file sf\_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`.

8.3.2.140 `template<typename _Tp> std::complex<_Tp> std::__detail::__hurwitz_zeta ( _Tp __s, std::complex<_Tp> __a )`

Return the Hurwitz Zeta function for real  $s$  and complex  $a$ .

#### Parameters

<code>__s</code>	The real argument
<code>__a</code>	The complex parameter

**Todo** This `__hurwitz_zeta` prefactor is prone to overflow. positive integer orders  $s$ ?

Definition at line 1119 of file sf\_polylog.tcc.

References `__polylog_exp()`.

Referenced by `__psi()`.

8.3.2.141 `template<typename _Tp> _Tp std::__detail::__hurwitz_zeta_euler_maclaurin ( _Tp __s, _Tp __a )`

Return the Hurwitz zeta function  $\zeta(s, a)$  for all  $s \neq 1$  and  $a > -1$ .

#### See also

An efficient algorithm for accelerating the convergence of oscillatory series, useful for computing the polylogarithm and Hurwitz zeta functions, Linas Vep

#### Parameters

<code>__s</code>	The argument $s \neq 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 560 of file sf\_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

Referenced by `__hurwitz_zeta()`.

8.3.2.142 `template<typename _Tp> std::complex<_Tp> std::__detail::__hydrogen ( const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi )`

Definition at line 44 of file sf\_hydrogen.tcc.

References `__assoc_laguerre()`, `__psi()`, and `__sph_legendre()`.

8.3.2.143 `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

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

#### Returns

The confluent hypergeometric function.

Definition at line 776 of file sf\_hyperg.tcc.

References `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

8.3.2.144 `template<typename _Tp> _Tp std::__detail::__hyperg_luke ( _Tp __a, _Tp __b, _Tp __c, _Tp __xin )`

Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Definition at line 352 of file sf\_hyperg.tcc.

Referenced by `__hyperg()`.

8.3.2.145 `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  $a, b, c \neq$  negative integer.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

The reflection formula for nonintegral  $d = c - a - b$  is:

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)\Gamma(d)}{\Gamma(c-a)\Gamma(c-b)} {}_2F_1(a, b; 1-d; 1-x) + \frac{\Gamma(c)\Gamma(-d)}{\Gamma(a)\Gamma(b)} {}_2F_1(c-a, c-b; 1+d; 1-x)$$

The reflection formula for integral  $m = c - a - b$  is:

$${}_2F_1(a, b; a+b+m; x) = \frac{\Gamma(m)\Gamma(a+b+m)}{\Gamma(a+m)\Gamma(b+m)} \sum_{k=0}^{m-1} \frac{(m+a)_k(m+b)_k}{k!(1-m)_k} -$$

Definition at line 486 of file `sf_hyperg.tcc`.

References `__hyperg_series()`, `__log_gamma()`, `__log_gamma_sign()`, and `__psi()`.

Referenced by `__hyperg()`.

8.3.2.146 `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

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.



**Returns**

The confluent hypergeometric function.

Definition at line 321 of file sf\_hyperg.tcc.

Referenced by \_\_hyperg(), and \_\_hyperg\_reflect().

**8.3.2.147** `template<typename _Tp> std::tuple<_Tp, _Tp, _Tp> std::__detail::__jacobi_sncndn ( _Tp __k, _Tp __u )`

Return a tuple of the three primary Jacobi elliptic functions:  $sn(k, u)$ ,  $cn(k, u)$ ,  $dn(k, u)$ .

Definition at line 414 of file sf\_theta.tcc.

**8.3.2.148** `template<typename _Tp> _Tp std::__detail::__jacobi_zeta ( _Tp __k, _Tp __phi )`

Return the Jacobi zeta function.

Definition at line 971 of file sf\_ellint.tcc.

References \_\_comp\_ellint\_1(), and \_\_ellint\_rj().

**8.3.2.149** `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**

<code>__n</code>	The order of the Laguerre polynomial.
<code>__x</code>	The argument of the Laguerre polynomial.

**Returns**

The value of the Laguerre polynomial of order n and argument x.

Definition at line 321 of file sf\_laguerre.tcc.

8.3.2.150 `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

<code>__l</code>	The order of the Legendre function. $l \geq 0$ .
<code>__x</code>	The argument of the Legendre function. $ x  \leq 1$ .

Definition at line 123 of file `sf_legendre.tcc`.

8.3.2.151 `template<typename _Tp> _Tp std::__detail::__log_bincoef ( unsigned int __n, unsigned int __k )`

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

#### Parameters

<code>__n</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

#### Returns

The logarithm of the binomial coefficient.

Definition at line 1862 of file `sf_gamma.tcc`.

8.3.2.152 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial ( _Tp __x )`

Definition at line 2450 of file `sf_gamma.tcc`.

References `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

8.3.2.153 `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:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } -1!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m-1)!! = 1/(1(-1)(-3)\dots(-2m+1)(-2m-1)) = \frac{(-1)^m}{(2m-1)!!}$$

for  $n = -2m - 1$ .

Definition at line 2516 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`.

8.3.2.154 `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:

$$n! = 12\dots(n-1)n, 0! = 1$$

Definition at line 2440 of file `sf_gamma.tcc`.

References `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

8.3.2.155 `template<typename _Tp> _Tp std::__detail::__log_gamma ( _Tp __x )`

Return  $\log(|\Gamma(x)|)$ . This will return values even for  $x < 0$ . To recover the sign of  $\Gamma(x)$  for any argument use `__log_gamma_sign`.

#### Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

#### Returns

The logarithm of the gamma function.

Definition at line 1800 of file `sf_gamma.tcc`.

References `__log_gamma_lanczos()`.

Referenced by `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__gamma()`, `__hyperg()`, `__hyperg_reflect()`, `__log_↵double_factorial()`, `__log_factorial()`, `__log_pochhammer_u()`, `__poly_laguerre_large_n()`, `__psi()`, `__riemann_zeta()`, `↵__riemann_zeta_glob()`, and `__sph_legendre()`.

**8.3.2.156** `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

<code>↵__x</code>	The argument of the log of the gamma function.
-------------------	--

#### Returns

The logarithm of the gamma function.

Definition at line 1699 of file `sf_gamma.tcc`.

**8.3.2.157** `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos ( _Tp __x )`

Return  $\log(\Gamma(x))$  by the Lanczos method. This method dominates all others on the positive axis I think.

#### Parameters

<code>↵__x</code>	The argument of the log of the gamma function.
-------------------	--

#### Returns

The logarithm of the gamma function.

Definition at line 1755 of file `sf_gamma.tcc`.

Referenced by `__log_gamma()`.

**8.3.2.158** `template<typename _Tp > _Tp std::__detail::__log_gamma_sign ( _Tp __x )`

Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned.

#### Parameters

<code>↵__x</code>	The argument of the gamma function.
-------------------	-------------------------------------

**Returns**

The sign of the gamma function.

Definition at line 1831 of file sf\_gamma.tcc.

Referenced by \_\_hyperg(), \_\_hyperg\_reflect(), and \_\_pochhammer\_l().

**8.3.2.159** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge ( _Tp __z )`

Return  $\Gamma(z)$  by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[ \sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

**See also**

Spouge, J.L., Computation of the gamma, digamma, and trigamma functions. SIAM Journal on Numerical Analysis 31, 3 (1994), pp. 931-944

**Parameters**

<code>__z</code>	The argument of the gamma function.
------------------	-------------------------------------

**Returns**

The the gamma function.

Definition at line 1739 of file sf\_gamma.tcc.

**8.3.2.160** `template<typename _Tp> _Tp std::__detail::__log_pochhammer_l ( _Tp __a, _Tp __n )`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular,  $\ln[(a)_n] = \ln(n!) - \ln(a)$ . Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

Definition at line 2209 of file `sf_gamma.tcc`.

**8.3.2.161** `template<typename _Tp> _Tp std::__detail::__log_pochhammer_u( _Tp __a, _Tp __n )`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(a), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[ \begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 2144 of file `sf_gamma.tcc`.

References `__log_gamma()`.

**8.3.2.162** `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 531 of file sf\_expint.tcc.

References `__expint()`.

**8.3.2.163** `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	$\leftrightarrow$ __h	The scale parameter.
in	$\leftrightarrow$ __a	The integration limit.

**Returns**

The owens T function.

Definition at line 92 of file sf\_owens\_t.tcc.

References `__znorm1()`, and `__znorm2()`.

**8.3.2.164** `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 2013 of file sf\_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

8.3.2.165 `template<typename _Tp> _Tp std::__detail::__pochhammer_l ( _Tp __a, _Tp __n )`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular,  $(n)_n = n!$ .

Definition at line 2232 of file `sf_gamma.tcc`.

References `__log_gamma_sign()`.

8.3.2.166 `template<typename _Tp> _Tp std::__detail::__pochhammer_u ( _Tp __a, _Tp __n )`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

Definition at line 2170 of file `sf_gamma.tcc`.

8.3.2.167 `template<typename _Tp> _Tp std::__detail::__poly_hermite ( unsigned int __n, _Tp __x )`

This routine returns the Hermite polynomial of order  $n$ :  $H_n(x)$ .

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

The Hermite polynomial obeys a reflection formula:

$$H_n(-x) = (-1)^n H_n(x)$$

#### Parameters

$\longleftrightarrow$ <code>__n</code>	The order of the Hermite polynomial.
$\longleftrightarrow$ <code>__x</code>	The argument of the Hermite polynomial.



**Returns**

The value of the Hermite polynomial of order n and argument x.

Definition at line 179 of file sf\_hermite.tcc.

References `__poly_hermite_asymp()`, and `__poly_hermite_recursion()`.

**8.3.2.168** `template<typename _Tp> _Tp std::__detail::__poly_hermite_asymp ( unsigned int __n, _Tp __x )`

This routine returns the Hermite polynomial of large order n:  $H_n(x)$ . We assume here that  $x \geq 0$ .

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

see "Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, arXiv:math/0601078v1 [math.CA] 4 Jan 2006

**Parameters**

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

**Returns**

The value of the Hermite polynomial of order n and argument x.

Definition at line 113 of file sf\_hermite.tcc.

References `__airy()`.

Referenced by `__poly_hermite()`.

**8.3.2.169** `template<typename _Tp> _Tp std::__detail::__poly_hermite_recursion ( unsigned int __n, _Tp __x )`

This routine returns the Hermite polynomial of order n:  $H_n(x)$  by recursion on n.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

**Parameters**

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

**Returns**

The value of the Hermite polynomial of order  $n$  and argument  $x$ .

Definition at line 69 of file `sf_hermite.tcc`.

Referenced by `__poly_hermite()`.

**8.3.2.170** `template<typename _Tp> _Tp std::__detail::__poly_jacobi ( unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x )`

Compute the Jacobi polynomial by recursion on  $n$  :

$$2n(\alpha+\beta+n)(\alpha+\beta+2n-2)P_n^{(\alpha,\beta)}(x) = (\alpha+\beta+2n-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2n-2)(\alpha+\beta+2n))P_{n-1}^{(\alpha,\beta)}(x) - 2(\alpha+n-1)(\beta+n-1)(\alpha+$$

Definition at line 57 of file `sf_jacobi.tcc`.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

**8.3.2.171** `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre ( unsigned int __n, _Tpa __alpha1, _Tp __x )`

This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha$ :  $L_n^{alpha}(x)$ .

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**

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

## Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

## Returns

The value of the Laguerre function of order  $n$ , degree  $\alpha$ , and argument  $x$ .

Definition at line 248 of file `sf_laguerre.tcc`.

References `__poly_laguerre_hyperg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

**8.3.2.172** `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_hyperg ( unsigned int __n, _Tpa __alpha1, _Tp __x )`

Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where  $(\alpha)_n$  is the Pochhammer symbol and  ${}_1F_1(a; c; x)$  is the confluent hypergeometric function.

This function assumes  $x \neq 0$ .

This is from the GNU Scientific Library.

## Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

## Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

## Returns

The value of the Laguerre function of order  $n$ , degree  $\alpha$ , and argument  $x$ .

Definition at line 129 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

8.3.2.173 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_large_n ( unsigned __n, _Tpa __alpha1, _Tp __x )`

This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha > -1$  for large  $n$ . Abramowitz & Stegun, 13.5.21.

#### Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

#### Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

#### Returns

The value of the Laguerre function of order  $n$ , degree  $\alpha$ , and argument  $x$ .

This is from the GNU Scientific Library.

Definition at line 72 of file `sf_laguerre.tcc`.

References `__log_gamma()`.

Referenced by `__poly_laguerre()`.

8.3.2.174 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_recursion ( unsigned int __n, _Tpa __alpha1, _Tp __x )`

This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha$ :  $L_n^\alpha(x)$  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

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

## Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

## Returns

The value of the Laguerre function of order  $n$ , degree  $\alpha$ , and argument  $x$ .

Definition at line 187 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

8.3.2.175 `template<typename _Tp> _Tp std::__detail::__poly_legendre_p ( unsigned int __l, _Tp __x )`

Return the Legendre polynomial by upward recursion on order  $l$ .

The Legendre function of order  $l$  and argument  $x$ ,  $P_l(x)$ , is defined by:

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l$$

## Parameters

<code>__l</code>	The order of the Legendre polynomial. $l \geq 0$ .
<code>__x</code>	The argument of the Legendre polynomial. $ x  \leq 1$ .

Definition at line 73 of file `sf_legendre.tcc`.

Referenced by `__assoc_legendre_p()`, and `__sph_legendre()`.

8.3.2.176 `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  $m \leq n$ , and real radial argument  $\rho$ .

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k! (\frac{n+m}{2} - k)! (\frac{n-m}{2} - k)!} \rho^{n-2k}$$

for  $n - m$  even and identically 0 for  $n - m$  odd. The radial polynomials can be related to the Zernike polynomials:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative  $m, n$ .

See also

[zernike](#) for details on the Zernike polynomials.

#### Template Parameters

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

#### Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The non-negative azimuthal order
<code>__rho</code>	The radial argument

Definition at line 138 of file `sf_jacobi.tcc`.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyl()`.

**8.3.2.177** `template<typename _Tp> _Tp std::__detail::__polylog ( _Tp __s, _Tp __x )`

Return the polylog  $\text{Li}_s(x)$  for two real arguments.

#### Parameters

<code>__s</code>	The real index.
<code>__x</code>	The real argument.

#### Returns

The complex value of the polylogarithm.

Definition at line 1072 of file sf\_polylog.tcc.

References `__fpequal()`, and `__polylog_exp()`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, and `__polylog()`.

**8.3.2.178** `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

<code>__s</code>	The real index.
<code>__w</code>	The complex argument.

#### Returns

The complex value of the polylogarithm.

Definition at line 1102 of file sf\_polylog.tcc.

References `__fpequal()`, `__polylog()`, and `__polylog_exp()`.

**8.3.2.179** `template<typename _Tp, typename ArgType> __gnu_cxx::__promote_num_t<std::complex<_Tp>, ArgType>  
std::__detail::__polylog_exp ( _Tp __s, ArgType __w )`

This is the frontend function which calculates  $Li_s(e^w)$ . First we branch into different parts depending on the properties of  $s$ . This function is the same irrespective of a real or complex  $w$ , hence the template parameter `ArgType`.

#### Note

: I *really* wish we could return a `variant<Tp, std::complex<Tp>>`.

#### Parameters

<code>__s</code>	The real order.
<code>__w</code>	The real or complex argument.

#### Returns

The real or complex value of  $Li_s(e^w)$ .

Definition at line 1039 of file sf\_polylog.tcc.

References `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_↵_real_neg()`, and `__polylog_exp_real_pos()`.

Referenced by `__bose_einstein()`, `__clausen()`, `__clausen_c()`, `__clausen_s()`, `__fermi_dirac()`, `__hurwitz_zeta()`, and `__polylog()`.

**8.3.2.180** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_asymp ( _Tp __s, std::complex<_Tp> ↵ __w )`

This function implements the asymptotic series for the polylog. It is given by

$$2 \sum_{k=0}^{\infty} \zeta(2k) w^{s-2k} / \Gamma(s-2k+1) - i\pi w^{(s-1)} / \Gamma(s)$$

for  $Re(w) >> 1$

Don't check this against Mathematica 8. For real u the imaginary part of the polylog is given by  $Im(Li_s(e^u)) = -\pi u^{s-1} / \Gamma(s)$  Check this relation for any benchmark that you use. The use of `evenzeta` leads to a speedup of about 1000.

#### Parameters

<code>↵ _s</code>	the real index s.
<code>↵ _w</code>	the large complex argument w.

#### Returns

the value of the polylogarithm.

Definition at line 686 of file sf\_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↵_real_pos()`.

**8.3.2.181** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg ( int __s, std::complex<_Tp> ↵ __w )`

This treats the case where s is a negative integer.



## Parameters

$\_s$	a negative integer.
$\_w$	an arbitrary complex number

## Returns

the value of the polylogarithm.

Definition at line 856 of file sf\_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

Referenced by `__polylog_exp()`.

8.3.2.182 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg ( const int __s, _Tp __w )`

This treats the case where `s` is a negative integer and `w` is a real.

## Parameters

$\_s$	a negative integer.
$\_w$	the argument.

## Returns

the value of the polylogarithm.

Definition at line 898 of file sf\_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

8.3.2.183 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos ( unsigned int __s, std::complex<_Tp> __w )`

Here `s` is a positive integer and the function descends into the different kernels depending on `w`.

## Parameters

$\_s$	a positive integer.
$\_w$	an arbitrary complex number.

**Returns**

The value of the polylogarithm.

Definition at line 767 of file sf\_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_↵negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

**8.3.2.184** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos ( unsigned int __s, _Tp __w )`

Here `s` is a positive integer and the function descends into the different kernels depending on `w`.

**Parameters**

<code>↵__s</code>	a positive integer
<code>↵__w</code>	an arbitrary real argument <code>w</code>

**Returns**

the value of the polylogarithm.

Definition at line 815 of file sf\_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `↵riemann_zeta()`.

**8.3.2.185** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg ( _Tp __s, std::complex<_Tp> __w )`

This function treats the cases of negative real index `s`. Theoretical convergence is present for  $|w| < 2\pi$ . We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^{(s-1)} + (2\pi)^{(s-1)}/\pi A_p(w)$$

$$A_p(w) = \sum_k \Gamma(1+k-s)/k! \sin(\pi/2 * (s-k))(w/2/\pi)^k \zeta(1+k-s)$$

**Parameters**

<code>↵__s</code>	The real index
<code>↵__w</code>	The complex argument

## Returns

The value of the polylogarithm.

Definition at line 346 of file sf\_polylog.tcc.

References `__fpequal()`, `__riemann_zeta()`, and `__riemann_zeta_m_1()`.

Referenced by `__polylog_exp_int_neg()`, and `__polylog_exp_real_neg()`.

**8.3.2.186** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg ( int __s, std::complex<_Tp> __w )`

This function treats the cases of negative integer index s and branches accordingly

## Parameters

<code>__s</code>	the integer index s.
<code>__w</code>	The Argument w

## Returns

The value of the Polylogarithm evaluated by a suitable function.

Definition at line 564 of file sf\_polylog.tcc.

References `__polylog_exp_neg_even()`, and `__polylog_exp_neg_odd()`.

**8.3.2.187** `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_even ( unsigned int __n, std::complex<_Tp> __w )`

This function treats the cases of negative integer index s which are multiples of two.

In that case the sine occuring in the expansion occasionally takes on the value zero. We use that to provide an optimized series for  $p = 2n$ :

In the template parameter sigma we transport whether  $p = 4k$  ( $\sigma = 1$ ) or  $p = 4k + 2$  ( $\sigma = -1$ )

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}(-p)!/(2\pi)^{(p-1)/2}(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi/w))$$

and

$$B_p(w) = -2(2\pi)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} (\zeta(2+2k-p) - 1)$$

This is suitable for  $|w| < 2\pi$  The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - \sigma(2\pi)^p/\pi \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} \zeta(2+2k-p)$$

## Parameters

$\leftrightarrow$ _n	the integral index $n = 4k$ .
$\leftrightarrow$ _w	The complex argument w

## Returns

the value of the Polylogarithm.

Definition at line 450 of file sf\_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

**8.3.2.188** `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_odd ( unsigned int __n, std::complex<_Tp> __w )`

This function treats the cases of negative integer index s which are odd.

In that case the sine occuring in the expansion occasionally vanishes. We use that to provide an optimized series for  $p = 1 + 2k$ : In the template parameter sigma we transport whether  $p = 1 + 4k$  ( $\sigma = 1$ ) or  $p = 3 + 4k$  ( $\sigma = -1$ )

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} + \sigma * A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}\Gamma(1-p)(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi i/w))$$

and

$$B_p(w) = 2(2\pi i)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-w^2/4/\pi^2)^k (\zeta(1+2k-p) - T p1)$$

This is suitable for  $|w| < 2\pi$ . The use of evenzeta gives a speedup of about 50 The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p) * (-w)^{p-1} - \sigma 2(2\pi)^{(p-1)} * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-1)^k (w/2/\pi)^{(2k)} \zeta(1+2k-p)$$

## Parameters

$\leftrightarrow$ _n	the integral index n = 4k.
$\leftrightarrow$ _w	The complex argument w.

## Returns

The value of the Polylogarithm.

Definition at line 517 of file sf\_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

8.3.2.189 `template<typename _PowTp, typename _Tp> _Tp std::__detail::__polylog_exp_negative_real_part ( _PowTp __s, _Tp __w )`

Theoretical convergence for  $\text{Re}(w) < 0$ .

Seems to beat the other expansions for  $\text{Re}(w) < -\pi/2 - \pi/5$ . Note that this is an implementation of the basic series:

$$Li_s(e^z) = \sum_{k=1} e^{(k * z) * k^{-s}}$$

## Parameters

<code>__s</code>	is an arbitrary type, integral or float.
<code>__w</code>	something with a negative real part.

## Returns

the value of the polylogarithm.

Definition at line 737 of file sf\_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

8.3.2.190 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos ( unsigned int __s, std::complex<_Tp> __w )`

This function treats the cases of positive integer index  $s$ .

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{(s-1)} / (s-1)!$$

The radius of convergence is  $|w| < 2\pi i$ . 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**

$\leftrightarrow$ _s	the index s.
$\leftrightarrow$ _w	the argument w.

**Returns**

the value of the polylogarithm.

Definition at line 206 of file sf\_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

8.3.2.191 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos ( unsigned int __s, _Tp __w )`

This function treats the cases of positive integer index s for real w.

This specialization is worthwhile to catch the differing behaviour of  $\log(x)$ .

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{s-1} / (s-1)!$$

The radius of convergence is  $|w| < 2\pi$ . Note that this series involves a  $\log(-x)$ . The use of `evenzeta` yields a speedup of about 2.5. gcc and Mathematica differ in their implementation of  $\log(e^{i\pi})$ : gcc:  $\log(e^{i\pi}) = + - i\pi$  whereas Mathematica doesn't preserve the sign in this case:  $\log(e^{i\pi}) = + i\pi$

**Parameters**

$\leftrightarrow$ _s	the index.
$\leftrightarrow$ _w	the argument

**Returns**

the value of the Polylogarithm

Definition at line 279 of file sf\_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

8.3.2.192 `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

<code>__s</code>	the positive real index s.
<code>__w</code>	The complex argument w.

