

# 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 . . . . .	5
1.3.1	Argument Promotion . . . . .	5
1.3.2	NaN Arguments . . . . .	5
1.4	Implementation . . . . .	5
1.5	Testing . . . . .	5
1.6	General Bibliography . . . . .	6
<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>Hierarchical Index</b>	<b>13</b>
5.1	Class Hierarchy . . . . .	13
<b>6</b>	<b>Class Index</b>	<b>15</b>
6.1	Class List . . . . .	15

<b>7 File Index</b>	<b>17</b>
7.1 File List . . . . .	17
<b>8 Module Documentation</b>	<b>19</b>
8.1 C++ Mathematical Special Functions . . . . .	19
8.1.1 Detailed Description . . . . .	19
8.2 C++17/IS29124 Mathematical Special Functions . . . . .	20
8.2.1 Detailed Description . . . . .	22
8.2.2 Function Documentation . . . . .	22
8.2.2.1 assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x) . . . . .	22
8.2.2.2 assoc_laguerref(unsigned int __n, unsigned int __m, float __x) . . . . .	23
8.2.2.3 assoc_laguerrel(unsigned int __n, unsigned int __m, long double __x) . . . . .	23
8.2.2.4 assoc_legendre(unsigned int __l, unsigned int __m, _Tp __x) . . . . .	23
8.2.2.5 assoc_legendref(unsigned int __l, unsigned int __m, float __x) . . . . .	24
8.2.2.6 assoc_legendrel(unsigned int __l, unsigned int __m, long double __x) . . . . .	24
8.2.2.7 beta(_Tpa __a, _Tpb __b) . . . . .	24
8.2.2.8 betaf(float __a, float __b) . . . . .	25
8.2.2.9 betal(long double __a, long double __b) . . . . .	25
8.2.2.10 comp_ellint_1(_Tp __k) . . . . .	25
8.2.2.11 comp_ellint_1f(float __k) . . . . .	26
8.2.2.12 comp_ellint_1l(long double __k) . . . . .	26
8.2.2.13 comp_ellint_2(_Tp __k) . . . . .	26
8.2.2.14 comp_ellint_2f(float __k) . . . . .	27
8.2.2.15 comp_ellint_2l(long double __k) . . . . .	27
8.2.2.16 comp_ellint_3(_Tp __k, _Tpn __nu) . . . . .	27
8.2.2.17 comp_ellint_3f(float __k, float __nu) . . . . .	28
8.2.2.18 comp_ellint_3l(long double __k, long double __nu) . . . . .	28
8.2.2.19 cyl_bessel_i(_Tpn __nu, _Tp __x) . . . . .	28

8.2.2.20	<a href="#">cyl_bessel_if(float __nu, float __x)</a>	29
8.2.2.21	<a href="#">cyl_bessel_il(long double __nu, long double __x)</a>	29
8.2.2.22	<a href="#">cyl_bessel_j(_Tpnu __nu, _Tp __x)</a>	29
8.2.2.23	<a href="#">cyl_bessel_jf(float __nu, float __x)</a>	30
8.2.2.24	<a href="#">cyl_bessel_jl(long double __nu, long double __x)</a>	30
8.2.2.25	<a href="#">cyl_bessel_k(_Tpnu __nu, _Tp __x)</a>	30
8.2.2.26	<a href="#">cyl_bessel_kf(float __nu, float __x)</a>	31
8.2.2.27	<a href="#">cyl_bessel_kl(long double __nu, long double __x)</a>	31
8.2.2.28	<a href="#">cyl_neumann(_Tpnu __nu, _Tp __x)</a>	31
8.2.2.29	<a href="#">cyl_neumannf(float __nu, float __x)</a>	32
8.2.2.30	<a href="#">cyl_neumannl(long double __nu, long double __x)</a>	32
8.2.2.31	<a href="#">ellint_1(_Tp __k, _Tpp __phi)</a>	32
8.2.2.32	<a href="#">ellint_1f(float __k, float __phi)</a>	33
8.2.2.33	<a href="#">ellint_1l(long double __k, long double __phi)</a>	33
8.2.2.34	<a href="#">ellint_2(_Tp __k, _Tpp __phi)</a>	33
8.2.2.35	<a href="#">ellint_2f(float __k, float __phi)</a>	34
8.2.2.36	<a href="#">ellint_2l(long double __k, long double __phi)</a>	34
8.2.2.37	<a href="#">ellint_3(_Tp __k, _Tpn __nu, _Tpp __phi)</a>	35
8.2.2.38	<a href="#">ellint_3f(float __k, float __nu, float __phi)</a>	35
8.2.2.39	<a href="#">ellint_3l(long double __k, long double __nu, long double __phi)</a>	36
8.2.2.40	<a href="#">expint(_Tp __x)</a>	36
8.2.2.41	<a href="#">expintf(float __x)</a>	36
8.2.2.42	<a href="#">expintl(long double __x)</a>	36
8.2.2.43	<a href="#">hermite(unsigned int __n, _Tp __x)</a>	37
8.2.2.44	<a href="#">hermitef(unsigned int __n, float __x)</a>	37
8.2.2.45	<a href="#">hermitel(unsigned int __n, long double __x)</a>	37
8.2.2.46	<a href="#">laguerre(unsigned int __n, _Tp __x)</a>	38
8.2.2.47	<a href="#">laguerref(unsigned int __n, float __x)</a>	38

8.2.2.48	<a href="#">laguerrel(unsigned int __n, long double __x)</a>	38
8.2.2.49	<a href="#">legendre(unsigned int __l, _Tp __x)</a>	39
8.2.2.50	<a href="#">legendref(unsigned int __l, float __x)</a>	39
8.2.2.51	<a href="#">legendrel(unsigned int __l, long double __x)</a>	39
8.2.2.52	<a href="#">riemann_zeta(_Tp __s)</a>	40
8.2.2.53	<a href="#">riemann_zetaf(float __s)</a>	40
8.2.2.54	<a href="#">riemann_zetal(long double __s)</a>	40
8.2.2.55	<a href="#">sph_bessel(unsigned int __n, _Tp __x)</a>	41
8.2.2.56	<a href="#">sph_besself(unsigned int __n, float __x)</a>	41
8.2.2.57	<a href="#">sph_bessell(unsigned int __n, long double __x)</a>	41
8.2.2.58	<a href="#">sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</a>	42
8.2.2.59	<a href="#">sph_legendref(unsigned int __l, unsigned int __m, float __theta)</a>	42
8.2.2.60	<a href="#">sph_legendrel(unsigned int __l, unsigned int __m, long double __theta)</a>	42
8.2.2.61	<a href="#">sph_neumann(unsigned int __n, _Tp __x)</a>	43
8.2.2.62	<a href="#">sph_neumannf(unsigned int __n, float __x)</a>	43
8.2.2.63	<a href="#">sph_neumannl(unsigned int __n, long double __x)</a>	43
8.3	<a href="#">GNU Extended Mathematical Special Functions</a>	44
8.3.1	<a href="#">Detailed Description</a>	53
8.3.2	<a href="#">Enumeration Type Documentation</a>	53
8.3.2.1	<a href="#">anonymous enum</a>	53
8.3.3	<a href="#">Function Documentation</a>	53
8.3.3.1	<a href="#">airy_ai(_Tp __x)</a>	53
8.3.3.2	<a href="#">airy_ai(std::complex&lt; _Tp &gt; __x)</a>	54
8.3.3.3	<a href="#">airy_aif(float __x)</a>	54
8.3.3.4	<a href="#">airy_ail(long double __x)</a>	54
8.3.3.5	<a href="#">airy_bi(_Tp __x)</a>	55
8.3.3.6	<a href="#">airy_bi(std::complex&lt; _Tp &gt; __x)</a>	55
8.3.3.7	<a href="#">airy_bif(float __x)</a>	55

8.3.3.8	<a href="#">airy_bil(long double __x)</a>	56
8.3.3.9	<a href="#">bernoulli(unsigned int __n)</a>	56
8.3.3.10	<a href="#">bernoullif(unsigned int __n)</a>	56
8.3.3.11	<a href="#">bernoullil(unsigned int __n)</a>	56
8.3.3.12	<a href="#">bincoef(unsigned int __n, unsigned int __k)</a>	57
8.3.3.13	<a href="#">bincoeff(unsigned int __n, unsigned int __k)</a>	57
8.3.3.14	<a href="#">bincoeffl(unsigned int __n, unsigned int __k)</a>	57
8.3.3.15	<a href="#">bose_einstein(_Tps __s, _Tp __x)</a>	57
8.3.3.16	<a href="#">bose_einsteinf(float __s, float __x)</a>	57
8.3.3.17	<a href="#">bose_einsteinl(long double __s, long double __x)</a>	57
8.3.3.18	<a href="#">chebyshev_t(unsigned int __n, _Tp __x)</a>	57
8.3.3.19	<a href="#">chebyshev_tf(unsigned int __n, float __x)</a>	58
8.3.3.20	<a href="#">chebyshev_tl(unsigned int __n, long double __x)</a>	58
8.3.3.21	<a href="#">chebyshev_u(unsigned int __n, _Tp __x)</a>	58
8.3.3.22	<a href="#">chebyshev_uf(unsigned int __n, float __x)</a>	59
8.3.3.23	<a href="#">chebyshev_ul(unsigned int __n, long double __x)</a>	59
8.3.3.24	<a href="#">chebyshev_v(unsigned int __n, _Tp __x)</a>	59
8.3.3.25	<a href="#">chebyshev_vf(unsigned int __n, float __x)</a>	60
8.3.3.26	<a href="#">chebyshev_vl(unsigned int __n, long double __x)</a>	60
8.3.3.27	<a href="#">chebyshev_w(unsigned int __n, _Tp __x)</a>	60
8.3.3.28	<a href="#">chebyshev_wf(unsigned int __n, float __x)</a>	61
8.3.3.29	<a href="#">chebyshev_wl(unsigned int __n, long double __x)</a>	61
8.3.3.30	<a href="#">clausen(unsigned int __m, _Tp __w)</a>	61
8.3.3.31	<a href="#">clausen(unsigned int __m, std::complex&lt; _Tp &gt; __w)</a>	62
8.3.3.32	<a href="#">clausen_c(unsigned int __m, _Tp __w)</a>	62
8.3.3.33	<a href="#">clausen_cf(unsigned int __m, float __w)</a>	63
8.3.3.34	<a href="#">clausen_cl(unsigned int __m, long double __w)</a>	63
8.3.3.35	<a href="#">clausen_s(unsigned int __m, _Tp __w)</a>	63

8.3.3.36	<code>clausen_sf(unsigned int __m, float __w)</code>	64
8.3.3.37	<code>clausen_sl(unsigned int __m, long double __w)</code>	64
8.3.3.38	<code>clausenf(unsigned int __m, float __w)</code>	64
8.3.3.39	<code>clausenf(unsigned int __m, std::complex&lt; float &gt; __w)</code>	65
8.3.3.40	<code>clausenl(unsigned int __m, long double __w)</code>	65
8.3.3.41	<code>clausenl(unsigned int __m, std::complex&lt; long double &gt; __w)</code>	65
8.3.3.42	<code>comp_ellint_d(_Tk __k)</code>	65
8.3.3.43	<code>comp_ellint_df(float __k)</code>	66
8.3.3.44	<code>comp_ellint_dl(long double __k)</code>	66
8.3.3.45	<code>comp_ellint_rf(float __x, float __y)</code>	66
8.3.3.46	<code>comp_ellint_rf(long double __x, long double __y)</code>	66
8.3.3.47	<code>comp_ellint_rf(_Tx __x, _Ty __y)</code>	66
8.3.3.48	<code>comp_ellint_rg(float __x, float __y)</code>	67
8.3.3.49	<code>comp_ellint_rg(long double __x, long double __y)</code>	67
8.3.3.50	<code>comp_ellint_rg(_Tx __x, _Ty __y)</code>	67
8.3.3.51	<code>conf_hyperg(_Tpa __a, _Tpc __c, _Tp __x)</code>	68
8.3.3.52	<code>conf_hyperg_lim(_Tpc __c, _Tp __x)</code>	68
8.3.3.53	<code>conf_hyperg_limf(float __c, float __x)</code>	69
8.3.3.54	<code>conf_hyperg_liml(long double __c, long double __x)</code>	69
8.3.3.55	<code>conf_hypergf(float __a, float __c, float __x)</code>	69
8.3.3.56	<code>conf_hypergl(long double __a, long double __c, long double __x)</code>	69
8.3.3.57	<code>cos_pi(_Tp __x)</code>	69
8.3.3.58	<code>cos_pif(float __x)</code>	70
8.3.3.59	<code>cos_pil(long double __x)</code>	70
8.3.3.60	<code>cosh_pi(_Tp __x)</code>	70
8.3.3.61	<code>cosh_pif(float __x)</code>	71
8.3.3.62	<code>cosh_pil(long double __x)</code>	71
8.3.3.63	<code>coshint(_Tp __x)</code>	71



8.3.3.64	<code>coshintf(float __x)</code>	72
8.3.3.65	<code>coshintl(long double __x)</code>	72
8.3.3.66	<code>cosint(_Tp __x)</code>	72
8.3.3.67	<code>cosintf(float __x)</code>	73
8.3.3.68	<code>cosintl(long double __x)</code>	73
8.3.3.69	<code>cyl_hankel_1(_Tpnu __nu, _Tp __z)</code>	73
8.3.3.70	<code>cyl_hankel_1(std::complex&lt; _Tpnu &gt; __nu, std::complex&lt; _Tp &gt; __x)</code>	74
8.3.3.71	<code>cyl_hankel_1f(float __nu, float __z)</code>	74
8.3.3.72	<code>cyl_hankel_1f(std::complex&lt; float &gt; __nu, std::complex&lt; float &gt; __x)</code>	74
8.3.3.73	<code>cyl_hankel_1l(long double __nu, long double __z)</code>	75
8.3.3.74	<code>cyl_hankel_1l(std::complex&lt; long double &gt; __nu, std::complex&lt; long double &gt; __x)</code>	75
8.3.3.75	<code>cyl_hankel_2(_Tpnu __nu, _Tp __z)</code>	75
8.3.3.76	<code>cyl_hankel_2(std::complex&lt; _Tpnu &gt; __nu, std::complex&lt; _Tp &gt; __x)</code>	76
8.3.3.77	<code>cyl_hankel_2f(float __nu, float __z)</code>	76
8.3.3.78	<code>cyl_hankel_2f(std::complex&lt; float &gt; __nu, std::complex&lt; float &gt; __x)</code>	76
8.3.3.79	<code>cyl_hankel_2l(long double __nu, long double __z)</code>	77
8.3.3.80	<code>cyl_hankel_2l(std::complex&lt; long double &gt; __nu, std::complex&lt; long double &gt; __x)</code>	77
8.3.3.81	<code>dawson(_Tp __x)</code>	77
8.3.3.82	<code>dawsonf(float __x)</code>	77
8.3.3.83	<code>dawsonl(long double __x)</code>	78
8.3.3.84	<code>dilog(_Tp __x)</code>	78
8.3.3.85	<code>dilogf(float __x)</code>	78
8.3.3.86	<code>dilogl(long double __x)</code>	79
8.3.3.87	<code>dirichlet_beta(_Tp __s)</code>	79
8.3.3.88	<code>dirichlet_betaf(float __s)</code>	79
8.3.3.89	<code>dirichlet_betall(long double __s)</code>	79
8.3.3.90	<code>dirichlet_eta(_Tp __s)</code>	80
8.3.3.91	<code>dirichlet_etaf(float __s)</code>	80

8.3.3.92	<code>dirichlet_etal(long double __s)</code>	80
8.3.3.93	<code>dirichlet_lambda(_Tp __s)</code>	80
8.3.3.94	<code>dirichlet_lambdaf(float __s)</code>	81
8.3.3.95	<code>dirichlet_lambdal(long double __s)</code>	81
8.3.3.96	<code>double_factorial(int __n)</code>	81
8.3.3.97	<code>double_factorialf(int __n)</code>	81
8.3.3.98	<code>double_factoriall(int __n)</code>	81
8.3.3.99	<code>ellint_cel(_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)</code>	81
8.3.3.100	<code>ellint_celf(float __k_c, float __p, float __a, float __b)</code>	82
8.3.3.101	<code>ellint_cell(long double __k_c, long double __p, long double __a, long double __b)</code>	82
8.3.3.102	<code>ellint_d(_Tk __k, _Tphi __phi)</code>	82
8.3.3.103	<code>ellint_df(float __k, float __phi)</code>	83
8.3.3.104	<code>ellint_dl(long double __k, long double __phi)</code>	83
8.3.3.105	<code>ellint_el1(_Tp __x, _Tk __k_c)</code>	83
8.3.3.106	<code>ellint_el1f(float __x, float __k_c)</code>	84
8.3.3.107	<code>ellint_el1l(long double __x, long double __k_c)</code>	84
8.3.3.108	<code>ellint_el2(_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)</code>	84
8.3.3.109	<code>ellint_el2f(float __x, float __k_c, float __a, float __b)</code>	85
8.3.3.110	<code>ellint_el2l(long double __x, long double __k_c, long double __a, long double __b)</code>	85
8.3.3.111	<code>ellint_el3(_Tx __x, _Tk __k_c, _Tp __p)</code>	85
8.3.3.112	<code>ellint_el3f(float __x, float __k_c, float __p)</code>	86
8.3.3.113	<code>ellint_el3l(long double __x, long double __k_c, long double __p)</code>	86
8.3.3.114	<code>ellint_rc(_Tp __x, _Up __y)</code>	86
8.3.3.115	<code>ellint_rcf(float __x, float __y)</code>	87
8.3.3.116	<code>ellint_rcl(long double __x, long double __y)</code>	87
8.3.3.117	<code>ellint_rd(_Tp __x, _Up __y, _Vp __z)</code>	87
8.3.3.118	<code>ellint_rdf(float __x, float __y, float __z)</code>	88
8.3.3.119	<code>ellint_rdl(long double __x, long double __y, long double __z)</code>	88

8.3.3.120 ellint_rf(_Tp __x, _Up __y, _Vp __z) . . . . .	88
8.3.3.121 ellint_rff(float __x, float __y, float __z) . . . . .	89
8.3.3.122 ellint_rfl(long double __x, long double __y, long double __z) . . . . .	89
8.3.3.123 ellint_rg(_Tp __x, _Up __y, _Vp __z) . . . . .	89
8.3.3.124 ellint_rgf(float __x, float __y, float __z) . . . . .	90
8.3.3.125 ellint_rgl(long double __x, long double __y, long double __z) . . . . .	90
8.3.3.126 ellint_rj(_Tp __x, _Up __y, _Vp __z, _Wp __p) . . . . .	90
8.3.3.127 ellint_rjf(float __x, float __y, float __z, float __p) . . . . .	91
8.3.3.128 ellint_rjl(long double __x, long double __y, long double __z, long double __p) . . . . .	91
8.3.3.129 ellnome(_Tp __k) . . . . .	91
8.3.3.130 ellnomef(float __k) . . . . .	92
8.3.3.131 ellnomel(long double __k) . . . . .	92
8.3.3.132 expint(unsigned int __n, _Tp __x) . . . . .	92
8.3.3.133 expintf(unsigned int __n, float __x) . . . . .	93
8.3.3.134 expintl(unsigned int __n, long double __x) . . . . .	93
8.3.3.135 factorial(unsigned int __n) . . . . .	93
8.3.3.136 factorialf(unsigned int __n) . . . . .	93
8.3.3.137 factoriall(unsigned int __n) . . . . .	93
8.3.3.138 fermi_dirac(_Tps __s, _Tp __x) . . . . .	93
8.3.3.139 fermi_diracf(float __s, float __x) . . . . .	93
8.3.3.140 fermi_diracl(long double __s, long double __x) . . . . .	93
8.3.3.141 fresnel_c(_Tp __x) . . . . .	93
8.3.3.142 fresnel_cf(float __x) . . . . .	94
8.3.3.143 fresnel_cl(long double __x) . . . . .	94
8.3.3.144 fresnel_s(_Tp __x) . . . . .	94
8.3.3.145 fresnel_sf(float __x) . . . . .	94
8.3.3.146 fresnel_sl(long double __x) . . . . .	94
8.3.3.147 gegenbauer(unsigned int __n, _Talpha __alpha, _Tp __x) . . . . .	95

8.3.3.148 gegenbauerf(unsigned int __n, float __alpha, float __x) . . . . .	95
8.3.3.149 gegenbauerl(unsigned int __n, long double __alpha, long double __x) . . . . .	95
8.3.3.150 heuman_lambda(_Tk __k, _Tphi __phi) . . . . .	96
8.3.3.151 heuman_lambdaf(float __k, float __phi) . . . . .	96
8.3.3.152 heuman_lambdal(long double __k, long double __phi) . . . . .	96
8.3.3.153 hurwitz_zeta(_Tp __s, _Up __a) . . . . .	96
8.3.3.154 hurwitz_zeta(_Tp __s, std::complex< _Up > __a) . . . . .	97
8.3.3.155 hurwitz_zetaf(float __s, float __a) . . . . .	97
8.3.3.156 hurwitz_zetal(long double __s, long double __a) . . . . .	97
8.3.3.157 hyperg(_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x) . . . . .	97
8.3.3.158 hypergf(float __a, float __b, float __c, float __x) . . . . .	98
8.3.3.159 hypergl(long double __a, long double __b, long double __c, long double __x) . . . . .	98
8.3.3.160 ibeta(_Ta __a, _Tb __b, _Tp __x) . . . . .	98
8.3.3.161 ibetac(_Ta __a, _Tb __b, _Tp __x) . . . . .	99
8.3.3.162 ibetacf(float __a, float __b, float __x) . . . . .	99
8.3.3.163 ibetacl(long double __a, long double __b, long double __x) . . . . .	99
8.3.3.164 ibetaf(float __a, float __b, float __x) . . . . .	100
8.3.3.165 ibetal(long double __a, long double __b, long double __x) . . . . .	100
8.3.3.166 jacobi(unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x) . . . . .	100
8.3.3.167 jacobi_cn(_Kp __k, _Up __u) . . . . .	101
8.3.3.168 jacobi_cnf(float __k, float __u) . . . . .	101
8.3.3.169 jacobi_cnl(long double __k, long double __u) . . . . .	101
8.3.3.170 jacobi_dn(_Kp __k, _Up __u) . . . . .	102
8.3.3.171 jacobi_dnf(float __k, float __u) . . . . .	102
8.3.3.172 jacobi_dnl(long double __k, long double __u) . . . . .	102
8.3.3.173 jacobi_sn(_Kp __k, _Up __u) . . . . .	103
8.3.3.174 jacobi_snf(float __k, float __u) . . . . .	103
8.3.3.175 jacobi_snl(long double __k, long double __u) . . . . .	103

8.3.3.176 jacobi_zeta(_Tk __k, _Tphi __phi) . . . . .	104
8.3.3.177 jacobi_zetaf(float __k, float __phi) . . . . .	104
8.3.3.178 jacobi_zetal(long double __k, long double __phi) . . . . .	104
8.3.3.179 jacobif(unsigned __n, float __alpha, float __beta, float __x) . . . . .	104
8.3.3.180 jacobil(unsigned __n, long double __alpha, long double __beta, long double __x) . . . . .	105
8.3.3.181 lbincoef(unsigned int __n, unsigned int __k) . . . . .	105
8.3.3.182 lbincoeff(unsigned int __n, unsigned int __k) . . . . .	105
8.3.3.183 lbincoefl(unsigned int __n, unsigned int __k) . . . . .	105
8.3.3.184 ldouble_factorial(int __n) . . . . .	105
8.3.3.185 ldouble_factorialf(int __n) . . . . .	105
8.3.3.186 ldouble_factoriall(int __n) . . . . .	105
8.3.3.187 legendre_q(unsigned int __n, _Tp __x) . . . . .	105
8.3.3.188 legendre_qf(unsigned int __n, float __x) . . . . .	106
8.3.3.189 legendre_ql(unsigned int __n, long double __x) . . . . .	106
8.3.3.190 lfactorial(unsigned int __n) . . . . .	106
8.3.3.191 lfactorialf(unsigned int __n) . . . . .	107
8.3.3.192 lfactoriall(unsigned int __n) . . . . .	107
8.3.3.193 lgamma(std::complex< _Ta > __a) . . . . .	107
8.3.3.194 lgammaf(std::complex< float > __a) . . . . .	107
8.3.3.195 lgammal(std::complex< long double > __a) . . . . .	107
8.3.3.196 logint(_Tp __x) . . . . .	107
8.3.3.197 logintf(float __x) . . . . .	108
8.3.3.198 logintl(long double __x) . . . . .	108
8.3.3.199 lpochhammer(_Tp __a, _Tn __n) . . . . .	108
8.3.3.200 lpochhammer_lower(_Tp __a, _Tn __n) . . . . .	108
8.3.3.201 lpochhammer_lowerf(float __a, float __n) . . . . .	108
8.3.3.202 lpochhammer_lowerl(long double __a, long double __n) . . . . .	109
8.3.3.203 lpochhammererf(float __a, float __n) . . . . .	109