#### Returns

the value of the polylogarithm.

Definition at line 603 of file sf\_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

8.3.2.193 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg ( _Tp __s, std::complex<_Tp> __w )`

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

#### Parameters

<code>__s</code>	A negative real value that does not reduce to a negative integer.
<code>__w</code>	The complex argument.

#### Returns

The value of the polylogarithm.

Definition at line 985 of file sf\_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_↔negative_real_part()`.

Referenced by `__polylog_exp()`.

**8.3.2.194** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg ( _Tp __s, _Tp __w )`

Return the polylog where `s` is a negative real value and for real argument. Now we branch depending on the properties of `w` in the specific functions.

#### Parameters

<code>__↔ _s</code>	A negative real value.
<code>__↔ _w</code>	A real argument.

#### Returns

The value of the polylogarithm.

Definition at line 1013 of file `sf_polylog.tcc`.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

**8.3.2.195** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos ( _Tp __s, std::complex<_Tp> __w )`

Return the polylog where `s` is a positive real value and for complex argument.

#### Parameters

<code>__↔ _s</code>	A positive real number.
<code>__↔ _w</code>	the complex argument.

#### Returns

The value of the polylogarithm.

Definition at line 922 of file `sf_polylog.tcc`.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_↔negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.



8.3.2.196 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos ( _Tp __s, _Tp __w )`

Return the polylog where s is a positive real value and the argument is real.

#### Parameters

<code>__s</code>	A positive real number tht does not reduce to an integer.
<code>__w</code>	The real argument w.

#### Returns

The value of the polylogarithm.

Definition at line 956 of file sf\_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

8.3.2.197 `template<typename _Tp> _Tp std::__detail::__psi ( _Tp __x )`

Return the digamma function. The digamma or  $\psi(x)$  function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

.

Definition at line 2330 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, and `__psi_asymp()`.

Referenced by `__expint_En_series()`, `__hydrogen()`, `__hyperg_reflect()`, and `__psi()`.

8.3.2.198 `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:

$$\psi^{(n)}(x) = (-1)^{n+1} n! \zeta(m+1, x)$$

Definition at line 2395 of file sf\_gamma.tcc.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

**8.3.2.199** `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 2299 of file `sf_gamma.tcc`.

Referenced by `__psi()`.

**8.3.2.200** `template<typename _Tp> _Tp std::__detail::__psi_series ( _Tp __x )`

Return the digamma function by series expansion. The digamma or  $\psi(x)$  function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

.

The series is given by:

$$\psi(x) = -\gamma_E - \frac{1}{x} \sum_{k=1}^{\infty} \frac{x-1}{(k+1)(x+k)}$$

Definition at line 2268 of file `sf_gamma.tcc`.

**8.3.2.201** `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 2044 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

**8.3.2.202** `template<typename _Tp> _Tp std::__detail::__riemann_zeta ( _Tp __s )`

Return the Riemann zeta function  $\zeta(s)$ .

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } s > 1 \quad \frac{(2\pi)^s}{\pi} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } s < 1$$

For  $s < 1$  use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

## Parameters

$\leftrightarrow$	The argument
$s$	

Definition at line 505 of file sf\_zeta.tcc.

References `__gamma()`, `__log_gamma()`, `__riemann_zeta_glob()`, `__riemann_zeta_product()`, and `__riemann_zeta_ $\leftrightarrow$ sum()`.

Referenced by `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__polylog_exp_real_pos()`, and `evenzeta()`.

### 8.3.2.203 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_alt ( _Tp $s$ )`

Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For  $s < 1$  use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 329 of file sf\_zeta.tcc.

### 8.3.2.204 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_euler_maclaurin ( _Tp $s$ )`

Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .

This is a specialization of the code for the Hurwitz zeta function.

Definition at line 282 of file sf\_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

### 8.3.2.205 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_glob ( _Tp $s$ )`

Evaluate the Riemann zeta function by series for all  $s \neq 1$ . 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:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For  $s < 1$  use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 374 of file sf\_zeta.tcc.

References `__log_gamma()`.

Referenced by `__riemann_zeta()`.

8.3.2.206 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1 ( _Tp __s )`

Return the Riemann zeta function  $\zeta(s) - 1$ .

#### Parameters

$\leftarrow$ __s	The argument $s! = 1$
---------------------	-----------------------

Definition at line 672 of file sf\_zeta.tcc.

References `__riemann_zeta_m_1_sum()`, `_S_num_zetam1`, and `_S_zetam1`.

Referenced by `__polylog_exp_neg()`.

8.3.2.207 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1_sum ( _Tp __s )`

Return the Riemann zeta function  $\zeta(s) - 1$  by summation for  $s > 1$ . This is a small remainder for large  $s$ .

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

#### Parameters

$\leftarrow$ __s	The argument $s! = 1$
---------------------	-----------------------

Definition at line 645 of file sf\_zeta.tcc.

Referenced by `__riemann_zeta_m_1()`.

8.3.2.208 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_product ( _Tp __s )`

Compute the Riemann zeta function  $\zeta(s)$  using the product over prime factors.

$$\zeta(s) = \prod_{i=1}^{\infty} \frac{1}{1 - p_i^{-s}}$$

where  $p_i$  are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For  $s < 1$  use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

## Parameters

$\_s$	The argument
-------	--------------

Definition at line 463 of file sf\_zeta.tcc.

Referenced by \_\_riemann\_zeta().

**8.3.2.209** `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:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For  $s < 1$  use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 254 of file sf\_zeta.tcc.

Referenced by \_\_riemann\_zeta().

**8.3.2.210** `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc ( _Tp __a, _Tp __x )`

Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

.

Definition at line 51 of file sf\_cardinal.tcc.

**8.3.2.211** `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc ( _Tp __x )`

Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

.

Definition at line 98 of file sf\_cardinal.tcc.

**8.3.2.212** `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc_pi ( _Tp __x )`

Return the unnormalized sinus cardinal function

$$\text{sinc}_{\pi}(x) = \frac{\sin(x)}{x}$$

.

Definition at line 78 of file `sf_cardinal.tcc`.

**8.3.2.213** `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__sincosint ( _Tp __x )`

This function returns the sine  $Si(x)$  and cosine  $Ci(x)$  integrals as a `pair`.

The sine integral is defined by:

$$Si(x) = \int_0^x dt \frac{\sin(t)}{t}$$

The cosine integral is defined by:

$$Ci(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cos(t) - 1}{t}$$

Definition at line 227 of file `sf_trigint.tcc`.

References `__sincosint_asymp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

**8.3.2.214** `template<typename _Tp> void std::__detail::__sincosint_asymp ( _Tp __t, _Tp & _Si, _Tp & _Ci )`

This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for  $x > 50$ .

Definition at line 163 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

**8.3.2.215** `template<typename _Tp> void std::__detail::__sincosint_cont_frac ( _Tp __t, _Tp & _Si, _Tp & _Ci )`

This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by continued fraction for positive argument.

Definition at line 55 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

8.3.2.216 `template<typename _Tp> void std::__detail::__sincosint_series ( _Tp __t, _Tp & _Si, _Tp & _Ci )`

This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by series summation for positive argument.

Definition at line 98 of file sf\_trigint.tcc.

Referenced by `__sincosint()`.

8.3.2.217 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc ( _Tp __a, _Tp __x )`

Return the generalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

.

Definition at line 124 of file sf\_cardinal.tcc.

8.3.2.218 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc ( _Tp __x )`

Return the normalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

.

Definition at line 167 of file sf\_cardinal.tcc.

8.3.2.219 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc_pi ( _Tp __x )`

Return the unnormalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_\pi(x) = \frac{\sinh(x)}{x}$$

.

Definition at line 149 of file sf\_cardinal.tcc.

8.3.2.220 `template<typename _Tp> _Tp std::__detail::__sinhint ( const _Tp __x )`

Return the hyperbolic sine integral  $li(x)$ .

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) - E_1(x))/2$$

## Parameters

$\leftrightarrow$ _x	The argument of the hyperbolic sine integral function.
-------------------------	--

## Returns

The hyperbolic sine integral.

Definition at line 577 of file sf\_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

**8.3.2.221** `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

$\leftrightarrow$ _n	The non-negative integral order
$\leftrightarrow$ _x	The non-negative real argument

## Returns

The output spherical Bessel function.

Definition at line 703 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`.

**8.3.2.222** `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	$\leftrightarrow$ _n	The order for which the spherical Bessel function is evaluated.
in	$\leftrightarrow$ _z	The argument at which the spherical Bessel function is evaluated.



## Returns

The complex spherical Bessel function.

Definition at line 1314 of file sf\_hankel.tcc.

References `__sph_hankel()`.

```
8.3.2.223  template<typename _Tp> void std::__detail::__sph_bessel_ik( unsigned int __n, _Tp __x, _Tp & __i_n, _Tp & __k_n,
    _Tp & __ip_n, _Tp & __kp_n )
```

Compute the spherical modified Bessel functions  $i_n(x)$  and  $k_n(x)$  and their first derivatives  $i'_n(x)$  and  $k'_n(x)$  respectively.

## Parameters

<code>__n</code>	The order of the modified spherical Bessel function.
<code>__x</code>	The argument of the modified spherical Bessel function.
<code>__i_n</code>	The output regular modified spherical Bessel function.
<code>__k_n</code>	The output irregular modified spherical Bessel function.
<code>__ip<sub>↔</sub> _n</code>	The output derivative of the regular modified spherical Bessel function.
<code>__kp<sub>↔</sub> _n</code>	The output derivative of the irregular modified spherical Bessel function.

Definition at line 456 of file sf\_mod\_bessel.tcc.

References `__cyl_bessel_ik()`.

```
8.3.2.224  template<typename _Tp> void std::__detail::__sph_bessel_jn( unsigned int __n, _Tp __x, _Tp & __j_n, _Tp & __n_n,
    _Tp & __jp_n, _Tp & __np_n )
```

Compute the spherical Bessel  $j_n(x)$  and Neumann  $n_n(x)$  functions and their first derivatives  $j'_n(x)$  and  $n'_n(x)$  respectively.

## Parameters

	<code>__n</code>	The order of the spherical Bessel function.
	<code>__x</code>	The argument of the spherical Bessel function.
out	<code>__j_n</code>	The output spherical Bessel function.
out	<code>__n_n</code>	The output spherical Neumann function.
out	<code>__jp<sub>↔</sub> _n</code>	The output derivative of the spherical Bessel function.
out	<code>__np<sub>↔</sub> _n</code>	The output derivative of the spherical Neumann function.

Definition at line 668 of file sf\_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
8.3.2.225 template<typename _Tp> void std::__detail::__sph_hankel ( unsigned int __n, std::complex< _Tp> __z,
    std::complex< _Tp> & _H1, std::complex< _Tp> & _H1p, std::complex< _Tp> & _H2, std::complex< _Tp> & _H2p
    )
```

Helper to compute complex spherical Hankel functions and their derivatives.

#### Parameters

in	<code>__n</code>	The order for which the spherical Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the spherical Hankel functions are evaluated.
out	<code>_H1</code>	The spherical Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the spherical Hankel function of the first kind.
out	<code>_H2</code>	The spherical Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the spherical Hankel function of the second kind.

Definition at line 1258 of file sf\_hankel.tcc.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
8.3.2.226 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

<code>__n</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

#### Returns

The output spherical Neumann function.

Definition at line 772 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`.

**8.3.2.227** `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	$\leftrightarrow$ __n	The order for which the spherical Hankel function of the first kind is evaluated.
in	$\leftrightarrow$ __z	The argument at which the spherical Hankel function of the first kind is evaluated.

#### Returns

The complex spherical Hankel function of the first kind.

Definition at line 1282 of file sf\_hankel.tcc.

References `__sph_hankel()`.

**8.3.2.228** `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  $h_n^{(2)}(x)$ .

The spherical Hankel function of the second kind is defined by:

$$h_n^{(2)}(x) = j_n(x) - in_n(x)$$

#### Parameters

$\leftrightarrow$ __n	The non-negative integral order
$\leftrightarrow$ __x	The non-negative real argument

#### Returns

The output spherical Neumann function.

Definition at line 804 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`.

8.3.2.229 `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	$\leftrightarrow$ __n	The order for which the spherical Hankel function of the second kind is evaluated.
in	$\leftrightarrow$ __z	The argument at which the spherical Hankel function of the second kind is evaluated.

#### Returns

The complex spherical Hankel function of the second kind.

Definition at line 1298 of file sf\_hankel.tcc.

References `__sph_hankel()`.

8.3.2.230 `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  $\theta$ ,  $\phi$  is defined by:

$$Y_l^m(\theta, \phi) = (-1)^m \left[ \frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^{|m|}(\cos \theta) \exp^{im\phi}$$

#### Parameters

__l	The order of the spherical harmonic function. $l \geq 0$ .
__m	The order of the spherical harmonic function. $m \leq l$ .
__theta	The radian polar angle argument of the spherical harmonic function.
__phi	The radian azimuthal angle argument of the spherical harmonic function.

Definition at line 350 of file sf\_legendre.tcc.

References `__sph_legendre()`.

8.3.2.231 `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  $\theta$  is defined as  $Y_l^m(\theta, 0)$  where

$$Y_l^m(\theta, \phi) = (-1)^m \left[ \frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^m(\cos \theta) \exp^{im\phi}$$

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

<code>__l</code>	The order of the spherical associated Legendre function. $l \geq 0$ .
<code>__m</code>	The order of the spherical associated Legendre function. $m \leq l$ .
<code>__theta</code>	The radian polar angle argument of the spherical associated Legendre function.

Definition at line 253 of file `sf_legendre.tcc`.

References `__log_gamma()`, and `__poly_legendre_p()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

#### 8.3.2.232 `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

<code>__n</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

#### Returns

The output spherical Neumann function.

Definition at line 740 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

8.3.2.233 `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	$\leftrightarrow$ _n	The order for which the spherical Neumann function is evaluated.
in	$\leftrightarrow$ _z	The argument at which the spherical Neumann function is evaluated.

## Returns

The complex spherical Neumann function.

Definition at line 1330 of file sf\_hankel.tcc.

References `__sph_hankel()`.

8.3.2.234 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf ( _Tp __t, unsigned int __nu )`

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right)A(t|\nu) =$$

## Parameters

__t	
__nu	

Definition at line 301 of file sf\_beta.tcc.

References `__beta_inc()`.

8.3.2.235 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc ( _Tp __t, unsigned int __nu )`

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) = 1 - A(t|\nu)$$

## Parameters

__t	
__nu	

Definition at line 324 of file sf\_beta.tcc.

References `__beta_inc()`.

**8.3.2.236** `template<typename _Tp> _Tp std::__detail::__theta_1 ( _Tp __nu, _Tp __x )`

Return the exponential theta-1 function of period `nu` and argument `x`.

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

#### Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 190 of file sf\_theta.tcc.

References `__theta_2()`.

Referenced by `__theta_s()`.

**8.3.2.237** `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

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 162 of file sf\_theta.tcc.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.



8.3.2.238 `template<typename _Tp> _Tp std::__detail::__theta_2_asyp ( _Tp __nu, _Tp __x )`

Compute and return the  $\theta_2$  function by series expansion.

Definition at line 103 of file sf\_theta.tcc.

Referenced by `__theta_2()`.

8.3.2.239 `template<typename _Tp> _Tp std::__detail::__theta_2_sum ( _Tp __nu, _Tp __x )`

Compute and return the  $\theta_1$  function by series expansion.

Definition at line 49 of file sf\_theta.tcc.

Referenced by `__theta_2()`.

8.3.2.240 `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

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 216 of file sf\_theta.tcc.

References `__theta_3_asyp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

8.3.2.241 `template<typename _Tp> _Tp std::__detail::__theta_3_asyp ( _Tp __nu, _Tp __x )`

Compute and return the  $\theta_3$  function by asymptotic series expansion.

Definition at line 128 of file sf\_theta.tcc.

Referenced by `__theta_3()`.

8.3.2.242 `template<typename _Tp> _Tp std::__detail::__theta_3_sum ( _Tp __nu, _Tp __x )`

Compute and return the  $\theta_3$  function by series expansion.

Definition at line 77 of file sf\_theta.tcc.

Referenced by `__theta_3()`.

8.3.2.243 `template<typename _Tp> _Tp std::__detail::__theta_4 ( _Tp __nu, _Tp __x )`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

#### Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 244 of file sf\_theta.tcc.

References `__theta_3()`.

Referenced by `__theta_n()`.

8.3.2.244 `template<typename _Tp> _Tp std::__detail::__theta_c ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_c$  function

Definition at line 337 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

8.3.2.245 `template<typename _Tp> _Tp std::__detail::__theta_d ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_d$  function

Definition at line 362 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

8.3.2.246 `template<typename _Tp> _Tp std::__detail::__theta_n ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_n$  function

Definition at line 387 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

8.3.2.247 `template<typename _Tp> _Tp std::__detail::__theta_s ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_s$  function

Definition at line 311 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

8.3.2.248 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__zernike ( unsigned int __n, int __m, _Tp __rho, _Tp __phi )`

Return the Zernicke polynomial  $Z_n^m(\rho, \phi)$  for non-negative degree  $n$ , signed order  $m$ , and real radial argument  $\rho$  and azimuthal angle  $\phi$ .

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  $m \leq n$  and where  $R_n^m(\rho)$  is the radial polynomial (

See also

[\\_\\_poly\\_radial\\_jacobi](#)).

#### Template Parameters

<code>_Tp</code>	The real type of the radial coordinate and azimuthal angle
------------------	--

#### Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 183 of file sf\_jacobi.tcc.

References `__poly_radial_jacobi()`.

**8.3.2.249** `template<typename _Tp> _Tp std::__detail::__znorm1 ( _Tp __x )`

Definition at line 58 of file sf\_owens\_t.tcc.

Referenced by `__owens_t()`.

**8.3.2.250** `template<typename _Tp> _Tp std::__detail::__znorm2 ( _Tp __x )`

Definition at line 47 of file sf\_owens\_t.tcc.

Referenced by `__owens_t()`.

**8.3.2.251** `template<typename _Tp = double> _Tp std::__detail::evenzeta ( unsigned int __k )`

A function to calculate the values of zeta at even positive integers. For values smaller than thirty a table is used.

#### Parameters

<code>__k</code>	an integer at which we evaluate the Riemann zeta function.
------------------	--

#### Returns

$zeta(k)$

Definition at line 156 of file sf\_polylog.tcc.

References `__riemann_zeta()`.

### 8.3.3 Variable Documentation

**8.3.3.1** `constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100`

Coefficients for Euler-Maclaurin summation of zeta functions.

$$B_{2j}/(2j)!$$

where  $B_k$  are the Bernoulli numbers.

Definition at line 65 of file sf\_zeta.tcc.

8.3.3.2 constexpr **\_Factorial\_table**<long double> std::\_\_detail::\_S\_double\_factorial\_table[301]

Definition at line 274 of file sf\_gamma.tcc.

Referenced by \_\_double\_factorial(), and \_\_log\_double\_factorial().

8.3.3.3 constexpr long double std::\_\_detail::\_S\_Euler\_Maclaurin\_zeta[\_Num\_Euler\_Maclaurin\_zeta]

Definition at line 68 of file sf\_zeta.tcc.

Referenced by \_\_hurwitz\_zeta\_euler\_maclaurin(), and \_\_riemann\_zeta\_euler\_maclaurin().

8.3.3.4 constexpr **\_Factorial\_table**<long double> std::\_\_detail::\_S\_factorial\_table[171]

Definition at line 84 of file sf\_gamma.tcc.

Referenced by \_\_factorial(), and \_\_log\_factorial().

8.3.3.5 constexpr **\_Factorial\_table**<long double> std::\_\_detail::\_S\_neg\_double\_factorial\_table[999]

Definition at line 595 of file sf\_gamma.tcc.

Referenced by \_\_double\_factorial(), and \_\_log\_double\_factorial().

8.3.3.6 template<typename \_Tp > constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials = 0

Definition at line 259 of file sf\_gamma.tcc.

8.3.3.7 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< double > = 301

Definition at line 264 of file sf\_gamma.tcc.

8.3.3.8 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< float > = 57

Definition at line 262 of file sf\_gamma.tcc.

8.3.3.9 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< long double > = 301

Definition at line 266 of file sf\_gamma.tcc.

**8.3.3.10** `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_factorials = 0`

Definition at line 69 of file `sf_gamma.tcc`.

**8.3.3.11** `template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`

Definition at line 74 of file `sf_gamma.tcc`.

**8.3.3.12** `template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`

Definition at line 72 of file `sf_gamma.tcc`.

**8.3.3.13** `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 76 of file `sf_gamma.tcc`.

**8.3.3.14** `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 579 of file `sf_gamma.tcc`.

**8.3.3.15** `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 584 of file `sf_gamma.tcc`.

**8.3.3.16** `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 582 of file `sf_gamma.tcc`.

**8.3.3.17** `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 586 of file `sf_gamma.tcc`.

**8.3.3.18** `constexpr size_t std::__detail::_S_num_zetam1 = 33`

Table of  $\zeta(n)$  - 1 from 2 - 32. MPFR - 128 bits.

Definition at line 592 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

**8.3.3.19** `constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]`

Definition at line 596 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

## Chapter 9

# Class Documentation

### 9.1 `std::__detail::_Factorial_table<_Tp>` Struct Template Reference

#### Public Attributes

- `_Tp __factorial`
- `_Tp __log_factorial`
- `unsigned int __n`

#### 9.1.1 Detailed Description

```
template<typename _Tp>  
struct std::__detail::_Factorial_table<_Tp>
```

Definition at line 61 of file `sf_gamma.tcc`.

#### 9.1.2 Member Data Documentation

9.1.2.1 `template<typename _Tp> _Tp std::__detail::_Factorial_table<_Tp>::__factorial`

Definition at line 64 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__double_factorial()`.

9.1.2.2 `template<typename _Tp> _Tp std::__detail::_Factorial_table<_Tp>::__log_factorial`

Definition at line 65 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__log_double_factorial()`.

### 9.1.2.3 `template<typename _Tp> unsigned int std::__detail::__Factorial_table<_Tp>::__n`

Definition at line 63 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__bernoulli()`, `std::__detail::__bernoulli_2n()`, `std::__detail::__bernoulli_series()`, `std::__detail::__bincoef()`, `std::__detail::__double_factorial()`, `std::__detail::__factorial()`, `std::__detail::__gamma_cont_frac()`, `std::__detail::__gamma_series()`, `std::__detail::__log_double_factorial()`, `std::__detail::__log_factorial()`, and `std::__detail::__psi()`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)



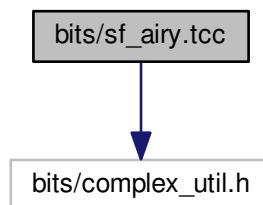
## Chapter 10

# File Documentation

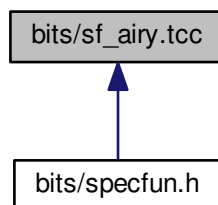
### 10.1 bits/sf\_airy.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf\_airy.tcc:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

## Functions

- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`  
*This function computes the Airy function  $Ai(z)$  and its first derivative in the complex  $z$ -plane.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_airy\_ai (std::complex< _Tp > __z)`  
*Return the complex Airy  $Ai$  function.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_asymp\_absarg\_ge\_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, int __sign=-1)`  
*This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|arg(z)| < 2 * \pi/3$ . For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from  $abs(z)$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_asymp\_absarg\_lt\_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip)`  
*This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|arg(-z)| < \pi/3$ . For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from  $|z|$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_bessel\_i (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_lp1d3, std::complex< _Tp > &_lm1d3, std::complex< _Tp > &_lp2d3, std::complex< _Tp > &_lm2d3)`
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_bessel\_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)`  
*Compute approximations to the modified Bessel functions of the second kind of orders  $1/3$  and  $2/3$  for moderate arguments.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_airy\_bi (std::complex< _Tp > __z)`  
*Return the complex Airy  $Bi$  function.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_hyperg\_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`  
*This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders  $\nu = + - 1/3$  and  $\nu = -2/3$ . That is,  $A(z)/B(z)$ , Where  $A(z)$  and  $B(z)$  are cubic polynomials with real coefficients, approximates*

$$\frac{\Gamma(\nu+1)}{(z/2)^{\nu}} I_{\nu}(z) = {}_0F_1(; \nu+1; z^2/4),$$

where the function on the right is a confluent hypergeometric limit function. For  $|z| \leq 1/4$  and  $|arg(z)| \leq \pi/2$ , the approximations are accurate to about 16 decimals.

### 10.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

### 10.1.2 Macro Definition Documentation

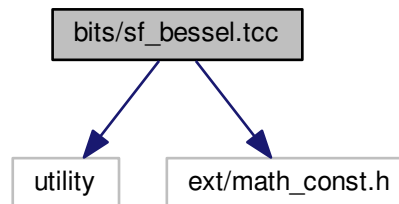
#### 10.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

Definition at line 31 of file sf\_airy.tcc.

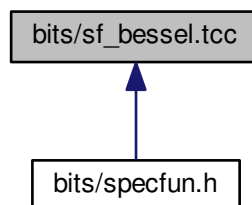
## 10.2 bits/sf\_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
```

Include dependency graph for sf\_bessel.tcc:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`  
*This routine returns the cylindrical Bessel functions of order  $\nu$ :  $J_\nu$  or  $I_\nu$  by series expansion.*
- `template<typename _Tp >`  
`_Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`  
*Return the Bessel function of order  $\nu$ :  $J_\nu(x)$ .*
- `template<typename _Tp >`  
`void std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
*Return the cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.*
- `template<typename _Tp >`  
`void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
*This routine computes the asymptotic cylindrical Bessel and Neumann functions of order  $\nu$ :  $J_\nu(x)$ ,  $N_\nu(x)$ . Use this for  $x \gg nu^2 + 1$ .*
- `template<typename _Tp >`  
`void std::__detail::__cyl_bessel_jn_steel (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
*Compute the Bessel  $J_\nu(x)$  and Neumann  $N_\nu(x)$  functions and their first derivatives  $J'_\nu(x)$  and  $N'_\nu(x)$  respectively. These four functions are computed together for numerical stability.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)`  
*Return the cylindrical Hankel function of the first kind  $H_\nu^{(1)}(x)$ .*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)`  
*Return the cylindrical Hankel function of the second kind  $H_\nu^{(2)}(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__cyl_neumann_n (_Tp __nu, _Tp __x)`  
*Return the Neumann function of order  $\nu$ :  $N_\nu(x)$ .*
- `template<typename _Tp >`  
`void std::__detail::__gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`  
*Compute the gamma functions required by the Temme series expansions of  $N_\nu(x)$  and  $K_\nu(x)$ .*

$$\Gamma_1 = \frac{1}{2\mu} \left[ \frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[ \frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where  $-1/2 \leq \mu \leq 1/2$  is  $\mu = \nu - N$  and  $N$  is the nearest integer to  $\nu$ . The values of  $\Gamma(1+\mu)$  and  $\Gamma(1-\mu)$  are returned as well.

- `template<typename _Tp >`  
`_Tp std::__detail::__sph_bessel (unsigned int __n, _Tp __x)`  
*Return the spherical Bessel function  $j_n(x)$  of order  $n$  and non-negative real argument  $x$ .*
- `template<typename _Tp >`  
`void std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`  
*Compute the spherical Bessel  $j_n(x)$  and Neumann  $n_n(x)$  functions and their first derivatives  $j'_n(x)$  and  $n'_n(x)$  respectively.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`  
*Return the spherical Hankel function of the first kind  $h_n^{(1)}(x)$ .*
- `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  $h_n^{(2)}(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__sph_neumann (unsigned int __n, _Tp __x)`  
*Return the spherical Neumann function  $n_n(x)$  of order  $n$  and non-negative real argument  $x$ .*

### 10.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.2.2 Macro Definition Documentation

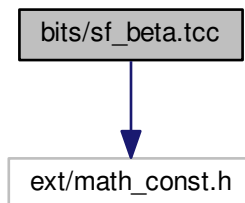
#### 10.2.2.1 `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file `sf_bessel.tcc`.

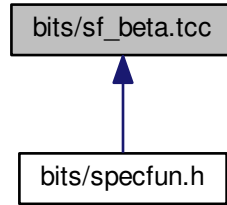
## 10.3 bits/sf\_beta.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_beta.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_BETA\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta\_gamma (_Tp __a, _Tp __b)`  
*Return the beta function:  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta\_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta\_inc\_cont\_frac (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta\_lgamma (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$  using the log gamma functions.*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_beta\_product (_Tp __a, _Tp __b)`  
*Return the beta function  $B(x, y)$  using the product form.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::\_\_detail::\_\_binomial\_cdf (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the binomial cumulative distribution function.*

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the complementary binomial cumulative distribution function.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution probability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution probability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`  
*Return the Students T probability function.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`  
*Return the complement of the Students T probability function.*

### 10.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.3.2 Macro Definition Documentation

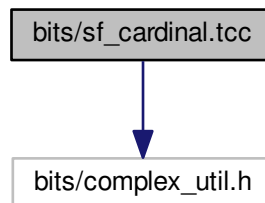
#### 10.3.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file `sf_beta.tcc`.

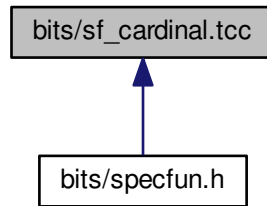
## 10.4 bits/sf\_cardinal.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for `sf_cardinal.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_CARDINAL\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::\_\_detail::\_\_sinc (_Tp __a, _Tp __x)`  
*Return the generalized sinus cardinal function*

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::\_\_detail::\_\_sinc (_Tp __x)`  
*Return the normalized sinus cardinal function*

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::\_\_detail::\_\_sinc\_pi (_Tp __x)`  
*Return the unnormalized sinus cardinal function*

$$\text{sinc}_\pi(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::\_\_detail::\_\_sinhc (_Tp __a, _Tp __x)`



*Return the generalized hyperbolic sinus cardinal function*

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __x)`

*Return the normalized hyperbolic sinus cardinal function*

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)`

*Return the unnormalized hyperbolic sinus cardinal function*

$$\operatorname{sinhc}_\pi(x) = \frac{\sinh(x)}{x}$$

## 10.4.1 Macro Definition Documentation

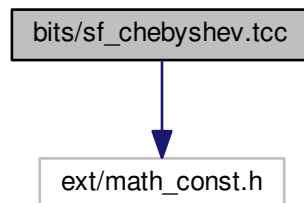
### 10.4.1.1 `#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1`

Definition at line 30 of file `sf_cardinal.tcc`.

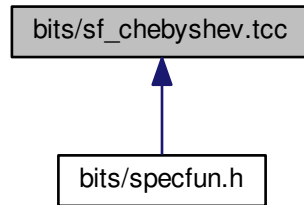
## 10.5 bits/sf\_chebyshev.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_chebyshev.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_CHEBYSHEV\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_chebyshev\\_recur \(unsigned int \\_\\_n, \\_Tp \\_\\_x, \\_Tp \\_C0, \\_Tp \\_C1\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_chebyshev\\_t \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_chebyshev\\_u \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_chebyshev\\_v \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_chebyshev\\_w \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)

### 10.5.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

## 10.5.2 Macro Definition Documentation

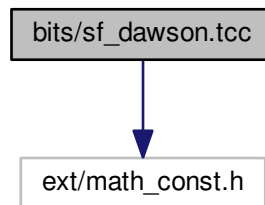
### 10.5.2.1 `#define _GLIBCXX_SF_CHEBYSHEV_TCC 1`

Definition at line 31 of file sf\_chebyshev.tcc.

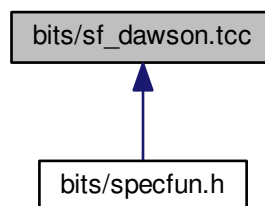
## 10.6 bits/sf\_dawson.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_dawson.tcc:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- `#define _GLIBCXX_SF_DAWSON_TCC 1`

## Functions

- `template<typename _Tp >  
_Tp std::__detail::__dawson (_Tp __x)`  
*Return the Dawson integral,  $F(x)$ , for real argument  $x$ .*
- `template<typename _Tp >  
_Tp std::__detail::__dawson_cont_frac (_Tp __x)`  
*Compute the Dawson integral using a sampling theorem representation.*
- `template<typename _Tp >  
_Tp std::__detail::__dawson_series (_Tp __x)`  
*Compute the Dawson integral using the series expansion.*

### 10.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.6.2 Macro Definition Documentation

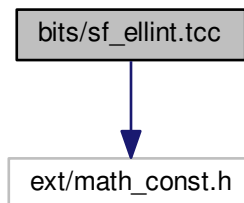
#### 10.6.2.1 `#define _GLIBCXX_SF_DAWSON_TCC 1`

Definition at line 31 of file `sf_dawson.tcc`.

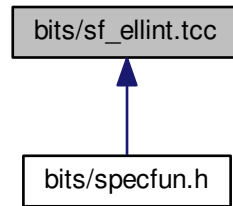
## 10.7 bits/sf\_ellint.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_ellint.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_ELLINT\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_1 \(\\_Tp \\_\\_k\)](#)  
*Return the complete elliptic integral of the first kind  $K(k)$  using the Carlson formulation.*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_2 \(\\_Tp \\_\\_k\)](#)  
*Return the complete elliptic integral of the second kind  $E(k)$  using the Carlson formulation.*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_3 \(\\_Tp \\_\\_k, \\_Tp \\_\\_nu\)](#)  
*Return the complete elliptic integral of the third kind  $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$  using the Carlson formulation.*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_d \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_rf \(\\_Tp \\_\\_x, \\_Tp \\_\\_y\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_comp\\_ellint\\_rg \(\\_Tp \\_\\_x, \\_Tp \\_\\_y\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellint\\_1 \(\\_Tp \\_\\_k, \\_Tp \\_\\_phi\)](#)  
*Return the incomplete elliptic integral of the first kind  $F(k, \phi)$  using the Carlson formulation.*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellint\\_2 \(\\_Tp \\_\\_k, \\_Tp \\_\\_phi\)](#)

*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$  using the Carlson formulation.*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`

*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$  using the Carlson formulation.*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)`

*Return the Carlson elliptic function  $R_C(x, y) = R_F(x, y, y)$  where  $R_F(x, y, z)$  is the Carlson elliptic function of the first kind.*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)`

*Return the Carlson elliptic function of the second kind  $R_D(x, y, z) = R_J(x, y, z, z)$  where  $R_J(x, y, z, p)$  is the Carlson elliptic function of the third kind.*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

*Return the Carlson elliptic function  $R_F(x, y, z)$  of the first kind.*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

*Return the symmetric Carlson elliptic function of the second kind  $R_G(x, y, z)$ .*

- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`

*Return the Carlson elliptic function  $R_J(x, y, z, p)$  of the third kind.*

- `template<typename _Tp >`  
`_Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

## 10.7.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

## 10.7.2 Macro Definition Documentation

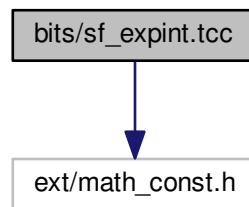
### 10.7.2.1 `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

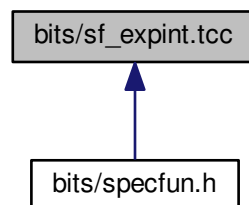
## 10.8 bits/sf\_expint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_expint.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_EXPINT\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__coshint (const _Tp __x)`  
*Return the hyperbolic cosine integral  $li(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint (_Tp __x)`  
*Return the exponential integral  $Ei(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_asymp (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  for large argument.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_E1 (_Tp __x)`  
*Return the exponential integral  $E_1(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_E1_asymp (_Tp __x)`  
*Return the exponential integral  $E_1(x)$  by asymptotic expansion.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_E1_series (_Tp __x)`  
*Return the exponential integral  $E_1(x)$  by series summation. This should be good for  $x < 1$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_Ei (_Tp __x)`  
*Return the exponential integral  $Ei(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_Ei_asymp (_Tp __x)`  
*Return the exponential integral  $Ei(x)$  by asymptotic expansion.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_Ei_series (_Tp __x)`  
*Return the exponential integral  $Ei(x)$  by series summation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by continued fractions.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by recursion. Use upward recursion for  $x < n$  and downward recursion (Miller's algorithm) otherwise.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  by series summation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__expint_large_n (unsigned int __n, _Tp __x)`  
*Return the exponential integral  $E_n(x)$  for large order.*
- `template<typename _Tp >`  
`_Tp std::__detail::__logint (const _Tp __x)`  
*Return the logarithmic integral  $li(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__sinhint (const _Tp __x)`  
*Return the hyperbolic sine integral  $li(x)$ .*



### 10.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.8.2 Macro Definition Documentation

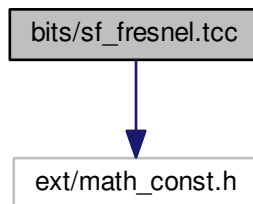
#### 10.8.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

Definition at line 47 of file `sf_expint.tcc`.

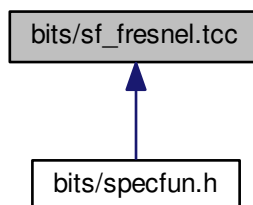
## 10.9 bits/sf\_fresnel.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_fresnel.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_FRESNEL\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_fresnel (const _Tp __x)`  
*Return the Fresnel cosine and sine integrals as a complex number  $C(x) + iS(x)$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_fresnel\_cont\_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`  
*This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_fresnel\_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`  
*This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.*

### 10.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.9.2 Macro Definition Documentation

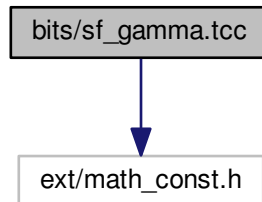
#### 10.9.2.1 [#define \\_GLIBCXX\\_SF\\_FRESNEL\\_TCC 1](#)

Definition at line 31 of file `sf_fresnel.tcc`.

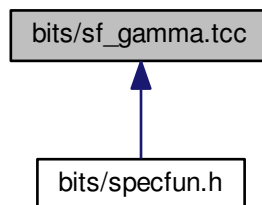
## 10.10 bits/sf\_gamma.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_gamma.tcc:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [std::\\_\\_detail::\\_Factorial\\_table< \\_Tp >](#)

### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_GAMMA\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int __n)`  
*This returns Bernoulli number  $B_n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int __n)`  
*This returns Bernoulli number  $B_n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`  
*This returns Bernoulli numbers from a table or by summation for larger values.*
- `template<typename _Tp >`  
`_Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)`  
*Return the binomial coefficient. The binomial coefficient is given by:*

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`  
*Return the double factorial of the integer  $n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`  
*Return the factorial of the integer  $n$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__gamma (_Tp __x)`  
*Return  $\Gamma(x)$ .*
- `template<typename _Tp >`  
`std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)`  
*Return the lower incomplete gamma function. The lower incomplete gamma function is defined by*

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`  
`std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp __a, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)`  
*Return the upper incomplete gamma function. The lower incomplete gamma function is defined by*

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`  
`_Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)`  
*Return the logarithm of the binomial coefficient. The binomial coefficient is given by:*

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`  
*Return the logarithm of the double factorial of the integer n.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`  
*Return the logarithm of the factorial of the integer n.*
- `template<typename _Tp >`  
`_Tp std::__detail::__log_gamma (_Tp __x)`  
*Return  $\log(|\Gamma(x)|)$ . This will return values even for  $x < 0$ . To recover the sign of  $\Gamma(x)$  for any argument use `__log_gamma_sign`.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`  
*Return  $\log(\Gamma(x))$  by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)`  
*Return  $\log(\Gamma(x))$  by the Lanczos method. This method dominates all others on the positive axis I think.*
- `template<typename _Tp >`  
`_Tp std::__detail::__log_gamma_sign (_Tp __x)`  
*Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)`  
*Return  $\Gamma(z)$  by the Spouge algorithm:*

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[ \sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

- `template<typename _Tp >`  
`_Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)`  
*Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by*

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular,  $(n)_n = n!$ . Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`  
`_Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(n), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[ \begin{matrix} a \\ n \end{matrix} \right]$$

, 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::__pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular,  $\$f[(n)_n = n! \$f]$ .

- `template<typename _Tp >`  
`_Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[ \begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`  
`_Tp std::__detail::__psi (_Tp __x)`

Return the digamma function. The digamma or  $\psi(x)$  function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

.

- `template<typename _Tp >`  
`_Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

*Return the polygamma function  $\psi^{(n)}(x)$ .*

- `template<typename _Tp >`  
`_Tp std::__detail::__psi_asymp (_Tp __x)`

*Return the digamma function for large argument. The digamma or  $\psi(x)$  function is defined by*

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- `template<typename _Tp >`  
`_Tp std::__detail::__psi_series (_Tp __x)`

*Return the digamma function by series expansion. The digamma or  $\psi(x)$  function is defined by*

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- `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.*

## Variables

- `constexpr _Factorial_table< long double > std::__detail::__S_double_factorial_table [301]`
- `constexpr _Factorial_table< long double > std::__detail::__S_factorial_table [171]`
- `constexpr _Factorial_table< long double > std::__detail::__S_neg_double_factorial_table [999]`
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::__S_num_double_factorials = 0`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_double_factorials< double > = 301`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_double_factorials< float > = 57`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::__S_num_factorials = 0`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_factorials< double > = 171`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_factorials< float > = 35`
- `template<>`  
`constexpr std::size_t std::__detail::__S_num_factorials< long double > = 171`
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials = 0`

- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

### 10.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.10.2 Macro Definition Documentation

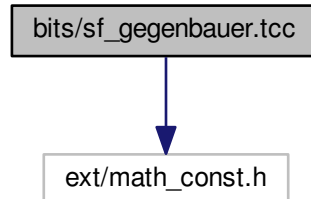
#### 10.10.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

## 10.11 `bits/sf_gegenbauer.tcc` File Reference

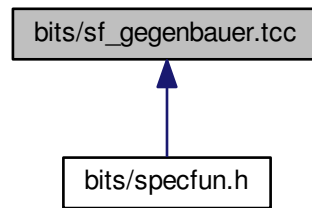
```
#include <ext/math_const.h>
```

Include dependency graph for `sf_gegenbauer.tcc`:





This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_GEGENBAUER\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_gegenbauer\\_poly \(unsigned int \\_\\_n, \\_Tp \\_\\_alpha, \\_Tp \\_\\_x\)](#)

### 10.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.11.2 Macro Definition Documentation

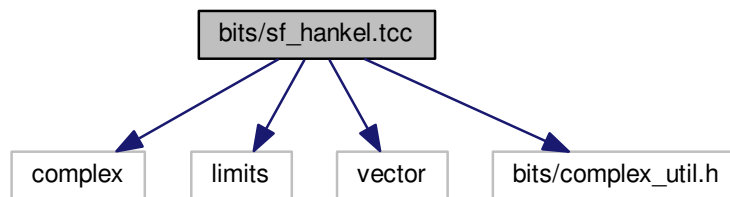
#### 10.11.2.1 [#define \\_GLIBCXX\\_SF\\_GEGENBAUER\\_TCC 1](#)

Definition at line 31 of file `sf_gegenbauer.tcc`.

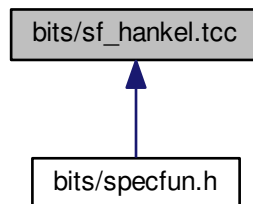
## 10.12 bits/sf\_hankel.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include <bits/complex_util.h>
```

Include dependency graph for sf\_hankel.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HANKEL\\_TCC](#) 1

## Functions

- `template<typename _Tp >`  
`void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`  
*Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Bessel function.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Hankel function of the first kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Hankel function of the second kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Neumann function.*
- `template<typename _Tp >`  
`void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`  
`void std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &_p, std::complex< _Tp > &_p2, std::complex< _Tp > &_nup2, std::complex< _Tp > &_num2, std::complex< _Tp > &_num1d3, std::complex< _Tp > &_num2d3, std::complex< _Tp > &_num4d3, std::complex< _Tp > &_zeta, std::complex< _Tp > &_zetaphf, std::complex< _Tp > &_zetamhf, std::complex< _Tp > &_zetam3hf, std::complex< _Tp > &_zetrat)`  
*Compute parameters depending on `z` and `nu` that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.*
- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`  
*This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.*
- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`  
*Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.*

- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __←`  
`eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std←`  
`::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp >`  
`&__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std←`  
`::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp`  
`> &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`  

*Compute outer factors and associated functions of  $z$  and  $nu$  appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of  $z$  and  $nu$  returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.*
- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex<`  
`_Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp,`  
`std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp`  
`> __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp >`  
`&__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)`  

*Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to  $n$  terms (less than 5) to achieve relative error  $eps$ .*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)`  

*Return the complex spherical Bessel function.*
- `template<typename _Tp >`  
`void std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &__H1, std←`  
`::complex< _Tp > &__H1p, std::complex< _Tp > &__H2, std::complex< _Tp > &__H2p)`  

*Helper to compute complex spherical Hankel functions and their derivatives.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`  

*Return the complex spherical Hankel function of the first kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`  

*Return the complex spherical Hankel function of the second kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)`  

*Return the complex spherical Neumann function.*

### 10.12.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

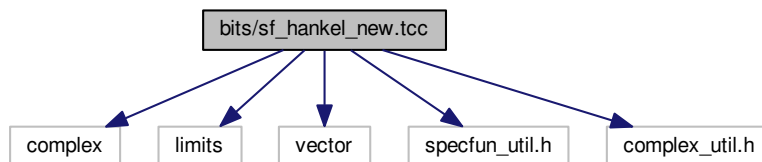
### 10.12.2 Macro Definition Documentation

#### 10.12.2.1 `#define GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

## 10.13 bits/sf\_hankel\_new.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include "specfun_util.h"
#include "complex_util.h"
Include dependency graph for sf_hankel_new.tcc:
```



### Macros

- `#define GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1`

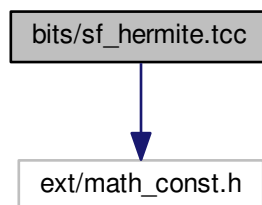
#### 10.13.1 Macro Definition Documentation

##### 10.13.1.1 `#define GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1`

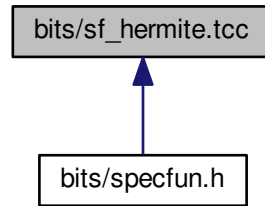
Definition at line 31 of file `sf_hankel_new.tcc`.