8.3.3.204 lpochhammer(long double __a, long double __n) . . . . .	109
8.3.3.205 owens_t(_Tph __h, _Tpa __a) . . . . .	109
8.3.3.206 owens_tf(float __h, float __a) . . . . .	109
8.3.3.207 owens_tl(long double __h, long double __a) . . . . .	110
8.3.3.208 pgamma(_Ta __a, _Tp __x) . . . . .	110
8.3.3.209 pgammaf(float __a, float __x) . . . . .	110
8.3.3.210 pgammal(long double __a, long double __x) . . . . .	110
8.3.3.211 pochhammer(_Tp __a, _Tn __n) . . . . .	110
8.3.3.212 pochhammer_lower(_Tp __a, _Tn __n) . . . . .	110
8.3.3.213 pochhammer_lowerf(float __a, float __n) . . . . .	110
8.3.3.214 pochhammer_lowerl(long double __a, long double __n) . . . . .	110
8.3.3.215 pochhammerf(float __a, float __n) . . . . .	111
8.3.3.216 pochhammerl(long double __a, long double __n) . . . . .	111
8.3.3.217 polylog(_Tp __s, _Wp __w) . . . . .	111
8.3.3.218 polylog(_Tp __s, std::complex< _Tp > __w) . . . . .	111
8.3.3.219 polylogf(float __s, float __w) . . . . .	112
8.3.3.220 polylogf(float __s, std::complex< float > __w) . . . . .	112
8.3.3.221 polylogl(long double __s, long double __w) . . . . .	112
8.3.3.222 polylogl(long double __s, std::complex< long double > __w) . . . . .	112
8.3.3.223 psi(_Tp __x) . . . . .	112
8.3.3.224 psif(float __x) . . . . .	113
8.3.3.225 psil(long double __x) . . . . .	113
8.3.3.226 qgamma(_Ta __a, _Tp __x) . . . . .	113
8.3.3.227 qgammaf(float __a, float __x) . . . . .	113
8.3.3.228 qgammal(long double __a, long double __x) . . . . .	113
8.3.3.229 radpoly(unsigned int __n, unsigned int __m, _Tp __rho) . . . . .	114
8.3.3.230 radpolyf(unsigned int __n, unsigned int __m, float __rho) . . . . .	114
8.3.3.231 radpolyl(unsigned int __n, unsigned int __m, long double __rho) . . . . .	115

8.3.3.232 <code>sin_pi(_Tp __x)</code> . . . . .	115
8.3.3.233 <code>sin_pif(float __x)</code> . . . . .	115
8.3.3.234 <code>sin_pil(long double __x)</code> . . . . .	116
8.3.3.235 <code>sinc(_Tp __x)</code> . . . . .	116
8.3.3.236 <code>sinc_pi(_Tp __x)</code> . . . . .	116
8.3.3.237 <code>sinc_pif(float __x)</code> . . . . .	117
8.3.3.238 <code>sinc_pil(long double __x)</code> . . . . .	117
8.3.3.239 <code>sincf(float __x)</code> . . . . .	117
8.3.3.240 <code>sincl(long double __x)</code> . . . . .	117
8.3.3.241 <code>sincos(double __x)</code> . . . . .	118
8.3.3.242 <code>sincos(_Tp __x)</code> . . . . .	118
8.3.3.243 <code>sincos_pi(_Tp __x)</code> . . . . .	118
8.3.3.244 <code>sincos_pif(float __x)</code> . . . . .	118
8.3.3.245 <code>sincos_pil(long double __x)</code> . . . . .	118
8.3.3.246 <code>sincosf(float __x)</code> . . . . .	119
8.3.3.247 <code>sincosl(long double __x)</code> . . . . .	119
8.3.3.248 <code>sinh_pi(_Tp __x)</code> . . . . .	119
8.3.3.249 <code>sinh_pif(float __x)</code> . . . . .	119
8.3.3.250 <code>sinh_pil(long double __x)</code> . . . . .	120
8.3.3.251 <code>sinhc(_Tp __x)</code> . . . . .	120
8.3.3.252 <code>sinhc_pi(_Tp __x)</code> . . . . .	120
8.3.3.253 <code>sinhc_pif(float __x)</code> . . . . .	121
8.3.3.254 <code>sinhc_pil(long double __x)</code> . . . . .	121
8.3.3.255 <code>sinhcf(float __x)</code> . . . . .	121
8.3.3.256 <code>sinhcl(long double __x)</code> . . . . .	121
8.3.3.257 <code>sinhint(_Tp __x)</code> . . . . .	121
8.3.3.258 <code>sinhintf(float __x)</code> . . . . .	122
8.3.3.259 <code>sinhintl(long double __x)</code> . . . . .	122

8.3.3.260 <code>sinint(_Tp __x)</code> . . . . .	122
8.3.3.261 <code>sinintf(float __x)</code> . . . . .	123
8.3.3.262 <code>sinintl(long double __x)</code> . . . . .	123
8.3.3.263 <code>sph_bessel_i(unsigned int __n, _Tp __x)</code> . . . . .	123
8.3.3.264 <code>sph_bessel_if(unsigned int __n, float __x)</code> . . . . .	124
8.3.3.265 <code>sph_bessel_il(unsigned int __n, long double __x)</code> . . . . .	124
8.3.3.266 <code>sph_bessel_k(unsigned int __n, _Tp __x)</code> . . . . .	124
8.3.3.267 <code>sph_bessel_kf(unsigned int __n, float __x)</code> . . . . .	125
8.3.3.268 <code>sph_bessel_kl(unsigned int __n, long double __x)</code> . . . . .	125
8.3.3.269 <code>sph_hankel_1(unsigned int __n, _Tp __z)</code> . . . . .	125
8.3.3.270 <code>sph_hankel_1(unsigned int __n, std::complex&lt; _Tp &gt; __x)</code> . . . . .	126
8.3.3.271 <code>sph_hankel_1f(unsigned int __n, float __z)</code> . . . . .	126
8.3.3.272 <code>sph_hankel_1f(unsigned int __n, std::complex&lt; float &gt; __x)</code> . . . . .	127
8.3.3.273 <code>sph_hankel_1l(unsigned int __n, long double __z)</code> . . . . .	127
8.3.3.274 <code>sph_hankel_1l(unsigned int __n, std::complex&lt; long double &gt; __x)</code> . . . . .	127
8.3.3.275 <code>sph_hankel_2(unsigned int __n, _Tp __z)</code> . . . . .	127
8.3.3.276 <code>sph_hankel_2(unsigned int __n, std::complex&lt; _Tp &gt; __x)</code> . . . . .	128
8.3.3.277 <code>sph_hankel_2f(unsigned int __n, float __z)</code> . . . . .	128
8.3.3.278 <code>sph_hankel_2f(unsigned int __n, std::complex&lt; float &gt; __x)</code> . . . . .	129
8.3.3.279 <code>sph_hankel_2l(unsigned int __n, long double __z)</code> . . . . .	129
8.3.3.280 <code>sph_hankel_2l(unsigned int __n, std::complex&lt; long double &gt; __x)</code> . . . . .	129
8.3.3.281 <code>sph_harmonic(unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)</code> . . . . .	129
8.3.3.282 <code>sph_harmonicf(unsigned int __l, int __m, float __theta, float __phi)</code> . . . . .	130
8.3.3.283 <code>sph_harmonicl(unsigned int __l, int __m, long double __theta, long double __phi)</code> . . . . .	130
8.3.3.284 <code>tan_pi(_Tp __x)</code> . . . . .	130
8.3.3.285 <code>tan_pif(float __x)</code> . . . . .	131
8.3.3.286 <code>tan_pil(long double __x)</code> . . . . .	131
8.3.3.287 <code>tanh_pi(_Tp __x)</code> . . . . .	131



8.3.3.288	<code>tanh_pif(float __x)</code>	132
8.3.3.289	<code>tanh_pil(long double __x)</code>	132
8.3.3.290	<code>tgamma(std::complex&lt; _Ta &gt; __a)</code>	132
8.3.3.291	<code>tgamma(_Ta __a, _Tp __x)</code>	132
8.3.3.292	<code>tgamma_lower(_Ta __a, _Tp __x)</code>	133
8.3.3.293	<code>tgamma_lowerf(float __a, float __x)</code>	133
8.3.3.294	<code>tgamma_lowerl(long double __a, long double __x)</code>	133
8.3.3.295	<code>tgammaf(std::complex&lt; float &gt; __a)</code>	133
8.3.3.296	<code>tgammaf(float __a, float __x)</code>	134
8.3.3.297	<code>tgamma(std::complex&lt; long double &gt; __a)</code>	134
8.3.3.298	<code>tgamma(long double __a, long double __x)</code>	134
8.3.3.299	<code>theta_1(_Tpnu __nu, _Tp __x)</code>	134
8.3.3.300	<code>theta_1f(float __nu, float __x)</code>	135
8.3.3.301	<code>theta_1l(long double __nu, long double __x)</code>	135
8.3.3.302	<code>theta_2(_Tpnu __nu, _Tp __x)</code>	135
8.3.3.303	<code>theta_2f(float __nu, float __x)</code>	135
8.3.3.304	<code>theta_2l(long double __nu, long double __x)</code>	136
8.3.3.305	<code>theta_3(_Tpnu __nu, _Tp __x)</code>	136
8.3.3.306	<code>theta_3f(float __nu, float __x)</code>	136
8.3.3.307	<code>theta_3l(long double __nu, long double __x)</code>	137
8.3.3.308	<code>theta_4(_Tpnu __nu, _Tp __x)</code>	137
8.3.3.309	<code>theta_4f(float __nu, float __x)</code>	137
8.3.3.310	<code>theta_4l(long double __nu, long double __x)</code>	137
8.3.3.311	<code>theta_c(_Tp k, _Tp __x)</code>	138
8.3.3.312	<code>theta_cf(float __k, float __x)</code>	138
8.3.3.313	<code>theta_cl(long double __k, long double __x)</code>	138
8.3.3.314	<code>theta_d(_Tp k, _Tp __x)</code>	138
8.3.3.315	<code>theta_df(float __k, float __x)</code>	139
8.3.3.316	<code>theta_dl(long double __k, long double __x)</code>	139
8.3.3.317	<code>theta_n(_Tp k, _Tp __x)</code>	139
8.3.3.318	<code>theta_nf(float __k, float __x)</code>	140
8.3.3.319	<code>theta_nl(long double __k, long double __x)</code>	140
8.3.3.320	<code>theta_s(_Tp k, _Tp __x)</code>	140
8.3.3.321	<code>theta_sf(float __k, float __x)</code>	140
8.3.3.322	<code>theta_sl(long double __k, long double __x)</code>	141
8.3.3.323	<code>zernike(unsigned int __n, int __m, _Trho __rho, _Tphi __phi)</code>	141
8.3.3.324	<code>zernikef(unsigned int __n, int __m, float __rho, float __phi)</code>	142
8.3.3.325	<code>zernikel(unsigned int __n, int __m, long double __rho, long double __phi)</code>	142

<b>9 Namespace Documentation</b>	<b>143</b>
9.1 <code>__gnu_cxx</code> Namespace Reference	143
9.2 <code>std</code> Namespace Reference	152
9.3 <code>std::__detail</code> Namespace Reference	154
9.3.1 Enumeration Type Documentation	174
9.3.1.1 anonymous enum	174
9.3.2 Function Documentation	174
9.3.2.1 <code>__airy(_Tp __z, _Tp &amp;_Ai, _Tp &amp;_Bi, _Tp &amp;_Aip, _Tp &amp;_Bip)</code>	174
9.3.2.2 <code>__airy_ai(std::complex&lt; _Tp &gt; __z)</code>	175
9.3.2.3 <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>	175
9.3.2.4 <code>__airy_bi(std::complex&lt; _Tp &gt; __z)</code>	176
9.3.2.5 <code>__assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)</code>	176
9.3.2.6 <code>__assoc_legendre_p(unsigned int __l, unsigned int __m, _Tp __x)</code>	176
9.3.2.7 <code>__bernoulli(int __n)</code>	178
9.3.2.8 <code>__bernoulli_2n(int __n)</code>	178
9.3.2.9 <code>__bernoulli_series(unsigned int __n)</code>	179
9.3.2.10 <code>__beta(_Tp __a, _Tp __b)</code>	179
9.3.2.11 <code>__beta_gamma(_Tp __a, _Tp __b)</code>	180
9.3.2.12 <code>__beta_inc(_Tp __a, _Tp __b, _Tp __x)</code>	180
9.3.2.13 <code>__beta_lgamma(_Tp __a, _Tp __b)</code>	181
9.3.2.14 <code>__beta_product(_Tp __a, _Tp __b)</code>	181
9.3.2.15 <code>__bincoef(unsigned int __n, unsigned int __k)</code>	182
9.3.2.16 <code>__bincoef(_Tp __nu, unsigned int __k)</code>	182
9.3.2.17 <code>__binomial_cdf(_Tp __p, unsigned int __n, unsigned int __k)</code>	183
9.3.2.18 <code>__binomial_cdfc(_Tp __p, unsigned int __n, unsigned int __k)</code>	183
9.3.2.19 <code>__binomial_pdf(_Tp __p, unsigned int __n, unsigned int __k)</code>	184
9.3.2.20 <code>__bose_einstein(_Sp __s, _Tp __x)</code>	184

9.3.2.21	<code>__chebyshev_recur(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)</code>	185
9.3.2.22	<code>__chebyshev_t(unsigned int __n, _Tp __x)</code>	185
9.3.2.23	<code>__chebyshev_u(unsigned int __n, _Tp __x)</code>	186
9.3.2.24	<code>__chebyshev_v(unsigned int __n, _Tp __x)</code>	186
9.3.2.25	<code>__chebyshev_w(unsigned int __n, _Tp __x)</code>	187
9.3.2.26	<code>__chi_squared_pdf(_Tp __chi2, unsigned int __nu)</code>	187
9.3.2.27	<code>__chi_squared_pdfc(_Tp __chi2, unsigned int __nu)</code>	188
9.3.2.28	<code>__chshint(_Tp __x, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	188
9.3.2.29	<code>__chshint_cont_frac(_Tp __t, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	188
9.3.2.30	<code>__chshint_series(_Tp __t, _Tp &amp;_Chi, _Tp &amp;_Shi)</code>	189
9.3.2.31	<code>__clamp_0_m2pi(std::complex&lt; _Tp &gt; __w)</code>	189
9.3.2.32	<code>__clamp_pi(std::complex&lt; _Tp &gt; __w)</code>	189
9.3.2.33	<code>__clausen(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	189
9.3.2.34	<code>__clausen(unsigned int __m, _Tp __w)</code>	189
9.3.2.35	<code>__clausen_c(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	190
9.3.2.36	<code>__clausen_c(unsigned int __m, _Tp __w)</code>	190
9.3.2.37	<code>__clausen_s(unsigned int __m, std::complex&lt; _Tp &gt; __w)</code>	191
9.3.2.38	<code>__clausen_s(unsigned int __m, _Tp __w)</code>	191
9.3.2.39	<code>__comp_ellint_1(_Tp __k)</code>	192
9.3.2.40	<code>__comp_ellint_2(_Tp __k)</code>	192
9.3.2.41	<code>__comp_ellint_3(_Tp __k, _Tp __nu)</code>	193
9.3.2.42	<code>__comp_ellint_d(_Tp __k)</code>	193
9.3.2.43	<code>__comp_ellint_rf(_Tp __x, _Tp __y)</code>	194
9.3.2.44	<code>__comp_ellint_rg(_Tp __x, _Tp __y)</code>	194
9.3.2.45	<code>__conf_hyperg(_Tp __a, _Tp __c, _Tp __x)</code>	194
9.3.2.46	<code>__conf_hyperg_lim(_Tp __c, _Tp __x)</code>	194
9.3.2.47	<code>__conf_hyperg_lim_series(_Tp __c, _Tp __x)</code>	195
9.3.2.48	<code>__conf_hyperg_luke(_Tp __a, _Tp __c, _Tp __xin)</code>	195

9.3.2.49	<code>__conf_hyperg_series(_Tp __a, _Tp __c, _Tp __x)</code>	195
9.3.2.50	<code>__cos_pi(_Tp __x)</code>	196
9.3.2.51	<code>__cos_pi(std::complex&lt; _Tp &gt; __z)</code>	196
9.3.2.52	<code>__cosh_pi(_Tp __x)</code>	196
9.3.2.53	<code>__cosh_pi(std::complex&lt; _Tp &gt; __z)</code>	197
9.3.2.54	<code>__coshint(const _Tp __x)</code>	197
9.3.2.55	<code>__cyl_bessel(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	197
9.3.2.56	<code>__cyl_bessel_i(_Tp __nu, _Tp __x)</code>	198
9.3.2.57	<code>__cyl_bessel_ij_series(_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)</code>	198
9.3.2.58	<code>__cyl_bessel_ik(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	199
9.3.2.59	<code>__cyl_bessel_ik_asymp(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	199
9.3.2.60	<code>__cyl_bessel_ik_steel(_Tp __nu, _Tp __x, _Tp &amp;_lnu, _Tp &amp;_knu, _Tp &amp;_lpnu, _Tp &amp;_kpnu)</code>	200
9.3.2.61	<code>__cyl_bessel_j(_Tp __nu, _Tp __x)</code>	200
9.3.2.62	<code>__cyl_bessel_jn(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	201
9.3.2.63	<code>__cyl_bessel_jn_asymp(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	201
9.3.2.64	<code>__cyl_bessel_jn_neg_arg(_Tp __nu, _Tp __x, std::complex&lt; _Tp &gt; &amp;_jnu, std::complex&lt; _Tp &gt; &amp;_nnu, std::complex&lt; _Tp &gt; &amp;_jpnu, std::complex&lt; _Tp &gt; &amp;_npnu)</code>	201
9.3.2.65	<code>__cyl_bessel_jn_steel(_Tp __nu, _Tp __x, _Tp &amp;_jnu, _Tp &amp;_nnu, _Tp &amp;_jpnu, _Tp &amp;_npnu)</code>	201
9.3.2.66	<code>__cyl_bessel_k(_Tp __nu, _Tp __x)</code>	202
9.3.2.67	<code>__cyl_hankel_1(_Tp __nu, _Tp __x)</code>	202
9.3.2.68	<code>__cyl_hankel_1(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	203
9.3.2.69	<code>__cyl_hankel_2(_Tp __nu, _Tp __x)</code>	203
9.3.2.70	<code>__cyl_hankel_2(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	204
9.3.2.71	<code>__cyl_neumann(std::complex&lt; _Tp &gt; __nu, std::complex&lt; _Tp &gt; __z)</code>	204
9.3.2.72	<code>__cyl_neumann_n(_Tp __nu, _Tp __x)</code>	205

9.3.2.73	<code>__dawson(_Tp __x)</code>	205
9.3.2.74	<code>__dawson_cont_frac(_Tp __x)</code>	205
9.3.2.75	<code>__dawson_series(_Tp __x)</code>	206
9.3.2.76	<code>__debye_region(std::complex&lt; _Tp &gt; __alpha, int &amp;__indexr, char &amp;__aorb)</code>	206
9.3.2.77	<code>__dilog(_Tp __x)</code>	206
9.3.2.78	<code>__dirichlet_beta(std::complex&lt; _Tp &gt; __w)</code>	206
9.3.2.79	<code>__dirichlet_beta(_Tp __w)</code>	207
9.3.2.80	<code>__dirichlet_eta(std::complex&lt; _Tp &gt; __w)</code>	207
9.3.2.81	<code>__dirichlet_eta(_Tp __w)</code>	208
9.3.2.82	<code>__dirichlet_lambda(_Tp __w)</code>	208
9.3.2.83	<code>__double_factorial(int __n)</code>	209
9.3.2.84	<code>__ellint_1(_Tp __k, _Tp __phi)</code>	209
9.3.2.85	<code>__ellint_2(_Tp __k, _Tp __phi)</code>	209
9.3.2.86	<code>__ellint_3(_Tp __k, _Tp __nu, _Tp __phi)</code>	210
9.3.2.87	<code>__ellint_cel(_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)</code>	210
9.3.2.88	<code>__ellint_d(_Tp __k, _Tp __phi)</code>	211
9.3.2.89	<code>__ellint_el1(_Tp __x, _Tp __k_c)</code>	211
9.3.2.90	<code>__ellint_el2(_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)</code>	211
9.3.2.91	<code>__ellint_el3(_Tp __x, _Tp __k_c, _Tp __p)</code>	211
9.3.2.92	<code>__ellint_rc(_Tp __x, _Tp __y)</code>	211
9.3.2.93	<code>__ellint_rd(_Tp __x, _Tp __y, _Tp __z)</code>	212
9.3.2.94	<code>__ellint_rf(_Tp __x, _Tp __y, _Tp __z)</code>	213
9.3.2.95	<code>__ellint_rg(_Tp __x, _Tp __y, _Tp __z)</code>	213
9.3.2.96	<code>__ellint_rj(_Tp __x, _Tp __y, _Tp __z, _Tp __p)</code>	214
9.3.2.97	<code>__ellnome(_Tp __k)</code>	215
9.3.2.98	<code>__ellnome_k(_Tp __k)</code>	215
9.3.2.99	<code>__ellnome_series(_Tp __k)</code>	215
9.3.2.100	<code>__expint(unsigned int __n, _Tp __x)</code>	215

9.3.2.101	<code>__expint(_Tp __x)</code>	216
9.3.2.102	<code>__expint_asyp(unsigned int __n, _Tp __x)</code>	216
9.3.2.103	<code>__expint_E1(_Tp __x)</code>	217
9.3.2.104	<code>__expint_E1_asyp(_Tp __x)</code>	217
9.3.2.105	<code>__expint_E1_series(_Tp __x)</code>	218
9.3.2.106	<code>__expint_Ei(_Tp __x)</code>	218
9.3.2.107	<code>__expint_Ei_asyp(_Tp __x)</code>	219
9.3.2.108	<code>__expint_Ei_series(_Tp __x)</code>	219
9.3.2.109	<code>__expint_En_cont_frac(unsigned int __n, _Tp __x)</code>	220
9.3.2.110	<code>__expint_En_recursion(unsigned int __n, _Tp __x)</code>	220
9.3.2.111	<code>__expint_En_series(unsigned int __n, _Tp __x)</code>	221
9.3.2.112	<code>__expint_large_n(unsigned int __n, _Tp __x)</code>	221
9.3.2.113	<code>__exponential_cdf(_Tp __lambda, _Tp __x)</code>	222
9.3.2.114	<code>__exponential_pdf(_Tp __lambda, _Tp __x)</code>	222
9.3.2.115	<code>__factorial(unsigned int __n)</code>	222
9.3.2.116	<code>__fermi_dirac(_Sp __s, _Tp __x)</code>	223
9.3.2.117	<code>__fisher_f_cdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	223
9.3.2.118	<code>__fisher_f_cdfc(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	224
9.3.2.119	<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>	224
9.3.2.120	<code>__fresnel(const _Tp __x)</code>	224
9.3.2.121	<code>__fresnel_cont_frac(const _Tp __ax, _Tp &amp;__Cf, _Tp &amp;__Sf)</code>	225
9.3.2.122	<code>__fresnel_series(const _Tp __ax, _Tp &amp;__Cf, _Tp &amp;__Sf)</code>	225
9.3.2.123	<code>__gamma(_Tp __x)</code>	225
9.3.2.124	<code>__gamma_cdf(_Tp __alpha, _Tp __beta, _Tp __x)</code>	226
9.3.2.125	<code>__gamma_cdfc(_Tp __alpha, _Tp __beta, _Tp __x)</code>	226
9.3.2.126	<code>__gamma_cont_frac(_Tp __a, _Tp __x)</code>	226
9.3.2.127	<code>__gamma_pdf(_Tp __alpha, _Tp __beta, _Tp __x)</code>	227

9.3.2.128	<code>__gamma_series(_Tp __a, _Tp __x)</code>	227
9.3.2.129	<code>__gamma_temme(_Tp __mu, _Tp &amp;__gam1, _Tp &amp;__gam2, _Tp &amp;__gampl, _Tp &amp;__gammi)</code>	227
9.3.2.130	<code>__gauss(_Tp __x)</code>	228
9.3.2.131	<code>__gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)</code>	228
9.3.2.132	<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>	228
9.3.2.133	<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 __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>	229
9.3.2.134	<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;__zetrat)</code>	229
9.3.2.135	<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>	230
9.3.2.136	<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>	230
9.3.2.137	<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>	231
9.3.2.138	<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>	231
9.3.2.139	<code>__harmonic_number(unsigned int __n)</code>	232
9.3.2.140	<code>__heuman_lambda(_Tp __k, _Tp __phi)</code>	232
9.3.2.141	<code>__hurwitz_zeta(_Tp __s, _Tp __a)</code>	232

9.3.2.142	<code>__hurwitz_zeta_euler_maclaurin(_Tp __s, _Tp __a)</code>	232
9.3.2.143	<code>__hurwitz_zeta_polylog(_Tp __s, std::complex&lt;_Tp&gt; __a)</code>	233
9.3.2.144	<code>__hydrogen(unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi)</code>	233
9.3.2.145	<code>__hyperg(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	234
9.3.2.146	<code>__hyperg_luke(_Tp __a, _Tp __b, _Tp __c, _Tp __xin)</code>	234
9.3.2.147	<code>__hyperg_reflect(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	234
9.3.2.148	<code>__hyperg_series(_Tp __a, _Tp __b, _Tp __c, _Tp __x)</code>	235
9.3.2.149	<code>__ibeta_cont_frac(_Tp __a, _Tp __b, _Tp __x)</code>	236
9.3.2.150	<code>__jacobi_sncndn(_Tp __k, _Tp __u)</code>	237
9.3.2.151	<code>__jacobi_zeta(_Tp __k, _Tp __phi)</code>	237
9.3.2.152	<code>__laguerre(unsigned int __n, _Tp __x)</code>	237
9.3.2.153	<code>__legendre_q(unsigned int __l, _Tp __x)</code>	238
9.3.2.154	<code>__log_bincoef(unsigned int __n, unsigned int __k)</code>	238
9.3.2.155	<code>__log_bincoef(_Tp __nu, unsigned int __k)</code>	239
9.3.2.156	<code>__log_bincoef_sign(_Tp __nu, unsigned int __k)</code>	239
9.3.2.157	<code>__log_bincoef_sign(std::complex&lt;_Tp&gt; __nu, unsigned int __k)</code>	240
9.3.2.158	<code>__log_double_factorial(_Tp __x)</code>	240
9.3.2.159	<code>__log_double_factorial(int __n)</code>	240
9.3.2.160	<code>__log_factorial(unsigned int __n)</code>	240
9.3.2.161	<code>__log_gamma(_Tp __x)</code>	240
9.3.2.162	<code>__log_gamma(std::complex&lt;_Tp&gt; __x)</code>	241
9.3.2.163	<code>__log_gamma1p_lanczos(_Tp __z)</code>	241
9.3.2.164	<code>__log_gamma1p_spouge(_Tp __z)</code>	242
9.3.2.165	<code>__log_gamma_bernoulli(_Tp __x)</code>	242
9.3.2.166	<code>__log_gamma_sign(_Tp __x)</code>	243
9.3.2.167	<code>__log_gamma_sign(std::complex&lt;_Tp&gt; __x)</code>	243
9.3.2.168	<code>__log_pochhammer(_Tp __a, _Tp __n)</code>	243



9.3.2.169	<code>__log_pochhammer_lower(_Tp __a, _Tp __n)</code>	244
9.3.2.170	<code>__logint(const _Tp __x)</code>	244
9.3.2.171	<code>__lognormal_cdf(_Tp __mu, _Tp __sigma, _Tp __x)</code>	245
9.3.2.172	<code>__lognormal_pdf(_Tp __nu, _Tp __sigma, _Tp __x)</code>	245
9.3.2.173	<code>__normal_cdf(_Tp __mu, _Tp __sigma, _Tp __x)</code>	245
9.3.2.174	<code>__normal_pdf(_Tp __nu, _Tp __sigma, _Tp __x)</code>	245
9.3.2.175	<code>__owens_t(_Tp __h, _Tp __a)</code>	245
9.3.2.176	<code>__pgamma(_Tp __a, _Tp __x)</code>	246
9.3.2.177	<code>__pochhammer(_Tp __a, _Tp __n)</code>	246
9.3.2.178	<code>__pochhammer_lower(_Tp __a, _Tp __n)</code>	247
9.3.2.179	<code>__polar_pi(_Tp __rho, _Tp __phi_pi)</code>	247
9.3.2.180	<code>__poly_hermite(unsigned int __n, _Tp __x)</code>	247
9.3.2.181	<code>__poly_hermite_asymp(unsigned int __n, _Tp __x)</code>	248
9.3.2.182	<code>__poly_hermite_recursion(unsigned int __n, _Tp __x)</code>	248
9.3.2.183	<code>__poly_jacobi(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)</code>	249
9.3.2.184	<code>__poly_laguerre(unsigned int __n, _Tp __alpha1, _Tp __x)</code>	249
9.3.2.185	<code>__poly_laguerre_hyperg(unsigned int __n, _Tp __alpha1, _Tp __x)</code>	250
9.3.2.186	<code>__poly_laguerre_large_n(unsigned __n, _Tp __alpha1, _Tp __x)</code>	250
9.3.2.187	<code>__poly_laguerre_recursion(unsigned int __n, _Tp __alpha1, _Tp __x)</code>	251
9.3.2.188	<code>__poly_legendre_p(unsigned int __l, _Tp __x)</code>	252
9.3.2.189	<code>__poly_radial_jacobi(unsigned int __n, unsigned int __m, _Tp __rho)</code>	252
9.3.2.190	<code>__polylog(_Tp __s, _Tp __x)</code>	253
9.3.2.191	<code>__polylog(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	253
9.3.2.192	<code>__polylog_exp(_Tp __s, ArgType __w)</code>	254
9.3.2.193	<code>__polylog_exp_asymp(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	255
9.3.2.194	<code>__polylog_exp_int_neg(int __s, std::complex&lt; _Tp &gt; __w)</code>	255
9.3.2.195	<code>__polylog_exp_int_neg(const int __s, _Tp __w)</code>	256
9.3.2.196	<code>__polylog_exp_int_pos(unsigned int __s, std::complex&lt; _Tp &gt; __w)</code>	256

9.3.2.197	<code>__polylog_exp_int_pos(unsigned int __s, _Tp __w)</code>	257
9.3.2.198	<code>__polylog_exp_neg(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	257
9.3.2.199	<code>__polylog_exp_neg(int __s, std::complex&lt; _Tp &gt; __w)</code>	258
9.3.2.200	<code>__polylog_exp_neg_even(unsigned int __n, std::complex&lt; _Tp &gt; __w)</code>	259
9.3.2.201	<code>__polylog_exp_neg_odd(unsigned int __n, std::complex&lt; _Tp &gt; __w)</code>	260
9.3.2.202	<code>__polylog_exp_negative_real_part(_PowTp __s, _Tp __w)</code>	261
9.3.2.203	<code>__polylog_exp_pos(unsigned int __s, std::complex&lt; _Tp &gt; __w)</code>	261
9.3.2.204	<code>__polylog_exp_pos(unsigned int __s, _Tp __w)</code>	262
9.3.2.205	<code>__polylog_exp_pos(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	262
9.3.2.206	<code>__polylog_exp_real_neg(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	263
9.3.2.207	<code>__polylog_exp_real_neg(_Tp __s, _Tp __w)</code>	263
9.3.2.208	<code>__polylog_exp_real_pos(_Tp __s, std::complex&lt; _Tp &gt; __w)</code>	264
9.3.2.209	<code>__polylog_exp_real_pos(_Tp __s, _Tp __w)</code>	264
9.3.2.210	<code>__psi(unsigned int __n)</code>	265
9.3.2.211	<code>__psi(_Tp __x)</code>	265
9.3.2.212	<code>__psi(unsigned int __n, _Tp __x)</code>	265
9.3.2.213	<code>__psi_asymp(_Tp __x)</code>	266
9.3.2.214	<code>__psi_series(_Tp __x)</code>	266
9.3.2.215	<code>__qgamma(_Tp __a, _Tp __x)</code>	266
9.3.2.216	<code>__rice_pdf(_Tp __nu, _Tp __sigma, _Tp __x)</code>	267
9.3.2.217	<code>__riemann_zeta(_Tp __s)</code>	267
9.3.2.218	<code>__riemann_zeta_alt(_Tp __s)</code>	267
9.3.2.219	<code>__riemann_zeta_euler_maclaurin(_Tp __s)</code>	268
9.3.2.220	<code>__riemann_zeta_glob(_Tp __s)</code>	268
9.3.2.221	<code>__riemann_zeta_m_1(_Tp __s)</code>	268
9.3.2.222	<code>__riemann_zeta_m_1_sum(_Tp __s)</code>	269
9.3.2.223	<code>__riemann_zeta_product(_Tp __s)</code>	269
9.3.2.224	<code>__riemann_zeta_sum(_Tp __s)</code>	270

9.3.2.225 <code>__sin_pi(_Tp __x)</code> . . . . .	270
9.3.2.226 <code>__sin_pi(std::complex&lt; _Tp &gt; __z)</code> . . . . .	270
9.3.2.227 <code>__sinc(_Tp __x)</code> . . . . .	270
9.3.2.228 <code>__sinc_pi(_Tp __x)</code> . . . . .	271
9.3.2.229 <code>__sincos(_Tp __x)</code> . . . . .	271
9.3.2.230 <code>__sincos(float __x)</code> . . . . .	271
9.3.2.231 <code>__sincos(double __x)</code> . . . . .	271
9.3.2.232 <code>__sincos(long double __x)</code> . . . . .	271
9.3.2.233 <code>__sincos_pi(_Tp __x)</code> . . . . .	271
9.3.2.234 <code>__sincosint(_Tp __x)</code> . . . . .	272
9.3.2.235 <code>__sincosint_asymp(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code> . . . . .	272
9.3.2.236 <code>__sincosint_cont_frac(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code> . . . . .	272
9.3.2.237 <code>__sincosint_series(_Tp __t, _Tp &amp;_Si, _Tp &amp;_Ci)</code> . . . . .	272
9.3.2.238 <code>__sinh_pi(_Tp __x)</code> . . . . .	273
9.3.2.239 <code>__sinh_pi(std::complex&lt; _Tp &gt; __z)</code> . . . . .	273
9.3.2.240 <code>__sinhc(_Tp __x)</code> . . . . .	273
9.3.2.241 <code>__sinhc_pi(_Tp __x)</code> . . . . .	273
9.3.2.242 <code>__sinhint(const _Tp __x)</code> . . . . .	273
9.3.2.243 <code>__sph_bessel(unsigned int __n, _Tp __x)</code> . . . . .	274
9.3.2.244 <code>__sph_bessel(unsigned int __n, std::complex&lt; _Tp &gt; __z)</code> . . . . .	274
9.3.2.245 <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> . . . . .	275
9.3.2.246 <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> . . . . .	275
9.3.2.247 <code>__sph_bessel_jn_neg_arg(unsigned int __n, _Tp __x, std::complex&lt; _Tp &gt; &amp;__j_n, std::complex&lt; _Tp &gt; &amp;__n_n, std::complex&lt; _Tp &gt; &amp;__jp_n, std::complex&lt; _Tp &gt; &amp;__np_n)</code> . . . . .	276
9.3.2.248 <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> . . . . .	276
9.3.2.249 <code>__sph_hankel_1(unsigned int __n, _Tp __x)</code> . . . . .	276

9.3.2.250	<code>__sph_hankel_1(unsigned int __n, std::complex&lt; _Tp &gt; __z)</code>	277
9.3.2.251	<code>__sph_hankel_2(unsigned int __n, _Tp __x)</code>	277
9.3.2.252	<code>__sph_hankel_2(unsigned int __n, std::complex&lt; _Tp &gt; __z)</code>	278
9.3.2.253	<code>__sph_harmonic(unsigned int __l, int __m, _Tp __theta, _Tp __phi)</code>	278
9.3.2.254	<code>__sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</code>	279
9.3.2.255	<code>__sph_neumann(unsigned int __n, _Tp __x)</code>	279
9.3.2.256	<code>__sph_neumann(unsigned int __n, std::complex&lt; _Tp &gt; __z)</code>	280
9.3.2.257	<code>__student_t_cdf(_Tp __t, unsigned int __nu)</code>	280
9.3.2.258	<code>__student_t_cdfc(_Tp __t, unsigned int __nu)</code>	281
9.3.2.259	<code>__tan_pi(_Tp __x)</code>	281
9.3.2.260	<code>__tan_pi(std::complex&lt; _Tp &gt; __z)</code>	281
9.3.2.261	<code>__tanh_pi(_Tp __x)</code>	282
9.3.2.262	<code>__tanh_pi(std::complex&lt; _Tp &gt; __z)</code>	282
9.3.2.263	<code>__tgamma(_Tp __a, _Tp __x)</code>	282
9.3.2.264	<code>__tgamma_lower(_Tp __a, _Tp __x)</code>	282
9.3.2.265	<code>__theta_1(_Tp __nu, _Tp __x)</code>	282
9.3.2.266	<code>__theta_2(_Tp __nu, _Tp __x)</code>	283
9.3.2.267	<code>__theta_2_asymp(_Tp __nu, _Tp __x)</code>	283
9.3.2.268	<code>__theta_2_sum(_Tp __nu, _Tp __x)</code>	283
9.3.2.269	<code>__theta_3(_Tp __nu, _Tp __x)</code>	284
9.3.2.270	<code>__theta_3_asymp(_Tp __nu, _Tp __x)</code>	284
9.3.2.271	<code>__theta_3_sum(_Tp __nu, _Tp __x)</code>	284
9.3.2.272	<code>__theta_4(_Tp __nu, _Tp __x)</code>	284
9.3.2.273	<code>__theta_c(_Tp __k, _Tp __x)</code>	285
9.3.2.274	<code>__theta_d(_Tp __k, _Tp __x)</code>	285
9.3.2.275	<code>__theta_n(_Tp __k, _Tp __x)</code>	285
9.3.2.276	<code>__theta_s(_Tp __k, _Tp __x)</code>	285
9.3.2.277	<code>__weibull_cdf(_Tp __a, _Tp __b, _Tp __x)</code>	286

9.3.2.278	<code>__weibull_pdf(_Tp __a, _Tp __b, _Tp __x)</code>	286
9.3.2.279	<code>__zernike(unsigned int __n, int __m, _Tp __rho, _Tp __phi)</code>	286
9.3.2.280	<code>__znorm1(_Tp __x)</code>	287
9.3.2.281	<code>__znorm2(_Tp __x)</code>	287
9.3.2.282	<code>evenzeta(unsigned int __k)</code>	287
9.3.3	Variable Documentation	287
9.3.3.1	<code>__max_FGH</code>	287
9.3.3.2	<code>__max_FGH&lt; double &gt;</code>	288
9.3.3.3	<code>__max_FGH&lt; float &gt;</code>	288
9.3.3.4	<code>_Num_Euler_Maclaurin_zeta</code>	288
9.3.3.5	<code>_S_double_factorial_table</code>	288
9.3.3.6	<code>_S_Euler_Maclaurin_zeta</code>	288
9.3.3.7	<code>_S_factorial_table</code>	288
9.3.3.8	<code>_S_harmonic_denom</code>	288
9.3.3.9	<code>_S_harmonic_numer</code>	289
9.3.3.10	<code>_S_neg_double_factorial_table</code>	289
9.3.3.11	<code>_S_num_double_factorials</code>	289
9.3.3.12	<code>_S_num_double_factorials&lt; double &gt;</code>	289
9.3.3.13	<code>_S_num_double_factorials&lt; float &gt;</code>	289
9.3.3.14	<code>_S_num_double_factorials&lt; long double &gt;</code>	289
9.3.3.15	<code>_S_num_factorials</code>	289
9.3.3.16	<code>_S_num_factorials&lt; double &gt;</code>	289
9.3.3.17	<code>_S_num_factorials&lt; float &gt;</code>	289
9.3.3.18	<code>_S_num_factorials&lt; long double &gt;</code>	290
9.3.3.19	<code>_S_num_harmonic_numer</code>	290
9.3.3.20	<code>_S_num_neg_double_factorials</code>	290
9.3.3.21	<code>_S_num_neg_double_factorials&lt; double &gt;</code>	290
9.3.3.22	<code>_S_num_neg_double_factorials&lt; float &gt;</code>	290
9.3.3.23	<code>_S_num_neg_double_factorials&lt; long double &gt;</code>	290
9.3.3.24	<code>_S_num_zetam1</code>	290
9.3.3.25	<code>_S_zetam1</code>	290

<b>10 Class Documentation</b>	<b>291</b>
10.1 <code>__gnu_cxx::__sincos_t&lt;_Tp&gt;</code> Struct Template Reference	291
10.1.1 Detailed Description	291
10.1.2 Member Data Documentation	291
10.1.2.1 <code>cos_value</code>	291
10.1.2.2 <code>sin_value</code>	291
10.2 <code>std::__detail::_Airy&lt;_Tp&gt;</code> Class Template Reference	292
10.2.1 Detailed Description	292
10.2.2 Member Typedef Documentation	293
10.2.2.1 <code>scalar_type</code>	293
10.2.2.2 <code>value_type</code>	293
10.2.3 Constructor & Destructor Documentation	293
10.2.3.1 <code>_Airy()=default</code>	293
10.2.3.2 <code>_Airy(const _Airy &amp;)=default</code>	293
10.2.3.3 <code>_Airy(_Airy &amp;&amp;)=default</code>	293
10.2.4 Member Function Documentation	293
10.2.4.1 <code>operator()(value_type __y) const</code>	293
10.2.5 Member Data Documentation	293
10.2.5.1 <code>_S_2pi_3</code>	293
10.2.5.2 <code>_S_5pi_6</code>	293
10.2.5.3 <code>_S_cNaN</code>	294
10.2.5.4 <code>_S_i</code>	294
10.2.5.5 <code>_S_NaN</code>	294
10.2.5.6 <code>_S_pi</code>	294
10.2.5.7 <code>_S_pi_3</code>	294
10.2.5.8 <code>_S_pi_6</code>	294
10.2.5.9 <code>_S_sqrt_pi</code>	294
10.2.5.10 <code>inner_radius</code>	294

10.2.5.11	<code>outer_radius</code>	295
10.3	<code>std::__detail::_Airy_asymp&lt; _Tp &gt;</code> Class Template Reference	295
10.3.1	Detailed Description	296
10.3.2	Member Typedef Documentation	296
10.3.2.1	<code>__cmplx</code>	296
10.3.3	Constructor & Destructor Documentation	296
10.3.3.1	<code>_Airy_asymp()</code> =default	296
10.3.4	Member Function Documentation	296
10.3.4.1	<code>_S_absarg_ge_pio3(std::complex&lt; _Tp &gt; __z) const</code>	296
10.3.4.2	<code>_S_absarg_lt_pio3(std::complex&lt; _Tp &gt; __z) const</code>	297
10.3.4.3	<code>operator()(std::complex&lt; _Tp &gt; __t, bool __return_fock_airy=false) const</code>	298
10.4	<code>std::__detail::_Airy_asymp_data&lt; _Tp &gt;</code> Struct Template Reference	298
10.4.1	Detailed Description	298
10.5	<code>std::__detail::_Airy_asymp_data&lt; double &gt;</code> Struct Template Reference	299
10.5.1	Detailed Description	299
10.5.2	Member Data Documentation	299
10.5.2.1	<code>_S_c</code>	299
10.5.2.2	<code>_S_d</code>	299
10.5.2.3	<code>_S_max_cd</code>	299
10.6	<code>std::__detail::_Airy_asymp_data&lt; float &gt;</code> Struct Template Reference	300
10.6.1	Detailed Description	300
10.6.2	Member Data Documentation	300
10.6.2.1	<code>_S_c</code>	300
10.6.2.2	<code>_S_d</code>	300
10.6.2.3	<code>_S_max_cd</code>	300
10.7	<code>std::__detail::_Airy_asymp_data&lt; long double &gt;</code> Struct Template Reference	300
10.7.1	Detailed Description	301
10.7.2	Member Data Documentation	301

10.7.2.1	<a href="#">_S_c</a>	301
10.7.2.2	<a href="#">_S_d</a>	301
10.7.2.3	<a href="#">_S_max_cd</a>	301
10.8	<a href="#">std::__detail::_Airy_asymp_series&lt;_Sum&gt; Class Template Reference</a>	301
10.8.1	<a href="#">Detailed Description</a>	301
10.8.2	<a href="#">Member Typedef Documentation</a>	302
10.8.2.1	<a href="#">scalar_type</a>	302
10.8.2.2	<a href="#">value_type</a>	302
10.8.3	<a href="#">Constructor &amp; Destructor Documentation</a>	302
10.8.3.1	<a href="#">_Airy_asymp_series(_Sum __proto)</a>	302
10.8.3.2	<a href="#">_Airy_asymp_series(const _Airy_asymp_series &amp;)=default</a>	302
10.8.3.3	<a href="#">_Airy_asymp_series(_Airy_asymp_series &amp;&amp;)=default</a>	302
10.8.4	<a href="#">Member Function Documentation</a>	302
10.8.4.1	<a href="#">operator()(value_type __y)</a>	302
10.8.5	<a href="#">Member Data Documentation</a>	303
10.8.5.1	<a href="#">_S_sqrt_pi</a>	303
10.9	<a href="#">std::__detail::_Airy_default_radii&lt;_Tp&gt; Struct Template Reference</a>	303
10.9.1	<a href="#">Detailed Description</a>	303
10.10	<a href="#">std::__detail::_Airy_default_radii&lt;double&gt; Struct Template Reference</a>	303
10.10.1	<a href="#">Detailed Description</a>	303
10.10.2	<a href="#">Member Data Documentation</a>	304
10.10.2.1	<a href="#">inner_radius</a>	304
10.10.2.2	<a href="#">outer_radius</a>	304
10.11	<a href="#">std::__detail::_Airy_default_radii&lt;float&gt; Struct Template Reference</a>	304
10.11.1	<a href="#">Detailed Description</a>	304
10.11.2	<a href="#">Member Data Documentation</a>	304
10.11.2.1	<a href="#">inner_radius</a>	304
10.11.2.2	<a href="#">outer_radius</a>	304



10.12std::__detail::_Airy_default_radii< long double > Struct Template Reference . . . . .	305
10.12.1 Detailed Description . . . . .	305
10.12.2 Member Data Documentation . . . . .	305
10.12.2.1 inner_radius . . . . .	305
10.12.2.2 outer_radius . . . . .	305
10.13std::__detail::_Airy_series< _Tp > Class Template Reference . . . . .	305
10.13.1 Detailed Description . . . . .	306
10.13.2 Member Function Documentation . . . . .	306
10.13.2.1 _S_Ai(std::complex< _Tp > __t) . . . . .	306
10.13.2.2 _S_Airy(std::complex< _Tp > __t) . . . . .	307
10.13.2.3 _S_Bi(std::complex< _Tp > __t) . . . . .	307
10.13.2.4 _S_FGH(std::complex< _Tp > __t) . . . . .	308
10.13.2.5 _S_Fock(std::complex< _Tp > __t) . . . . .	308
10.13.2.6 _S_Scorer(std::complex< _Tp > __t) . . . . .	309
10.13.2.7 _S_Scorer2(std::complex< _Tp > __t) . . . . .	309
10.13.3 Member Data Documentation . . . . .	310
10.13.3.1 _N_FGH . . . . .	310
10.13.3.2 _S_Ai0 . . . . .	310
10.13.3.3 _S_Aip0 . . . . .	310
10.13.3.4 _S_Bi0 . . . . .	310
10.13.3.5 _S_Bip0 . . . . .	310
10.13.3.6 _S_eps . . . . .	310
10.13.3.7 _S_Gi0 . . . . .	310
10.13.3.8 _S_Gip0 . . . . .	310
10.13.3.9 _S_Hi0 . . . . .	311
10.13.3.10 _S_Hip0 . . . . .	311
10.13.3.11 _S_i . . . . .	311
10.13.3.12 _S_pi . . . . .	311

10.13.3.13 <code>S_sqrt_pi</code> . . . . .	311
10.14 <code>std::__detail::_AiryAuxilliaryState&lt;_Tp&gt;</code> Struct Template Reference . . . . .	311
10.14.1 Detailed Description . . . . .	312
10.14.2 Member Typedef Documentation . . . . .	312
10.14.2.1 <code>_Val</code> . . . . .	312
10.14.3 Member Data Documentation . . . . .	312
10.14.3.1 <code>fai</code> . . . . .	312
10.14.3.2 <code>faip</code> . . . . .	312
10.14.3.3 <code>gai</code> . . . . .	312
10.14.3.4 <code>gaip</code> . . . . .	312
10.14.3.5 <code>hai</code> . . . . .	312
10.14.3.6 <code>haip</code> . . . . .	313
10.14.3.7 <code>z</code> . . . . .	313
10.15 <code>std::__detail::_AiryState&lt;_Tp&gt;</code> Struct Template Reference . . . . .	313
10.15.1 Detailed Description . . . . .	313
10.15.2 Member Typedef Documentation . . . . .	314
10.15.2.1 <code>_Val</code> . . . . .	314
10.15.3 Member Function Documentation . . . . .	314
10.15.3.1 <code>true_Wronskian()</code> . . . . .	314
10.15.3.2 <code>Wronskian() const</code> . . . . .	314
10.15.4 Member Data Documentation . . . . .	314
10.15.4.1 <code>Ai</code> . . . . .	314
10.15.4.2 <code>Aip</code> . . . . .	314
10.15.4.3 <code>Bi</code> . . . . .	314
10.15.4.4 <code>Bip</code> . . . . .	314
10.15.4.5 <code>z</code> . . . . .	315
10.16 <code>std::__detail::_Factorial_table&lt;_Tp&gt;</code> Struct Template Reference . . . . .	315
10.16.1 Detailed Description . . . . .	315

10.16.2 Member Data Documentation . . . . .	315
10.16.2.1 __factorial . . . . .	315
10.16.2.2 __log_factorial . . . . .	315
10.16.2.3 __n . . . . .	316
10.17std::__detail::__GammaLanczos< _Tp > Struct Template Reference . . . . .	316
10.17.1 Detailed Description . . . . .	316
10.18std::__detail::__GammaLanczos< double > Struct Template Reference . . . . .	316
10.18.1 Detailed Description . . . . .	316
10.18.2 Member Data Documentation . . . . .	317
10.18.2.1 _S_cheby . . . . .	317
10.18.2.2 _S_g . . . . .	317
10.19std::__detail::__GammaLanczos< float > Struct Template Reference . . . . .	317
10.19.1 Detailed Description . . . . .	317
10.19.2 Member Data Documentation . . . . .	318
10.19.2.1 _S_cheby . . . . .	318
10.19.2.2 _S_g . . . . .	318
10.20std::__detail::__GammaLanczos< long double > Struct Template Reference . . . . .	318
10.20.1 Detailed Description . . . . .	318
10.20.2 Member Data Documentation . . . . .	319
10.20.2.1 _S_cheby . . . . .	319
10.20.2.2 _S_g . . . . .	319
10.21std::__detail::__GammaSpouge< _Tp > Struct Template Reference . . . . .	319
10.21.1 Detailed Description . . . . .	319
10.22std::__detail::__GammaSpouge< double > Struct Template Reference . . . . .	320
10.22.1 Detailed Description . . . . .	320
10.22.2 Member Data Documentation . . . . .	320
10.22.2.1 _S_cheby . . . . .	320
10.23std::__detail::__GammaSpouge< float > Struct Template Reference . . . . .	320
10.23.1 Detailed Description . . . . .	321
10.23.2 Member Data Documentation . . . . .	321
10.23.2.1 _S_cheby . . . . .	321
10.24std::__detail::__GammaSpouge< long double > Struct Template Reference . . . . .	321
10.24.1 Detailed Description . . . . .	321
10.24.2 Member Data Documentation . . . . .	322
10.24.2.1 _S_cheby . . . . .	322