## 10.14 bits/sf\_hermite.tcc File Reference

```
#include <ext/math_const.h>
Include dependency graph for sf_hermite.tcc:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HERMITE\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_poly\\_hermite \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)  
*This routine returns the Hermite polynomial of order  $n$ :  $H_n(x)$ .*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_poly\\_hermite\\_asymp \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)  
*This routine returns the Hermite polynomial of large order  $n$ :  $H_n(x)$ . We assume here that  $x \geq 0$ .*
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_poly\\_hermite\\_recursion \(unsigned int \\_\\_n, \\_Tp \\_\\_x\)](#)  
*This routine returns the Hermite polynomial of order  $n$ :  $H_n(x)$  by recursion on  $n$ .*

### 10.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

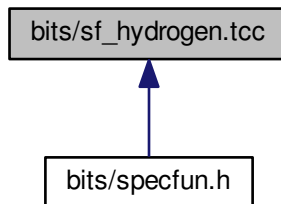
### 10.14.2 Macro Definition Documentation

#### 10.14.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_HERMITE\\_TCC 1](#)

Definition at line 42 of file `sf_hermite.tcc`.

## 10.15 bits/sf\_hydrogen.tcc File Reference

This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HYDROGEN\\_TCC 1](#)

### Functions

- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_hydrogen \(const unsigned int \\_\\_n, const unsigned int \\_\\_l, const unsigned int \\_\\_m, const \\_Tp \\_\\_Z, const \\_Tp \\_\\_r, const \\_Tp \\_\\_theta, const \\_Tp \\_\\_phi\)](#)

#### 10.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

#### 10.15.2 Macro Definition Documentation

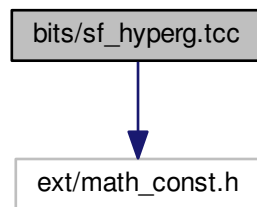
##### 10.15.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_HYDROGEN\\_TCC 1](#)

Definition at line 31 of file sf\_hydrogen.tcc.

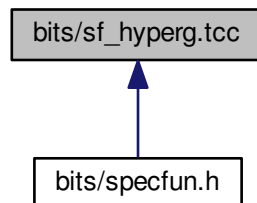
## 10.16 bits/sf\_hyperg.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_hyperg.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HYPERG\\_TCC](#) 1



## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`  
*Return the confluent hypergeometric function  ${}_1F_1(a; c; x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)`  
*Return the confluent hypergeometric limit function  ${}_0F_1(-; c; x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)`  
*This routine returns the confluent hypergeometric limit function by series expansion.*
- `template<typename _Tp >`  
`_Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`  
*Return the hypergeometric function  ${}_1F_1(a; c; x)$  by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.*
- `template<typename _Tp >`  
`_Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`  
*This routine returns the confluent hypergeometric function by series expansion.*
- `template<typename _Tp >`  
`_Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.*
- `template<typename _Tp >`  
`_Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for  $d = c - a - b$  not integral and formula 15.3.11 for  $d = c - a - b$  integral. This assumes  $a, b, c \neq$  negative integer.*
- `template<typename _Tp >`  
`_Tp std::__detail::__hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by series expansion.*

### 10.16.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.16.2 Macro Definition Documentation

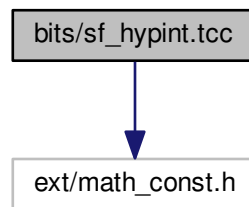
#### 10.16.2.1 `#define _GLIBCXX_BITS_SF_HYPERG_TCC 1`

Definition at line 44 of file `sf_hyperg.tcc`.

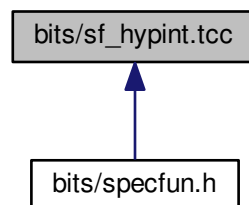
## 10.17 bits/sf\_hypint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_hypint.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_SF\\_HYPINT\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > std::__detail::__chshint ( _Tp __x, _Tp &_Chi, _Tp &_Shi)`  
*This function returns the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals as a pair.*
- `template<typename _Tp >`  
`void std::__detail::__chshint_cont_frac ( _Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by continued fraction for positive argument.*
- `template<typename _Tp >`  
`void std::__detail::__chshint_series ( _Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by series summation for positive argument.*

### 10.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.17.2 Macro Definition Documentation

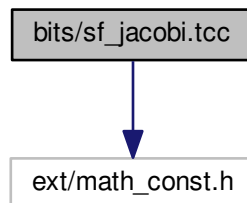
#### 10.17.2.1 `#define _GLIBCXX_SF_HYPINT_TCC 1`

Definition at line 31 of file `sf_hypint.tcc`.

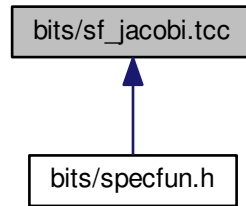
## 10.18 bits/sf\_jacobi.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_jacobi.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_JACOBI\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_poly\\_jacobi \(unsigned int \\_\\_n, \\_Tp \\_\\_alpha, \\_Tp \\_\\_beta, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_poly\\_radial\\_jacobi \(unsigned int \\_\\_n, unsigned int \\_\\_m, \\_Tp \\_\\_rho\)](#)
- [template<typename \\_Tp > \\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t< \\_Tp > std::\\_\\_detail::\\_\\_zernike \(unsigned int \\_\\_n, int \\_\\_m, \\_Tp \\_\\_rho, \\_Tp \\_\\_phi\)](#)

### 10.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.18.2 Macro Definition Documentation

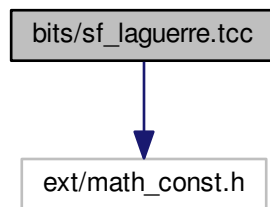
#### 10.18.2.1 [#define \\_GLIBCXX\\_SF\\_JACOBI\\_TCC 1](#)

Definition at line 31 of file `sf_jacobi.tcc`.

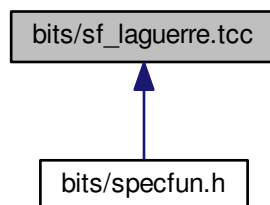
## 10.19 bits/sf\_laguerre.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_laguerre.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_LAGUERRE\\_TCC](#) 1

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__assoc_laguerre` (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_x)  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $m$ :  $L_n^m(x)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__laguerre` (unsigned int \_\_n, \_Tp \_\_x)  
*This routine returns the Laguerre polynomial of order  $n$ :  $L_n(x)$ .*
- `template<typename _Tpa, typename _Tp >`  
`_Tp std::__detail::__poly_laguerre` (unsigned int \_\_n, \_Tpa \_\_alpha1, \_Tp \_\_x)  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha$ :  $L_n^\alpha(x)$ .*
- `template<typename _Tpa, typename _Tp >`  
`_Tp std::__detail::__poly_laguerre_hyperg` (unsigned int \_\_n, \_Tpa \_\_alpha1, \_Tp \_\_x)  
*Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.*
- `template<typename _Tpa, typename _Tp >`  
`_Tp std::__detail::__poly_laguerre_large_n` (unsigned \_\_n, \_Tpa \_\_alpha1, \_Tp \_\_x)  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha > -1$  for large  $n$ . Abramowitz & Stegun, 13.5.21.*
- `template<typename _Tpa, typename _Tp >`  
`_Tp std::__detail::__poly_laguerre_recursion` (unsigned int \_\_n, \_Tpa \_\_alpha1, \_Tp \_\_x)  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $\alpha$ :  $L_n^\alpha(x)$  by recursion.*

### 10.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.19.2 Macro Definition Documentation

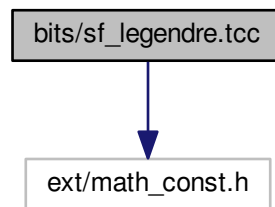
#### 10.19.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

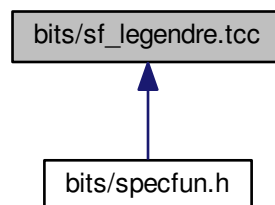
## 10.20 bits/sf\_legendre.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf\_legendre.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_LEGENDRE\\_TCC](#) 1

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__assoc_legendre_p` (unsigned int \_\_l, unsigned int \_\_m, \_Tp \_\_x)  
*Return the associated Legendre function by recursion on l and downward recursion on m.*
- `template<typename _Tp >`  
`_Tp std::__detail::__legendre_q` (unsigned int \_\_l, \_Tp \_\_x)  
*Return the Legendre function of the second kind by upward recursion on order l.*
- `template<typename _Tp >`  
`_Tp std::__detail::__poly_legendre_p` (unsigned int \_\_l, \_Tp \_\_x)  
*Return the Legendre polynomial by upward recursion on order l.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sph_harmonic` (unsigned int \_\_l, int \_\_m, \_Tp \_\_theta, \_Tp \_\_phi)  
*Return the spherical harmonic function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__sph_legendre` (unsigned int \_\_l, unsigned int \_\_m, \_Tp \_\_theta)  
*Return the spherical associated Legendre function.*

### 10.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

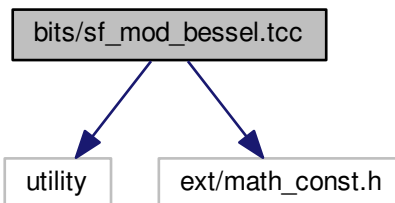
### 10.20.2 Macro Definition Documentation

#### 10.20.2.1 `#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1`

Definition at line 47 of file `sf_legendre.tcc`.

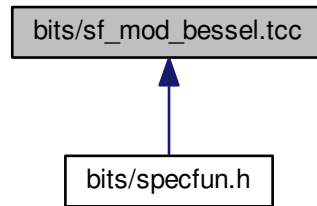
## 10.21 bits/sf\_mod\_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
Include dependency graph for sf_mod_bessel.tcc:
```





This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_MOD\\_BESSEL\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)`  
*Compute the Airy functions  $Ai(x)$  and  $Bi(x)$  and their first derivatives  $Ai'(x)$  and  $Bi'(x)$  respectively.*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_cyl\_bessel\_i (_Tp __nu, _Tp __x)`  
*Return the regular modified Bessel function of order  $\nu$ :  $I_\nu(x)$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_cyl\_bessel\_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`  
*Return the modified cylindrical Bessel functions and their derivatives of order  $\nu$  by various means.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_cyl\_bessel\_ik\_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`  
*This routine computes the asymptotic modified cylindrical Bessel and functions of order  $\nu$ :  $I_\nu(x)$ ,  $N_\nu(x)$ . Use this for  $x \gg nu^2 + 1$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_cyl\_bessel\_ik\_steeds (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`  
*Compute the modified Bessel functions  $I_\nu(x)$  and  $K_\nu(x)$  and their first derivatives  $I'_\nu(x)$  and  $K'_\nu(x)$  respectively. These four functions are computed together for numerical stability.*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_cyl\_bessel\_k (_Tp __nu, _Tp __x)`

Return the irregular modified Bessel function  $K_\nu(x)$  of order  $\nu$ .

- `template<typename _Tp >`  
`void std::__detail::__fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`

Compute the Fock-type Airy functions  $w_1(x)$  and  $w_2(x)$  and their first derivatives  $w_1'(x)$  and  $w_2'(x)$  respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

- `template<typename _Tp >`  
`void std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`

Compute the spherical modified Bessel functions  $i_n(x)$  and  $k_n(x)$  and their first derivatives  $i_n'(x)$  and  $k_n'(x)$  respectively.

### 10.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

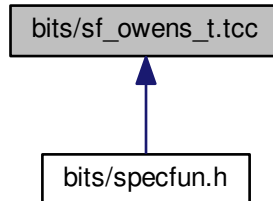
### 10.21.2 Macro Definition Documentation

#### 10.21.2.1 `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

## 10.22 bits/sf\_owens\_t.tcc File Reference

This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_OWENS\\_T\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_gauss \(\\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_owens\\_t \(\\_Tp \\_\\_h, \\_Tp \\_\\_a\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_znorm1 \(\\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_znorm2 \(\\_Tp \\_\\_x\)](#)

### 10.22.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

### 10.22.2 Macro Definition Documentation

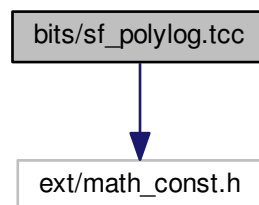
#### 10.22.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_OWENS\\_T\\_TCC 1](#)

Definition at line 31 of file `sf_owens_t.tcc`.

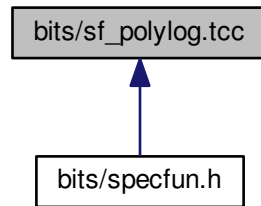
## 10.23 bits/sf\_polylog.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_polylog.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_POLYLOG\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_bose\\_einstein \(\\_Tp \\_\\_s, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_clamp\\_0\\_m2pi \(std::complex< \\_Tp > \\_\\_w\)](#)
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_clamp\\_pi \(std::complex< \\_Tp > \\_\\_w\)](#)
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_clausen \(unsigned int \\_\\_m, std::complex< \\_Tp > \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_clausen \(unsigned int \\_\\_m, \\_Tp \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_clausen\\_c \(unsigned int \\_\\_m, std::complex< \\_Tp > \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_clausen\\_c \(unsigned int \\_\\_m, \\_Tp \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_clausen\\_s \(unsigned int \\_\\_m, std::complex< \\_Tp > \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_clausen\\_s \(unsigned int \\_\\_m, \\_Tp \\_\\_w\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_dirichlet\\_beta \(std::complex< \\_Tp > \\_\\_w\)](#)

- `template<typename _Tp >`  
`_Tp std::__detail::__dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp std::__detail::__dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`  
`_Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)`
- `template<typename _Tp >`  
`bool std::__detail::__fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`  
`bool std::__detail::__fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`  
`bool std::__detail::__fpimag (const _Tp)`
- `template<typename _Tp >`  
`bool std::__detail::__fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`  
`bool std::__detail::__fpreal (const _Tp)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`  
`_Tp std::__detail::__polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`  
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > std::__detail::__polylog_exp (_Tp __s, 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_int_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`  
`std::complex< _Tp > std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`  
`std::complex< _Tp > std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _PowTp, typename _Tp >`  
`_Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp = double>`  
`_Tp std::__detail::evenzeta (unsigned int __k)`

### 10.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

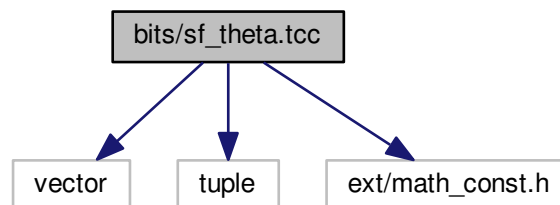
### 10.23.2 Macro Definition Documentation

#### 10.23.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

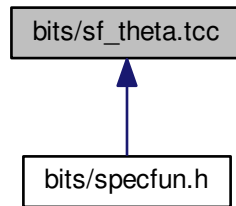
Definition at line 41 of file `sf_polylog.tcc`.

## 10.24 bits/sf\_theta.tcc File Reference

```
#include <vector>
#include <tuple>
#include <ext/math_const.h>
Include dependency graph for sf_theta.tcc:
```



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_THETA\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome\\_k \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome\\_series \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > std::tuple< \\_Tp, \\_Tp, \\_Tp > std::\\_\\_detail::\\_\\_jacobi\\_sncndn \(\\_Tp \\_\\_k, \\_Tp \\_\\_u\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_1 \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_2 \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_2\\_asymp \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_2\\_sum \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_3 \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_theta\\_3\\_asymp \(\\_Tp \\_\\_nu, \\_Tp \\_\\_x\)](#)

- `template<typename _Tp >`  
`_Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__theta_s (_Tp __k, _Tp __x)`

### 10.24.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.24.2 Macro Definition Documentation

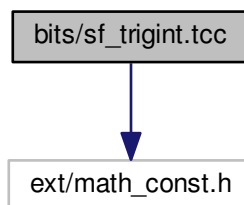
#### 10.24.2.1 `#define _GLIBCXX_SF_THETA_TCC 1`

Definition at line 31 of file `sf_theta.tcc`.

## 10.25 `bits/sf_trigint.tcc` File Reference

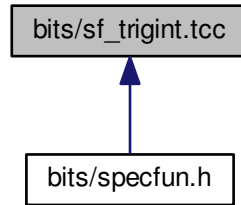
```
#include <ext/math_const.h>
```

Include dependency graph for `sf_trigint.tcc`:





This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_SF\\_TRIGINT\\_TCC 1](#)

## Enumerations

- [enum { std::\\_\\_detail::SININT, std::\\_\\_detail::COSINT }](#)

## Functions

- [template<typename \\_Tp > std::pair< \\_Tp, \\_Tp > std::\\_\\_detail::\\_\\_sincosint \( \\_Tp \\_\\_x\)](#)  
*This function returns the sine  $Si(x)$  and cosine  $Ci(x)$  integrals as a pair.*
- [template<typename \\_Tp > void std::\\_\\_detail::\\_\\_sincosint\\_asymp \( \\_Tp \\_\\_t, \\_Tp &\\_Si, \\_Tp &\\_Ci\)](#)  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by asymptotic series summation for positive argument.*
- [template<typename \\_Tp > void std::\\_\\_detail::\\_\\_sincosint\\_cont\\_frac \( \\_Tp \\_\\_t, \\_Tp &\\_Si, \\_Tp &\\_Ci\)](#)  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by continued fraction for positive argument.*
- [template<typename \\_Tp > void std::\\_\\_detail::\\_\\_sincosint\\_series \( \\_Tp \\_\\_t, \\_Tp &\\_Si, \\_Tp &\\_Ci\)](#)  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by series summation for positive argument.*

### 10.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.25.2 Macro Definition Documentation

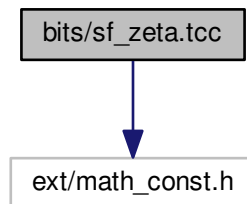
#### 10.25.2.1 `#define _GLIBCXX_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

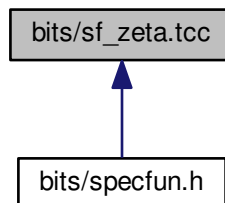
## 10.26 `bits/sf_zeta.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_zeta.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__dilog (_Tp __x)`  
*Compute the dilogarithm function  $Li_2(x)$  by summation for  $x \leq 1$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__hurwitz_zeta (_Tp __s, _Tp __a)`  
*Return the Hurwitz zeta function  $\zeta(s, a)$  for all  $s \neq 1$  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  $s \neq 1$  and  $a > -1$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta (_Tp __s)`  
*Return the Riemann zeta function  $\zeta(s)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_alt (_Tp __s)`  
*Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_euler_maclaurin (_Tp __s)`  
*Evaluate the Riemann zeta function  $\zeta(s)$  by an alternate series for  $s > 0$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_glob (_Tp __s)`  
*Evaluate the Riemann zeta function by series for all  $s \neq 1$ . Convergence is great until largish negative numbers. Then the convergence of the  $> 0$  sum gets better.*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`  
*Return the Riemann zeta function  $\zeta(s) - 1$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)`  
*Return the Riemann zeta function  $\zeta(s) - 1$  by summation for  $s > 1$ . This is a small remainder for large  $s$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_product (_Tp __s)`  
*Compute the Riemann zeta function  $\zeta(s)$  using the product over prime factors.*
- `template<typename _Tp >`  
`_Tp std::__detail::__riemann_zeta_sum (_Tp __s)`  
*Compute the Riemann zeta function  $\zeta(s)$  by summation for  $s > 1$ .*

## Variables

- constexpr size\_t [std::\\_\\_detail::\\_Num\\_Euler\\_Maclaurin\\_zeta](#) = 100
- constexpr long double [std::\\_\\_detail::\\_S\\_Euler\\_Maclaurin\\_zeta](#) [[\\_Num\\_Euler\\_Maclaurin\\_zeta](#)]
- constexpr size\_t [std::\\_\\_detail::\\_S\\_num\\_zetam1](#) = 33
- constexpr long double [std::\\_\\_detail::\\_S\\_zetam1](#) [[\\_S\\_num\\_zetam1](#)]

### 10.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.26.2 Macro Definition Documentation

#### 10.26.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

## 10.27 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_gamma.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_dawson.tcc>
#include <bits/sf_ellint.tcc>
#include <bits/sf_expint.tcc>
#include <bits/sf_fresnel.tcc>
#include <bits/sf_gegenbauer.tcc>
#include <bits/sf_hyperg.tcc>
#include <bits/sf_hypint.tcc>
#include <bits/sf_jacobi.tcc>
#include <bits/sf_laguerre.tcc>
#include <bits/sf_legendre.tcc>
#include <bits/sf_hydrogen.tcc>
#include <bits/sf_mod_bessel.tcc>
#include <bits/sf_hermite.tcc>
#include <bits/sf_theta.tcc>
#include <bits/sf_trigint.tcc>
#include <bits/sf_zeta.tcc>
#include <bits/sf_owens_t.tcc>
#include <bits/sf_polylog.tcc>
#include <bits/sf_airy.tcc>
#include <bits/sf_hankel.tcc>
#include <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`

## Enumerations

- enum { [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_SN](#), [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_CN](#), [\\_\\_gnu\\_cxx::GLIBCXX\\_JACOBI\\_DN](#) }