<b>11 File Documentation</b>	<b>323</b>
11.1 bits/sf_airy.tcc File Reference	323
11.1.1 Detailed Description	325
11.1.2 Macro Definition Documentation	325
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC	325
11.2 bits/sf_bessel.tcc File Reference	325
11.2.1 Detailed Description	327
11.2.2 Macro Definition Documentation	327
11.2.2.1 _GLIBCXX_BITS_SF_BESSEL_TCC	327
11.3 bits/sf_beta.tcc File Reference	328
11.3.1 Detailed Description	329
11.3.2 Macro Definition Documentation	329
11.3.2.1 _GLIBCXX_BITS_SF_BETA_TCC	329
11.4 bits/sf_cardinal.tcc File Reference	329
11.4.1 Macro Definition Documentation	331
11.4.1.1 _GLIBCXX_BITS_SF_CARDINAL_TCC	331
11.5 bits/sf_chebyshev.tcc File Reference	331
11.5.1 Detailed Description	332
11.5.2 Macro Definition Documentation	332
11.5.2.1 _GLIBCXX_BITS_SF_CHEBYSHEV_TCC	332
11.6 bits/sf_dawson.tcc File Reference	332
11.6.1 Detailed Description	333
11.6.2 Macro Definition Documentation	333
11.6.2.1 _GLIBCXX_BITS_SF_DAWSON_TCC	333
11.7 bits/sf_distributions.tcc File Reference	334
11.7.1 Detailed Description	336
11.7.2 Macro Definition Documentation	336
11.7.2.1 _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC	336

11.8 bits/sf_ellint.tcc File Reference . . . . .	336
11.8.1 Detailed Description . . . . .	339
11.8.2 Macro Definition Documentation . . . . .	339
11.8.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC . . . . .	339
11.9 bits/sf_expint.tcc File Reference . . . . .	339
11.9.1 Detailed Description . . . . .	341
11.9.2 Macro Definition Documentation . . . . .	341
11.9.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC . . . . .	341
11.10 bits/sf_fresnel.tcc File Reference . . . . .	341
11.10.1 Detailed Description . . . . .	342
11.10.2 Macro Definition Documentation . . . . .	342
11.10.2.1 _GLIBCXX_BITS_SF_FRESNEL_TCC . . . . .	342
11.11 bits/sf_gamma.tcc File Reference . . . . .	343
11.11.1 Detailed Description . . . . .	349
11.11.2 Macro Definition Documentation . . . . .	349
11.11.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC . . . . .	349
11.12 bits/sf_gegenbauer.tcc File Reference . . . . .	350
11.12.1 Detailed Description . . . . .	351
11.12.2 Macro Definition Documentation . . . . .	351
11.12.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC . . . . .	351
11.13 bits/sf_hankel.tcc File Reference . . . . .	351
11.13.1 Detailed Description . . . . .	353
11.13.2 Macro Definition Documentation . . . . .	354
11.13.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC . . . . .	354
11.14 bits/sf_hermite.tcc File Reference . . . . .	354
11.14.1 Detailed Description . . . . .	355
11.14.2 Macro Definition Documentation . . . . .	355
11.14.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC . . . . .	355

11.15bits/sf_hydrogen.tcc File Reference . . . . .	355
11.15.1 Detailed Description . . . . .	356
11.15.2 Macro Definition Documentation . . . . .	356
11.15.2.1 _GLIBCXX_BITS_SF_HYDROGEN_TCC . . . . .	356
11.16bits/sf_hyperg.tcc File Reference . . . . .	357
11.16.1 Detailed Description . . . . .	358
11.16.2 Macro Definition Documentation . . . . .	358
11.16.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC . . . . .	358
11.17bits/sf_hypint.tcc File Reference . . . . .	359
11.17.1 Detailed Description . . . . .	360
11.17.2 Macro Definition Documentation . . . . .	360
11.17.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC . . . . .	360
11.18bits/sf_jacobi.tcc File Reference . . . . .	360
11.18.1 Detailed Description . . . . .	361
11.18.2 Macro Definition Documentation . . . . .	361
11.18.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC . . . . .	361
11.19bits/sf_laguerre.tcc File Reference . . . . .	362
11.19.1 Detailed Description . . . . .	363
11.19.2 Macro Definition Documentation . . . . .	363
11.19.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC . . . . .	363
11.20bits/sf_legendre.tcc File Reference . . . . .	363
11.20.1 Detailed Description . . . . .	365
11.20.2 Macro Definition Documentation . . . . .	365
11.20.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC . . . . .	365
11.21bits/sf_mod_bessel.tcc File Reference . . . . .	365
11.21.1 Detailed Description . . . . .	367
11.21.2 Macro Definition Documentation . . . . .	367
11.21.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC . . . . .	367

11.22bits/sf_owens.t.tcc File Reference . . . . .	367
11.22.1 Detailed Description . . . . .	368
11.22.2 Macro Definition Documentation . . . . .	368
11.22.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC . . . . .	368
11.23bits/sf_polylog.tcc File Reference . . . . .	368
11.23.1 Detailed Description . . . . .	371
11.23.2 Macro Definition Documentation . . . . .	371
11.23.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC . . . . .	371
11.24bits/sf_theta.tcc File Reference . . . . .	371
11.24.1 Detailed Description . . . . .	373
11.24.2 Macro Definition Documentation . . . . .	373
11.24.2.1 _GLIBCXX_BITS_SF_THETA_TCC . . . . .	373
11.25bits/sf_trig.tcc File Reference . . . . .	373
11.25.1 Detailed Description . . . . .	375
11.25.2 Macro Definition Documentation . . . . .	375
11.25.2.1 _GLIBCXX_BITS_SF_TRIG_TCC . . . . .	375
11.26bits/sf_trigint.tcc File Reference . . . . .	375
11.26.1 Detailed Description . . . . .	377
11.26.2 Macro Definition Documentation . . . . .	377
11.26.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC . . . . .	377
11.27bits/sf_zeta.tcc File Reference . . . . .	377
11.27.1 Detailed Description . . . . .	379
11.27.2 Macro Definition Documentation . . . . .	379
11.27.2.1 _GLIBCXX_BITS_SF_ZETA_TCC . . . . .	379
11.28bits/specfun.h File Reference . . . . .	380
11.28.1 Detailed Description . . . . .	391
11.28.2 Macro Definition Documentation . . . . .	391
11.28.2.1 __cpp_lib_math_special_functions . . . . .	391
11.28.2.2 __STDCPP_MATH_SPEC_FUNCS__ . . . . .	391





# 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
- [bose\\_einstein](#) - Bose-Einstein integrals
- [chebyshev\\_t](#) - Chebyshev polynomials of the first kind
- [chebyshev\\_u](#) - Chebyshev polynomials of the second kind
- [chebyshev\\_v](#) - Chebyshev polynomials of the third kind
- [chebyshev\\_w](#) - Chebyshev polynomials of the fourth kind
- [clausen](#) - Clausen integrals
- [clausen\\_c](#) - Clausen cosine integrals
- [clausen\\_s](#) - Clausen sine integrals
- [comp\\_ellint\\_d](#) - Incomplete Legendre D elliptic integral
- [conf\\_hyperg\\_lim](#) - Confluent hypergeometric limit functions
- [cos\\_pi](#) - Reperiodized cosine function.
- [cosh\\_pi](#) - Reperiodized hyperbolic cosine function.
- [coshint](#) - Hyperbolic cosine integral

- [cosint](#) - Cosine integral
- [cyl\\_hankel\\_1](#) - Cylindrical Hankel functions of the first kind
- [cyl\\_hankel\\_2](#) - Cylindrical Hankel functions of the second kind
- [dawson](#) - Dawson integrals
- [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
- [ellnome](#) - Elliptic nome
- [expint](#) - Exponential integrals
- [factorial](#) - Factorials
- [fermi\\_dirac](#) - Fermi-Dirac integrals
- [fresnel\\_c](#) - Fresnel cosine integrals
- [fresnel\\_s](#) - Fresnel sine integrals
- [pgamma](#) - Regularized lower incomplete gamma functions
- [qgamma](#) - Regularized 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
- `lgamma` - Log gamma for complex arguments
- `lpochhammer_lower` - Log lower Pochhammer functions
- `lpochhammer` - Log upper Pochhammer functions
- `owens_t` - Owens T functions
- `pochhammer_lower` - Lower Pochhammer functions
- `pochhammer` - Upper Pochhammer functions
- `psi` - Psi or digamma function
- `radpoly` - Radial polynomials
- `sinhc` - Hyperbolic sinus cardinal function
- `sinhc_pi` -
- `sinc` - Normalized sinus cardinal function
- `sincos` - Sine + cosine function
- `sincos_pi` - Reperiodized sine + cosine function
- `sin_pi` - Reperiodized sine function.
- `sinh_pi` - Reperiodized hyperbolic sine function.
- `sinc_pi` - Sinus cardinal function
- `sinhint` - Hyperbolic sine integral
- `sinint` - Sine integral
- `sph_bessel_i` - Spherical regular modified Bessel functions
- `sph_bessel_k` - Spherical irregular modified Bessel functions
- `sph_hankel_1` - Spherical Hankel functions of the first kind
- `sph_hankel_2` - Spherical Hankel functions of the first kind
- `sph_harmonic` - Spherical
- `tan_pi` - Reperiodized tangent function.
- `tanh_pi` - Reperiodized hyperbolic tangent function.
- `tgamma` - Gamma for complex arguments
- `tgamma` - Upper incomplete gamma functions
- `tgamma_lower` - Lower incomplete gamma functions
- `theta_1` - Exponential theta function 1
- `theta_2` - Exponential theta function 2
- `theta_3` - Exponential theta function 3
- `theta_4` - Exponential theta function 4
- `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/scientific/), [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\\_polylog](#) ([\\_Tp \\_\\_s](#), [std::complex<\\_Tp> \\_\\_a](#))

This [\\_\\_hurwitz\\_zeta\\_polylog](#) prefactor is prone to overflow. positive integer orders  $s$ ?

#### Member [std::\\_\\_detail::\\_\\_Airy\\_asymp<\\_Tp>::\\_\\_S\\_absarg\\_lt\\_pio3](#) ([std::complex<\\_Tp> \\_\\_z](#)) const

Revisit these numbers of terms for the Airy asymptotic expansions.

#### Member [std::\\_\\_detail::\\_\\_Airy\\_series<\\_Tp>::\\_\\_S\\_Scorer](#) ([std::complex<\\_Tp> \\_\\_t](#))

Find out what is wrong with the  $H_i = f_{ai} + g_{ai} + h_{ai}$  scorer function.





## Chapter 3

# Module Index

### 3.1 Modules

Here is a list of all modules:

C++ Mathematical Special Functions . . . . .	<a href="#">19</a>
C++17/IS29124 Mathematical Special Functions . . . . .	<a href="#">20</a>
GNU Extended Mathematical Special Functions . . . . .	<a href="#">44</a>



## Chapter 4

# Namespace Index

### 4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">__gnu_cxx</a>	.....	143
<a href="#">std</a>	.....	152
<a href="#">std::__detail</a>	.....	154



## Chapter 5

# Hierarchical Index

### 5.1 Class Hierarchy

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

<code>__gnu_cxx::__sincos_t&lt; _Tp &gt;</code> . . . . .	291
<code>std::__detail::_Airy&lt; _Tp &gt;</code> . . . . .	292
<code>std::__detail::_Airy_asymp_data&lt; _Tp &gt;</code> . . . . .	298
<code>std::__detail::_Airy_asymp&lt; _Tp &gt;</code> . . . . .	295
<code>std::__detail::_Airy_asymp_data&lt; double &gt;</code> . . . . .	299
<code>std::__detail::_Airy_asymp_data&lt; float &gt;</code> . . . . .	300
<code>std::__detail::_Airy_asymp_data&lt; long double &gt;</code> . . . . .	300
<code>std::__detail::_Airy_asymp_series&lt; _Sum &gt;</code> . . . . .	301
<code>std::__detail::_Airy_default_radii&lt; _Tp &gt;</code> . . . . .	303
<code>std::__detail::_Airy_default_radii&lt; double &gt;</code> . . . . .	303
<code>std::__detail::_Airy_default_radii&lt; float &gt;</code> . . . . .	304
<code>std::__detail::_Airy_default_radii&lt; long double &gt;</code> . . . . .	305
<code>std::__detail::_Airy_series&lt; _Tp &gt;</code> . . . . .	305
<code>std::__detail::_AiryAuxilliaryState&lt; _Tp &gt;</code> . . . . .	311
<code>std::__detail::_AiryState&lt; _Tp &gt;</code> . . . . .	313
<code>std::__detail::_Factorial_table&lt; _Tp &gt;</code> . . . . .	315
<code>std::__detail::_GammaLanczos&lt; _Tp &gt;</code> . . . . .	316
<code>std::__detail::_GammaLanczos&lt; double &gt;</code> . . . . .	316
<code>std::__detail::_GammaLanczos&lt; float &gt;</code> . . . . .	317
<code>std::__detail::_GammaLanczos&lt; long double &gt;</code> . . . . .	318
<code>std::__detail::_GammaSpouge&lt; _Tp &gt;</code> . . . . .	319
<code>std::__detail::_GammaSpouge&lt; double &gt;</code> . . . . .	320
<code>std::__detail::_GammaSpouge&lt; float &gt;</code> . . . . .	320
<code>std::__detail::_GammaSpouge&lt; long double &gt;</code> . . . . .	321



## Chapter 6

# Class Index

### 6.1 Class List

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

<a href="#">__gnu_cxx::__sincos_t&lt; _Tp &gt;</a>	291
<a href="#">std::__detail::__Airy&lt; _Tp &gt;</a>	292
<a href="#">std::__detail::__Airy_asymp&lt; _Tp &gt;</a>	295
<a href="#">std::__detail::__Airy_asymp_data&lt; _Tp &gt;</a>	298
<a href="#">std::__detail::__Airy_asymp_data&lt; double &gt;</a>	299
<a href="#">std::__detail::__Airy_asymp_data&lt; float &gt;</a>	300
<a href="#">std::__detail::__Airy_asymp_data&lt; long double &gt;</a>	300
<a href="#">std::__detail::__Airy_asymp_series&lt; _Sum &gt;</a>	301
<a href="#">std::__detail::__Airy_default_radii&lt; _Tp &gt;</a>	303
<a href="#">std::__detail::__Airy_default_radii&lt; double &gt;</a>	303
<a href="#">std::__detail::__Airy_default_radii&lt; float &gt;</a>	304
<a href="#">std::__detail::__Airy_default_radii&lt; long double &gt;</a>	305
<a href="#">std::__detail::__Airy_series&lt; _Tp &gt;</a>	305
<a href="#">std::__detail::__AiryAuxilliaryState&lt; _Tp &gt;</a>	311
<a href="#">std::__detail::__AiryState&lt; _Tp &gt;</a>	313
<a href="#">std::__detail::__Factorial_table&lt; _Tp &gt;</a>	315
<a href="#">std::__detail::__GammaLanczos&lt; _Tp &gt;</a>	316
<a href="#">std::__detail::__GammaLanczos&lt; double &gt;</a>	316
<a href="#">std::__detail::__GammaLanczos&lt; float &gt;</a>	317
<a href="#">std::__detail::__GammaLanczos&lt; long double &gt;</a>	318
<a href="#">std::__detail::__GammaSpouge&lt; _Tp &gt;</a>	319
<a href="#">std::__detail::__GammaSpouge&lt; double &gt;</a>	320
<a href="#">std::__detail::__GammaSpouge&lt; float &gt;</a>	320
<a href="#">std::__detail::__GammaSpouge&lt; long double &gt;</a>	321





## Chapter 7

# File Index

### 7.1 File List

Here is a list of all files with brief descriptions:

<a href="#">bits/sf_airy.tcc</a>	323
<a href="#">bits/sf_bessel.tcc</a>	325
<a href="#">bits/sf_beta.tcc</a>	328
<a href="#">bits/sf_cardinal.tcc</a>	329
<a href="#">bits/sf_chebyshev.tcc</a>	331
<a href="#">bits/sf_dawson.tcc</a>	332
<a href="#">bits/sf_distributions.tcc</a>	334
<a href="#">bits/sf_ellint.tcc</a>	336
<a href="#">bits/sf_expint.tcc</a>	339
<a href="#">bits/sf_fresnel.tcc</a>	341
<a href="#">bits/sf_gamma.tcc</a>	343
<a href="#">bits/sf_gegenbauer.tcc</a>	350
<a href="#">bits/sf_hankel.tcc</a>	351
<a href="#">bits/sf_hermite.tcc</a>	354
<a href="#">bits/sf_hydrogen.tcc</a>	355
<a href="#">bits/sf_hyperg.tcc</a>	357
<a href="#">bits/sf_hypint.tcc</a>	359
<a href="#">bits/sf_jacobi.tcc</a>	360
<a href="#">bits/sf_laguerre.tcc</a>	362
<a href="#">bits/sf_legendre.tcc</a>	363
<a href="#">bits/sf_mod_bessel.tcc</a>	365
<a href="#">bits/sf_owens_t.tcc</a>	367
<a href="#">bits/sf_polylog.tcc</a>	368
<a href="#">bits/sf_theta.tcc</a>	371
<a href="#">bits/sf_trig.tcc</a>	373
<a href="#">bits/sf_trigint.tcc</a>	375
<a href="#">bits/sf_zeta.tcc</a>	377
<a href="#">bits/specfun.h</a>	380

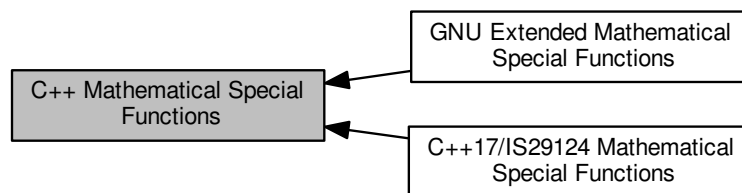


## Chapter 8

# Module Documentation

### 8.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



#### Modules

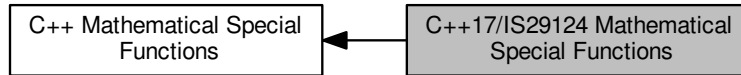
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

#### 8.1.1 Detailed Description

A collection of advanced mathematical special functions.

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

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



### Functions

- `template<typename _Tp >`  
`__gnu_cxx::__promote< _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)

## 8.2.1 Detailed Description

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

## 8.2.2 Function Documentation

**8.2.2.1** `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 $\geq$ 0.
$\leftrightarrow$ __m	The degree of the Laguerre function, __m $\geq$ 0.
$\leftrightarrow$ __x	The argument of the Laguerre function, __x $\geq$ 0.

## Exceptions

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

Definition at line 389 of file specfun.h.

**8.2.2.2** `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 341 of file specfun.h.

**8.2.2.3** `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 352 of file specfun.h.

**8.2.2.4** `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 437 of file `specfun.h`.

**8.2.2.5** `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 404 of file `specfun.h`.

**8.2.2.6** `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 415 of file `specfun.h`.

**8.2.2.7** `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 482 of file `specfun.h`.

8.2.2.8 `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 451 of file `specfun.h`.

8.2.2.9 `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 461 of file `specfun.h`.

8.2.2.10 `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 530 of file `specfun.h`.

**8.2.2.11** `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 497 of file `specfun.h`.

**8.2.2.12** `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 507 of file `specfun.h`.

**8.2.2.13** `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 577 of file `specfun.h`.

**8.2.2.14** `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 545 of file `specfun.h`.

**8.2.2.15** `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 555 of file `specfun.h`.

**8.2.2.16** `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 628 of file `specfun.h`.

**8.2.2.17** `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 592 of file `specfun.h`.

**8.2.2.18** `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 602 of file `specfun.h`.

**8.2.2.19** `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 674 of file `specfun.h`.

**8.2.2.20** `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 643 of file `specfun.h`.

**8.2.2.21** `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 653 of file `specfun.h`.

**8.2.2.22** `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 720 of file `specfun.h`.

**8.2.2.23** `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 689 of file `specfun.h`.

**8.2.2.24** `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 699 of file `specfun.h`.

**8.2.2.25** `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 772 of file `specfun.h`.

**8.2.2.26** `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 735 of file `specfun.h`.

**8.2.2.27** `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 745 of file `specfun.h`.

**8.2.2.28** `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 820 of file `specfun.h`.

**8.2.2.29** `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 787 of file `specfun.h`.

**8.2.2.30** `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 797 of file `specfun.h`.

**8.2.2.31** `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 868 of file `specfun.h`.

**8.2.2.32** `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 835 of file `specfun.h`.

**8.2.2.33** `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 845 of file `specfun.h`.

**8.2.2.34** `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 916 of file `specfun.h`.

**8.2.2.35** `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 883 of file `specfun.h`.

**8.2.2.36** `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 893 of file `specfun.h`.

8.2.2.37 `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 969 of file `specfun.h`.

8.2.2.38 `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 931 of file `specfun.h`.

**8.2.2.39** `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 941 of file specfun.h.

**8.2.2.40** `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 1009 of file specfun.h.

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

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

See also

[expint](#) for details.

Definition at line 983 of file specfun.h.

**8.2.2.42** `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 993 of file specfun.h.

**8.2.2.43** `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 1057 of file `specfun.h`.

**8.2.2.44** `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 1024 of file `specfun.h`.

**8.2.2.45** `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 1034 of file `specfun.h`.

**8.2.2.46** `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 1101 of file `specfun.h`.

**8.2.2.47** `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 1072 of file `specfun.h`.

**8.2.2.48** `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 1082 of file `specfun.h`.

**8.2.2.49** `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 1146 of file `specfun.h`.

**8.2.2.50** `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 1116 of file `specfun.h`.

**8.2.2.51** `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 1126 of file `specfun.h`.

**8.2.2.52** `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 1197 of file `specfun.h`.

**8.2.2.53** `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 1161 of file `specfun.h`.

**8.2.2.54** `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 1171 of file `specfun.h`.



**8.2.2.55** `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 1241 of file `specfun.h`.

**8.2.2.56** `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 1212 of file `specfun.h`.

**8.2.2.57** `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 1222 of file `specfun.h`.

**8.2.2.58** `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 1288 of file `specfun.h`.

**8.2.2.59** `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 1256 of file `specfun.h`.

**8.2.2.60** `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 1267 of file `specfun.h`.

**8.2.2.61** `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 1332 of file `specfun.h`.

**8.2.2.62** `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 1303 of file `specfun.h`.

**8.2.2.63** `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`  $x \geq 0$ .

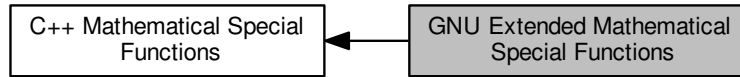
See also

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

Definition at line 1313 of file `specfun.h`.

## 8.3 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](#), [\\_\\_gnu\\_cxx::\\_\\_GLIBCXX\\_JACOBI\\_DN](#) }

### Functions

- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::airy\\_ai](#) (\_Tp \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::airy\\_ai](#) (std::complex<\_Tp> \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_aif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_ail](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::airy\\_bi](#) (\_Tp \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::airy\\_bi](#) (std::complex<\_Tp> \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_bif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_bil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::bernoulli](#) (unsigned int \_\_n)
- float [\\_\\_gnu\\_cxx::bernoullif](#) (unsigned int \_\_n)
- long double [\\_\\_gnu\\_cxx::bernoullil](#) (unsigned int \_\_n)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::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 \_Tps, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tps, \\_Tp>](#) [\\_\\_gnu\\_cxx::bose\\_einstein](#) (\_Tps \_\_s, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::bose\\_einsteinf](#) (float \_\_s, float \_\_x)
- long double [\\_\\_gnu\\_cxx::bose\\_einsteinl](#) (long double \_\_s, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::chebyshev\\_t](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_tf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_tl](#) (unsigned int \_\_n, long double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_u (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_uf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_ul (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_wl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tk > __gnu_cxx::comp_ellint_d (_Tk __k)`
- `float __gnu_cxx::comp_ellint_df (float __k)`
- `long double __gnu_cxx::comp_ellint_dl (long double __k)`
- `float __gnu_cxx::comp_ellint_rf (float __x, float __y)`
- `long double __gnu_cxx::comp_ellint_rf (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf (_Tx __x, _Ty __y)`
- `float __gnu_cxx::comp_ellint_rg (float __x, float __y)`
- `long double __gnu_cxx::comp_ellint_rg (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)`
- `template<typename _Tpa, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_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< _Tp >::type __gnu_cxx::cos_pi (_Tp __x)`
- `float __gnu_cxx::cos_pif (float __x)`

- long double [\\_\\_gnu\\_cxx::cos\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::\\_\\_type \\_\\_gnu\\_cxx::cosh\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::cosh\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::cosh\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp> \\_\\_gnu\\_cxx::coshint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::coshintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::coshintl](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp> \\_\\_gnu\\_cxx::cosint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::cosintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::cosintl](#) (long double \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tpnu, \\_Tp>](#) > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1](#) (\_Tpnu \_\_nu, \_Tp \_\_z)
- template<typename \_Tpnu, typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tpnu, \\_Tp>](#) > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1](#) (std::complex< \_Tpnu > \_\_nu, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1f](#) (float \_\_nu, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1f](#) (std::complex< float > \_\_nu, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1l](#) (long double \_\_nu, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_1l](#) (std::complex< long double > \_\_nu, std::complex< long double > \_\_x)
- template<typename \_Tpnu, typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tpnu, \\_Tp>](#) > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2](#) (\_Tpnu \_\_nu, \_Tp \_\_z)
- template<typename \_Tpnu, typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tpnu, \\_Tp>](#) > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2](#) (std::complex< \_Tpnu > \_\_nu, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2f](#) (float \_\_nu, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2f](#) (std::complex< float > \_\_nu, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2l](#) (long double \_\_nu, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::cyl\\_hankel\\_2l](#) (std::complex< long double > \_\_nu, std::complex< long double > \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp> \\_\\_gnu\\_cxx::dawson](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::dawsonf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::dawsonl](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp> \\_\\_gnu\\_cxx::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\\_betalf](#) (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\\_etalf](#) (long double \_\_s)
- template<typename \_Tp >  
  \_Tp [\\_\\_gnu\\_cxx::dirichlet\\_lambda](#) (\_Tp \_\_s)
- float [\\_\\_gnu\\_cxx::dirichlet\\_lambdaf](#) (float \_\_s)

- long double [\\_\\_gnu\\_cxx::dirichlet\\_lambdal](#) (long double \_\_s)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::double\\_factorial](#) (int \_\_n)
- float [\\_\\_gnu\\_cxx::double\\_factorialf](#) (int \_\_n)
- long double [\\_\\_gnu\\_cxx::double\\_factoriall](#) (int \_\_n)
- template<typename \_Tk, typename \_Tp, typename \_Ta, typename \_Tb >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tk, \_Tp, \_Ta, \_Tb > [\\_\\_gnu\\_cxx::ellint\\_cel](#) (\_Tk \_\_k\_c, \_Tp \_\_p, \_Ta \_\_a, \_Tb \_\_b)
- float [\\_\\_gnu\\_cxx::ellint\\_celf](#) (float \_\_k\_c, float \_\_p, float \_\_a, float \_\_b)
- long double [\\_\\_gnu\\_cxx::ellint\\_cell](#) (long double \_\_k\_c, long double \_\_p, long double \_\_a, long double \_\_b)
- template<typename \_Tk, typename \_Tphi >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tk, \_Tphi > [\\_\\_gnu\\_cxx::ellint\\_d](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::ellint\\_df](#) (float \_\_k, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::ellint\\_dl](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Tk >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Tk > [\\_\\_gnu\\_cxx::ellint\\_el1](#) (\_Tp \_\_x, \_Tk \_\_k\_c)
- float [\\_\\_gnu\\_cxx::ellint\\_el1f](#) (float \_\_x, float \_\_k\_c)
- long double [\\_\\_gnu\\_cxx::ellint\\_el1l](#) (long double \_\_x, long double \_\_k\_c)
- template<typename \_Tp, typename \_Tk, typename \_Ta, typename \_Tb >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Tk, \_Ta, \_Tb > [\\_\\_gnu\\_cxx::ellint\\_el2](#) (\_Tp \_\_x, \_Tk \_\_k\_c, \_Ta \_\_a, \_Tb \_\_b)
- float [\\_\\_gnu\\_cxx::ellint\\_el2f](#) (float \_\_x, float \_\_k\_c, float \_\_a, float \_\_b)
- long double [\\_\\_gnu\\_cxx::ellint\\_el2l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_a, long double \_\_b)
- template<typename \_Tx, typename \_Tk, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tx, \_Tk, \_Tp > [\\_\\_gnu\\_cxx::ellint\\_el3](#) (\_Tx \_\_x, \_Tk \_\_k\_c, \_Tp \_\_p)
- float [\\_\\_gnu\\_cxx::ellint\\_el3f](#) (float \_\_x, float \_\_k\_c, float \_\_p)
- long double [\\_\\_gnu\\_cxx::ellint\\_el3l](#) (long double \_\_x, long double \_\_k\_c, long double \_\_p)
- template<typename \_Tp, typename \_Up >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Up > [\\_\\_gnu\\_cxx::ellint\\_rc](#) (\_Tp \_\_x, \_Up \_\_y)
- float [\\_\\_gnu\\_cxx::ellint\\_rcf](#) (float \_\_x, float \_\_y)
- long double [\\_\\_gnu\\_cxx::ellint\\_rcl](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Up, \_Vp > [\\_\\_gnu\\_cxx::ellint\\_rd](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rdf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rdl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Up, \_Vp > [\\_\\_gnu\\_cxx::ellint\\_rf](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rff](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rfl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Up, \_Vp > [\\_\\_gnu\\_cxx::ellint\\_rg](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [\\_\\_gnu\\_cxx::ellint\\_rgf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [\\_\\_gnu\\_cxx::ellint\\_rgl](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp, typename \_Wp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp, \_Up, \_Vp, \_Wp > [\\_\\_gnu\\_cxx::ellint\\_rj](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z, \_Wp \_\_p)
- float [\\_\\_gnu\\_cxx::ellint\\_rjf](#) (float \_\_x, float \_\_y, float \_\_z, float \_\_p)
- long double [\\_\\_gnu\\_cxx::ellint\\_rjl](#) (long double \_\_x, long double \_\_y, long double \_\_z, long double \_\_p)
- template<typename \_Tp >  
  \_Tp [\\_\\_gnu\\_cxx::ellnome](#) (\_Tp \_\_k)
- float [\\_\\_gnu\\_cxx::ellnomef](#) (float \_\_k)
- long double [\\_\\_gnu\\_cxx::ellnomel](#) (long double \_\_k)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::expint (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::expintf (unsigned int __n, float __x)`
- `long double __gnu_cxx::expintl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_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 _Tps, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::fermi_dirac (_Tps __s, _Tp __x)`
- `float __gnu_cxx::fermi_diracf (float __s, float __x)`
- `long double __gnu_cxx::fermi_diracl (long double __s, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::fresnel_c (_Tp __x)`
- `float __gnu_cxx::fresnel_cf (float __x)`
- `long double __gnu_cxx::fresnel_cl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::fresnel_s (_Tp __x)`
- `float __gnu_cxx::fresnel_sf (float __x)`
- `long double __gnu_cxx::fresnel_sl (long double __x)`
- `template<typename _Talpha, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)`
- `float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)`
- `long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)`
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::heuman_lambdaf (float __k, float __phi)`
- `long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)`
- `template<typename _Tp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta (_Tp __s, _Up __a)`
- `template<typename _Tp, typename _Up >`  
`std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_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_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetacf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetacl (long double __a, long double __b, long double __x)`
- `float __gnu_cxx::ibetaf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`



- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_cn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_cnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_dn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_dnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_sn ( _Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_snf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta ( _Tk __k, _Tphi __phi)`
- `float __gnu_cxx::jacobi_zetaf (float __k, float __phi)`
- `long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)`
- `float __gnu_cxx::jacobi_f (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobi_l (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)`
- `float __gnu_cxx::lbincoefficient (unsigned int __n, unsigned int __k)`
- `long double __gnu_cxx::lbincoefficientl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_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_fp_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_fp_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 _Ta >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::lgamma (std::complex< _Ta > __a)`
- `std::complex< float > __gnu_cxx::lgammaf (std::complex< float > __a)`
- `std::complex< long double > __gnu_cxx::lgammal (std::complex< long double > __a)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::logint ( _Tp __x)`
- `float __gnu_cxx::logintf (float __x)`
- `long double __gnu_cxx::logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Tn > __gnu_cxx::lpochhammer ( _Tp __a, _Tn __n)`
- `template<typename _Tp, typename _Tn >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Tn > __gnu_cxx::lpochhammer_lower ( _Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_lowerf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_lowerl (long double __a, long double __n)`
- `float __gnu_cxx::lpochhammerf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammerl (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`  
`__gnu_cxx::__promote_fp_t< _Tph, _Tpa > __gnu_cxx::owens_t ( _Tph __h, _Tpa __a)`

- float [\\_\\_gnu\\_cxx::owens\\_tf](#) (float \_\_h, float \_\_a)
- long double [\\_\\_gnu\\_cxx::owens\\_tl](#) (long double \_\_h, long double \_\_a)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Ta, \\_Tp>](#) [\\_\\_gnu\\_cxx::pgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::pgammaf](#) (float \_\_a, float \_\_x)
- long double [\\_\\_gnu\\_cxx::pgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Tn>](#) [\\_\\_gnu\\_cxx::pochhammer](#) (\_Tp \_\_a, \_Tn \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Tn>](#) [\\_\\_gnu\\_cxx::pochhammer\\_lower](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_lowerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_lowerl](#) (long double \_\_a, long double \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammerl](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Wp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Wp>](#) [\\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, \_Wp \_\_w)
- template<typename \_Tp, typename \_Wp >  
[std::complex<\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Wp>>](#) [\\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, [std::complex<\\_Tp>](#) \_\_w)
- float [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, float \_\_w)
- [std::complex<float>](#) [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, [std::complex<float>](#) \_\_w)
- long double [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, long double \_\_w)
- [std::complex<long double>](#) [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, [std::complex<long double>](#) \_\_w)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::psi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::psif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::psil](#) (long double \_\_x)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Ta, \\_Tp>](#) [\\_\\_gnu\\_cxx::qgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::qgammaf](#) (float \_\_a, float \_\_x)
- long double [\\_\\_gnu\\_cxx::qgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::radpoly](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_rho)
- float [\\_\\_gnu\\_cxx::radpolyf](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_rho)
- long double [\\_\\_gnu\\_cxx::radpolyl](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_rho)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote<\\_Tp>::type](#) [\\_\\_gnu\\_cxx::sin\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sin\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sin\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::sinc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::sinc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sincf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sincl](#) (long double \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t<double>](#) [\\_\\_gnu\\_cxx::sincos](#) (double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_sincos\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::sincos](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_sincos\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::sincos\\_pi](#) (\_Tp \_\_x)

- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t](#)< float > [\\_\\_gnu\\_cxx::sincos\\_pif](#) (float \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t](#)< long double > [\\_\\_gnu\\_cxx::sincos\\_pil](#) (long double \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t](#)< float > [\\_\\_gnu\\_cxx::sincosf](#) (float \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t](#)< long double > [\\_\\_gnu\\_cxx::sincosl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote](#)< \_Tp >::\_\_type [\\_\\_gnu\\_cxx::sinh\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinh\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinh\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sinhc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sinhc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinhc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sinhcf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhcl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sinhint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinhintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhintl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sinint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinintl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sph\\_bessel\\_i](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sph\\_bessel\\_if](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::sph\\_bessel\\_il](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [\\_\\_gnu\\_cxx::sph\\_bessel\\_k](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sph\\_bessel\\_kf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::sph\\_bessel\\_kl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1](#) (unsigned int \_\_n, \_Tp \_\_z)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1](#) (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1f](#) (unsigned int \_\_n, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1f](#) (unsigned int \_\_n, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1l](#) (unsigned int \_\_n, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_1l](#) (unsigned int \_\_n, std::complex< long double > \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2](#) (unsigned int \_\_n, \_Tp \_\_z)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2](#) (unsigned int \_\_n, std::complex< \_Tp > \_\_x)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2f](#) (unsigned int \_\_n, float \_\_z)
- std::complex< float > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2f](#) (unsigned int \_\_n, std::complex< float > \_\_x)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2l](#) (unsigned int \_\_n, long double \_\_z)
- std::complex< long double > [\\_\\_gnu\\_cxx::sph\\_hankel\\_2l](#) (unsigned int \_\_n, std::complex< long double > \_\_x)

- `template<typename _Ttheta, typename _Tphi >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic` (unsigned int \_\_l,  
int \_\_m, \_Ttheta \_\_theta, \_Tphi \_\_phi)
- `std::complex< float > __gnu_cxx::sph_harmonicf` (unsigned int \_\_l, int \_\_m, float \_\_theta, float \_\_phi)
- `std::complex< long double > __gnu_cxx::sph_harmonicl` (unsigned int \_\_l, int \_\_m, long double \_\_theta, long  
double \_\_phi)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type __gnu_cxx::tan_pi` (\_Tp \_\_x)
- `float __gnu_cxx::tan_pif` (float \_\_x)
- `long double __gnu_cxx::tan_pil` (long double \_\_x)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type __gnu_cxx::tanh_pi` (\_Tp \_\_x)
- `float __gnu_cxx::tanh_pif` (float \_\_x)
- `long double __gnu_cxx::tanh_pil` (long double \_\_x)
- `template<typename _Ta >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma` (std::complex< \_Ta > \_\_a)
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma` (\_Ta \_\_a, \_Tp \_\_x)
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower` (\_Ta \_\_a, \_Tp \_\_x)
- `float __gnu_cxx::tgamma_lowerf` (float \_\_a, float \_\_x)
- `long double __gnu_cxx::tgamma_lowerl` (long double \_\_a, long double \_\_x)
- `std::complex< float > __gnu_cxx::tgammaf` (std::complex< float > \_\_a)
- `float __gnu_cxx::tgammaf` (float \_\_a, float \_\_x)
- `std::complex< long double > __gnu_cxx::tgammal` (std::complex< long double > \_\_a)
- `long double __gnu_cxx::tgammal` (long double \_\_a, long double \_\_x)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_1` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float __gnu_cxx::theta_1f` (float \_\_nu, float \_\_x)
- `long double __gnu_cxx::theta_1l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_2` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float __gnu_cxx::theta_2f` (float \_\_nu, float \_\_x)
- `long double __gnu_cxx::theta_2l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_3` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float __gnu_cxx::theta_3f` (float \_\_nu, float \_\_x)
- `long double __gnu_cxx::theta_3l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_4` (\_Tpnu \_\_nu, \_Tp \_\_x)
- `float __gnu_cxx::theta_4f` (float \_\_nu, float \_\_x)
- `long double __gnu_cxx::theta_4l` (long double \_\_nu, long double \_\_x)
- `template<typename _Tp_k, typename _Tp >`  
`__gnu_cxx::__promote_fp_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_fp_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_fp_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 `_Tpk`, typename `_Tp` >  
`__gnu_cxx::__promote_fp_t<_Tpk, _Tp>` `__gnu_cxx::theta_s` (`_Tpk` `__k`, `_Tp` `__x`)
- float `__gnu_cxx::theta_sf` (float `__k`, float `__x`)
- long double `__gnu_cxx::theta_sl` (long double `__k`, long double `__x`)
- template<typename `_Trho`, typename `_Tphi` >  
`__gnu_cxx::__promote_fp_t<_Trho, _Tphi>` `__gnu_cxx::zernike` (unsigned int `__n`, int `__m`, `_Trho` `__rho`, `_Tphi` `__phi`)
- float `__gnu_cxx::zernikef` (unsigned int `__n`, int `__m`, float `__rho`, float `__phi`)
- long double `__gnu_cxx::zernikel` (unsigned int `__n`, int `__m`, long double `__rho`, long double `__phi`)

### 8.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

### 8.3.2 Enumeration Type Documentation

#### 8.3.2.1 anonymous enum

Enumerator

`_GLIBCXX_JACOBI_SN`  
`_GLIBCXX_JACOBI_CN`  
`_GLIBCXX_JACOBI_DN`

Definition at line 1782 of file `specfun.h`.

### 8.3.3 Function Documentation

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

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

The Airy function is defined by:

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

Template Parameters

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

## Parameters

<a href="#"><math>\leftrightarrow</math></a>	The argument
<code>__x</code>	

Definition at line 2764 of file specfun.h.

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

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

The Airy function is defined by:

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

## Template Parameters

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

## Parameters

<a href="#"><math>\leftrightarrow</math></a>	The complex argument
<code>__x</code>	

Definition at line 2786 of file specfun.h.

```
8.3.3.3 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 2729 of file specfun.h.

```
8.3.3.4 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 2743 of file specfun.h.

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

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

The Airy function is defined by:

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

#### Template Parameters

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

#### Parameters

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

Definition at line 2836 of file specfun.h.

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

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

The Airy function is defined by:

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

#### Template Parameters

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

#### Parameters

<code>↵ _x</code>	The complex argument
-----------------------	----------------------

Definition at line 2859 of file specfun.h.

8.3.3.7 `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 2800 of file specfun.h.

**8.3.3.8** `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 2814 of file specfun.h.

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

Return the Bernoulli number of integer order  $n$ .

The Bernoulli numbers are defined by

Parameters

<code>__n</code>	The order.
------------------	------------

Definition at line 3932 of file specfun.h.

**8.3.3.10** `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 3907 of file specfun.h.

**8.3.3.11** `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 3917 of file specfun.h.



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

Definition at line 3872 of file specfun.h.

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

Definition at line 3860 of file specfun.h.

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

Definition at line 3864 of file specfun.h.

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

Definition at line 5448 of file specfun.h.

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

Definition at line 5439 of file specfun.h.

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

Definition at line 5443 of file specfun.h.

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

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

The Chebyshev polynomial of the first kind is defined by:

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

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

Template Parameters

<code>_Tp</code>	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 1983 of file specfun.h.

**8.3.3.19** `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 1954 of file specfun.h.

**8.3.3.20** `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 1964 of file specfun.h.

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

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

The Chebyshev polynomial of the second kind is defined by:

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

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

## Template Parameters

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

## Parameters

$\leftrightarrow$ __n	The non-negative integral order
$\leftrightarrow$ __x	The real argument $-1 \leq x \leq +1$

Definition at line 2027 of file specfun.h.

**8.3.3.22** `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 1998 of file specfun.h.

**8.3.3.23** `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 2008 of file specfun.h.

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

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

The Chebyshev polynomial of the third kind is defined by:

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

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

## 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 2072 of file specfun.h.

**8.3.3.25** `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 2042 of file specfun.h.

**8.3.3.26** `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 2052 of file specfun.h.

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

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

## Template Parameters

<code>_Tp</code>	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 2117 of file specfun.h.

**8.3.3.28** `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 2087 of file specfun.h.

**8.3.3.29** `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 2097 of file specfun.h.

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

Return the Clausen function  $Cl_n(w)$  of integer order  $m$  and real argument  $w$ .

The Clausen function is defined by

$$Cl_n(w) = S_n(w) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^n} \text{ for even } m = C_n(w) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^n} \text{ for odd } m$$

## Template Parameters

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

## Parameters

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

Definition at line 4943 of file specfun.h.

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

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

The Clausen function is defined by

$$Cl_n(w) = S_n(w) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^n} \text{ for even } m = C_n(w) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^n} \text{ for odd } m$$

## Template Parameters

__Tp	The real type of the complex components
------	---

## Parameters

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

Definition at line 4987 of file specfun.h.

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

Return the Clausen cosine function  $C_n(w)$  of order  $m$  and real argument  $w$ .

The Clausen cosine function is defined by

$$C_n(w) = \sum_{k=1}^{\infty} \frac{\cos(kx)}{k^n}$$

## Template Parameters

__Tp	The real type of the argument
------	-------------------------------

## Parameters

$\leftrightarrow$ <code>_m</code>	The unsigned integer order
$\leftrightarrow$ <code>_w</code>	The real argument

Definition at line 4899 of file specfun.h.

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

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

See also

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

Definition at line 4871 of file specfun.h.

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

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

See also

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

Definition at line 4881 of file specfun.h.

**8.3.3.35** `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::clausen_s ( unsigned int __m, _Tp __w ) [inline]`

Return the Clausen sine function  $S_n(w)$  of order  $m$  and real argument  $w$ .

The Clausen sine function is defined by

$$S_n(w) = \sum_{k=1}^{\infty} \frac{\sin(kx)}{k^n}$$

## Template Parameters

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

## Parameters

$\_m$	The unsigned integer order
$\_w$	The real argument

Definition at line 4856 of file specfun.h.

8.3.3.36 `float __gnu_cxx::clausen_sf ( unsigned int  $\_m$ , float  $\_w$  ) [inline]`

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

## See also

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

Definition at line 4828 of file specfun.h.

8.3.3.37 `long double __gnu_cxx::clausen_sl ( unsigned int  $\_m$ , long double  $\_w$  ) [inline]`

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

## See also

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

Definition at line 4838 of file specfun.h.

8.3.3.38 `float __gnu_cxx::clausenf ( unsigned int  $\_m$ , float  $\_w$  ) [inline]`

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

## See also

[clausen](#) for details.

Definition at line 4914 of file specfun.h.



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

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

See also

[clausen](#) for details.

Definition at line 4958 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 4924 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 4968 of file `specfun.h`.

8.3.3.42 `template<typename _Tk> __gnu_cxx::__promote_fp_t<_Tk> __gnu_cxx::comp_ellint_d ( _Tk __k ) [inline]`

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

The complete Legendre elliptic integral  $D$  is defined by

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

Template Parameters

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

## Parameters

$\leftrightarrow$ __k	The modulus $-1 \leq \text{__k} \leq +1$
--------------------------	--

Definition at line 4134 of file specfun.h.

8.3.3.43 `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 4107 of file specfun.h.

8.3.3.44 `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 4117 of file specfun.h.

8.3.3.45 `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 3054 of file specfun.h.

8.3.3.46 `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 3064 of file specfun.h.

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

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

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

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

## Parameters

$\leftrightarrow$ _x	The first argument.
$\leftrightarrow$ _y	The second argument.

Definition at line 3082 of file specfun.h.

**8.3.3.48** `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 3287 of file specfun.h.

**8.3.3.49** `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 3296 of file specfun.h.

**8.3.3.50** `template<typename _Tx, typename _Ty> __gnu_cxx::__promote_fp_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rg ( _Tx _x, _Ty _y ) [inline]`

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

The complete Carlson elliptic function is defined by:

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

## Parameters

$\leftrightarrow$ _x	The first argument.
$\leftrightarrow$ _y	The second argument.

Definition at line 3315 of file specfun.h.

**8.3.3.51** `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

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

Definition at line 1397 of file specfun.h.

**8.3.3.52** `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

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

Definition at line 1493 of file specfun.h.

8.3.3.53 `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 1464 of file `specfun.h`.

8.3.3.54 `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 1474 of file `specfun.h`.

8.3.3.55 `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 1365 of file `specfun.h`.

8.3.3.56 `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 1376 of file `specfun.h`.

8.3.3.57 `template<typename _Tp> __gnu_cxx::__promote<_Tp>::__type __gnu_cxx::cos_pi ( _Tp __x ) [inline]`

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

The reperiodized cosine function is defined by:

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

## Template Parameters

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

## Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 5574 of file `specfun.h`.

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

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

See also

[cos\\_pi](#) for more details.

Definition at line 5547 of file `specfun.h`.

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

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

See also

[cos\\_pi](#) for more details.

Definition at line 5557 of file `specfun.h`.

**8.3.3.60** `template<typename _Tp> __gnu_cxx::__promote<_Tp>::__type __gnu_cxx::cosh_pi ( _Tp __x ) [inline]`

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

The reperiodized hyperbolic cosine function is defined by:

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

## Template Parameters

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

## Parameters

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

Definition at line 5616 of file specfun.h.

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

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

## See also

[cosh\\_pi](#) for more details.

Definition at line 5589 of file specfun.h.

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

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

## See also

[cosh\\_pi](#) for more details.

Definition at line 5599 of file specfun.h.

**8.3.3.63** `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::coshint ( _Tp __x ) [inline]`

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

The hyperbolic cosine integral is defined by

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

## Template Parameters

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

## Parameters

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

Definition at line 1775 of file specfun.h.

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

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

See also

[coshint](#) for details.

Definition at line 1747 of file specfun.h.

8.3.3.65 `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 1757 of file specfun.h.

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

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

The cosine integral is defined by

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

## Parameters

<code>_↔</code>	The real upper integration limit
<code>_x</code>	

Definition at line 1692 of file specfun.h.



8.3.3.67 `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 1666 of file `specfun.h`.

8.3.3.68 `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 1676 of file `specfun.h`.

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

Return the cylindrical Hankel function of the first kind  $H_n^{(1)}(x)$  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 2466 of file specfun.h.

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

Return the complex cylindrical Hankel function of the first kind  $H_\nu^{(1)}(x)$  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 4411 of file specfun.h.

```
8.3.3.71  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 2433 of file specfun.h.

```
8.3.3.72  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 4380 of file specfun.h.

8.3.3.73 `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 2444 of file specfun.h.

8.3.3.74 `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 4391 of file specfun.h.

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

Return the cylindrical Hankel function of the second kind  $H_n^{(2)}(x)$  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 2515 of file specfun.h.

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

Return the complex cylindrical Hankel function of the second kind  $H_\nu^{(2)}(x)$  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 4458 of file specfun.h.

```
8.3.3.77  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 2482 of file specfun.h.

```
8.3.3.78  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 4427 of file specfun.h.

8.3.3.79 `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 2493 of file `specfun.h`.

8.3.3.80 `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 4438 of file `specfun.h`.

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

<code>__x</code>	The argument $-inf < x < inf$ .
------------------	---------------------------------

Definition at line 3637 of file `specfun.h`.

8.3.3.82 `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 3609 of file specfun.h.

**8.3.3.83** `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 3618 of file specfun.h.

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

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

The dilogarithm is defined by:

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

Parameters

<code>__x</code>	The argument.
------------------	---------------

Definition at line 3039 of file specfun.h.

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

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

See also

[dilog](#) for details.

Definition at line 3013 of file specfun.h.

**8.3.3.86** `long double __gnu_cxx::dilog( long double __x )` `[inline]`

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

See also

[dilog](#) for details.

Definition at line 3023 of file `specfun.h`.

**8.3.3.87** `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

<code>↔</code>	
<code>__s</code>	

Definition at line 4774 of file `specfun.h`.

**8.3.3.88** `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 4745 of file `specfun.h`.

**8.3.3.89** `long double __gnu_cxx::dirichlet_betad( long double __s )` `[inline]`

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

See also

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

Definition at line 4754 of file `specfun.h`.

**8.3.3.90** `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

$\longleftrightarrow$	
$s$	

Definition at line 4731 of file specfun.h.

**8.3.3.91** `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 4701 of file specfun.h.

**8.3.3.92** `long double __gnu_cxx::dirichlet_etalf ( long double __s ) [inline]`

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

See also

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

Definition at line 4710 of file specfun.h.

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

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

The Dirichlet lambda function is defined by

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



## Parameters

$\_ \leftrightarrow$	
$\_s$	

Definition at line 4813 of file specfun.h.

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

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

See also

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

Definition at line 4788 of file specfun.h.