## Functions

- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::airy\\_ai](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_aif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_ail](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::airy\\_bi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_bif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_bil](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::\\_\\_type std::assoc\\_laguerre](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_x)
- float [std::assoc\\_laguerref](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_x)
- long double [std::assoc\\_laguerrel](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::\\_\\_type std::assoc\\_legendre](#) (unsigned int \_\_l, unsigned int \_\_m, \_Tp \_\_x)
- float [std::assoc\\_legendref](#) (unsigned int \_\_l, unsigned int \_\_m, float \_\_x)
- long double [std::assoc\\_legendrel](#) (unsigned int \_\_l, unsigned int \_\_m, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::bernoulli](#) (unsigned int \_\_n)
- float [\\_\\_gnu\\_cxx::bernoullif](#) (unsigned int \_\_n)
- long double [\\_\\_gnu\\_cxx::bernoullil](#) (unsigned int \_\_n)
- template<typename \_Tpa, typename \_Tpb >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_2<\\_Tpa, \\_Tpb>::\\_\\_type std::beta](#) (\_Tpa \_\_a, \_Tpb \_\_b)
- float [std::betaf](#) (float \_\_a, float \_\_b)
- long double [std::betal](#) (long double \_\_a, long double \_\_b)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::bincoef](#) (unsigned int \_\_n, unsigned int \_\_k)
- float [\\_\\_gnu\\_cxx::bincoeff](#) (unsigned int \_\_n, unsigned int \_\_k)
- long double [\\_\\_gnu\\_cxx::bincoefl](#) (unsigned int \_\_n, unsigned int \_\_k)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_t](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_tf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_tl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_u](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_uf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_ul](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_v](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_vf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_vl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::chebyshev\\_w](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_wf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_wl](#) (unsigned int \_\_n, long double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen` (unsigned int \_\_m, \_Tp \_\_w)
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen` (unsigned int \_\_m, std::complex< \_Tp > \_\_w)
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c` (unsigned int \_\_m, \_Tp \_\_w)
- `float __gnu_cxx::clausen_cf` (unsigned int \_\_m, float \_\_w)
- `long double __gnu_cxx::clausen_cl` (unsigned int \_\_m, long double \_\_w)
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s` (unsigned int \_\_m, \_Tp \_\_w)
- `float __gnu_cxx::clausen_sf` (unsigned int \_\_m, float \_\_w)
- `long double __gnu_cxx::clausen_sl` (unsigned int \_\_m, long double \_\_w)
- `float __gnu_cxx::clausenf` (unsigned int \_\_m, float \_\_w)
- `std::complex< float > __gnu_cxx::clausenf` (unsigned int \_\_m, std::complex< float > \_\_w)
- `long double __gnu_cxx::clausenl` (unsigned int \_\_m, long double \_\_w)
- `std::complex< long double > __gnu_cxx::clausenl` (unsigned int \_\_m, std::complex< long double > \_\_w)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (\_Tp \_\_k)
- `float std::comp_ellint_1f` (float \_\_k)
- `long double std::comp_ellint_1l` (long double \_\_k)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (\_Tp \_\_k)
- `float std::comp_ellint_2f` (float \_\_k)
- `long double std::comp_ellint_2l` (long double \_\_k)
- `template<typename _Tp, typename _Tpn >`  
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (\_Tp \_\_k, \_Tpn \_\_nu)
- `float std::comp_ellint_3f` (float \_\_k, float \_\_nu)
- Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for float modulus  $k$ .*
- `long double std::comp_ellint_3l` (long double \_\_k, long double \_\_nu)
- Return the complete elliptic integral of the third kind  $\Pi(k, \nu)$  for long double modulus  $k$ .*
- `template<typename _Tk >`  
`__gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d` (\_Tk \_\_k)
- `float __gnu_cxx::comp_ellint_df` (float \_\_k)
- `long double __gnu_cxx::comp_ellint_dl` (long double \_\_k)
- `float __gnu_cxx::comp_ellint_rf` (float \_\_x, float \_\_y)
- `long double __gnu_cxx::comp_ellint_rf` (long double \_\_x, long double \_\_y)
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf` (\_Tx \_\_x, \_Ty \_\_y)
- `float __gnu_cxx::comp_ellint_rg` (float \_\_x, float \_\_y)
- `long double __gnu_cxx::comp_ellint_rg` (long double \_\_x, long double \_\_y)
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg` (\_Tx \_\_x, \_Ty \_\_y)
- `template<typename _Tpa, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg` (\_Tpa \_\_a, \_Tpc \_\_c, \_Tp \_\_x)
- `template<typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg_lim` (\_Tpc \_\_c, \_Tp \_\_x)
- `float __gnu_cxx::conf_hyperg_limf` (float \_\_c, float \_\_x)
- `long double __gnu_cxx::conf_hyperg_liml` (long double \_\_c, long double \_\_x)
- `float __gnu_cxx::conf_hypergf` (float \_\_a, float \_\_c, float \_\_x)
- `long double __gnu_cxx::conf_hypergl` (long double \_\_a, long double \_\_c, long double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::coshint ( _Tp __x)`
- `float __gnu_cxx::coshintf (float __x)`
- `long double __gnu_cxx::coshintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::cosint ( _Tp __x)`
- `float __gnu_cxx::cosintf (float __x)`
- `long double __gnu_cxx::cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_if (float __nu, float __x)`
- `long double std::cyl_bessel_il (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_jf (float __nu, float __x)`
- `long double std::cyl_bessel_jl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_k ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_kf (float __nu, float __x)`
- `long double std::cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 ( _Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 ( _Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_neumann ( _Tpnu __nu, _Tp __x)`
- `float std::cyl_neumannf (float __nu, float __x)`
- `long double std::cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson ( _Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma ( _Tp __z)`



- float [\\_\\_gnu\\_cxx::digammaf](#) (float \_\_z)
- long double [\\_\\_gnu\\_cxx::digammal](#) (long double \_\_z)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::dilog](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::dilogf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::dilogl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_Tp](#) [\\_\\_gnu\\_cxx::dirichlet\\_beta](#) (\_Tp \_\_s)
- float [\\_\\_gnu\\_cxx::dirichlet\\_betaf](#) (float \_\_s)
- long double [\\_\\_gnu\\_cxx::dirichlet\\_betall](#) (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\\_etall](#) (long double \_\_s)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::double\\_factorial](#) (int \_\_n)
- float [\\_\\_gnu\\_cxx::double\\_factorialf](#) (int \_\_n)
- long double [\\_\\_gnu\\_cxx::double\\_factoriall](#) (int \_\_n)
- template<typename \_Tp, typename \_Tpp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_2<\\_Tp, \\_Tpp>::\\_\\_type](#) [std::ellint\\_1](#) (\_Tp \_\_k, \_Tpp \_\_phi)
- float [std::ellint\\_1f](#) (float \_\_k, float \_\_phi)
- long double [std::ellint\\_1l](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Tpp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_2<\\_Tp, \\_Tpp>::\\_\\_type](#) [std::ellint\\_2](#) (\_Tp \_\_k, \_Tpp \_\_phi)
- float [std::ellint\\_2f](#) (float \_\_k, float \_\_phi)
- Return the incomplete elliptic integral of the second kind  $E(k, \phi)$  for float argument.*
- long double [std::ellint\\_2l](#) (long double \_\_k, long double \_\_phi)
- Return the incomplete elliptic integral of the second kind  $E(k, \phi)$ .*
- template<typename \_Tp, typename \_Tpn, typename \_Tpp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_3<\\_Tp, \\_Tpn, \\_Tpp>::\\_\\_type](#) [std::ellint\\_3](#) (\_Tp \_\_k, \_Tpn \_\_nu, \_Tpp \_\_phi)
- Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- float [std::ellint\\_3f](#) (float \_\_k, float \_\_nu, float \_\_phi)
- Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$  for float argument.*
- long double [std::ellint\\_3l](#) (long double \_\_k, long double \_\_nu, long double \_\_phi)
- Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$ .*
- template<typename \_Tk, typename \_Tp, typename \_Ta, typename \_Tb >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tk, \\_Tp, \\_Ta, \\_Tb>](#) [\\_\\_gnu\\_cxx::ellint\\_cel](#) (\_Tk \_\_k\_c, \_Tp \_\_p, \_Ta \_\_a, \_Tb \_\_b)
- float [\\_\\_gnu\\_cxx::ellint\\_celf](#) (float \_\_k\_c, float \_\_p, float \_\_a, float \_\_b)
- long double [\\_\\_gnu\\_cxx::ellint\\_cell](#) (long double \_\_k\_c, long double \_\_p, long double \_\_a, long double \_\_b)
- template<typename \_Tk, typename \_Tphi >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tk, \\_Tphi>](#) [\\_\\_gnu\\_cxx::ellint\\_d](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::ellint\\_df](#) (float \_\_k, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::ellint\\_dl](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Tk >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tk>](#) [\\_\\_gnu\\_cxx::ellint\\_el1](#) (\_Tp \_\_x, \_Tk \_\_k\_c)
- float [\\_\\_gnu\\_cxx::ellint\\_el1f](#) (float \_\_x, float \_\_k\_c)
- long double [\\_\\_gnu\\_cxx::ellint\\_el1l](#) (long double \_\_x, long double \_\_k\_c)
- template<typename \_Tp, typename \_Tk, typename \_Ta, typename \_Tb >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tk, \\_Ta, \\_Tb>](#) [\\_\\_gnu\\_cxx::ellint\\_el2](#) (\_Tp \_\_x, \_Tk \_\_k\_c, \_Ta \_\_a, \_Tb \_\_b)

- float [\\_\\_gnu\\_cxx::ellint\\_el2f](#) (float \_\_x, float \_\_k\_c, float \_\_a, float \_\_b)
- long double [\\_\\_gnu\\_cxx::ellint\\_el2l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_a, long double \_\_b)
- template<typename \_Tx, typename \_Tk, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tx, \\_Tk, \\_Tp>](#) [\\_\\_gnu\\_cxx::ellint\\_el3](#) (\_Tx \_\_x, \_Tk \_\_k\_c, \_Tp \_\_p)
- float [\\_\\_gnu\\_cxx::ellint\\_el3f](#) (float \_\_x, float \_\_k\_c, float \_\_p)
- long double [\\_\\_gnu\\_cxx::ellint\\_el3l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_p)
- template<typename \_Tp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up>](#) [\\_\\_gnu\\_cxx::ellint\\_rc](#) (\_Tp \_\_x, \_Up \_\_y)
- float [\\_\\_gnu\\_cxx::ellint\\_rcf](#) (float \_\_x, float \_\_y)
- long double [\\_\\_gnu\\_cxx::ellint\\_rcl](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up, \\_Vp>](#) [\\_\\_gnu\\_cxx::ellint\\_rd](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rdf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rdl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up, \\_Vp>](#) [\\_\\_gnu\\_cxx::ellint\\_rf](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rff](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rfl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up, \\_Vp>](#) [\\_\\_gnu\\_cxx::ellint\\_rg](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rgf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rgl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp, typename \_Wp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Up, \\_Vp, \\_Wp>](#) [\\_\\_gnu\\_cxx::ellint\\_rj](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z, \_Wp \_\_p)
- float [\\_\\_gnu\\_cxx::ellint\\_rjf](#) (float \_\_x, float \_\_y, float \_\_z, float \_\_p)
- long double [\\_\\_gnu\\_cxx::ellint\\_rjl](#) (long double \_\_x, long double \_\_y, long double \_\_z, long double \_\_p)
- template<typename \_Tp >  
[\\_Tp](#) [\\_\\_gnu\\_cxx::ellnome](#) (\_Tp \_\_k)
- float [\\_\\_gnu\\_cxx::ellnomef](#) (float \_\_k)
- long double [\\_\\_gnu\\_cxx::ellnomel](#) (long double \_\_k)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::type](#) [std::expint](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::expint](#) (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 \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::factorial](#) (unsigned int \_\_n)
- float [\\_\\_gnu\\_cxx::factorialf](#) (unsigned int \_\_n)
- long double [\\_\\_gnu\\_cxx::factoriall](#) (unsigned int \_\_n)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::fresnel\\_c](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::fresnel\\_cf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::fresnel\\_cl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::fresnel\\_s](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::fresnel\\_sf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::fresnel\\_sl](#) (long double \_\_x)

- `template<typename _Tn, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_l ( _Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_lf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ll (long double __n, long double __x)`
- `template<typename _Tn, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_u ( _Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_uf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ul (long double __n, long double __x)`
- `template<typename _Talpha, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)`
- `float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)`
- `long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)`
- `float std::hermitef (unsigned int __n, float __x)`
- `long double std::hermitel (unsigned int __n, long double __x)`
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda ( _Tk __k, _Tphi __phi)`
- `float __gnu_cxx::heuman_lambdaf (float __k, float __phi)`
- `long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)`
- `template<typename _Tp, typename _Up >`  
`__gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta ( _Tp __s, _Up __a)`
- `template<typename _Tp, typename _Up >`  
`std::complex< _Tp > __gnu_cxx::hurwitz_zeta ( _Tp __s, std::complex< _Up > __a)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg ( _Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)`
- `long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta ( _Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac ( _Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetacf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetac (long double __a, long double __b, long double __x)`
- `float __gnu_cxx::ibetaf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_cnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_dnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)`

- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_snf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta ( _Tk __k, _Tphi __phi)`
- `float __gnu_cxx::jacobi_zetaf (float __k, float __phi)`
- `long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)`
- `float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::laguerre (unsigned int __n, _Tp __x)`
- `float std::laguerref (unsigned int __n, float __x)`
- `long double std::laguerrel (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)`
- `float __gnu_cxx::lbincoeff (unsigned int __n, unsigned int __k)`
- `long double __gnu_cxx::lbincoeffl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)`
- `float __gnu_cxx::ldouble_factorialf (int __n)`
- `long double __gnu_cxx::ldouble_factoriall (int __n)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::legendre (unsigned int __l, _Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::legendre_qf (unsigned int __n, float __x)`
- `long double __gnu_cxx::legendre_ql (unsigned int __n, long double __x)`
- `float std::legendref (unsigned int __l, float __x)`
- `long double std::legendrel (unsigned int __l, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)`
- `float __gnu_cxx::lfactorialf (unsigned int __n)`
- `long double __gnu_cxx::lfactoriall (unsigned int __n)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint ( _Tp __x)`
- `float __gnu_cxx::logintf (float __x)`
- `long double __gnu_cxx::logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l ( _Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_lf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_u ( _Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_uf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`  
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > __gnu_cxx::owens_t ( _Tph __h, _Tpa __a)`
- `float __gnu_cxx::owens_tf (float __h, float __a)`
- `long double __gnu_cxx::owens_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_num_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 \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn> \\_\\_gnu\\_cxx::pochhammer\\_l](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_lf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_ll](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Tn> \\_\\_gnu\\_cxx::pochhammer\\_u](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_uf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_ul](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Wp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Wp> \\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, \_Wp \_\_w)
- template<typename \_Tp, typename \_Wp >  
[std::complex<\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp, \\_Wp>> \\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, std::complex<\_Tp> \_\_w)
- float [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, float \_\_w)
- std::complex< float > [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, std::complex< float > \_\_w)
- long double [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, long double \_\_w)
- std::complex< long double > [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, std::complex< long double > \_\_w)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::psi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::psif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::psil](#) (long double \_\_x)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_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\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::radpoly](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_rho)
- float [\\_\\_gnu\\_cxx::radpolyf](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_rho)
- long double [\\_\\_gnu\\_cxx::radpolyl](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_rho)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::\\_\\_type std::riemann\\_zeta](#) (\_Tp \_\_s)
- float [std::riemann\\_zetaf](#) (float \_\_s)
- long double [std::riemann\\_zetal](#) (long double \_\_s)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::sinc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::sinc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sincf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sincl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::sinhc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_num\\_t<\\_Tp> \\_\\_gnu\\_cxx::sinhc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinhc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sinhcf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhcl](#) (long double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhint ( _Tp __x)`
- `float __gnu_cxx::sinhintf (float __x)`
- `long double __gnu_cxx::sinhintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinint ( _Tp __x)`
- `float __gnu_cxx::sinintf (float __x)`
- `long double __gnu_cxx::sinintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::sph_bessel (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x)`
- `float std::sph_besself (unsigned int __n, float __x)`
- `long double std::sph_bessell (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Ttheta, typename _Tphi >`  
`std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)`
- `std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)`
- `std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`
- `float std::sph_legendref (unsigned int __l, unsigned int __m, float __theta)`
- `long double std::sph_legendrel (unsigned int __l, unsigned int __m, long double __theta)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type std::sph_neumann (unsigned int __n, _Tp __x)`
- `float std::sph_neumannf (unsigned int __n, float __x)`
- `long double std::sph_neumannl (unsigned int __n, long double __x)`

- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_1f (float __nu, float __x)`
- `long double __gnu_cxx::theta_1l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_2f (float __nu, float __x)`
- `long double __gnu_cxx::theta_2l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_3f (float __nu, float __x)`
- `long double __gnu_cxx::theta_3l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_4f (float __nu, float __x)`
- `long double __gnu_cxx::theta_4l (long double __nu, long double __x)`
- `template<typename _Tp_k, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_c (_Tp_k __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 _Tp_k, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_d (_Tp_k __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 _Tp_k, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_n (_Tp_k __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 _Tp_k, typename _Tp>`  
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_s (_Tp_k __k, _Tp __x)`
- `float __gnu_cxx::theta_sf (float __k, float __x)`
- `long double __gnu_cxx::theta_sl (long double __k, long double __x)`
- `template<typename _Trho, typename _Tphi>`  
`__gnu_cxx::__promote_num_t< _Trho, _Tphi > __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)`
- `float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)`
- `long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)`

### 10.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 10.27.2 Macro Definition Documentation

#### 10.27.2.1 `#define __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

#### 10.27.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file `specfun.h`.





# Index

`_GLIBCXX_BITS_SF_AIRY_TCC`  
    [sf\\_airy.tcc, 267](#)  
`_GLIBCXX_BITS_SF_BESSEL_TCC`  
    [sf\\_bessel.tcc, 269](#)  
`_GLIBCXX_BITS_SF_BETA_TCC`  
    [sf\\_beta.tcc, 271](#)  
`_GLIBCXX_BITS_SF_CARDINAL_TCC`  
    [sf\\_cardinal.tcc, 273](#)  
`_GLIBCXX_BITS_SF_ELLINT_TCC`  
    [sf\\_ellint.tcc, 278](#)  
`_GLIBCXX_BITS_SF_EXPINT_TCC`  
    [sf\\_expint.tcc, 281](#)  
`_GLIBCXX_BITS_SF_GAMMA_TCC`  
    [sf\\_gamma.tcc, 288](#)  
`_GLIBCXX_BITS_SF_HANKEL_NEW_TCC`  
    [sf\\_hankel\\_new.tcc, 293](#)  
`_GLIBCXX_BITS_SF_HANKEL_TCC`  
    [sf\\_hankel.tcc, 292](#)  
`_GLIBCXX_BITS_SF_HERMITE_TCC`  
    [sf\\_hermite.tcc, 294](#)  
`_GLIBCXX_BITS_SF_HYDROGEN_TCC`  
    [sf\\_hydrogen.tcc, 295](#)  
`_GLIBCXX_BITS_SF_HYPERG_TCC`  
    [sf\\_hyperg.tcc, 297](#)  
`_GLIBCXX_BITS_SF_LAGUERRE_TCC`  
    [sf\\_laguerre.tcc, 302](#)  
`_GLIBCXX_BITS_SF_LEGENDRE_TCC`  
    [sf\\_legendre.tcc, 304](#)  
`_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`  
    [sf\\_mod\\_bessel.tcc, 306](#)  
`_GLIBCXX_BITS_SF_OWENS_T_TCC`  
    [sf\\_owens\\_t.tcc, 307](#)  
`_GLIBCXX_BITS_SF_POLYLOG_TCC`  
    [sf\\_polylog.tcc, 310](#)  
`_GLIBCXX_BITS_SF_ZETA_TCC`  
    [sf\\_zeta.tcc, 316](#)  
`_GLIBCXX_JACOBI_CN`  
    GNU Extended Mathematical Special Functions, [50](#)  
`_GLIBCXX_JACOBI_DN`  
    GNU Extended Mathematical Special Functions, [50](#)  
`_GLIBCXX_JACOBI_SN`  
    GNU Extended Mathematical Special Functions, [50](#)  
`_GLIBCXX_SF_CHEBYSHEV_TCC`  
    [sf\\_chebyshev.tcc, 275](#)  
`_GLIBCXX_SF_DAWSON_TCC`  
    [sf\\_dawson.tcc, 276](#)  
`_GLIBCXX_SF_FRESNEL_TCC`  
    [sf\\_fresnel.tcc, 282](#)  
`_GLIBCXX_SF_GEGENBAUER_TCC`  
    [sf\\_gegenbauer.tcc, 289](#)  
`_GLIBCXX_SF_HYPINT_TCC`  
    [sf\\_hypint.tcc, 299](#)  
`_GLIBCXX_SF_JACOBI_TCC`  
    [sf\\_jacobi.tcc, 300](#)  
`_GLIBCXX_SF_THETA_TCC`  
    [sf\\_theta.tcc, 312](#)  
`_GLIBCXX_SF_TRIGINT_TCC`  
    [sf\\_trigint.tcc, 314](#)  
`_Num_Euler_Maclaurin_zeta`  
    std::\_\_detail, [260](#)  
`_S_Euler_Maclaurin_zeta`  
    std::\_\_detail, [261](#)  
`_S_double_factorial_table`  
    std::\_\_detail, [260](#)  
`_S_factorial_table`  
    std::\_\_detail, [261](#)  
`_S_neg_double_factorial_table`  
    std::\_\_detail, [261](#)  
`_S_num_double_factorials`  
    std::\_\_detail, [261](#)  
`_S_num_double_factorials< double >`  
    std::\_\_detail, [261](#)  
`_S_num_double_factorials< float >`  
    std::\_\_detail, [261](#)  
`_S_num_double_factorials< long double >`  
    std::\_\_detail, [261](#)  
`_S_num_factorials`  
    std::\_\_detail, [261](#)  
`_S_num_factorials< double >`  
    std::\_\_detail, [262](#)  
`_S_num_factorials< float >`  
    std::\_\_detail, [262](#)  
`_S_num_factorials< long double >`  
    std::\_\_detail, [262](#)  
`_S_num_neg_double_factorials`  
    std::\_\_detail, [262](#)  
`_S_num_neg_double_factorials< double >`  
    std::\_\_detail, [262](#)  
`_S_num_neg_double_factorials< float >`  
    std::\_\_detail, [262](#)