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

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

See also

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

Definition at line 4797 of file specfun.h.

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

Definition at line 3809 of file specfun.h.

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

Definition at line 3797 of file specfun.h.

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

Definition at line 3801 of file specfun.h.

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

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

The Bulirsch complete elliptic integral is defined by

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

## Parameters

<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 4364 of file specfun.h.

8.3.3.100 `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 4332 of file specfun.h.

8.3.3.101 `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 4341 of file specfun.h.

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

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

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 4177 of file specfun.h.

8.3.3.103 `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 4149 of file specfun.h.

8.3.3.104 `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 4159 of file specfun.h.

8.3.3.105 `template<typename _Tp, typename _Tk> __gnu_cxx::__promote_fp_t<_Tp, _Tk> __gnu_cxx::ellint_el1 ( _Tp __x, _Tk __k_c ) [inline]`

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

The Bulirsch elliptic integral of the first kind is defined by

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

## Parameters

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

Definition at line 4223 of file specfun.h.

8.3.3.106 `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 4193 of file specfun.h.

8.3.3.107 `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 4204 of file specfun.h.

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

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

The Bulirsch elliptic integral of the second kind is defined by

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

#### Parameters

<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 4269 of file specfun.h.

8.3.3.109 `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 4238 of file specfun.h.

8.3.3.110 `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 4248 of file specfun.h.

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

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

The Bulirsch elliptic integral of the third kind is defined by

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

#### Parameters

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

Definition at line 4316 of file specfun.h.

8.3.3.112 `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 4285 of file `specfun.h`.

8.3.3.113 `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 4296 of file `specfun.h`.

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

#### Parameters

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

Definition at line 3174 of file specfun.h.

8.3.3.115 `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 3140 of file specfun.h.

8.3.3.116 `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 3149 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

#### Parameters

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

Definition at line 3273 of file specfun.h.

8.3.3.118 `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 3237 of file specfun.h.

8.3.3.119 `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 3246 of file specfun.h.

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

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

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

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

#### Parameters

<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 3126 of file specfun.h.



8.3.3.121 `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 3097 of file `specfun.h`.

8.3.3.122 `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 3107 of file `specfun.h`.

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

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

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

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

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 3364 of file specfun.h.

8.3.3.124 `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 3329 of file specfun.h.

8.3.3.125 `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 3338 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

#### Parameters

<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 3223 of file specfun.h.

8.3.3.127 `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 3188 of file specfun.h.

8.3.3.128 `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 3197 of file specfun.h.

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

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

The elliptic nome function is defined by

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

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

#### Template Parameters

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

#### Parameters

<code>↵ _k</code>	The modulus $-1 \leq k \leq +1$
-----------------------	---------------------------------

Definition at line 5201 of file specfun.h.

8.3.3.130 `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 5174 of file specfun.h.

8.3.3.131 `long double __gnu_cxx::ellnomel ( long double __k ) [inline]`

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

See also

[ellnome](#) for details.

Definition at line 5184 of file specfun.h.

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

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

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

In particular

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

#### Template Parameters

<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 3683 of file specfun.h.

8.3.3.133 `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 3652 of file `specfun.h`.

8.3.3.134 `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 3662 of file `specfun.h`.

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

Definition at line 3788 of file `specfun.h`.

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

Definition at line 3776 of file `specfun.h`.

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

Definition at line 3780 of file `specfun.h`.

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

Definition at line 5430 of file `specfun.h`.

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

Definition at line 5421 of file `specfun.h`.

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

Definition at line 5425 of file `specfun.h`.

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

Return the Fresnel cosine integral of argument  $x$ .

The Fresnel cosine integral is defined by

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

## Parameters

$\leftrightarrow$	The argument
$x$	

Definition at line 3595 of file specfun.h.

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

Definition at line 3576 of file specfun.h.

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

Definition at line 3580 of file specfun.h.

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

Return the Fresnel sine integral of argument  $x$ .

The Fresnel sine integral is defined by

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

## Parameters

$\leftrightarrow$	The argument
$x$	

Definition at line 3567 of file specfun.h.

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

Definition at line 3548 of file specfun.h.

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

Definition at line 3552 of file specfun.h.

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

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

The Gegenbauer 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 2225 of file `specfun.h`.

8.3.3.148 `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 2192 of file `specfun.h`.

8.3.3.149 `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 2203 of file `specfun.h`.

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

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

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 4092 of file specfun.h.

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

Definition at line 4066 of file specfun.h.

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

Definition at line 4070 of file specfun.h.

8.3.3.153 `template<typename _Tp, typename _Up> __gnu_cxx::__promote_fp_t<_Tp, _Up> __gnu_cxx::hurwitz_zeta ( _Tp __s, _Up __a ) [inline]`

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

The the Hurwitz zeta function is defined by

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



## Parameters

$\leftrightarrow$ _s	The argument
$\leftrightarrow$ _a	The parameter

Definition at line 3405 of file specfun.h.

```
8.3.3.154 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 3419 of file specfun.h.

```
8.3.3.155 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 3379 of file specfun.h.

```
8.3.3.156 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 3389 of file specfun.h.

```
8.3.3.157 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

$\leftrightarrow$ _a	The first numeratorial parameter
$\leftrightarrow$ _b	The second numeratorial parameter
$\leftrightarrow$ _c	The denominatorial parameter
$\leftrightarrow$ _x	The argument

Definition at line 1446 of file specfun.h.

8.3.3.158 `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 1413 of file specfun.h.

8.3.3.159 `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 1424 of file specfun.h.

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

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

The regularized incomplete beta function is defined by

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

where

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

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

## Parameters

$\leftrightarrow$ _a	The first parameter
$\leftrightarrow$ _b	The second parameter
$\leftrightarrow$ _x	The argument

Definition at line 3508 of file specfun.h.

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

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

The regularized complementary incomplete beta function is defined by

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

## Parameters

$\leftrightarrow$ _a	The parameter
$\leftrightarrow$ _b	The parameter
$\leftrightarrow$ _x	The argument

Definition at line 3539 of file specfun.h.

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

Definition at line 3517 of file specfun.h.

References `__gnu_cxx::ibetaf()`.

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

Definition at line 3521 of file specfun.h.

References `__gnu_cxx::ibetal()`.

8.3.3.164 `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 3474 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetaf()`.

8.3.3.165 `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 3484 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetac()`.

8.3.3.166 `template<typename _Talpha, typename _Tbeta, typename _Tp> __gnu_cxx::__promote_fp_t<_Talpha, _Tbeta, _Tp> __gnu_cxx::jacobi ( unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x ) [inline]`

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

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

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

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 2177 of file `specfun.h`.

References `std::__detail::__beta()`.

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

Return the Jacobi elliptic  $cn(k, u)$  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 1886 of file `specfun.h`.

```
8.3.3.168 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 1850 of file `specfun.h`.

```
8.3.3.169 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 1863 of file `specfun.h`.

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

Return the Jacobi elliptic  $dn(k, u)$  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 1938 of file `specfun.h`.

8.3.3.171 `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 1902 of file `specfun.h`.

8.3.3.172 `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 1915 of file `specfun.h`.

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

Return the Jacobi elliptic  $sn(k, u)$  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 1834 of file `specfun.h`.

8.3.3.174 `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 1798 of file `specfun.h`.

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

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

#### See also

[jacobi\\_sn](#) for details.

Definition at line 1811 of file `specfun.h`.

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

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

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 4057 of file `specfun.h`.

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

Definition at line 4032 of file `specfun.h`.

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

Definition at line 4036 of file `specfun.h`.

8.3.3.179 `float __gnu_cxx::jacobi ( 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 2133 of file `specfun.h`.

References `std::__detail::__beta()`.



8.3.3.180 `long double __gnu_cxx::jacobi ( 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 2144 of file `specfun.h`.

References `std::__detail::__beta()`.

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

Definition at line 3893 of file `specfun.h`.

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

Definition at line 3881 of file `specfun.h`.

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

Definition at line 3885 of file `specfun.h`.

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

Definition at line 3851 of file `specfun.h`.

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

Definition at line 3839 of file `specfun.h`.

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

Definition at line 3843 of file `specfun.h`.

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

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

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

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

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

## Template Parameters

<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 3981 of file `specfun.h`.

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

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

See also

[legendre\\_q](#) for details.

Definition at line 3947 of file `specfun.h`.

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

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

See also

[legendre\\_q](#) for details.

Definition at line 3957 of file `specfun.h`.

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

Definition at line 3830 of file `specfun.h`.

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

Definition at line 3818 of file specfun.h.

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

Definition at line 3822 of file specfun.h.

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

Return the logarithm of the gamma function for complex argument.

Definition at line 2892 of file specfun.h.

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

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

See also

[lgamma](#) for details.

Definition at line 2874 of file specfun.h.

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

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

See also

[lgamma](#) for details.

Definition at line 2884 of file specfun.h.

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

Return the logarithmic integral of argument  $x$ .

The logarithmic integral is defined by

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

## Parameters

$\_x$	The real upper integration limit
-------	----------------------------------

Definition at line 1613 of file specfun.h.

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

Return the logarithmic integral of argument  $x$ .

## See also

[logint](#) for details.

Definition at line 1589 of file specfun.h.

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

Return the logarithmic integral of argument  $x$ .

## See also

[logint](#) for details.

Definition at line 1598 of file specfun.h.

8.3.3.199 `template<typename _Tp, typename _Tn> __gnu_cxx::__promote_fp_t<_Tp, _Tn> __gnu_cxx::lpochhammer ( _Tp __a, _Tn __n ) [inline]`

Definition at line 3704 of file specfun.h.

8.3.3.200 `template<typename _Tp, typename _Tn> __gnu_cxx::__promote_fp_t<_Tp, _Tn> __gnu_cxx::lpochhammer_lower ( _Tp __a, _Tn __n ) [inline]`

Definition at line 3725 of file specfun.h.

8.3.3.201 `float __gnu_cxx::lpochhammer_lowerf ( float __a, float __n ) [inline]`

Definition at line 3713 of file specfun.h.

8.3.3.202 `long double __gnu_cxx::lpochhammer_lowerl ( long double __a, long double __n ) [inline]`

Definition at line 3717 of file specfun.h.

8.3.3.203 `float __gnu_cxx::lpochhammerf ( float __a, float __n ) [inline]`

Definition at line 3692 of file specfun.h.

8.3.3.204 `long double __gnu_cxx::lpochhammerl ( long double __a, long double __n ) [inline]`

Definition at line 3696 of file specfun.h.

8.3.3.205 `template<typename _Tph, typename _Tpa> __gnu_cxx::__promote_fp_t<_Tph, _Tpa> __gnu_cxx::owens_t ( _Tph __h, _Tpa __a ) [inline]`

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

The Owens T function is defined by

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

#### Parameters

$\leftrightarrow$ __h	The shape factor
$\leftrightarrow$ __a	The integration limit

Definition at line 5412 of file specfun.h.

8.3.3.206 `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 5384 of file specfun.h.

8.3.3.207 `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 5394 of file `specfun.h`.

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

Definition at line 4002 of file `specfun.h`.

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

Definition at line 3990 of file `specfun.h`.

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

Definition at line 3994 of file `specfun.h`.

8.3.3.211 `template<typename _Tp, typename _Tn> __gnu_cxx::__promote_fp_t<_Tp, _Tn> __gnu_cxx::pochhammer ( _Tp __a, _Tn __n ) [inline]`

Definition at line 3746 of file `specfun.h`.

8.3.3.212 `template<typename _Tp, typename _Tn> __gnu_cxx::__promote_fp_t<_Tp, _Tn> __gnu_cxx::pochhammer_lower ( _Tp __a, _Tn __n ) [inline]`

Definition at line 3767 of file `specfun.h`.

8.3.3.213 `float __gnu_cxx::pochhammer_lowerf ( float __a, float __n ) [inline]`

Definition at line 3755 of file `specfun.h`.

8.3.3.214 `long double __gnu_cxx::pochhammer_lowerl ( long double __a, long double __n ) [inline]`

Definition at line 3759 of file `specfun.h`.

8.3.3.215 `float __gnu_cxx::pochhammerf ( float __a, float __n ) [inline]`

Definition at line 3734 of file specfun.h.

8.3.3.216 `long double __gnu_cxx::pochhammerl ( long double __a, long double __n ) [inline]`

Definition at line 3738 of file specfun.h.

8.3.3.217 `template<typename _Tp, typename _Wp> __gnu_cxx::__promote_fp_t<_Tp, _Wp> __gnu_cxx::polylog ( _Tp __s, _Wp __w ) [inline]`

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

The polylogarithm function is defined by

#### Parameters

$\leftarrow$ __s	
$\leftarrow$ __w	

Definition at line 4647 of file specfun.h.

8.3.3.218 `template<typename _Tp, typename _Wp> std::complex<__gnu_cxx::__promote_fp_t<_Tp, _Wp>> __gnu_cxx::polylog ( _Tp __s, std::complex<_Tp> __w ) [inline]`

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

The polylogarithm function is defined by

#### Parameters

$\leftarrow$ __s	
$\leftarrow$ __w	

Definition at line 4687 of file specfun.h.

8.3.3.219 `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 4620 of file specfun.h.

8.3.3.220 `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 4660 of file specfun.h.

8.3.3.221 `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 4630 of file specfun.h.

8.3.3.222 `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 4670 of file specfun.h.

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

Return the psi or digamma function of argument  $x$ .

The the psi or digamma function is defined by

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



## Parameters

$\leftrightarrow$	The parameter
$x$	

Definition at line 3459 of file specfun.h.

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

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

See also

[psi](#) for details.

Definition at line 3433 of file specfun.h.

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

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

See also

[psi](#) for details.

Definition at line 3443 of file specfun.h.

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

Definition at line 4023 of file specfun.h.

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

Definition at line 4011 of file specfun.h.

**8.3.3.228** `long double __gnu_cxx::qgamma ( long double __a, long double __x ) [inline]`

Definition at line 4015 of file specfun.h.

8.3.3.229 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::radpoly ( unsigned int __n, unsigned int __m, _Tp __rho ) [inline]`

Return the radial polynomial  $R_n^m(\rho)$  for non-negative degree  $n$ , order  $m \leq n$ , 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 2335 of file `specfun.h`.

8.3.3.230 `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 2296 of file `specfun.h`.

References `std::__detail::__poly_radial_jacobi()`.

8.3.3.231 `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 2307 of file specfun.h.

References `std::__detail::__poly_radial_jacobi()`.

8.3.3.232 `template<typename _Tp> __gnu_cxx::__promote<_Tp>::__type __gnu_cxx::sin_pi ( _Tp __x ) [inline]`

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

The reperiodized sine function is defined by:

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

Template Parameters

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

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 5490 of file specfun.h.

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

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

See also

[sin\\_pi](#) for more details.

Definition at line 5463 of file specfun.h.

8.3.3.234 `long double __gnu_cxx::sin_pi ( long double __x ) [inline]`

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

See also

[sin\\_pi](#) for more details.

Definition at line 5473 of file `specfun.h`.

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

Return the sinus cardinal function  $\text{sinc}_\pi(x)$  for real argument `__x`. The sinus cardinal function is defined by:

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

Template Parameters

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

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 1534 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 1575 of file specfun.h.

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

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

See also

[sinc](#) for details.

Definition at line 1549 of file specfun.h.

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

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

See also

[sinc](#) for details.

Definition at line 1559 of file specfun.h.

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

Return the sinus cardinal function  $\text{sinc}_\pi(x)$  for `float` argument `__x`.

See also

[sinc\\_pi](#) for details.

Definition at line 1508 of file specfun.h.

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

Return the sinus cardinal function  $\text{sinc}_\pi(x)$  for `long double` argument `__x`.

See also

[sinc\\_pi](#) for details.

Definition at line 1518 of file specfun.h.

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

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

See also

[sincos](#) for details.

Definition at line 5728 of file `specfun.h`.

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

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

$$\text{sincos}(x) = \sin(x), \cos(x)$$

Definition at line 5739 of file `specfun.h`.

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

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

$$\text{sincos}_\pi(x) = \sin(\pi x), \cos(\pi x)$$

Definition at line 5770 of file `specfun.h`.

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

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

See also

[sincos\\_pi](#) for details.

Definition at line 5748 of file `specfun.h`.

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

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

See also

[sincos\\_pi](#) for details.

Definition at line 5758 of file `specfun.h`.

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

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

Definition at line 5710 of file `specfun.h`.

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

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

See also

[sincos](#) for details.

Definition at line 5719 of file `specfun.h`.

8.3.3.248 `template<typename _Tp> __gnu_cxx::__promote<_Tp>::__type __gnu_cxx::sinh_pi ( _Tp __x ) [inline]`

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

The reperiodized hyperbolic sine function is defined by:

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

#### Template Parameters

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

#### Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 5532 of file `specfun.h`.

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

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

See also

[sinh\\_pi](#) for more details.

Definition at line 5505 of file `specfun.h`.

8.3.3.250 `long double __gnu_cxx::sinh_pi ( long double __x ) [inline]`

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

See also

[sinh\\_pi](#) for more details.

Definition at line 5515 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 2417 of file `specfun.h`.

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

Return the hyperbolic sinus cardinal function  $\operatorname{sinhc}_{\pi}(x)$  for real argument `__x`. The sinus cardinal function is defined by:

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

Template Parameters

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

Parameters

<code>__x</code>	The argument
------------------	--------------



Definition at line 2376 of file specfun.h.

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

Return the hyperbolic sinus cardinal function  $\operatorname{sinhc}_{\pi}(x)$  for `float` argument `__x`.

See also

[sinhc\\_pi](#) for details.

Definition at line 2350 of file specfun.h.

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

Return the hyperbolic sinus cardinal function  $\operatorname{sinhc}_{\pi}(x)$  for `long double` argument `__x`.

See also

[sinhc\\_pi](#) for details.

Definition at line 2360 of file specfun.h.

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

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

See also

[sinhc](#) for details.

Definition at line 2391 of file specfun.h.

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

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

See also

[sinhc](#) for details.

Definition at line 2401 of file specfun.h.

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

Return the hyperbolic sine integral  $\operatorname{Shi}(x)$  of real argument  $x$ .

The hyperbolic sine integral is defined by

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

## Template Parameters

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

## Parameters

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

Definition at line 1733 of file `specfun.h`.

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

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

See also

[sinhint](#) for details.

Definition at line 1706 of file `specfun.h`.

**8.3.3.259** `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 1716 of file `specfun.h`.

**8.3.3.260** `template<typename _Tp> __gnu_cxx::_promote_fp_t<_Tp> __gnu_cxx::sinint ( _Tp __x ) [inline]`

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

The sine integral is defined by

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

## Parameters

<code>↔ _x</code>	The real upper integration limit
-----------------------	----------------------------------

Definition at line 1652 of file specfun.h.

8.3.3.261 `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 1627 of file specfun.h.

8.3.3.262 `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 1637 of file specfun.h.

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

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

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 2655 of file `specfun.h`.

8.3.3.264 `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 2616 of file `specfun.h`.

8.3.3.265 `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 2631 of file `specfun.h`.

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

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

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

$\leftrightarrow$ __n	The integral order $n \geq 0$
$\leftrightarrow$ __x	The real argument $x \geq 0$

## Exceptions

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

Definition at line 2712 of file `specfun.h`.

8.3.3.267 `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 2673 of file `specfun.h`.

8.3.3.268 `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 2688 of file `specfun.h`.

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

Return the spherical Hankel function of the first kind  $h_n^{(1)}(x)$  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 2558 of file `specfun.h`.

```
8.3.3.270  template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_1 ( unsigned
            int __n, std::complex<_Tp> __x )  [inline]
```

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

<code>_↵ _n</code>	The integral order $\geq 0$
<code>_↵ _x</code>	The complex argument

Definition at line 4506 of file `specfun.h`.

```
8.3.3.271  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 2530 of file `specfun.h`.

8.3.3.272 `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 4474 of file `specfun.h`.

8.3.3.273 `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 2540 of file `specfun.h`.

8.3.3.274 `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 4485 of file `specfun.h`.

8.3.3.275 `template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_2( unsigned int __n, _Tp __z ) [inline]`

Return the spherical Hankel function of the second kind  $h_n^{(2)}(x)$  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 2601 of file `specfun.h`.

```
8.3.3.276  template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_2 ( unsigned
            int __n, std::complex<_Tp> __x )  [inline]
```

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

The spherical Hankel function of the second kind is defined by

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

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

## Parameters

<code>_↵ _n</code>	The integral order $\geq 0$
<code>_↵ _x</code>	The complex argument

Definition at line 4554 of file `specfun.h`.

```
8.3.3.277  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 2573 of file `specfun.h`.



8.3.3.278 `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 4522 of file `specfun.h`.

8.3.3.279 `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 2583 of file `specfun.h`.

8.3.3.280 `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 4533 of file `specfun.h`.

8.3.3.281 `template<typename _Ttheta , typename _Tphi > std::complex<__gnu_cxx::__promote_fp_t<_Ttheta, _Tphi> > __gnu_cxx::sph_harmonic ( unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi ) [inline]`

Return the complex spherical harmonic function of degree  $l$ , order  $m$ , and real zenith angle  $\theta$ , 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 4605 of file `specfun.h`.

```
8.3.3.282  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 4569 of file `specfun.h`.

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

Return the complex spherical harmonic function of degree  $l$ , order  $m$ , and `long double` zenith angle  $\theta$ , and azimuth angle  $\phi$ .

## See also

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

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

```
8.3.3.284  template<typename _Tp> __gnu_cxx::promote<_Tp>::__type __gnu_cxx::tan_pi ( _Tp __x )  [inline]
```

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

The reperiodized tangent function is defined by:

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

## Template Parameters

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

## Parameters

$\leftrightarrow$	The argument
$x$	

Definition at line 5658 of file specfun.h.

8.3.3.285 `float __gnu_cxx::tan_pif ( float  $x$  )` `[inline]`

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

## See also

[tan\\_pi](#) for more details.

Definition at line 5631 of file specfun.h.

8.3.3.286 `long double __gnu_cxx::tan_pil ( long double  $x$  )` `[inline]`

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

## See also

[tan\\_pi](#) for more details.

Definition at line 5641 of file specfun.h.

8.3.3.287 `template<typename  $Tp$ > __gnu_cxx::__promote< $Tp$ >::__type __gnu_cxx::tanh_pi (  $Tp$   $x$  )` `[inline]`

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

The reperiodized hyperbolic tangent function is defined by:

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

## Template Parameters

$Tp$	The floating-point type of the argument $x$ .
------	---

## Parameters

$\leftrightarrow$	The argument
$x$	

Definition at line 5700 of file specfun.h.

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

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

See also

[tanh\\_pi](#) for more details.

Definition at line 5673 of file specfun.h.

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

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

See also

[tanh\\_pi](#) for more details.

Definition at line 5683 of file specfun.h.

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

Return the gamma function for complex argument.

Definition at line 2924 of file specfun.h.

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

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

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

Definition at line 2961 of file specfun.h.

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

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

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

Definition at line 2998 of file `specfun.h`.

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

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

See also

[tgamma\\_lower](#) for details.

Definition at line 2976 of file `specfun.h`.

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

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

See also

[tgamma\\_lower](#) for details.

Definition at line 2986 of file `specfun.h`.

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

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

See also

[lgamma](#) for details.

Definition at line 2906 of file `specfun.h`.

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

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

See also

[tgamma](#) for details.

Definition at line 2939 of file `specfun.h`.

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

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

See also

[lgamma](#) for details.

Definition at line 2916 of file `specfun.h`.

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

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

See also

[tgamma](#) for details.

Definition at line 2949 of file `specfun.h`.

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

Return the exponential theta-1 function  $\theta_1(\nu, x)$  of period  $nu$  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 5030 of file specfun.h.

8.3.3.300 `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 5002 of file specfun.h.

8.3.3.301 `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 5012 of file specfun.h.

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

Return the exponential theta-2 function  $\theta_2(\nu, x)$  of period  $nu$  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 5073 of file specfun.h.

8.3.3.303 `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 5045 of file specfun.h.

```
8.3.3.304 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 5055 of file specfun.h.

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

Return the exponential theta-3 function  $\theta_3(\nu, x)$  of period  $nu$  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 5116 of file specfun.h.

```
8.3.3.306 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 5088 of file specfun.h.



8.3.3.307 `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 5098 of file specfun.h.

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

Return the exponential theta-4 function  $\theta_4(\nu, x)$  of period  $nu$  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 5159 of file specfun.h.

8.3.3.309 `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 5131 of file specfun.h.

8.3.3.310 `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 5141 of file specfun.h.

8.3.3.311 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_fp_t<_Tp, _Tp> __gnu_cxx::theta_c ( _Tp __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 5285 of file specfun.h.

8.3.3.312 `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 5258 of file specfun.h.

8.3.3.313 `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 5268 of file specfun.h.

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

$\leftrightarrow$ __k	The modulus $-1 \leq k \leq +1$
$\leftrightarrow$ __x	The argument

Definition at line 5327 of file specfun.h.

8.3.3.315 `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 5300 of file specfun.h.

8.3.3.316 `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 5310 of file specfun.h.

8.3.3.317 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_fp_t<_Tp, _Tp> __gnu_cxx::theta_n ( _Tp __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

$\leftrightarrow$ __k	The modulus $-1 \leq k \leq +1$
$\leftrightarrow$ __x	The argument

Definition at line 5369 of file specfun.h.

8.3.3.318 `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 5342 of file specfun.h.

8.3.3.319 `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 5352 of file specfun.h.

8.3.3.320 `template<typename _Tp, typename _Tp> __gnu_cxx::_promote_fp_t<_Tp, _Tp> __gnu_cxx::theta_s ( _Tp __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 5243 of file specfun.h.

8.3.3.321 `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 5216 of file specfun.h.

**8.3.3.322** `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 5226 of file specfun.h.

**8.3.3.323** `template<typename _Trho, typename _Tphi> __gnu_cxx::__promote_fp_t<_Trho, _Tphi> __gnu_cxx::zernike ( unsigned int __n, int __m, _Trho __rho, _Tphi __phi ) [inline]`

Return the Zernicke polynomial  $Z_n^m(\rho, \phi)$  for non-negative degree  $n$ , signed order  $m$ , and real radial argument  $\rho$  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 2280 of file specfun.h.

8.3.3.324 `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 2241 of file specfun.h.

8.3.3.325 `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 2252 of file specfun.h.

## Chapter 9

# Namespace Documentation

### 9.1 `__gnu_cxx` Namespace Reference

#### Classes

- struct [\\_\\_sincos\\_t](#)

#### Enumerations

- enum { [\\_GLIBCXX\\_JACOBI\\_SN](#), [\\_GLIBCXX\\_JACOBI\\_CN](#), [\\_GLIBCXX\\_JACOBI\\_DN](#) }

#### Functions

- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [airy\\_ai](#) (\_Tp \_\_x)
- template<typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [airy\\_ai](#) (std::complex< \_Tp > \_\_x)
- float [airy\\_aif](#) (float \_\_x)
- long double [airy\\_ail](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [airy\\_bi](#) (\_Tp \_\_x)
- template<typename \_Tp >  
  std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > > [airy\\_bi](#) (std::complex< \_Tp > \_\_x)
- float [airy\\_bif](#) (float \_\_x)
- long double [airy\\_bil](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)< \_Tp > [bernoulli](#) (unsigned int \_\_n)
- float [bernoullif](#) (unsigned int \_\_n)
- long double [bernoullil](#) (unsigned int \_\_n)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_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 \_Tps, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tps, \_Tp > [bose\\_einstein](#) (\_Tps \_\_s, \_Tp \_\_x)
- float [bose\\_einsteinf](#) (float \_\_s, float \_\_x)
- long double [bose\\_einsteinl](#) (long double \_\_s, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [chebyshev\\_t](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [chebyshev\\_tf](#) (unsigned int \_\_n, float \_\_x)
- long double [chebyshev\\_tl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [chebyshev\\_u](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [chebyshev\\_uf](#) (unsigned int \_\_n, float \_\_x)
- long double [chebyshev\\_ul](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [chebyshev\\_v](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [chebyshev\\_vf](#) (unsigned int \_\_n, float \_\_x)
- long double [chebyshev\\_vl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [chebyshev\\_w](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [chebyshev\\_wf](#) (unsigned int \_\_n, float \_\_x)
- long double [chebyshev\\_wl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [clausen](#) (unsigned int \_\_m, \_Tp \_\_w)
- template<typename \_Tp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > > [clausen](#) (unsigned int \_\_m, std::complex< \_Tp > \_\_w)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_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\_fp\_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\_fp\_t< \_Tk > [comp\\_ellint\\_d](#) (\_Tk \_\_k)
- float [comp\\_ellint\\_df](#) (float \_\_k)
- long double [comp\\_ellint\\_dl](#) (long double \_\_k)
- float [comp\\_ellint\\_rf](#) (float \_\_x, float \_\_y)
- long double [comp\\_ellint\\_rf](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tx, typename \_Ty >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tx, \_Ty > [comp\\_ellint\\_rf](#) (\_Tx \_\_x, \_Ty \_\_y)
- float [comp\\_ellint\\_rg](#) (float \_\_x, float \_\_y)
- long double [comp\\_ellint\\_rg](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tx, typename \_Ty >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tx, \_Ty > [comp\\_ellint\\_rg](#) (\_Tx \_\_x, \_Ty \_\_y)
- template<typename \_Tpa, typename \_Tpc, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_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< _Tp >::__type cos\_pi (_Tp __x)`
- `float cos\_pif (float __x)`
- `long double cos\_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type cosh\_pi (_Tp __x)`
- `float cosh\_pif (float __x)`
- `long double cosh\_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > coshint (_Tp __x)`
- `float coshintf (float __x)`
- `long double coshintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > cosint (_Tp __x)`
- `float cosintf (float __x)`
- `long double cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl\_hankel\_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl\_hankel\_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > cyl\_hankel\_1f (float __nu, float __z)`
- `std::complex< float > cyl\_hankel\_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl\_hankel\_1l (long double __nu, long double __z)`
- `std::complex< long double > cyl\_hankel\_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl\_hankel\_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl\_hankel\_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > cyl\_hankel\_2f (float __nu, float __z)`
- `std::complex< float > cyl\_hankel\_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl\_hankel\_2l (long double __nu, long double __z)`
- `std::complex< long double > cyl\_hankel\_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > dawson (_Tp __x)`
- `float dawsonf (float __x)`
- `long double dawsonl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > dilog (_Tp __x)`
- `float dilogf (float __x)`
- `long double dilogl (long double __x)`
- `template<typename _Tp >`  
`_Tp dirichlet\_beta (_Tp __s)`

- float [dirichlet\\_betal](#) (float \_\_s)
- long double [dirichlet\\_betal](#) (long double \_\_s)
- template<typename \_Tp >  
  \_Tp [dirichlet\\_eta](#) (\_Tp \_\_s)
- float [dirichlet\\_etaf](#) (float \_\_s)
- long double [dirichlet\\_etaf](#) (long double \_\_s)
- template<typename \_Tp >  
  \_Tp [dirichlet\\_lambda](#) (\_Tp \_\_s)
- float [dirichlet\\_lambdaf](#) (float \_\_s)
- long double [dirichlet\\_lambdaf](#) (long double \_\_s)
- template<typename \_Tp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp > [double\\_factorial](#) (int \_\_n)
- float [double\\_factorialf](#) (int \_\_n)
- long double [double\\_factorialf](#) (int \_\_n)
- template<typename \_Tk, typename \_Tp, typename \_Ta, typename \_Tb >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tk, \_Tp, \_Ta, \_Tb > [ellint\\_cel](#) (\_Tk \_\_k\_c, \_Tp \_\_p, \_Ta \_\_a, \_Tb \_\_b)
- float [ellint\\_celf](#) (float \_\_k\_c, float \_\_p, float \_\_a, float \_\_b)
- long double [ellint\\_celf](#) (long double \_\_k\_c, long double \_\_p, long double \_\_a, long double \_\_b)
- template<typename \_Tk, typename \_Tphi >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tk, \_Tphi > [ellint\\_d](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [ellint\\_df](#) (float \_\_k, float \_\_phi)
- long double [ellint\\_df](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Tk >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Tk > [ellint\\_el1](#) (\_Tp \_\_x, \_Tk \_\_k\_c)
- float [ellint\\_el1f](#) (float \_\_x, float \_\_k\_c)
- long double [ellint\\_el1f](#) (long double \_\_x, long double \_\_k\_c)
- template<typename \_Tp, typename \_Tk, typename \_Ta, typename \_Tb >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Tk, \_Ta, \_Tb > [ellint\\_el2](#) (\_Tp \_\_x, \_Tk \_\_k\_c, \_Ta \_\_a, \_Tb \_\_b)
- float [ellint\\_el2f](#) (float \_\_x, float \_\_k\_c, float \_\_a, float \_\_b)
- long double [ellint\\_el2f](#) (long double \_\_x, long double \_\_k\_c, long double \_\_a, long double \_\_b)
- template<typename \_Tx, typename \_Tk, typename \_Tp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tx, \_Tk, \_Tp > [ellint\\_el3](#) (\_Tx \_\_x, \_Tk \_\_k\_c, \_Tp \_\_p)
- float [ellint\\_el3f](#) (float \_\_x, float \_\_k\_c, float \_\_p)
- long double [ellint\\_el3f](#) (long double \_\_x, long double \_\_k\_c, long double \_\_p)
- template<typename \_Tp, typename \_Up >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Up > [ellint\\_rc](#) (\_Tp \_\_x, \_Up \_\_y)
- float [ellint\\_rcf](#) (float \_\_x, float \_\_y)
- long double [ellint\\_rcf](#) (long double \_\_x, long double \_\_y)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Up, \_Vp > [ellint\\_rd](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rdf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rdf](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Up, \_Vp > [ellint\\_rf](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rff](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rff](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Up, \_Vp > [ellint\\_rg](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z)
- float [ellint\\_rgf](#) (float \_\_x, float \_\_y, float \_\_z)
- long double [ellint\\_rgf](#) (long double \_\_x, long double \_\_y, long double \_\_z)
- template<typename \_Tp, typename \_Up, typename \_Vp, typename \_Wp >  
  \_\_gnu\_cxx::\_\_promote\_fp\_t<\_Tp, \_Up, \_Vp, \_Wp > [ellint\\_rj](#) (\_Tp \_\_x, \_Up \_\_y, \_Vp \_\_z, \_Wp \_\_p)

- float [ellint\\_rjf](#) (float \_\_x, float \_\_y, float \_\_z, float \_\_p)
- long double [ellint\\_rjl](#) (long double \_\_x, long double \_\_y, long double \_\_z, long double \_\_p)
- template<typename \_Tp >  
  [\\_Tp ellnome](#) (\_Tp \_\_k)
- float [ellnomef](#) (float \_\_k)
- long double [ellnomel](#) (long double \_\_k)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp > expint](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [expintf](#) (unsigned int \_\_n, float \_\_x)
- long double [expintl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp > factorial](#) (unsigned int \_\_n)
- float [factorialf](#) (unsigned int \_\_n)
- long double [factoriall](#) (unsigned int \_\_n)
- template<typename \_Tps, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tps, \\_Tp > fermi\\_dirac](#) (\_Tps \_\_s, \_Tp \_\_x)
- float [fermi\\_diracf](#) (float \_\_s, float \_\_x)
- long double [fermi\\_diracl](#) (long double \_\_s, long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp > fresnel\\_c](#) (\_Tp \_\_x)
- float [fresnel\\_cf](#) (float \_\_x)
- long double [fresnel\\_cl](#) (long double \_\_x)
- template<typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp > fresnel\\_s](#) (\_Tp \_\_x)
- float [fresnel\\_sf](#) (float \_\_x)
- long double [fresnel\\_sl](#) (long double \_\_x)
- template<typename \_Talpha, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Talpha, \\_Tp > gegenbauer](#) (unsigned int \_\_n, \_Talpha \_\_alpha, \_Tp \_\_x)
- float [gegenbauerf](#) (unsigned int \_\_n, float \_\_alpha, float \_\_x)
- long double [gegenbauerl](#) (unsigned int \_\_n, long double \_\_alpha, long double \_\_x)
- template<typename \_Tk, typename \_Tphi >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tk, \\_Tphi > heuman\\_lambda](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [heuman\\_lambdaf](#) (float \_\_k, float \_\_phi)
- long double [heuman\\_lambdal](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Up >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Up > hurwitz\\_zeta](#) (\_Tp \_\_s, \_Up \_\_a)
- template<typename \_Tp, typename \_Up >  
  [std::complex<\\_Tp > hurwitz\\_zeta](#) (\_Tp \_\_s, std::complex<\_Up > \_\_a)
- float [hurwitz\\_zetaf](#) (float \_\_s, float \_\_a)
- long double [hurwitz\\_zetal](#) (long double \_\_s, long double \_\_a)
- template<typename \_Tpa, typename \_Tpb, typename \_Tpc, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_4<\\_Tpa, \\_Tpb, \\_Tpc, \\_Tp >::\\_\\_type hyperg](#) (\_Tpa \_\_a, \_Tpb \_\_b, \_Tpc \_\_c, \_Tp \_\_x)
- float [hyperg](#) (float \_\_a, float \_\_b, float \_\_c, float \_\_x)
- long double [hypergl](#) (long double \_\_a, long double \_\_b, long double \_\_c, long double \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Ta, \\_Tb, \\_Tp > ibeta](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
  [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Ta, \\_Tb, \\_Tp > ibetac](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- float [ibetacf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [ibetac](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- float [ibetaf](#) (float \_\_a, float \_\_b, float \_\_x)

- long double [ibetal](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- template<typename \_Talpha, typename \_Tbeta, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Talpha, \_Tbeta, \_Tp > [jacobi](#) (unsigned \_\_n, \_Talpha \_\_alpha, \_Tbeta \_\_beta, \_Tp \_\_x)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Kp, \_Up > [jacobi\\_cn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_cnf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_cnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Kp, \_Up > [jacobi\\_dn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_dnf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_dnl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Kp, typename \_Up >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Kp, \_Up > [jacobi\\_sn](#) (\_Kp \_\_k, \_Up \_\_u)
- float [jacobi\\_snf](#) (float \_\_k, float \_\_u)
- long double [jacobi\\_snl](#) (long double \_\_k, long double \_\_u)
- template<typename \_Tk, typename \_Tphi >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tk, \_Tphi > [jacobi\\_zeta](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [jacobi\\_zetaf](#) (float \_\_k, float \_\_phi)
- long double [jacobi\\_zetal](#) (long double \_\_k, long double \_\_phi)
- float [jacobif](#) (unsigned \_\_n, float \_\_alpha, float \_\_beta, float \_\_x)
- long double [jacobil](#) (unsigned \_\_n, long double \_\_alpha, long double \_\_beta, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [lbincoef](#) (unsigned int \_\_n, unsigned int \_\_k)
- float [lbincoeff](#) (unsigned int \_\_n, unsigned int \_\_k)
- long double [lbincoefl](#) (unsigned int \_\_n, unsigned int \_\_k)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [ldouble\\_factorial](#) (int \_\_n)
- float [ldouble\\_factorialf](#) (int \_\_n)
- long double [ldouble\\_factoriall](#) (int \_\_n)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [legendre\\_q](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [legendre\\_qf](#) (unsigned int \_\_n, float \_\_x)
- long double [legendre\\_ql](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [lfactorial](#) (unsigned int \_\_n)
- float [lfactorialf](#) (unsigned int \_\_n)
- long double [lfactoriall](#) (unsigned int \_\_n)
- template<typename \_Ta >  
std::complex< \_\_gnu\_cxx::\_\_promote\_fp\_t< \_Ta > > [lgamma](#) (std::complex< \_Ta > \_\_a)
- std::complex< float > [lgammaf](#) (std::complex< float > \_\_a)
- std::complex< long double > [lgammal](#) (std::complex< long double > \_\_a)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [logint](#) (\_Tp \_\_x)
- float [logintf](#) (float \_\_x)
- long double [logintl](#) (long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Tn > [lpochhammer](#) (\_Tp \_\_a, \_Tn \_\_n)
- template<typename \_Tp, typename \_Tn >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Tn > [lpochhammer\\_lower](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [lpochhammer\\_lowerf](#) (float \_\_a, float \_\_n)
- long double [lpochhammer\\_lowerl](#) (long double \_\_a, long double \_\_n)

- float [lpochhammerf](#) (float \_\_a, float \_\_n)
- long double [lpochhammerl](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tph, typename \_Tpa >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tph, \_Tpa > [owens\\_t](#) (\_Tph \_\_h, \_Tpa \_\_a)
- float [owens\\_tf](#) (float \_\_h, float \_\_a)
- long double [owens\\_tl](#) (long double \_\_h, long double \_\_a)
- template<typename \_Ta, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Ta, \_Tp > [pgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [pgammaf](#) (float \_\_a, float \_\_x)
- long double [pgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Tn > [pochhammer](#) (\_Tp \_\_a, \_Tn \_\_n)
- template<typename \_Tp, typename \_Tn >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Tn > [pochhammer\\_lower](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [pochhammer\\_lowerf](#) (float \_\_a, float \_\_n)
- long double [pochhammer\\_lowerl](#) (long double \_\_a, long double \_\_n)
- float [pochhammerf](#) (float \_\_a, float \_\_n)
- long double [pochhammerl](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Wp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Wp > [polylog](#) (\_Tp \_\_s, \_Wp \_\_w)
- template<typename \_Tp, typename \_Wp >  
std::complex< \_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp, \_Wp > > [polylog](#) (\_Tp \_\_s, std::complex< \_Tp > \_\_w)
- float [polylogf](#) (float \_\_s, float \_\_w)
- std::complex< float > [polylogf](#) (float \_\_s, std::complex< float > \_\_w)
- long double [polylogl](#) (long double \_\_s, long double \_\_w)
- std::complex< long double > [polylogl](#) (long double \_\_s, std::complex< long double > \_\_w)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [psi](#) (\_Tp \_\_x)
- float [psif](#) (float \_\_x)
- long double [psil](#) (long double \_\_x)
- template<typename \_Ta, typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Ta, \_Tp > [qgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [qgammaf](#) (float \_\_a, float \_\_x)
- long double [qgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [radpoly](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_rho)
- float [radpolyf](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_rho)
- long double [radpolyl](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_rho)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote< \_Tp >::\_\_type [sin\\_pi](#) (\_Tp \_\_x)
- float [sin\\_pif](#) (float \_\_x)
- long double [sin\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [sinc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
\_\_gnu\_cxx::\_\_promote\_fp\_t< \_Tp > [sinc\\_pi](#) (\_Tp \_\_x)
- float [sinc\\_pif](#) (float \_\_x)
- long double [sinc\\_pil](#) (long double \_\_x)
- float [sincf](#) (float \_\_x)
- long double [sincl](#) (long double \_\_x)
- \_\_gnu\_cxx::\_\_sincos\_t< double > [sincos](#) (double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__sincos_t< _Tp > sincos ( _Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__sincos_t< _Tp > sincos_pi ( _Tp __x)`
- `__gnu_cxx::__sincos_t< float > sincos_pif (float __x)`
- `__gnu_cxx::__sincos_t< long double > sincos_pil (long double __x)`
- `__gnu_cxx::__sincos_t< float > sincosf (float __x)`
- `__gnu_cxx::__sincos_t< long double > sincosl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type sinh_pi ( _Tp __x)`
- `float sinh_pif (float __x)`
- `long double sinh_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sinhc ( _Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sinhc_pi ( _Tp __x)`
- `float sinhc_pif (float __x)`
- `long double sinhc_pil (long double __x)`
- `float sinhcf (float __x)`
- `long double sinhcl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sinhint ( _Tp __x)`
- `float sinhintf (float __x)`
- `long double sinhintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sinint ( _Tp __x)`
- `float sinintf (float __x)`
- `long double sinintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sph_bessel_i (unsigned int __n, _Tp __x)`
- `float sph_bessel_if (unsigned int __n, float __x)`
- `long double sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > sph_bessel_k (unsigned int __n, _Tp __x)`
- `float sph_bessel_kf (unsigned int __n, float __x)`
- `long double sph_bessel_kl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > sph_hankel_1f (unsigned int __n, float __z)`
- `std::complex< float > sph_hankel_1f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > sph_hankel_1l (unsigned int __n, long double __z)`
- `std::complex< long double > sph_hankel_1l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > sph_hankel_2 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > sph_hankel_2f (unsigned int __n, float __z)`
- `std::complex< float > sph_hankel_2f (unsigned int __n, std::complex< float > __x)`

- `std::complex< long double > sph_hankel_2l` (unsigned int `__n`, long double `__z`)
- `std::complex< long double > sph_hankel_2l` (unsigned int `__n`, `std::complex< long double > __x`)
- `template<typename _Ttheta, typename _Tphi >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > sph_harmonic` (unsigned int `__l`, int `__m`, `←`  
`_Ttheta __theta`, `_Tphi __phi`)
- `std::complex< float > sph_harmonicf` (unsigned int `__l`, int `__m`, float `__theta`, float `__phi`)
- `std::complex< long double > sph_harmonicl` (unsigned int `__l`, int `__m`, long double `__theta`, long double `__phi`)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type tan_pi` (`_Tp __x`)
- float `tan_pif` (float `__x`)
- long double `tan_pil` (long double `__x`)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type tanh_pi` (`_Tp __x`)
- float `tanh_pif` (float `__x`)
- long double `tanh_pil` (long double `__x`)
- `template<typename _Ta >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > tgamma` (`std::complex< _Ta > __a`)
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > tgamma` (`_Ta __a`, `_Tp __x`)
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > tgamma_lower` (`_Ta __a`, `_Tp __x`)
- float `tgamma_lowerf` (float `__a`, float `__x`)
- long double `tgamma_lowerl` (long double `__a`, long double `__x`)
- `std::complex< float > tgammaf` (`std::complex< float > __a`)
- float `tgammaf` (float `__a`, float `__x`)
- `std::complex< long double > tgamma` (`std::complex< long double > __a`)
- long double `tgamma` (long double `__a`, long double `__x`)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > theta_1` (`_Tpnu __nu`, `_Tp __x`)
- float `theta_1f` (float `__nu`, float `__x`)
- long double `theta_1l` (long double `__nu`, long double `__x`)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > theta_2` (`_Tpnu __nu`, `_Tp __x`)
- float `theta_2f` (float `__nu`, float `__x`)
- long double `theta_2l` (long double `__nu`, long double `__x`)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > theta_3` (`_Tpnu __nu`, `_Tp __x`)
- float `theta_3f` (float `__nu`, float `__x`)
- long double `theta_3l` (long double `__nu`, long double `__x`)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > theta_4` (`_Tpnu __nu`, `_Tp __x`)
- float `theta_4f` (float `__nu`, float `__x`)
- long double `theta_4l` (long double `__nu`, long double `__x`)
- `template<typename _Tpk, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > theta_c` (`_Tpk __k`, `_Tp __x`)
- float `theta_cf` (float `__k`, float `__x`)
- long double `theta_cl` (long double `__k`, long double `__x`)
- `template<typename _Tpk, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > theta_d` (`_Tpk __k`, `_Tp __x`)
- float `theta_df` (float `__k`, float `__x`)
- long double `theta_dl` (long double `__k`, long double `__x`)

- `template<typename _Tpk, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > theta\_n (_Tpk __k, _Tp __x)`
- `float theta\_nf (float __k, float __x)`
- `long double theta\_nl (long double __k, long double __x)`
- `template<typename _Tpk, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > theta\_s (_Tpk __k, _Tp __x)`
- `float theta\_sf (float __k, float __x)`
- `long double theta\_sl (long double __k, long double __x)`
- `template<typename _Trho, typename _Tphi >`  
`__gnu_cxx::__promote_fp_t< _Trho, _Tphi > zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)`
- `float zernikef (unsigned int __n, int __m, float __rho, float __phi)`
- `long double zernikel (unsigned int __n, int __m, long double __rho, long double __phi)`

## 9.2 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)

### 9.3 std::\_\_detail Namespace Reference

#### Classes

- class [\\_Airy](#)
- class [\\_Airy\\_asymp](#)
- struct [\\_Airy\\_asymp\\_data](#)
- struct [\\_Airy\\_asymp\\_data< double >](#)
- struct [\\_Airy\\_asymp\\_data< float >](#)
- struct [\\_Airy\\_asymp\\_data< long double >](#)
- class [\\_Airy\\_asymp\\_series](#)
- struct [\\_Airy\\_default\\_radii](#)
- struct [\\_Airy\\_default\\_radii< double >](#)
- struct [\\_Airy\\_default\\_radii< float >](#)
- struct [\\_Airy\\_default\\_radii< long double >](#)
- class [\\_Airy\\_series](#)
- struct [\\_AiryAuxilliaryState](#)
- struct [\\_AiryState](#)
- struct [\\_Factorial\\_table](#)
- struct [\\_GammaLanczos](#)
- struct [\\_GammaLanczos< double >](#)
- struct [\\_GammaLanczos< float >](#)
- struct [\\_GammaLanczos< long double >](#)
- struct [\\_GammaSpouge](#)
- struct [\\_GammaSpouge< double >](#)
- struct [\\_GammaSpouge< float >](#)
- struct [\\_GammaSpouge< long double >](#)

#### 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 >`  
`std::complex< _Tp > __airy_ai (std::complex< _Tp > __z)`  
*Return the complex Airy Ai function.*
- `template<typename _Tp >`  
`void __airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`  
*Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.*
- `template<typename _Tp >`  
`std::complex< _Tp > __airy_bi (std::complex< _Tp > __z)`  
*Return the complex Airy Bi function.*
- `template<typename _Tp >`  
`_Tp __assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)`  
*This routine returns the associated Laguerre polynomial of order  $n$ , degree  $m$ :  $L_n^m(x)$ .*
- `template<typename _Tp >`  
`_Tp __assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`  
*Return the associated Legendre function by recursion on  $l$  and downward recursion on  $m$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli (int __n)`  
*This returns Bernoulli number  $B_n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_2n (int __n)`  
*This returns Bernoulli number  $B_{2n}$  at even integer arguments  $2n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)`  
*This returns Bernoulli numbers from a table or by summation for larger values.*
- `template<typename _Tp >`  
`_Tp __beta (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp __beta_gamma (_Tp __a, _Tp __b)`  
*Return the beta function:  $B(a, b)$ .*
- `template<typename _Tp >`  
`_Tp __beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __beta_lgamma (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$  using the log gamma functions.*
- `template<typename _Tp >`  
`_Tp __beta_product (_Tp __a, _Tp __b)`  
*Return the beta function  $B(x, y)$  using the product form.*
- `template<typename _Tp >`  
`_Tp __bincoef (unsigned int __n, unsigned int __k)`

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

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

The binomial coefficients are generated by:

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

- `template<typename _Tp >`  
`_Tp __bincoef (_Tp __nu, unsigned int __k)`

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

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

The binomial coefficients are generated by:

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

- `template<typename _Tp >`  
`_Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`  
Return the binomial cumulative distribution function.
- `template<typename _Tp >`  
`_Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`  
Return the complementary binomial cumulative distribution function.
- `template<typename _Tp >`  
`_Tp __binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`  
Return the binomial probability mass function.
- `template<typename _Sp, typename _Tp >`  
`_Tp __bose_einstein (_Sp __s, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`  
`_Tp __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chebyshev_w (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __chi_squared_pdf (_Tp __chi2, unsigned int __nu)`

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

- `template<typename _Tp >`  
`_Tp __chi_squared_pdfc (_Tp __chi2, unsigned int __nu)`

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

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`

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

- `template<typename _Tp >`  
`void __chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by continued fraction for positive argument.*
- `template<typename _Tp >`  
`void __chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`  
*This function computes the hyperbolic cosine  $Chi(x)$  and hyperbolic sine  $Shi(x)$  integrals by series summation for positive argument.*
- `template<typename _Tp >`  