- `__S_num_neg_double_factorials< long double >`
  - `std::__detail`, [262](#)
- `__S_num_zetam1`
  - `std::__detail`, [262](#)
- `__S_zetam1`
  - `std::__detail`, [262](#)
- `__STDCPP_MATH_SPEC_FUNCS__`
  - `specfun.h`, [327](#)
- `__airy`
  - `std::__detail`, [153](#)
- `__airy_ai`
  - `std::__detail`, [155](#)
- `__airy_arg`
  - `std::__detail`, [155](#)
- `__airy_asymp_absarg_ge_pio3`
  - `std::__detail`, [155](#)
- `__airy_asymp_absarg_lt_pio3`
  - `std::__detail`, [156](#)
- `__airy_bessel_i`
  - `std::__detail`, [156](#)
- `__airy_bessel_k`
  - `std::__detail`, [158](#)
- `__airy_bi`
  - `std::__detail`, [158](#)
- `__airy_hyperg_rational`
  - `std::__detail`, [158](#)
- `__assoc_laguerre`
  - `std::__detail`, [159](#)
- `__assoc_legendre_p`
  - `std::__detail`, [160](#)
- `__bernoulli`
  - `std::__detail`, [160](#)
- `__bernoulli_2n`
  - `std::__detail`, [161](#)
- `__bernoulli_series`
  - `std::__detail`, [161](#)
- `__beta`
  - `std::__detail`, [162](#)
- `__beta_gamma`
  - `std::__detail`, [162](#)
- `__beta_inc`
  - `std::__detail`, [163](#)
- `__beta_inc_cont_frac`
  - `std::__detail`, [163](#)
- `__beta_lgamma`
  - `std::__detail`, [164](#)
- `__beta_product`
  - `std::__detail`, [164](#)
- `__bincoef`
  - `std::__detail`, [165](#)
- `__binomial_cdf`
  - `std::__detail`, [165](#)
- `__binomial_cdfc`
  - `std::__detail`, [166](#)
- `__bose_einstein`
  - `std::__detail`, [166](#)
- `__chebyshev_recur`
  - `std::__detail`, [167](#)
- `__chebyshev_t`
  - `std::__detail`, [167](#)
- `__chebyshev_u`
  - `std::__detail`, [168](#)
- `__chebyshev_v`
  - `std::__detail`, [168](#)
- `__chebyshev_w`
  - `std::__detail`, [169](#)
- `__chshint`
  - `std::__detail`, [169](#)
- `__chshint_cont_frac`
  - `std::__detail`, [170](#)
- `__chshint_series`
  - `std::__detail`, [170](#)
- `__clamp_0_m2pi`
  - `std::__detail`, [170](#)
- `__clamp_pi`
  - `std::__detail`, [170](#)
- `__clausen`
  - `std::__detail`, [170](#), [171](#)
- `__clausen_c`
  - `std::__detail`, [171](#), [172](#)
- `__clausen_s`
  - `std::__detail`, [172](#), [173](#)
- `__comp_ellint_1`
  - `std::__detail`, [173](#)
- `__comp_ellint_2`
  - `std::__detail`, [174](#)
- `__comp_ellint_3`
  - `std::__detail`, [174](#)
- `__comp_ellint_d`
  - `std::__detail`, [175](#)
- `__comp_ellint_rf`
  - `std::__detail`, [175](#)
- `__comp_ellint_rg`
  - `std::__detail`, [175](#)
- `__conf_hyperg`
  - `std::__detail`, [175](#)
- `__conf_hyperg_lim`
  - `std::__detail`, [176](#)
- `__conf_hyperg_lim_series`
  - `std::__detail`, [176](#)
- `__conf_hyperg_luke`
  - `std::__detail`, [177](#)
- `__conf_hyperg_series`
  - `std::__detail`, [177](#)
- `__coshint`
  - `std::__detail`, [177](#)
- `__cpp_lib_math_special_functions`
  - `specfun.h`, [327](#)

- `__cyl_bessel`
  - `std::__detail`, 178
- `__cyl_bessel_i`
  - `std::__detail`, 178
- `__cyl_bessel_ij_series`
  - `std::__detail`, 179
- `__cyl_bessel_ik`
  - `std::__detail`, 179
- `__cyl_bessel_ik_asymp`
  - `std::__detail`, 181
- `__cyl_bessel_ik_steel`
  - `std::__detail`, 181
- `__cyl_bessel_j`
  - `std::__detail`, 182
- `__cyl_bessel_jn`
  - `std::__detail`, 182
- `__cyl_bessel_jn_asymp`
  - `std::__detail`, 182
- `__cyl_bessel_jn_steel`
  - `std::__detail`, 183
- `__cyl_bessel_k`
  - `std::__detail`, 183
- `__cyl_hankel_1`
  - `std::__detail`, 184
- `__cyl_hankel_2`
  - `std::__detail`, 185
- `__cyl_neumann`
  - `std::__detail`, 186
- `__cyl_neumann_n`
  - `std::__detail`, 186
- `__dawson`
  - `std::__detail`, 186
- `__dawson_cont_frac`
  - `std::__detail`, 187
- `__dawson_series`
  - `std::__detail`, 187
- `__debye_region`
  - `std::__detail`, 187
- `__dilog`
  - `std::__detail`, 187
- `__dirichlet_beta`
  - `std::__detail`, 188
- `__dirichlet_eta`
  - `std::__detail`, 189
- `__double_factorial`
  - `std::__detail`, 190
- `__ellint_1`
  - `std::__detail`, 190
- `__ellint_2`
  - `std::__detail`, 190
- `__ellint_3`
  - `std::__detail`, 191
- `__ellint_cel`
  - `std::__detail`, 191
- `__ellint_d`
  - `std::__detail`, 192
- `__ellint_el1`
  - `std::__detail`, 192
- `__ellint_el2`
  - `std::__detail`, 192
- `__ellint_el3`
  - `std::__detail`, 192
- `__ellint_rc`
  - `std::__detail`, 192
- `__ellint_rd`
  - `std::__detail`, 193
- `__ellint_rf`
  - `std::__detail`, 194
- `__ellint_rg`
  - `std::__detail`, 194
- `__ellint_rj`
  - `std::__detail`, 195
- `__ellnome`
  - `std::__detail`, 196
- `__ellnome_k`
  - `std::__detail`, 196
- `__ellnome_series`
  - `std::__detail`, 196
- `__expint`
  - `std::__detail`, 196, 197
- `__expint_E1`
  - `std::__detail`, 198
- `__expint_E1_asymp`
  - `std::__detail`, 198
- `__expint_E1_series`
  - `std::__detail`, 199
- `__expint_Ei`
  - `std::__detail`, 199
- `__expint_Ei_asymp`
  - `std::__detail`, 200
- `__expint_Ei_series`
  - `std::__detail`, 200
- `__expint_En_cont_frac`
  - `std::__detail`, 201
- `__expint_En_recursion`
  - `std::__detail`, 201
- `__expint_En_series`
  - `std::__detail`, 202
- `__expint_asymp`
  - `std::__detail`, 197
- `__expint_large_n`
  - `std::__detail`, 202
- `__factorial`
  - `std::__detail`, 203
  - `std::__detail::Factorial_table`, 263
- `__fermi_dirac`
  - `std::__detail`, 203
- `__fisher_f_cdf`

- std::\_\_detail, 204
- \_\_fisher\_f\_cdfc
  - std::\_\_detail, 204
- \_\_fock\_airy
  - std::\_\_detail, 205
- \_\_fpequal
  - std::\_\_detail, 205
- \_\_fpimag
  - std::\_\_detail, 205, 206
- \_\_fpreal
  - std::\_\_detail, 206
- \_\_fresnel
  - std::\_\_detail, 206
- \_\_fresnel\_cont\_frac
  - std::\_\_detail, 207
- \_\_fresnel\_series
  - std::\_\_detail, 207
- \_\_gamma
  - std::\_\_detail, 207
- \_\_gamma\_cont\_frac
  - std::\_\_detail, 208
- \_\_gamma\_l
  - std::\_\_detail, 208
- \_\_gamma\_series
  - std::\_\_detail, 208
- \_\_gamma\_temme
  - std::\_\_detail, 208
- \_\_gamma\_u
  - std::\_\_detail, 209
- \_\_gauss
  - std::\_\_detail, 209
- \_\_gegenbauer\_poly
  - std::\_\_detail, 209
- \_\_gnu\_cxx, 125
- \_\_hankel
  - std::\_\_detail, 210
- \_\_hankel\_debye
  - std::\_\_detail, 210
- \_\_hankel\_params
  - std::\_\_detail, 211
- \_\_hankel\_uniform
  - std::\_\_detail, 211
- \_\_hankel\_uniform\_olver
  - std::\_\_detail, 211
- \_\_hankel\_uniform\_outer
  - std::\_\_detail, 212
- \_\_hankel\_uniform\_sum
  - std::\_\_detail, 212
- \_\_heuman\_lambda
  - std::\_\_detail, 213
- \_\_hurwitz\_zeta
  - std::\_\_detail, 213, 214
- \_\_hurwitz\_zeta\_euler\_maclaurin
  - std::\_\_detail, 214
- \_\_hydrogen
  - std::\_\_detail, 214
- \_\_hyperg
  - std::\_\_detail, 215
- \_\_hyperg\_luke
  - std::\_\_detail, 215
- \_\_hyperg\_reflect
  - std::\_\_detail, 215
- \_\_hyperg\_series
  - std::\_\_detail, 216
- \_\_jacobi\_sncndn
  - std::\_\_detail, 217
- \_\_jacobi\_zeta
  - std::\_\_detail, 217
- \_\_laguerre
  - std::\_\_detail, 217
- \_\_legendre\_q
  - std::\_\_detail, 217
- \_\_log\_bincoef
  - std::\_\_detail, 218
- \_\_log\_double\_factorial
  - std::\_\_detail, 218
- \_\_log\_factorial
  - std::\_\_detail, 219
  - std::\_\_detail::\_Factorial\_table, 263
- \_\_log\_gamma
  - std::\_\_detail, 219
- \_\_log\_gamma\_bernoulli
  - std::\_\_detail, 220
- \_\_log\_gamma\_lanczos
  - std::\_\_detail, 220
- \_\_log\_gamma\_sign
  - std::\_\_detail, 220
- \_\_log\_gamma\_spouge
  - std::\_\_detail, 221
- \_\_log\_pochhammer\_l
  - std::\_\_detail, 221
- \_\_log\_pochhammer\_u
  - std::\_\_detail, 222
- \_\_logint
  - std::\_\_detail, 222
- \_\_n
  - std::\_\_detail::\_Factorial\_table, 263
- \_\_owens\_t
  - std::\_\_detail, 223
- \_\_pgamma
  - std::\_\_detail, 223
- \_\_pochhammer\_l
  - std::\_\_detail, 223
- \_\_pochhammer\_u
  - std::\_\_detail, 224
- \_\_poly\_hermite
  - std::\_\_detail, 224
- \_\_poly\_hermite\_asymp

- std::\_\_detail, [225](#)
- \_\_poly\_hermite\_recursion
  - std::\_\_detail, [225](#)
- \_\_poly\_jacobi
  - std::\_\_detail, [226](#)
- \_\_poly\_laguerre
  - std::\_\_detail, [226](#)
- \_\_poly\_laguerre\_hyperg
  - std::\_\_detail, [227](#)
- \_\_poly\_laguerre\_large\_n
  - std::\_\_detail, [228](#)
- \_\_poly\_laguerre\_recursion
  - std::\_\_detail, [228](#)
- \_\_poly\_legendre\_p
  - std::\_\_detail, [229](#)
- \_\_poly\_radial\_jacobi
  - std::\_\_detail, [229](#)
- \_\_polylog
  - std::\_\_detail, [230](#), [231](#)
- \_\_polylog\_exp
  - std::\_\_detail, [231](#)
- \_\_polylog\_exp\_asymp
  - std::\_\_detail, [232](#)
- \_\_polylog\_exp\_int\_neg
  - std::\_\_detail, [232](#), [233](#)
- \_\_polylog\_exp\_int\_pos
  - std::\_\_detail, [233](#), [234](#)
- \_\_polylog\_exp\_neg
  - std::\_\_detail, [234](#), [235](#)
- \_\_polylog\_exp\_neg\_even
  - std::\_\_detail, [235](#)
- \_\_polylog\_exp\_neg\_odd
  - std::\_\_detail, [236](#)
- \_\_polylog\_exp\_negative\_real\_part
  - std::\_\_detail, [237](#)
- \_\_polylog\_exp\_pos
  - std::\_\_detail, [237](#), [238](#)
- \_\_polylog\_exp\_real\_neg
  - std::\_\_detail, [239](#), [240](#)
- \_\_polylog\_exp\_real\_pos
  - std::\_\_detail, [240](#)
- \_\_psi
  - std::\_\_detail, [241](#)
- \_\_psi\_asymp
  - std::\_\_detail, [241](#)
- \_\_psi\_series
  - std::\_\_detail, [242](#)
- \_\_qgamma
  - std::\_\_detail, [242](#)
- \_\_riemann\_zeta
  - std::\_\_detail, [242](#)
- \_\_riemann\_zeta\_alt
  - std::\_\_detail, [243](#)
- \_\_riemann\_zeta\_euler\_maclaurin
  - std::\_\_detail, [243](#)
- \_\_riemann\_zeta\_glob
  - std::\_\_detail, [243](#)
- \_\_riemann\_zeta\_m\_1
  - std::\_\_detail, [243](#)
- \_\_riemann\_zeta\_m\_1\_sum
  - std::\_\_detail, [244](#)
- \_\_riemann\_zeta\_product
  - std::\_\_detail, [244](#)
- \_\_riemann\_zeta\_sum
  - std::\_\_detail, [245](#)
- \_\_sinc
  - std::\_\_detail, [245](#)
- \_\_sinc\_pi
  - std::\_\_detail, [245](#)
- \_\_sincosint
  - std::\_\_detail, [246](#)
- \_\_sincosint\_asymp
  - std::\_\_detail, [246](#)
- \_\_sincosint\_cont\_frac
  - std::\_\_detail, [246](#)
- \_\_sincosint\_series
  - std::\_\_detail, [246](#)
- \_\_sinhc
  - std::\_\_detail, [247](#)
- \_\_sinhc\_pi
  - std::\_\_detail, [247](#)
- \_\_sinhint
  - std::\_\_detail, [247](#)
- \_\_sph\_bessel
  - std::\_\_detail, [248](#)
- \_\_sph\_bessel\_ik
  - std::\_\_detail, [249](#)
- \_\_sph\_bessel\_jn
  - std::\_\_detail, [249](#)
- \_\_sph\_hankel
  - std::\_\_detail, [250](#)
- \_\_sph\_hankel\_1
  - std::\_\_detail, [250](#), [251](#)
- \_\_sph\_hankel\_2
  - std::\_\_detail, [251](#)
- \_\_sph\_harmonic
  - std::\_\_detail, [252](#)
- \_\_sph\_legendre
  - std::\_\_detail, [252](#)
- \_\_sph\_neumann
  - std::\_\_detail, [253](#)
- \_\_student\_t\_cdf
  - std::\_\_detail, [255](#)
- \_\_student\_t\_cdfc
  - std::\_\_detail, [255](#)
- \_\_theta\_1
  - std::\_\_detail, [256](#)
- \_\_theta\_2

- std::\_\_detail, [256](#)
- \_\_theta\_2\_asymp
  - std::\_\_detail, [256](#)
- \_\_theta\_2\_sum
  - std::\_\_detail, [257](#)
- \_\_theta\_3
  - std::\_\_detail, [257](#)
- \_\_theta\_3\_asymp
  - std::\_\_detail, [257](#)
- \_\_theta\_3\_sum
  - std::\_\_detail, [257](#)
- \_\_theta\_4
  - std::\_\_detail, [258](#)
- \_\_theta\_c
  - std::\_\_detail, [258](#)
- \_\_theta\_d
  - std::\_\_detail, [258](#)
- \_\_theta\_n
  - std::\_\_detail, [258](#)
- \_\_theta\_s
  - std::\_\_detail, [259](#)
- \_\_zernike
  - std::\_\_detail, [259](#)
- \_\_znorm1
  - std::\_\_detail, [260](#)
- \_\_znorm2
  - std::\_\_detail, [260](#)
- airy\_ai
  - GNU Extended Mathematical Special Functions, [50](#)
- airy\_aif
  - GNU Extended Mathematical Special Functions, [51](#)
- airy\_ail
  - GNU Extended Mathematical Special Functions, [51](#)
- airy\_bi
  - GNU Extended Mathematical Special Functions, [51](#)
- airy\_bif
  - GNU Extended Mathematical Special Functions, [52](#)
- airy\_bil
  - GNU Extended Mathematical Special Functions, [52](#)
- assoc\_laguerre
  - C++17/IS29124 Mathematical Special Functions, [20](#)
- assoc\_laguerref
  - C++17/IS29124 Mathematical Special Functions, [21](#)
- assoc\_laguerrel
  - C++17/IS29124 Mathematical Special Functions, [21](#)
- assoc\_legendre
  - C++17/IS29124 Mathematical Special Functions, [21](#)
- assoc\_legendref
  - C++17/IS29124 Mathematical Special Functions, [22](#)
- assoc\_legendrel
  - C++17/IS29124 Mathematical Special Functions, [22](#)
- bernoulli
  - GNU Extended Mathematical Special Functions, [52](#)
- bernoullif
  - GNU Extended Mathematical Special Functions, [52](#)
- bernoullil
  - GNU Extended Mathematical Special Functions, [53](#)
- beta
  - C++17/IS29124 Mathematical Special Functions, [22](#)
- betaf
  - C++17/IS29124 Mathematical Special Functions, [23](#)
- betal
  - C++17/IS29124 Mathematical Special Functions, [23](#)
- bincoef
  - GNU Extended Mathematical Special Functions, [53](#)
- bincoeff
  - GNU Extended Mathematical Special Functions, [53](#)
- bincoefl
  - GNU Extended Mathematical Special Functions, [53](#)
- bits/sf\_airy.tcc, [265](#)
- bits/sf\_bessel.tcc, [267](#)
- bits/sf\_beta.tcc, [269](#)
- bits/sf\_cardinal.tcc, [271](#)
- bits/sf\_chebyshev.tcc, [273](#)
- bits/sf\_dawson.tcc, [275](#)
- bits/sf\_ellint.tcc, [276](#)
- bits/sf\_expint.tcc, [279](#)
- bits/sf\_fresnel.tcc, [281](#)
- bits/sf\_gamma.tcc, [283](#)
- bits/sf\_gegenbauer.tcc, [288](#)
- bits/sf\_hankel.tcc, [290](#)
- bits/sf\_hankel\_new.tcc, [293](#)
- bits/sf\_hermite.tcc, [293](#)
- bits/sf\_hydrogen.tcc, [295](#)
- bits/sf\_hyperg.tcc, [296](#)
- bits/sf\_hypint.tcc, [298](#)
- bits/sf\_jacobi.tcc, [299](#)
- bits/sf\_laguerre.tcc, [301](#)
- bits/sf\_legendre.tcc, [303](#)
- bits/sf\_mod\_bessel.tcc, [304](#)
- bits/sf\_owens\_t.tcc, [306](#)
- bits/sf\_polylog.tcc, [307](#)
- bits/sf\_theta.tcc, [310](#)
- bits/sf\_trigint.tcc, [312](#)
- bits/sf\_zeta.tcc, [314](#)
- bits/specfun.h, [317](#)
- C++ Mathematical Special Functions, [17](#)
- C++17/IS29124 Mathematical Special Functions, [18](#)
  - assoc\_laguerre, [20](#)
  - assoc\_laguerref, [21](#)
  - assoc\_laguerrel, [21](#)
  - assoc\_legendre, [21](#)
  - assoc\_legendref, [22](#)
  - assoc\_legendrel, [22](#)
  - beta, [22](#)
  - betaf, [23](#)

- betal, [23](#)
- comp\_ellint\_1, [23](#)
- comp\_ellint\_1f, [24](#)
- comp\_ellint\_1l, [24](#)
- comp\_ellint\_2, [24](#)
- comp\_ellint\_2f, [25](#)
- comp\_ellint\_2l, [25](#)
- comp\_ellint\_3, [25](#)
- comp\_ellint\_3f, [26](#)
- comp\_ellint\_3l, [26](#)
- cyl\_bessel\_i, [26](#)
- cyl\_bessel\_if, [27](#)
- cyl\_bessel\_il, [27](#)
- cyl\_bessel\_j, [27](#)
- cyl\_bessel\_jf, [28](#)
- cyl\_bessel\_jl, [28](#)
- cyl\_bessel\_k, [28](#)
- cyl\_bessel\_kf, [29](#)
- cyl\_bessel\_kl, [29](#)
- cyl\_neumann, [29](#)
- cyl\_neumannf, [30](#)
- cyl\_neumannl, [30](#)
- ellint\_1, [30](#)
- ellint\_1f, [31](#)
- ellint\_1l, [31](#)
- ellint\_2, [31](#)
- ellint\_2f, [32](#)
- ellint\_2l, [32](#)
- ellint\_3, [32](#)
- ellint\_3f, [33](#)
- ellint\_3l, [33](#)
- expint, [34](#)
- expintf, [34](#)
- expintl, [34](#)
- hermite, [34](#)
- hermitef, [35](#)
- hermitel, [35](#)
- laguerre, [35](#)
- laguerref, [36](#)
- laguerrel, [36](#)
- legendre, [36](#)
- legendref, [37](#)
- legendrel, [37](#)
- riemann\_zeta, [37](#)
- riemann\_zetaf, [38](#)
- riemann\_zetal, [38](#)
- sph\_bessel, [38](#)
- sph\_besself, [39](#)
- sph\_bessell, [39](#)
- sph\_legendre, [39](#)
- sph\_legendref, [40](#)
- sph\_legendrel, [40](#)
- sph\_neumann, [40](#)
- sph\_neumannf, [41](#)
- sph\_neumannl, [41](#)
- COSINT
  - std::\_\_detail, [152](#)
- chebyshev\_t
  - GNU Extended Mathematical Special Functions, [53](#)
- chebyshev\_tf
  - GNU Extended Mathematical Special Functions, [54](#)
- chebyshev\_tl
  - GNU Extended Mathematical Special Functions, [54](#)
- chebyshev\_u
  - GNU Extended Mathematical Special Functions, [54](#)
- chebyshev\_uf
  - GNU Extended Mathematical Special Functions, [55](#)
- chebyshev\_ul
  - GNU Extended Mathematical Special Functions, [55](#)
- chebyshev\_v
  - GNU Extended Mathematical Special Functions, [55](#)
- chebyshev\_vf
  - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev\_vl
  - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev\_w
  - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev\_wf
  - GNU Extended Mathematical Special Functions, [57](#)
- chebyshev\_wl
  - GNU Extended Mathematical Special Functions, [57](#)
- clausen
  - GNU Extended Mathematical Special Functions, [57](#), [58](#)
- clausen\_c
  - GNU Extended Mathematical Special Functions, [58](#)
- clausen\_cf
  - GNU Extended Mathematical Special Functions, [58](#)
- clausen\_cl
  - GNU Extended Mathematical Special Functions, [58](#)
- clausen\_s
  - GNU Extended Mathematical Special Functions, [59](#)
- clausen\_sf
  - GNU Extended Mathematical Special Functions, [59](#)
- clausen\_sl
  - GNU Extended Mathematical Special Functions, [59](#)
- clausenf
  - GNU Extended Mathematical Special Functions, [59](#), [60](#)
- clausenl
  - GNU Extended Mathematical Special Functions, [60](#)
- comp\_ellint\_1
  - C++17/IS29124 Mathematical Special Functions, [23](#)
- comp\_ellint\_1f
  - C++17/IS29124 Mathematical Special Functions, [24](#)
- comp\_ellint\_1l
  - C++17/IS29124 Mathematical Special Functions, [24](#)
- comp\_ellint\_2

- C++17/IS29124 Mathematical Special Functions, [24](#)
- comp\_ellint\_2f
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- comp\_ellint\_2l
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- comp\_ellint\_3
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- comp\_ellint\_3f
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp\_ellint\_3l
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp\_ellint\_d
  - GNU Extended Mathematical Special Functions, [60](#)
- comp\_ellint\_df
  - GNU Extended Mathematical Special Functions, [61](#)
- comp\_ellint\_dl
  - GNU Extended Mathematical Special Functions, [61](#)
- comp\_ellint\_rf
  - GNU Extended Mathematical Special Functions, [61](#)
- comp\_ellint\_rg
  - GNU Extended Mathematical Special Functions, [62](#)
- conf\_hyperg
  - GNU Extended Mathematical Special Functions, [63](#)
- conf\_hyperg\_lim
  - GNU Extended Mathematical Special Functions, [63](#)
- conf\_hyperg\_limf
  - GNU Extended Mathematical Special Functions, [63](#)
- conf\_hyperg\_liml
  - GNU Extended Mathematical Special Functions, [64](#)
- conf\_hypergf
  - GNU Extended Mathematical Special Functions, [64](#)
- conf\_hypergl
  - GNU Extended Mathematical Special Functions, [64](#)
- coshint
  - GNU Extended Mathematical Special Functions, [64](#)
- coshintf
  - GNU Extended Mathematical Special Functions, [65](#)
- coshintl
  - GNU Extended Mathematical Special Functions, [65](#)
- cosint
  - GNU Extended Mathematical Special Functions, [65](#)
- cosintf
  - GNU Extended Mathematical Special Functions, [66](#)
- cosintl
  - GNU Extended Mathematical Special Functions, [66](#)
- cyl\_bessel\_i
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- cyl\_bessel\_if
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- cyl\_bessel\_il
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- cyl\_bessel\_j
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- cyl\_bessel\_jf
  - C++17/IS29124 Mathematical Special Functions, [28](#)
- cyl\_bessel\_k
  - C++17/IS29124 Mathematical Special Functions, [28](#)
- cyl\_bessel\_kf
  - C++17/IS29124 Mathematical Special Functions, [29](#)
- cyl\_bessel\_kl
  - C++17/IS29124 Mathematical Special Functions, [29](#)
- cyl\_hankel\_1
  - GNU Extended Mathematical Special Functions, [66](#), [67](#)
- cyl\_hankel\_1f
  - GNU Extended Mathematical Special Functions, [67](#)
- cyl\_hankel\_1l
  - GNU Extended Mathematical Special Functions, [68](#)
- cyl\_hankel\_2
  - GNU Extended Mathematical Special Functions, [68](#), [69](#)
- cyl\_hankel\_2f
  - GNU Extended Mathematical Special Functions, [69](#)
- cyl\_hankel\_2l
  - GNU Extended Mathematical Special Functions, [70](#)
- cyl\_neumann
  - C++17/IS29124 Mathematical Special Functions, [29](#)
- cyl\_neumannf
  - C++17/IS29124 Mathematical Special Functions, [30](#)
- cyl\_neumannl
  - C++17/IS29124 Mathematical Special Functions, [30](#)
- dawson
  - GNU Extended Mathematical Special Functions, [70](#)
- dawsonf
  - GNU Extended Mathematical Special Functions, [71](#)
- dawsonl
  - GNU Extended Mathematical Special Functions, [71](#)
- digamma
  - GNU Extended Mathematical Special Functions, [71](#)
- digammaf
  - GNU Extended Mathematical Special Functions, [71](#)
- digammal
  - GNU Extended Mathematical Special Functions, [71](#)
- dilog
  - GNU Extended Mathematical Special Functions, [71](#)
- dilogf
  - GNU Extended Mathematical Special Functions, [72](#)
- dilogl
  - GNU Extended Mathematical Special Functions, [72](#)
- dirichlet\_beta
  - GNU Extended Mathematical Special Functions, [72](#)
- dirichlet\_betaf
  - GNU Extended Mathematical Special Functions, [73](#)
- dirichlet\_betall
  - GNU Extended Mathematical Special Functions, [73](#)



- dirichlet\_eta
  - GNU Extended Mathematical Special Functions, [73](#)
- dirichlet\_etaf
  - GNU Extended Mathematical Special Functions, [73](#)
- dirichlet\_etat
  - GNU Extended Mathematical Special Functions, [74](#)
- double\_factorial
  - GNU Extended Mathematical Special Functions, [74](#)
- double\_factorialf
  - GNU Extended Mathematical Special Functions, [74](#)
- double\_factoriall
  - GNU Extended Mathematical Special Functions, [74](#)
- ellint\_1
  - C++17/IS29124 Mathematical Special Functions, [30](#)
- ellint\_1f
  - C++17/IS29124 Mathematical Special Functions, [31](#)
- ellint\_1l
  - C++17/IS29124 Mathematical Special Functions, [31](#)
- ellint\_2
  - C++17/IS29124 Mathematical Special Functions, [31](#)
- ellint\_2f
  - C++17/IS29124 Mathematical Special Functions, [32](#)
- ellint\_2l
  - C++17/IS29124 Mathematical Special Functions, [32](#)
- ellint\_3
  - C++17/IS29124 Mathematical Special Functions, [32](#)
- ellint\_3f
  - C++17/IS29124 Mathematical Special Functions, [33](#)
- ellint\_3l
  - C++17/IS29124 Mathematical Special Functions, [33](#)
- ellint\_cel
  - GNU Extended Mathematical Special Functions, [74](#)
- ellint\_celf
  - GNU Extended Mathematical Special Functions, [75](#)
- ellint\_cell
  - GNU Extended Mathematical Special Functions, [75](#)
- ellint\_d
  - GNU Extended Mathematical Special Functions, [75](#)
- ellint\_df
  - GNU Extended Mathematical Special Functions, [76](#)
- ellint\_dl
  - GNU Extended Mathematical Special Functions, [76](#)
- ellint\_el1
  - GNU Extended Mathematical Special Functions, [76](#)
- ellint\_el1f
  - GNU Extended Mathematical Special Functions, [77](#)
- ellint\_el1l
  - GNU Extended Mathematical Special Functions, [77](#)
- ellint\_el2
  - GNU Extended Mathematical Special Functions, [77](#)
- ellint\_el2f
  - GNU Extended Mathematical Special Functions, [77](#)
- ellint\_el2l
  - GNU Extended Mathematical Special Functions, [78](#)
- ellint\_el3
  - GNU Extended Mathematical Special Functions, [78](#)
- ellint\_el3f
  - GNU Extended Mathematical Special Functions, [78](#)
- ellint\_el3l
  - GNU Extended Mathematical Special Functions, [79](#)
- ellint\_rc
  - GNU Extended Mathematical Special Functions, [79](#)
- ellint\_rcf
  - GNU Extended Mathematical Special Functions, [80](#)
- ellint\_rcl
  - GNU Extended Mathematical Special Functions, [80](#)
- ellint\_rd
  - GNU Extended Mathematical Special Functions, [80](#)
- ellint\_rdf
  - GNU Extended Mathematical Special Functions, [81](#)
- ellint\_rdl
  - GNU Extended Mathematical Special Functions, [81](#)
- ellint\_rf
  - GNU Extended Mathematical Special Functions, [81](#)
- ellint\_rff
  - GNU Extended Mathematical Special Functions, [81](#)
- ellint\_rfl
  - GNU Extended Mathematical Special Functions, [82](#)
- ellint\_rg
  - GNU Extended Mathematical Special Functions, [82](#)
- ellint\_rgf
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_rgl
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_rj
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_rjf
  - GNU Extended Mathematical Special Functions, [84](#)
- ellint\_rjl
  - GNU Extended Mathematical Special Functions, [84](#)
- ellnome
  - GNU Extended Mathematical Special Functions, [84](#)
- ellnomef
  - GNU Extended Mathematical Special Functions, [84](#)
- ellnomel
  - GNU Extended Mathematical Special Functions, [84](#)
- evenzeta
  - std::\_\_detail, [260](#)
- expint
  - C++17/IS29124 Mathematical Special Functions, [34](#)
  - GNU Extended Mathematical Special Functions, [85](#)
- expintf
  - C++17/IS29124 Mathematical Special Functions, [34](#)
  - GNU Extended Mathematical Special Functions, [85](#)
- expintl
  - C++17/IS29124 Mathematical Special Functions, [34](#)
  - GNU Extended Mathematical Special Functions, [85](#)

- factorial
  - GNU Extended Mathematical Special Functions, 86
- factorialf
  - GNU Extended Mathematical Special Functions, 86
- factoriall
  - GNU Extended Mathematical Special Functions, 86
- fresnel\_c
  - GNU Extended Mathematical Special Functions, 86
- fresnel\_cf
  - GNU Extended Mathematical Special Functions, 86
- fresnel\_cl
  - GNU Extended Mathematical Special Functions, 86
- fresnel\_s
  - GNU Extended Mathematical Special Functions, 87
- fresnel\_sf
  - GNU Extended Mathematical Special Functions, 87
- fresnel\_sl
  - GNU Extended Mathematical Special Functions, 87
- GNU Extended Mathematical Special Functions, 42
  - \_GLIBCXX\_JACOBI\_CN, 50
  - \_GLIBCXX\_JACOBI\_DN, 50
  - \_GLIBCXX\_JACOBI\_SN, 50
  - airy\_ai, 50
  - airy\_aif, 51
  - airy\_ail, 51
  - airy\_bi, 51
  - airy\_bif, 52
  - airy\_bil, 52
  - bernoulli, 52
  - bernoullif, 52
  - bernoullil, 53
  - bincoef, 53
  - bincoeff, 53
  - bincoefl, 53
  - chebyshev\_t, 53
  - chebyshev\_tf, 54
  - chebyshev\_tl, 54
  - chebyshev\_u, 54
  - chebyshev\_uf, 55
  - chebyshev\_ul, 55
  - chebyshev\_v, 55
  - chebyshev\_vf, 56
  - chebyshev\_vl, 56
  - chebyshev\_w, 56
  - chebyshev\_wf, 57
  - chebyshev\_wl, 57
  - clausen, 57, 58
  - clausen\_c, 58
  - clausen\_cf, 58
  - clausen\_cl, 58
  - clausen\_s, 59
  - clausen\_sf, 59
  - clausen\_sl, 59
  - clausenf, 59, 60
  - clausenl, 60
  - comp\_ellint\_d, 60
  - comp\_ellint\_df, 61
  - comp\_ellint\_dl, 61
  - comp\_ellint\_rf, 61
  - comp\_ellint\_rg, 62
  - conf\_hyperg, 63
  - conf\_hyperg\_lim, 63
  - conf\_hyperg\_limf, 63
  - conf\_hyperg\_liml, 64
  - conf\_hypergf, 64
  - conf\_hypergl, 64
  - coshint, 64
  - coshintf, 65
  - coshintl, 65
  - cosint, 65
  - cosintf, 66
  - cosintl, 66
  - cyl\_hankel\_1, 66, 67
  - cyl\_hankel\_1f, 67
  - cyl\_hankel\_1l, 68
  - cyl\_hankel\_2, 68, 69
  - cyl\_hankel\_2f, 69
  - cyl\_hankel\_2l, 70
  - dawson, 70
  - dawsonf, 71
  - dawsonl, 71
  - digamma, 71
  - digammaf, 71
  - digammal, 71
  - dilog, 71
  - dilogf, 72
  - dilogl, 72
  - dirichlet\_beta, 72
  - dirichlet\_betaf, 73
  - dirichlet\_betall, 73
  - dirichlet\_eta, 73
  - dirichlet\_etaf, 73
  - dirichlet\_etall, 74
  - double\_factorial, 74
  - double\_factorialf, 74
  - double\_factoriall, 74
  - ellint\_cel, 74
  - ellint\_celf, 75
  - ellint\_cell, 75
  - ellint\_d, 75
  - ellint\_df, 76
  - ellint\_dl, 76
  - ellint\_el1, 76
  - ellint\_el1f, 77
  - ellint\_el1l, 77
  - ellint\_el2, 77
  - ellint\_el2f, 77

ellint\_el2l, 78  
ellint\_el3, 78  
ellint\_el3f, 78  
ellint\_el3l, 79  
ellint\_rc, 79  
ellint\_rcf, 80  
ellint\_rcl, 80  
ellint\_rd, 80  
ellint\_rdf, 81  
ellint\_rdl, 81  
ellint\_rf, 81  
ellint\_rff, 81  
ellint\_rfl, 82  
ellint\_rg, 82  
ellint\_rgf, 83  
ellint\_rgl, 83  
ellint\_rj, 83  
ellint\_rjf, 84  
ellint\_rjl, 84  
ellnome, 84  
ellnomef, 84  
ellnomel, 84  
expint, 85  
expintf, 85  
expintl, 85  
factorial, 86  
factorialf, 86  
factoriall, 86  
fresnel\_c, 86  
fresnel\_cf, 86  
fresnel\_cl, 86  
fresnel\_s, 87  
fresnel\_sf, 87  
fresnel\_sl, 87  
gamma\_l, 87  
gamma\_lf, 87  
gamma\_ll, 87  
gamma\_u, 87  
gamma\_uf, 88  
gamma\_ul, 88  
gegenbauer, 88  
gegenbauerf, 88  
gegenbauerl, 89  
heuman\_lambda, 89  
heuman\_lambdaf, 89  
heuman\_lambdal, 89  
hurwitz\_zeta, 90  
hurwitz\_zetaf, 90  
hurwitz\_zetal, 90  
hyperg, 91  
hypergf, 91  
hypergl, 91  
ibeta, 92  
ibetac, 92  
ibetacf, 93  
ibetacl, 93  
ibetaf, 93  
ibetal, 93  
jacobi, 93  
jacobi\_cn, 94  
jacobi\_cnf, 94  
jacobi\_cnl, 95  
jacobi\_dn, 95  
jacobi\_dnf, 95  
jacobi\_dnl, 96  
jacobi\_sn, 96  
jacobi\_snf, 96  
jacobi\_snl, 97  
jacobi\_zeta, 97  
jacobi\_zetaf, 97  
jacobi\_zetal, 97  
jacobif, 98  
jacobil, 98  
lbincoef, 98  
lbincoeff, 98  
lbincoefl, 98  
ldouble\_factorial, 98  
ldouble\_factorialf, 99  
ldouble\_factoriall, 99  
legendre\_q, 99  
legendre\_qf, 99  
legendre\_ql, 99  
lfactorial, 99  
lfactorialf, 99  
lfactoriall, 100  
logint, 100  
logintf, 100  
logintl, 100  
lpochhammer\_l, 100  
lpochhammer\_lf, 101  
lpochhammer\_ll, 101  
lpochhammer\_u, 101  
lpochhammer\_uf, 101  
lpochhammer\_ul, 101  
owens\_t, 101  
owens\_tf, 102  
owens\_tl, 102  
pgamma, 102  
pgammaf, 102  
pgammal, 102  
pochhammer\_l, 102  
pochhammer\_lf, 102  
pochhammer\_ll, 103  
pochhammer\_u, 103  
pochhammer\_uf, 103  
pochhammer\_ul, 103  
polylog, 103  
polylogf, 104

- polylogl, [104](#)
- psi, [105](#)
- psif, [105](#)
- psil, [105](#)
- qgamma, [105](#)
- qgammaf, [105](#)
- qgammal, [105](#)
- radpoly, [106](#)
- radpolyf, [106](#)
- radpolyl, [106](#)
- sinc, [107](#)
- sinc\_pi, [107](#)
- sinc\_pif, [107](#)
- sinc\_pil, [107](#)
- sincf, [107](#)
- sincl, [107](#)
- sinhc, [107](#)
- sinhc\_pi, [107](#)
- sinhc\_pif, [108](#)
- sinhc\_pil, [108](#)
- sinhcf, [108](#)
- sinhcl, [108](#)
- sinhint, [108](#)
- sinhintf, [108](#)
- sinhintl, [109](#)
- sinint, [109](#)
- sinintf, [109](#)
- sinintl, [109](#)
- sph\_bessel\_i, [110](#)
- sph\_bessel\_if, [110](#)
- sph\_bessel\_il, [110](#)
- sph\_bessel\_k, [111](#)
- sph\_bessel\_kf, [111](#)
- sph\_bessel\_kl, [111](#)
- sph\_hankel\_1, [112](#)
- sph\_hankel\_1f, [113](#)
- sph\_hankel\_1l, [113](#)
- sph\_hankel\_2, [114](#)
- sph\_hankel\_2f, [115](#)
- sph\_hankel\_2l, [115](#)
- sph\_harmonic, [116](#)
- sph\_harmonicf, [116](#)
- sph\_harmonicl, [116](#)
- theta\_1, [117](#)
- theta\_1f, [117](#)
- theta\_1l, [117](#)
- theta\_2, [117](#)
- theta\_2f, [118](#)
- theta\_2l, [118](#)
- theta\_3, [118](#)
- theta\_3f, [119](#)
- theta\_3l, [119](#)
- theta\_4, [119](#)
- theta\_4f, [120](#)
- theta\_4l, [120](#)
- theta\_c, [120](#)
- theta\_cf, [120](#)
- theta\_cl, [121](#)
- theta\_d, [121](#)
- theta\_df, [121](#)
- theta\_dl, [121](#)
- theta\_n, [122](#)
- theta\_nf, [122](#)
- theta\_nl, [122](#)
- theta\_s, [122](#)
- theta\_sf, [123](#)
- theta\_sl, [123](#)
- zernike, [123](#)
- zernikef, [124](#)
- zernikel, [124](#)
- gamma\_l
  - GNU Extended Mathematical Special Functions, [87](#)
- gamma\_lf
  - GNU Extended Mathematical Special Functions, [87](#)
- gamma\_ll
  - GNU Extended Mathematical Special Functions, [87](#)
- gamma\_u
  - GNU Extended Mathematical Special Functions, [87](#)
- gamma\_uf
  - GNU Extended Mathematical Special Functions, [88](#)
- gamma\_ul
  - GNU Extended Mathematical Special Functions, [88](#)
- gegenbauer
  - GNU Extended Mathematical Special Functions, [88](#)
- gegenbauerf
  - GNU Extended Mathematical Special Functions, [88](#)
- gegenbauerl
  - GNU Extended Mathematical Special Functions, [89](#)
- hermite
  - C++17/IS29124 Mathematical Special Functions, [34](#)
- hermitef
  - C++17/IS29124 Mathematical Special Functions, [35](#)
- hermitel
  - C++17/IS29124 Mathematical Special Functions, [35](#)
- heuman\_lambda
  - GNU Extended Mathematical Special Functions, [89](#)
- heuman\_lambdaf
  - GNU Extended Mathematical Special Functions, [89](#)
- heuman\_lambdal
  - GNU Extended Mathematical Special Functions, [89](#)
- hurwitz\_zeta
  - GNU Extended Mathematical Special Functions, [90](#)
- hurwitz\_zetaf
  - GNU Extended Mathematical Special Functions, [90](#)
- hurwitz\_zetal
  - GNU Extended Mathematical Special Functions, [90](#)
- hyperg

- GNU Extended Mathematical Special Functions, [91](#)
- hypergf
  - GNU Extended Mathematical Special Functions, [91](#)
- hypergl
  - GNU Extended Mathematical Special Functions, [91](#)
- ibeta
  - GNU Extended Mathematical Special Functions, [92](#)
- ibetac
  - GNU Extended Mathematical Special Functions, [92](#)
- ibetacf
  - GNU Extended Mathematical Special Functions, [93](#)
- ibetacl
  - GNU Extended Mathematical Special Functions, [93](#)
- ibetaf
  - GNU Extended Mathematical Special Functions, [93](#)
- ibetal
  - GNU Extended Mathematical Special Functions, [93](#)
- jacobi
  - GNU Extended Mathematical Special Functions, [93](#)
- jacobi\_cn
  - GNU Extended Mathematical Special Functions, [94](#)
- jacobi\_cnf
  - GNU Extended Mathematical Special Functions, [94](#)
- jacobi\_cnl
  - GNU Extended Mathematical Special Functions, [95](#)
- jacobi\_dn
  - GNU Extended Mathematical Special Functions, [95](#)
- jacobi\_dnf
  - GNU Extended Mathematical Special Functions, [95](#)
- jacobi\_dnl
  - GNU Extended Mathematical Special Functions, [96](#)
- jacobi\_sn
  - GNU Extended Mathematical Special Functions, [96](#)
- jacobi\_snf
  - GNU Extended Mathematical Special Functions, [96](#)
- jacobi\_snl
  - GNU Extended Mathematical Special Functions, [97](#)
- jacobi\_zeta
  - GNU Extended Mathematical Special Functions, [97](#)
- jacobi\_zetaf
  - GNU Extended Mathematical Special Functions, [97](#)
- jacobi\_zetal
  - GNU Extended Mathematical Special Functions, [97](#)
- jacobif
  - GNU Extended Mathematical Special Functions, [98](#)
- jacobil
  - GNU Extended Mathematical Special Functions, [98](#)
- laguerre
  - C++17/IS29124 Mathematical Special Functions, [35](#)
- laguerref
  - C++17/IS29124 Mathematical Special Functions, [36](#)
- laguerrel
  - C++17/IS29124 Mathematical Special Functions, [36](#)
- lbincoef
  - GNU Extended Mathematical Special Functions, [98](#)
- lbincoeff
  - GNU Extended Mathematical Special Functions, [98](#)
- lbincoefl
  - GNU Extended Mathematical Special Functions, [98](#)
- ldouble\_factorial
  - GNU Extended Mathematical Special Functions, [98](#)
- ldouble\_factorialf
  - GNU Extended Mathematical Special Functions, [99](#)
- ldouble\_factoriall
  - GNU Extended Mathematical Special Functions, [99](#)
- legendre
  - C++17/IS29124 Mathematical Special Functions, [36](#)
- legendre\_q
  - GNU Extended Mathematical Special Functions, [99](#)
- legendre\_qf
  - GNU Extended Mathematical Special Functions, [99](#)
- legendre\_ql
  - GNU Extended Mathematical Special Functions, [99](#)
- legendref
  - C++17/IS29124 Mathematical Special Functions, [37](#)
- legendrel
  - C++17/IS29124 Mathematical Special Functions, [37](#)
- lfactorial
  - GNU Extended Mathematical Special Functions, [99](#)
- lfactorialf
  - GNU Extended Mathematical Special Functions, [99](#)
- lfactoriall
  - GNU Extended Mathematical Special Functions, [100](#)
- logint
  - GNU Extended Mathematical Special Functions, [100](#)
- logintf
  - GNU Extended Mathematical Special Functions, [100](#)
- logintl
  - GNU Extended Mathematical Special Functions, [100](#)
- lpochhammer\_l
  - GNU Extended Mathematical Special Functions, [100](#)
- lpochhammer\_lf
  - GNU Extended Mathematical Special Functions, [101](#)
- lpochhammer\_ll
  - GNU Extended Mathematical Special Functions, [101](#)
- lpochhammer\_u
  - GNU Extended Mathematical Special Functions, [101](#)
- lpochhammer\_uf
  - GNU Extended Mathematical Special Functions, [101](#)
- lpochhammer\_ul
  - GNU Extended Mathematical Special Functions, [101](#)
- owens\_t
  - GNU Extended Mathematical Special Functions, [101](#)
- owens\_tf
  - GNU Extended Mathematical Special Functions, [102](#)

- owens\_tl
  - GNU Extended Mathematical Special Functions, [102](#)
- pgamma
  - GNU Extended Mathematical Special Functions, [102](#)
- pgammaf
  - GNU Extended Mathematical Special Functions, [102](#)
- pgammal
  - GNU Extended Mathematical Special Functions, [102](#)
- pochhammer\_l
  - GNU Extended Mathematical Special Functions, [102](#)
- pochhammer\_lf
  - GNU Extended Mathematical Special Functions, [102](#)
- pochhammer\_ll
  - GNU Extended Mathematical Special Functions, [103](#)
- pochhammer\_u
  - GNU Extended Mathematical Special Functions, [103](#)
- pochhammer\_uf
  - GNU Extended Mathematical Special Functions, [103](#)
- pochhammer\_ul
  - GNU Extended Mathematical Special Functions, [103](#)
- polylog
  - GNU Extended Mathematical Special Functions, [103](#)
- polylogf
  - GNU Extended Mathematical Special Functions, [104](#)
- polylogl
  - GNU Extended Mathematical Special Functions, [104](#)
- psi
  - GNU Extended Mathematical Special Functions, [105](#)
- psif
  - GNU Extended Mathematical Special Functions, [105](#)
- psil
  - GNU Extended Mathematical Special Functions, [105](#)
- qgamma
  - GNU Extended Mathematical Special Functions, [105](#)
- qgammaf
  - GNU Extended Mathematical Special Functions, [105](#)
- qgammal
  - GNU Extended Mathematical Special Functions, [105](#)
- radpoly
  - GNU Extended Mathematical Special Functions, [106](#)
- radpolyf
  - GNU Extended Mathematical Special Functions, [106](#)
- radpolyl
  - GNU Extended Mathematical Special Functions, [106](#)
- riemann\_zeta
  - C++17/IS29124 Mathematical Special Functions, [37](#)
- riemann\_zetaf
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- riemann\_zetal
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- SININT
  - std::\_\_detail, [152](#)
- sfairy.tcc
  - \_GLIBCXX\_BITS\_SF\_AIRY\_TCC, [267](#)
- sf\_bessel.tcc
  - \_GLIBCXX\_BITS\_SF\_BESSEL\_TCC, [269](#)
- sf\_beta.tcc
  - \_GLIBCXX\_BITS\_SF\_BETA\_TCC, [271](#)
- sf\_cardinal.tcc
  - \_GLIBCXX\_BITS\_SF\_CARDINAL\_TCC, [273](#)
- sf\_chebyshev.tcc
  - \_GLIBCXX\_SF\_CHEBYSHEV\_TCC, [275](#)
- sf\_dawson.tcc
  - \_GLIBCXX\_SF\_DAWSON\_TCC, [276](#)
- sf\_ellint.tcc
  - \_GLIBCXX\_BITS\_SF\_ELLINT\_TCC, [278](#)
- sf\_expint.tcc
  - \_GLIBCXX\_BITS\_SF\_EXPINT\_TCC, [281](#)
- sf\_fresnel.tcc
  - \_GLIBCXX\_SF\_FRESNEL\_TCC, [282](#)
- sf\_gamma.tcc
  - \_GLIBCXX\_BITS\_SF\_GAMMA\_TCC, [288](#)
- sf\_gegenbauer.tcc
  - \_GLIBCXX\_SF\_GEGENBAUER\_TCC, [289](#)
- sf\_hankel.tcc
  - \_GLIBCXX\_BITS\_SF\_HANKEL\_TCC, [292](#)
- sf\_hankel\_new.tcc
  - \_GLIBCXX\_BITS\_SF\_HANKEL\_NEW\_TCC, [293](#)
- sf\_hermite.tcc
  - \_GLIBCXX\_BITS\_SF\_HERMITE\_TCC, [294](#)
- sf\_hydrogen.tcc
  - \_GLIBCXX\_BITS\_SF\_HYDROGEN\_TCC, [295](#)
- sf\_hyperg.tcc
  - \_GLIBCXX\_BITS\_SF\_HYPERG\_TCC, [297](#)
- sf\_hypint.tcc
  - \_GLIBCXX\_SF\_HYPINT\_TCC, [299](#)
- sf\_jacobi.tcc
  - \_GLIBCXX\_SF\_JACOBI\_TCC, [300](#)
- sf\_laguerre.tcc
  - \_GLIBCXX\_BITS\_SF\_LAGUERRE\_TCC, [302](#)
- sf\_legendre.tcc
  - \_GLIBCXX\_BITS\_SF\_LEGENDRE\_TCC, [304](#)
- sf\_mod\_bessel.tcc
  - \_GLIBCXX\_BITS\_SF\_MOD\_BESSEL\_TCC, [306](#)
- sf\_owens\_t.tcc
  - \_GLIBCXX\_BITS\_SF\_OWENS\_T\_TCC, [307](#)
- sf\_polylog.tcc
  - \_GLIBCXX\_BITS\_SF\_POLYLOG\_TCC, [310](#)
- sf\_theta.tcc
  - \_GLIBCXX\_SF\_THETA\_TCC, [312](#)
- sf\_trigint.tcc
  - \_GLIBCXX\_SF\_TRIGINT\_TCC, [314](#)
- sf\_zeta.tcc
  - \_GLIBCXX\_BITS\_SF\_ZETA\_TCC, [316](#)
- sinc

- GNU Extended Mathematical Special Functions, [107](#)
- `sinc_pi`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sinc_pif`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sinc_pil`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sincf`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sincl`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sinhc`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sinhc_pi`
  - GNU Extended Mathematical Special Functions, [107](#)
- `sinhc_pif`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhc_pil`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhcf`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhcl`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhint`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhintf`
  - GNU Extended Mathematical Special Functions, [108](#)
- `sinhintl`
  - GNU Extended Mathematical Special Functions, [109](#)
- `sinint`
  - GNU Extended Mathematical Special Functions, [109](#)
- `sinintf`
  - GNU Extended Mathematical Special Functions, [109](#)
- `sinintl`
  - GNU Extended Mathematical Special Functions, [109](#)
- `specfun.h`
  - `__STDCPP_MATH_SPEC_FUNCS__`, [327](#)
  - `__cpp_lib_math_special_functions`, [327](#)
- `sph_bessel`
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- `sph_bessel_i`
  - GNU Extended Mathematical Special Functions, [110](#)
- `sph_bessel_if`
  - GNU Extended Mathematical Special Functions, [110](#)
- `sph_bessel_il`
  - GNU Extended Mathematical Special Functions, [110](#)
- `sph_bessel_k`
  - GNU Extended Mathematical Special Functions, [111](#)
- `sph_bessel_kf`
  - GNU Extended Mathematical Special Functions, [111](#)
- `sph_bessel_kl`
  - GNU Extended Mathematical Special Functions, [111](#)
- `sph_besself`
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- `sph_bessell`
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- `sph_hankel_1`
  - GNU Extended Mathematical Special Functions, [112](#)
- `sph_hankel_1f`
  - GNU Extended Mathematical Special Functions, [113](#)
- `sph_hankel_1l`
  - GNU Extended Mathematical Special Functions, [113](#)
- `sph_hankel_2`
  - GNU Extended Mathematical Special Functions, [114](#)
- `sph_hankel_2f`
  - GNU Extended Mathematical Special Functions, [115](#)
- `sph_hankel_2l`
  - GNU Extended Mathematical Special Functions, [115](#)
- `sph_harmonic`
  - GNU Extended Mathematical Special Functions, [116](#)
- `sph_harmonicf`
  - GNU Extended Mathematical Special Functions, [116](#)
- `sph_harmonicl`
  - GNU Extended Mathematical Special Functions, [116](#)
- `sph_legendre`
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- `sph_legendref`
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- `sph_legendrel`
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- `sph_neumann`
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- `sph_neumannf`
  - C++17/IS29124 Mathematical Special Functions, [41](#)
- `sph_neumannl`
  - C++17/IS29124 Mathematical Special Functions, [41](#)
- `std`, [133](#)
- `std::__detail`, [135](#)
- `_Num_Euler_Maclaurin_zeta`, [260](#)
- `_S_Euler_Maclaurin_zeta`, [261](#)
- `_S_double_factorial_table`, [260](#)
- `_S_factorial_table`, [261](#)
- `_S_neg_double_factorial_table`, [261](#)
- `_S_num_double_factorials`, [261](#)
- `_S_num_double_factorials< double >`, [261](#)
- `_S_num_double_factorials< float >`, [261](#)
- `_S_num_double_factorials< long double >`, [261](#)
- `_S_num_factorials`, [261](#)
- `_S_num_factorials< double >`, [262](#)
- `_S_num_factorials< float >`, [262](#)
- `_S_num_factorials< long double >`, [262](#)
- `_S_num_neg_double_factorials`, [262](#)
- `_S_num_neg_double_factorials< double >`, [262](#)
- `_S_num_neg_double_factorials< float >`, [262](#)
- `_S_num_neg_double_factorials< long double >`, [262](#)
- `_S_num_zetam1`, [262](#)
- `_S_zetam1`, [262](#)
- `__airy`, [153](#)



- \_\_airy\_ai, 155
- \_\_airy\_arg, 155
- \_\_airy\_asymp\_absarg\_ge\_pio3, 155
- \_\_airy\_asymp\_absarg\_lt\_pio3, 156
- \_\_airy\_bessel\_i, 156
- \_\_airy\_bessel\_k, 158
- \_\_airy\_bi, 158
- \_\_airy\_hyperg\_rational, 158
- \_\_assoc\_laguerre, 159
- \_\_assoc\_legendre\_p, 160
- \_\_bernoulli, 160
- \_\_bernoulli\_2n, 161
- \_\_bernoulli\_series, 161
- \_\_beta, 162
- \_\_beta\_gamma, 162
- \_\_beta\_inc, 163
- \_\_beta\_inc\_cont\_frac, 163
- \_\_beta\_lgamma, 164
- \_\_beta\_product, 164
- \_\_bincoef, 165
- \_\_binomial\_cdf, 165
- \_\_binomial\_cdfc, 166
- \_\_bose\_einstein, 166
- \_\_chebyshev\_recur, 167
- \_\_chebyshev\_t, 167
- \_\_chebyshev\_u, 168
- \_\_chebyshev\_v, 168
- \_\_chebyshev\_w, 169
- \_\_chshint, 169
- \_\_chshint\_cont\_frac, 170
- \_\_chshint\_series, 170
- \_\_clamp\_0\_m2pi, 170
- \_\_clamp\_pi, 170
- \_\_clausen, 170, 171
- \_\_clausen\_c, 171, 172
- \_\_clausen\_s, 172, 173
- \_\_comp\_ellint\_1, 173
- \_\_comp\_ellint\_2, 174
- \_\_comp\_ellint\_3, 174
- \_\_comp\_ellint\_d, 175
- \_\_comp\_ellint\_rf, 175
- \_\_comp\_ellint\_rg, 175
- \_\_conf\_hyperg, 175
- \_\_conf\_hyperg\_lim, 176
- \_\_conf\_hyperg\_lim\_series, 176
- \_\_conf\_hyperg\_luke, 177
- \_\_conf\_hyperg\_series, 177
- \_\_coshint, 177
- \_\_cyl\_bessel, 178
- \_\_cyl\_bessel\_i, 178
- \_\_cyl\_bessel\_ij\_series, 179
- \_\_cyl\_bessel\_ik, 179
- \_\_cyl\_bessel\_ik\_asymp, 181
- \_\_cyl\_bessel\_ik\_steel, 181
- \_\_cyl\_bessel\_j, 182
- \_\_cyl\_bessel\_jn, 182
- \_\_cyl\_bessel\_jn\_asymp, 182
- \_\_cyl\_bessel\_jn\_steel, 183
- \_\_cyl\_bessel\_k, 183
- \_\_cyl\_hankel\_1, 184
- \_\_cyl\_hankel\_2, 185
- \_\_cyl\_neumann, 186
- \_\_cyl\_neumann\_n, 186
- \_\_dawson, 186
- \_\_dawson\_cont\_frac, 187
- \_\_dawson\_series, 187
- \_\_debye\_region, 187
- \_\_dilog, 187
- \_\_dirichlet\_beta, 188
- \_\_dirichlet\_eta, 189
- \_\_double\_factorial, 190
- \_\_ellint\_1, 190
- \_\_ellint\_2, 190
- \_\_ellint\_3, 191
- \_\_ellint\_cel, 191
- \_\_ellint\_d, 192
- \_\_ellint\_el1, 192
- \_\_ellint\_el2, 192
- \_\_ellint\_el3, 192
- \_\_ellint\_rc, 192
- \_\_ellint\_rd, 193
- \_\_ellint\_rf, 194
- \_\_ellint\_rg, 194
- \_\_ellint\_rj, 195
- \_\_ellnome, 196
- \_\_ellnome\_k, 196
- \_\_ellnome\_series, 196
- \_\_expint, 196, 197
- \_\_expint\_E1, 198
- \_\_expint\_E1\_asymp, 198
- \_\_expint\_E1\_series, 199
- \_\_expint\_Ei, 199
- \_\_expint\_Ei\_asymp, 200
- \_\_expint\_Ei\_series, 200
- \_\_expint\_En\_cont\_frac, 201
- \_\_expint\_En\_recursion, 201
- \_\_expint\_En\_series, 202
- \_\_expint\_asymp, 197
- \_\_expint\_large\_n, 202
- \_\_factorial, 203
- \_\_fermi\_dirac, 203
- \_\_fisher\_f\_cdf, 204
- \_\_fisher\_f\_cdfc, 204
- \_\_fock\_airy, 205
- \_\_fpequal, 205
- \_\_fpimag, 205, 206
- \_\_fpreal, 206
- \_\_fresnel, 206



- [\\_\\_fresnel\\_cont\\_frac](#), 207
- [\\_\\_fresnel\\_series](#), 207
- [\\_\\_gamma](#), 207
- [\\_\\_gamma\\_cont\\_frac](#), 208
- [\\_\\_gamma\\_l](#), 208
- [\\_\\_gamma\\_series](#), 208
- [\\_\\_gamma\\_temme](#), 208
- [\\_\\_gamma\\_u](#), 209
- [\\_\\_gauss](#), 209
- [\\_\\_gegenbauer\\_poly](#), 209
- [\\_\\_hankel](#), 210
- [\\_\\_hankel\\_debye](#), 210
- [\\_\\_hankel\\_params](#), 211
- [\\_\\_hankel\\_uniform](#), 211
- [\\_\\_hankel\\_uniform\\_olver](#), 211
- [\\_\\_hankel\\_uniform\\_outer](#), 212
- [\\_\\_hankel\\_uniform\\_sum](#), 212
- [\\_\\_heuman\\_lambda](#), 213
- [\\_\\_hurwitz\\_zeta](#), 213, 214
- [\\_\\_hurwitz\\_zeta\\_euler\\_maclaurin](#), 214
- [\\_\\_hydrogen](#), 214
- [\\_\\_hyperg](#), 215
- [\\_\\_hyperg\\_luke](#), 215
- [\\_\\_hyperg\\_reflect](#), 215
- [\\_\\_hyperg\\_series](#), 216
- [\\_\\_jacobi\\_sncndn](#), 217
- [\\_\\_jacobi\\_zeta](#), 217
- [\\_\\_laguerre](#), 217
- [\\_\\_legendre\\_q](#), 217
- [\\_\\_log\\_bincoef](#), 218
- [\\_\\_log\\_double\\_factorial](#), 218
- [\\_\\_log\\_factorial](#), 219
- [\\_\\_log\\_gamma](#), 219
- [\\_\\_log\\_gamma\\_bernoulli](#), 220
- [\\_\\_log\\_gamma\\_lanczos](#), 220
- [\\_\\_log\\_gamma\\_sign](#), 220
- [\\_\\_log\\_gamma\\_spouge](#), 221
- [\\_\\_log\\_pochhammer\\_l](#), 221
- [\\_\\_log\\_pochhammer\\_u](#), 222
- [\\_\\_logint](#), 222
- [\\_\\_owens\\_t](#), 223
- [\\_\\_pgamma](#), 223
- [\\_\\_pochhammer\\_l](#), 223
- [\\_\\_pochhammer\\_u](#), 224
- [\\_\\_poly\\_hermite](#), 224
- [\\_\\_poly\\_hermite\\_asymp](#), 225
- [\\_\\_poly\\_hermite\\_recursion](#), 225
- [\\_\\_poly\\_jacobi](#), 226
- [\\_\\_poly\\_laguerre](#), 226
- [\\_\\_poly\\_laguerre\\_hyperg](#), 227
- [\\_\\_poly\\_laguerre\\_large\\_n](#), 228
- [\\_\\_poly\\_laguerre\\_recursion](#), 228
- [\\_\\_poly\\_legendre\\_p](#), 229
- [\\_\\_poly\\_radial\\_jacobi](#), 229
- [\\_\\_polylog](#), 230, 231
- [\\_\\_polylog\\_exp](#), 231
- [\\_\\_polylog\\_exp\\_asymp](#), 232
- [\\_\\_polylog\\_exp\\_int\\_neg](#), 232, 233
- [\\_\\_polylog\\_exp\\_int\\_pos](#), 233, 234
- [\\_\\_polylog\\_exp\\_neg](#), 234, 235
- [\\_\\_polylog\\_exp\\_neg\\_even](#), 235
- [\\_\\_polylog\\_exp\\_neg\\_odd](#), 236
- [\\_\\_polylog\\_exp\\_negative\\_real\\_part](#), 237
- [\\_\\_polylog\\_exp\\_pos](#), 237, 238
- [\\_\\_polylog\\_exp\\_real\\_neg](#), 239, 240
- [\\_\\_polylog\\_exp\\_real\\_pos](#), 240
- [\\_\\_psi](#), 241
- [\\_\\_psi\\_asymp](#), 241
- [\\_\\_psi\\_series](#), 242
- [\\_\\_qgamma](#), 242
- [\\_\\_riemann\\_zeta](#), 242
- [\\_\\_riemann\\_zeta\\_alt](#), 243
- [\\_\\_riemann\\_zeta\\_euler\\_maclaurin](#), 243
- [\\_\\_riemann\\_zeta\\_glob](#), 243
- [\\_\\_riemann\\_zeta\\_m\\_1](#), 243
- [\\_\\_riemann\\_zeta\\_m\\_1\\_sum](#), 244
- [\\_\\_riemann\\_zeta\\_product](#), 244
- [\\_\\_riemann\\_zeta\\_sum](#), 245
- [\\_\\_sinc](#), 245
- [\\_\\_sinc\\_pi](#), 245
- [\\_\\_sincosint](#), 246
- [\\_\\_sincosint\\_asymp](#), 246
- [\\_\\_sincosint\\_cont\\_frac](#), 246
- [\\_\\_sincosint\\_series](#), 246
- [\\_\\_sinhc](#), 247
- [\\_\\_sinhc\\_pi](#), 247
- [\\_\\_sinhint](#), 247
- [\\_\\_sph\\_bessel](#), 248
- [\\_\\_sph\\_bessel\\_ik](#), 249
- [\\_\\_sph\\_bessel\\_jn](#), 249
- [\\_\\_sph\\_hankel](#), 250
- [\\_\\_sph\\_hankel\\_1](#), 250, 251
- [\\_\\_sph\\_hankel\\_2](#), 251
- [\\_\\_sph\\_harmonic](#), 252
- [\\_\\_sph\\_legendre](#), 252
- [\\_\\_sph\\_neumann](#), 253
- [\\_\\_student\\_t\\_cdf](#), 255
- [\\_\\_student\\_t\\_cdfc](#), 255
- [\\_\\_theta\\_1](#), 256
- [\\_\\_theta\\_2](#), 256
- [\\_\\_theta\\_2\\_asymp](#), 256
- [\\_\\_theta\\_2\\_sum](#), 257
- [\\_\\_theta\\_3](#), 257
- [\\_\\_theta\\_3\\_asymp](#), 257
- [\\_\\_theta\\_3\\_sum](#), 257
- [\\_\\_theta\\_4](#), 258
- [\\_\\_theta\\_c](#), 258
- [\\_\\_theta\\_d](#), 258

- [\\_\\_theta\\_n](#), [258](#)
  - [\\_\\_theta\\_s](#), [259](#)
  - [\\_\\_zernike](#), [259](#)
  - [\\_\\_znorm1](#), [260](#)
  - [\\_\\_znorm2](#), [260](#)
- [COSINT](#), [152](#)
- [evenzeta](#), [260](#)
- [SININT](#), [152](#)
- [std::\\_\\_detail::\\_Factorial\\_table](#)
  - [\\_\\_factorial](#), [263](#)
  - [\\_\\_log\\_factorial](#), [263](#)
  - [\\_\\_n](#), [263](#)
- [std::\\_\\_detail::\\_Factorial\\_table<\\_Tp>](#), [263](#)
- [theta\\_1](#)
  - GNU Extended Mathematical Special Functions, [117](#)
- [theta\\_1f](#)
  - GNU Extended Mathematical Special Functions, [117](#)
- [theta\\_1l](#)
  - GNU Extended Mathematical Special Functions, [117](#)
- [theta\\_2](#)
  - GNU Extended Mathematical Special Functions, [117](#)
- [theta\\_2f](#)
  - GNU Extended Mathematical Special Functions, [118](#)
- [theta\\_2l](#)
  - GNU Extended Mathematical Special Functions, [118](#)
- [theta\\_3](#)
  - GNU Extended Mathematical Special Functions, [118](#)
- [theta\\_3f](#)
  - GNU Extended Mathematical Special Functions, [119](#)
- [theta\\_3l](#)
  - GNU Extended Mathematical Special Functions, [119](#)
- [theta\\_4](#)
  - GNU Extended Mathematical Special Functions, [119](#)
- [theta\\_4f](#)
  - GNU Extended Mathematical Special Functions, [120](#)
- [theta\\_4l](#)
  - GNU Extended Mathematical Special Functions, [120](#)
- [theta\\_c](#)
  - GNU Extended Mathematical Special Functions, [120](#)
- [theta\\_cf](#)
  - GNU Extended Mathematical Special Functions, [120](#)
- [theta\\_cl](#)
  - GNU Extended Mathematical Special Functions, [121](#)
- [theta\\_d](#)
  - GNU Extended Mathematical Special Functions, [121](#)
- [theta\\_df](#)
  - GNU Extended Mathematical Special Functions, [121](#)
- [theta\\_dl](#)
  - GNU Extended Mathematical Special Functions, [121](#)
- [theta\\_n](#)
  - GNU Extended Mathematical Special Functions, [122](#)
- [theta\\_nf](#)
  - GNU Extended Mathematical Special Functions, [122](#)
- [theta\\_nl](#)
  - GNU Extended Mathematical Special Functions, [122](#)
- [theta\\_s](#)
  - GNU Extended Mathematical Special Functions, [122](#)
- [theta\\_sf](#)
  - GNU Extended Mathematical Special Functions, [123](#)
- [theta\\_sl](#)
  - GNU Extended Mathematical Special Functions, [123](#)
- [zernike](#)
  - GNU Extended Mathematical Special Functions, [123](#)
- [zernikef](#)
  - GNU Extended Mathematical Special Functions, [124](#)
- [zernikel](#)
  - GNU Extended Mathematical Special Functions, [124](#)