`std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __clamp_pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __clausen_c (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen_c (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __clausen_s (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`_Tp __clausen_s (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`_Tp __comp_ellint_1 (_Tp __k)`  
*Return the complete elliptic integral of the first kind  $K(k)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp __comp_ellint_2 (_Tp __k)`  
*Return the complete elliptic integral of the second kind  $E(k)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp __comp_ellint_3 (_Tp __k, _Tp __nu)`  
*Return the complete elliptic integral of the third kind  $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp __comp_ellint_d (_Tp __k)`
- `template<typename _Tp >`  
`_Tp __comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >`  
`_Tp __comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >`  
`_Tp __conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`  
*Return the confluent hypergeometric function  ${}_1F_1(a; c; x)$ .*
- `template<typename _Tp >`  
`_Tp __conf_hyperg_lim (_Tp __c, _Tp __x)`  
*Return the confluent hypergeometric limit function  ${}_0F_1(-; c; x)$ .*
- `template<typename _Tp >`  
`_Tp __conf_hyperg_lim_series (_Tp __c, _Tp __x)`  
*This routine returns the confluent hypergeometric limit function by series expansion.*
- `template<typename _Tp >`  
`_Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`

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

- `template<typename _Tp >`  
`_Tp __conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`

This routine returns the confluent hypergeometric function by series expansion.

- `template<typename _Tp >`  
`_Tp __cos_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > __cos_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`_Tp __cosh_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > __cosh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`_Tp __coshint (const _Tp __x)`

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

- `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 and 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_neg_arg (_Tp __nu, _Tp __x, std::complex< _Tp > &_Jnu, std::complex< _Tp > &_Nnu, std::complex< _Tp > &_Jpnu, std::complex< _Tp > &_Npnu)`

Return the cylindrical Bessel functions and their derivatives of order  $\nu$  and argument  $x < 0$ .

- `template<typename _Tp >`  
`void __cyl_bessel_jn_steed (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`  
*Compute the Bessel  $J_\nu(x)$  and Neumann  $N_\nu(x)$  functions and their first derivatives  $J'_\nu(x)$  and  $N'_\nu(x)$  respectively. These four functions are computed together for numerical stability.*
- `template<typename _Tp >`  
`_Tp __cyl_bessel_k (_Tp __nu, _Tp __x)`  
*Return the irregular modified Bessel function  $K_\nu(x)$  of order  $\nu$ .*
- `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 >`  
`_Tp __dirichlet_lambda (_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 >`  
`_Tp __exponential_cdf (_Tp __lambda, _Tp __x)`  
*Return the exponential cumulative probability density function.*
- `template<typename _Tp >`  
`_Tp __exponential_pdf (_Tp __lambda, _Tp __x)`  
*Return the exponential probability density function.*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)`  
*Return the factorial of the integer  $n$ .*
- `template<typename _Sp, typename _Tp >`  
`_Tp __fermi_dirac (_Sp __s, _Tp __x)`
- `template<typename _Tp >`  
`_Tp __fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the  $F$ -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*

- `template<typename _Tp >`  
`_Tp __fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*

- `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 >`  
`std::complex< _Tp > __fresnel (const _Tp __x)`  
*Return the Fresnel cosine and sine integrals as a complex number  $\$f[C(x) + iS(x)]$ .*
- `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 the gamma function  $\Gamma(x)$ . The gamma function is defined by:*

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`  
`_Tp __gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma cumulative propability distribution function.*
- `template<typename _Tp >`  
`_Tp __gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma complementary cumulative propability distribution function.*
- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`  
*Return the incomplete gamma function by continued fraction.*
- `template<typename _Tp >`  
`_Tp __gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma propability distribution function.*
- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)`  
*Return the incomplete gamma function by series summation.*
- `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 __gauss (_Tp __x)`
- `template<typename _Tp >`  
`_Tp __gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`
- `template<typename _Tp >`  
`void __hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void __hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void __hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`  
*Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.*
- `template<typename _Tp >`  
`void __hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`  
*This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.*
- `template<typename _Tp >`  
`void __hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`  
*Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.*
- `template<typename _Tp >`  
`void __hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`  
*Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.*
- `template<typename _Tp >`  
`void __hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > &__num2, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)`  
*Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.*
- `template<typename _Tp >`  
`_Tp __harmonic_number (unsigned int __n)`
- `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 >`  
`_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 > __hurwitz_zeta_polylog (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`  
`std::complex< _Tp > __hydrogen (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp __hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$ .*
- `template<typename _Tp >`  
`_Tp __hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.*
- `template<typename _Tp >`  
`_Tp __hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`  
*Return the hypergeometric function  ${}_2F_1(a, b; c; x)$  by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for  $d = c - a - b$  not integral and formula 15.3.11 for  $d = c - a - b$  integral. This assumes  $a, b, c \neq$  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 >`  
`_Tp __ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`
- `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:*

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

*The binomial coefficients are generated by:*

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

- `template<typename _Tp >`  
`_Tp __log_bincoef (_Tp __nu, unsigned int __k)`

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

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

The binomial coefficients are generated by:

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

- `template<typename _Tp >`  
`_Tp __log_bincoef_sign (_Tp __nu, unsigned int __k)`  
 Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned.
- `template<typename _Tp >`  
`std::complex< _Tp > __log_bincoef_sign (std::complex< _Tp > __nu, unsigned int __k)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)`  
 Return the logarithm of the double factorial of the integer  $n$ .
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)`  
 Return the logarithm of the factorial of the integer  $n$ .
- `template<typename _Tp >`  
`_Tp __log_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 >`  
`std::complex< _Tp > __log_gamma (std::complex< _Tp > __x)`  
 Return  $\log(\Gamma(x))$  for complex argument.
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma1p_lanczos (_Tp __z)`  
 Return  $\log(\Gamma(x))$  by the Lanczos method. This method dominates all others on the positive axis I think.
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma1p_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 >`  
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)`  
 Return  $\log(\Gamma(x))$  by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.
- `template<typename _Tp >`  
`_Tp __log_gamma_sign (_Tp __x)`  
 Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned indicating  $\Gamma(x)$  is undefined.

- `template<typename _Tp >`  
`std::complex< _Tp > __log_gamma_sign (std::complex< _Tp > __x)`
- `template<typename _Tp >`  
`_Tp __log_pochhammer (_Tp __a, _Tp __n)`

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

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

,

$$\{ \begin{matrix} a \\ n \end{matrix} \}$$

, and others.

- `template<typename _Tp >`  
`_Tp __logint (const _Tp __x)`  
Return the logarithmic integral  $li(x)$ .
- `template<typename _Tp >`  
`_Tp __lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`  
Return the lognormal cumulative probability density function.
- `template<typename _Tp >`  
`_Tp __lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`  
Return the lognormal probability density function.
- `template<typename _Tp >`  
`_Tp __normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`  
Return the normal cumulative probability density function.
- `template<typename _Tp >`  
`_Tp __normal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`  
Return the normal probability density function.
- `template<typename _Tp >`  
`_Tp __owens_t (_Tp __h, _Tp __a)`
- `template<typename _Tp >`  
`_Tp __pgamma (_Tp __a, _Tp __x)`

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

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

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

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

is the lower incomplete gamma function.

- `template<typename _Tp >`  
`_Tp __pochhammer (_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.

- `template<typename _Tp >`  
`_Tp __pochhammer_lower (_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 >`  
`std::complex<_Tp> __polar_pi (_Tp __rho, _Tp __phi_pi)`
- `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_fp_t< std::complex< _Tp >, ArgType > __polylog_exp` (\_Tp \_\_s, ArgType \_\_w)
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_asymp` (\_Tp \_\_s, std::complex< \_Tp > \_\_w)
- `template<typename _Tp >`  
`std::complex< _Tp > __polylog_exp_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` (unsigned int \_\_n)



Return the digamma function of integral argument. The digamma or  $\psi(x)$  function is defined as the logarithmic derivative of the gamma function:

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

The digamma series for integral argument is given by:

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

The latter sum is called the harmonic number,  $H_n$ .

- `template<typename _Tp >`  
`_Tp __psi (_Tp __x)`

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

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

For negative argument the reflection formula is used:

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

- `template<typename _Tp >`  
`_Tp __psi (unsigned int __n, _Tp __x)`  
Return the polygamma function  $\psi^{(n)}(x)$ .

- `template<typename _Tp >`  
`_Tp __psi_asymp (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __psi_series (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __qgamma (_Tp __a, _Tp __x)`

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

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

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

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

is the upper incomplete gamma function.

- `template<typename _Tp >`  
`_Tp __rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

Return the Rice probability density function.

- `template<typename _Tp >`  
`_Tp __riemann_zeta (_Tp __s)`

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

- `template<typename _Tp >`  
`_Tp __riemann_zeta_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 >`  
`_Tp __sin_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > __sin_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`_gnu_cxx::__promote_fp_t< _Tp > __sinc (_Tp __x)`  
*Return the sinus cardinal function*

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$
- `template<typename _Tp >`  
`_gnu_cxx::__promote_fp_t< _Tp > __sinc_pi (_Tp __x)`  
*Return the reperiodized sinus cardinal function*

$$\text{sinc}_{\pi}(x) = \frac{\sin(\pi x)}{\pi x}$$
- `template<typename _Tp >`  
`_gnu_cxx::__sincos_t< _Tp > __sincos (_Tp __x)`
- `template<>`  
`_gnu_cxx::__sincos_t< float > __sincos (float __x)`
- `template<>`  
`_gnu_cxx::__sincos_t< double > __sincos (double __x)`
- `template<>`  
`_gnu_cxx::__sincos_t< long double > __sincos (long double __x)`
- `template<typename _Tp >`  
`_gnu_cxx::__sincos_t< _Tp > __sincos_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > __sincosint (_Tp __x)`  
*This function returns the sine  $Si(x)$  and cosine  $Ci(x)$  integrals as a pair.*

- `template<typename _Tp >`  
`void __sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by asymptotic series summation for positive argument.*
- `template<typename _Tp >`  
`void __sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by continued fraction for positive argument.*
- `template<typename _Tp >`  
`void __sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`  
*This function computes the sine  $Si(x)$  and cosine  $Ci(x)$  integrals by series summation for positive argument.*
- `template<typename _Tp >`  
`_Tp __sinh_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > __sinh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __sinhc (_Tp __x)`  
*Return the hyperbolic sinus cardinal function*  

$$\operatorname{sinhc}(x) = \frac{\sinh(x)}{x}$$
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __sinhc_pi (_Tp __x)`  
*Return the reperiodized hyperbolic sinus cardinal function*  

$$\operatorname{sinhc}_{\pi}(x) = \frac{\sinh(\pi x)}{\pi x}$$
- `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_bessel_jn_neg_arg (unsigned int __n, _Tp __x, std::complex< _Tp > &__j_n, std::complex< _Tp > &__n_n, std::complex< _Tp > &__jp_n, std::complex< _Tp > &__np_n)`
- `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 >`  
`_Tp __student_t_cdf (_Tp __t, unsigned int __nu)`

Return the Students  $T$  probability function.

- `template<typename _Tp >`  
`_Tp __student_t_cdfc (_Tp __t, unsigned int __nu)`

Return the complement of the Students  $T$  probability function.

- `template<typename _Tp >`  
`_Tp __tan_pi (_Tp __x)`

- `template<typename _Tp >`  
`std::complex< _Tp > __tan_pi (std::complex< _Tp > __z)`

- `template<typename _Tp >`  
`_Tp __tanh_pi (_Tp __x)`

- `template<typename _Tp >`  
`std::complex< _Tp > __tanh_pi (std::complex< _Tp > __z)`

- `template<typename _Tp >`  
`_Tp __tgamma (_Tp __a, _Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __tgamma_lower (_Tp __a, _Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp __theta_1 (_Tp __nu, _Tp __x)`

- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_2](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_2\\_asymp](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_2\\_sum](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_3](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_3\\_asymp](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_3\\_sum](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_4](#) (\_Tp \_\_nu, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_c](#) (\_Tp \_\_k, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_d](#) (\_Tp \_\_k, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_n](#) (\_Tp \_\_k, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_theta\\_s](#) (\_Tp \_\_k, \_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_weibull\\_cdf](#) (\_Tp \_\_a, \_Tp \_\_b, \_Tp \_\_x)  
*Return the Weibull cumulative probability density function.*
- template<typename \_Tp >  
\_Tp [\\_\\_weibull\\_pdf](#) (\_Tp \_\_a, \_Tp \_\_b, \_Tp \_\_x)  
*Return the Weibull probability density function.*
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t](#)<\_Tp> [\\_\\_zernike](#) (unsigned int \_\_n, int \_\_m, \_Tp \_\_rho, \_Tp \_\_phi)
- template<typename \_Tp >  
\_Tp [\\_\\_znorm1](#) (\_Tp \_\_x)
- template<typename \_Tp >  
\_Tp [\\_\\_znorm2](#) (\_Tp \_\_x)
- template<typename \_Tp = double>  
\_Tp [evenzeta](#) (unsigned int \_\_k)

## Variables

- template<typename \_Tp >  
constexpr int [\\_\\_max\\_FGH](#) = [\\_Airy\\_series](#)<\_Tp>::\_\_N\_FGH
- template<>  
constexpr int [\\_\\_max\\_FGH](#)< double > = 79
- template<>  
constexpr int [\\_\\_max\\_FGH](#)< float > = 15
- constexpr size\_t [\\_Num\\_Euler\\_Maclaurin\\_zeta](#) = 100
- constexpr [\\_Factorial\\_table](#)< long double > [\\_S\\_double\\_factorial\\_table](#) [301]
- constexpr long double [\\_S\\_Euler\\_Maclaurin\\_zeta](#) [[\\_Num\\_Euler\\_Maclaurin\\_zeta](#)]
- constexpr [\\_Factorial\\_table](#)< long double > [\\_S\\_factorial\\_table](#) [171]
- constexpr unsigned long long [\\_S\\_harmonic\\_denom](#) [[\\_S\\_num\\_harmonic\\_numer](#)]
- constexpr unsigned long long [\\_S\\_harmonic\\_numer](#) [[\\_S\\_num\\_harmonic\\_numer](#)]

- constexpr [\\_Factorial\\_table](#)< long double > [\\_S\\_neg\\_double\\_factorial\\_table](#) [999]
- template<typename [\\_Tp](#) >  
constexpr std::size\_t [\\_S\\_num\\_double\\_factorials](#) = 0
- template<>  
constexpr std::size\_t [\\_S\\_num\\_double\\_factorials](#)< double > = 301
- template<>  
constexpr std::size\_t [\\_S\\_num\\_double\\_factorials](#)< float > = 57
- template<>  
constexpr std::size\_t [\\_S\\_num\\_double\\_factorials](#)< long double > = 301
- template<typename [\\_Tp](#) >  
constexpr std::size\_t [\\_S\\_num\\_factorials](#) = 0
- template<>  
constexpr std::size\_t [\\_S\\_num\\_factorials](#)< double > = 171
- template<>  
constexpr std::size\_t [\\_S\\_num\\_factorials](#)< float > = 35
- template<>  
constexpr std::size\_t [\\_S\\_num\\_factorials](#)< long double > = 171
- constexpr unsigned long long [\\_S\\_num\\_harmonic\\_number](#) = 29
- template<typename [\\_Tp](#) >  
constexpr std::size\_t [\\_S\\_num\\_neg\\_double\\_factorials](#) = 0
- template<>  
constexpr std::size\_t [\\_S\\_num\\_neg\\_double\\_factorials](#)< double > = 150
- template<>  
constexpr std::size\_t [\\_S\\_num\\_neg\\_double\\_factorials](#)< float > = 27
- template<>  
constexpr std::size\_t [\\_S\\_num\\_neg\\_double\\_factorials](#)< long double > = 999
- constexpr size\_t [\\_S\\_num\\_zetam1](#) = 33
- constexpr long double [\\_S\\_zetam1](#) [[\\_S\\_num\\_zetam1](#)]

### 9.3.1 Enumeration Type Documentation

#### 9.3.1.1 anonymous enum

Enumerator

***SININT***

***COSINT***

Definition at line 45 of file sf\_trigint.tcc.

### 9.3.2 Function Documentation

#### 9.3.2.1 template<typename [\\_Tp](#) > void std::\_\_detail::\_\_airy ( [\\_Tp](#) [\\_z](#), [\\_Tp](#) & [\\_Ai](#), [\\_Tp](#) & [\\_Bi](#), [\\_Tp](#) & [\\_Aip](#), [\\_Tp](#) & [\\_Bip](#) )

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

## Parameters

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

Definition at line 500 of file sf\_mod\_bessel.tcc.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

Referenced by `__airy_ai()`, `__airy_bi()`, and `__poly_hermite_asymp()`.

9.3.2.2 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_ai ( std::complex<_Tp> __z )`

Return the complex Airy Ai function.

Definition at line 2641 of file sf\_airy.tcc.

References `__airy()`.

9.3.2.3 `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 217 of file sf\_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

9.3.2.4 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_bi ( std::complex<_Tp> __z )`

Return the complex Airy Bi function.

Definition at line 2653 of file `sf_airy.tcc`.

References `__airy()`.

9.3.2.5 `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 303 of file `sf_laguerre.tcc`.

Referenced by `__hydrogen()`.

9.3.2.6 `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

$\leftrightarrow$ _l	The order of the associated Legendre function. $l \geq 0$ .
$\leftrightarrow$ _m	The order of the associated Legendre function. $m \leq l$ .
$\leftrightarrow$ _x	The argument of the associated Legendre function. $ x  \leq 1$ .

Definition at line 178 of file sf\_legendre.tcc.

References `__poly_legendre_p()`.

### 9.3.2.7 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli ( int __n )`

This returns Bernoulli number  $B_n$ .

## Parameters

$\leftrightarrow$ _n	the order n of the Bernoulli number.
-------------------------	--------------------------------------

## Returns

The Bernoulli number of order n.

Definition at line 1678 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__n`.

### 9.3.2.8 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n ( int __n )`

This returns Bernoulli number  $B_{2n}$  at even integer arguments  $2n$ .

## Parameters

$\leftrightarrow$ _n	the half-order n of the Bernoulli number.
-------------------------	---

## Returns

The Bernoulli number of order 2n.

Definition at line 1691 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__n`.

### 9.3.2.9 `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

<code>__n</code>	the order $n$ of the Bernoulli number.
------------------	--

#### Returns

The Bernoulli number of order  $n$ .

Definition at line 1611 of file `sf_gamma.tcc`.

References `std::__detail::__Factorial_table< _Tp >::__n`.

### 9.3.2.10 `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

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

#### Returns

The beta function.

Definition at line 183 of file `sf_beta.tcc`.

References `__beta_lgamma()`.

Referenced by `__poly_jacobi()`, `__gnu_cxx::jacobi()`, `__gnu_cxx::jacobif()`, `__gnu_cxx::jacobil()`, and `std::__detail::__Airy< _Tp >::operator()()`.

9.3.2.11 `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

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

#### Returns

The beta function.

Definition at line 77 of file `sf_beta.tcc`.

References `__gamma()`.

9.3.2.12 `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 275 of file `sf_beta.tcc`.

References `__ibeta_cont_frac()`, `__log_gamma()`, and `__log_gamma_sign()`.

Referenced by `__binomial_cdf()`, `__binomial_cdfc()`, `__fisher_f_cdf()`, `__fisher_f_cdfc()`, `__student_t_cdf()`, and `__student_t_cdfc()`.

#### 9.3.2.13 `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

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

##### Returns

The beta function.

Definition at line 111 of file `sf_beta.tcc`.

References `__log_gamma()`, and `__log_gamma_sign()`.

Referenced by `__beta()`.

#### 9.3.2.14 `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

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

**Returns**

The beta function.

Definition at line 150 of file sf\_beta.tcc.

**9.3.2.15** `template<typename _Tp> _Tp std::__detail::__bincoef ( unsigned int __n, unsigned int __k )`

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

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

The binomial coefficients are generated by:

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

.

**Parameters**

$\longleftrightarrow$ __n	The first argument of the binomial coefficient.
$\longleftrightarrow$ __k	The second argument of the binomial coefficient.

**Returns**

The binomial coefficient.

Definition at line 2229 of file sf\_gamma.tcc.

References std::\_\_detail::\_Factorial\_table< \_Tp >::\_\_n.

**9.3.2.16** `template<typename _Tp> _Tp std::__detail::__bincoef ( _Tp __nu, unsigned int __k )`

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

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

The binomial coefficients are generated by:

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

.

**Parameters**

__nu	The real first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

**Returns**

The binomial coefficient.

Definition at line 2268 of file sf\_gamma.tcc.

References `__log_bincoef()`, `__log_bincoef_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

**9.3.2.17** `template<typename _Tp> _Tp std::__detail::__binomial_cdf ( _Tp __p, unsigned int __n, unsigned int __k )`

Return the binomial cumulative distribution function.

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

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

**Parameters**

$\leftarrow$ <code>__p</code>	
$\leftarrow$ <code>__n</code>	
$\leftarrow$ <code>__k</code>	

Definition at line 538 of file sf\_distributions.tcc.

References `__beta_inc()`.

**9.3.2.18** `template<typename _Tp> _Tp std::__detail::__binomial_cdfc ( _Tp __p, unsigned int __n, unsigned int __k )`

Return the complementary binomial cumulative distribution function.

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

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

**Parameters**

$\leftarrow$ <code>__p</code>	
$\leftarrow$ <code>__n</code>	
$\leftarrow$ <code>__k</code>	

Definition at line 568 of file sf\_distributions.tcc.

References `__beta_inc()`.

9.3.2.19 `template<typename _Tp> _Tp std::__detail::__binomial_pdf ( _Tp __p, unsigned int __n, unsigned int __k )`

Return the binomial probability mass function.

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

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

#### Parameters

<code>__p</code>	
<code>__n</code>	
<code>__k</code>	

Definition at line 501 of file sf\_distributions.tcc.

9.3.2.20 `template<typename _Sp, typename _Tp> _Tp std::__detail::__bose_einstein ( _Sp __s, _Tp __x )`

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

#### See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)  
<http://dlmf.nist.gov/25.12.16>

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

#### Parameters

<code>__s</code>	The order s >= 0.
<code>__x</code>	The real argument.



## Returns

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

Definition at line 1430 of file sf\_polylog.tcc.

References `__polylog_exp()`.

**9.3.2.21** `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 59 of file sf\_chebyshev.tcc.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

**9.3.2.22** `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

$\leftrightarrow$ _n	The non-negative integral order
$\leftrightarrow$ _x	The real argument $-1 \leq x \leq +1$

Definition at line 87 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**9.3.2.23** `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 116 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**9.3.2.24** `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:

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

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 146 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**9.3.2.25** `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 176 of file sf\_chebyshev.tcc.

References `__chebyshev_recur()`.

**9.3.2.26** `template<typename _Tp> _Tp std::__detail::__chi_squared_pdf( _Tp __chi2, unsigned int __nu )`

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

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

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

Definition at line 75 of file sf\_distributions.tcc.

References `__pgamma()`.

**9.3.2.27** `template<typename _Tp > _Tp std::__detail::__chi_squared_pdfc ( _Tp __chi2, unsigned int __nu )`

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

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

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

Definition at line 99 of file sf\_distributions.tcc.

References `__qgamma()`.

**9.3.2.28** `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.

The hyperbolic cosine integral is defined by:

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

The hyperbolic sine integral is defined by:

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

Definition at line 166 of file sf\_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

**9.3.2.29** `template<typename _Tp > void std::__detail::__chshint_cont_frac ( _Tp __t, _Tp & __Chi, _Tp & __Shi )`

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

Definition at line 53 of file sf\_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.30 `template<typename _Tp> void std::__detail::__chshint_series ( _Tp __t, _Tp & _Chi, _Tp & _Shi )`

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

Definition at line 96 of file sf\_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.31 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi ( std::complex<_Tp> __w )`

Definition at line 72 of file sf\_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

9.3.2.32 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi ( std::complex<_Tp> __w )`

Definition at line 59 of file sf\_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

9.3.2.33 `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

<code>__m</code>	The non-negative integral order.
<code>__w</code>	The complex argument.

#### Returns

The complex Clausen function.

Definition at line 1245 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.34 `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 1269 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.35 `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 1344 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.36 `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 1369 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.37 `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 $m \geq 1$ .
$\leftrightarrow$ _w	The complex argument.

## Returns

The Clausen sine sum  $Sl_m(w)$ ,

Definition at line 1294 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.38 `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 $m \geq 1$ .
$\leftrightarrow$ _w	The complex argument.

## Returns

The Clausen sine sum  $Sl_m(w)$ ,

Definition at line 1319 of file sf\_polylog.tcc.

References `__polylog_exp()`.

### 9.3.2.39 `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 568 of file sf\_ellint.tcc.

References `__comp_ellint_rf()`.

Referenced by `__ellint_1()`, `__ellnome_k()`, `__heuman_lambda()`, `__jacobi_zeta()`, `__theta_c()`, `__theta_d()`, `__theta_↔n()`, and `__theta_s()`.

### 9.3.2.40 `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

$\_k$	The modulus of the complete elliptic function.
-------	--

## Returns

The complete elliptic function of the second kind.

Definition at line 641 of file sf\_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

**9.3.2.41** `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 730 of file sf\_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

**9.3.2.42** `template<typename _Tp> _Tp std::__detail::__comp_ellint_d ( _Tp __k )`

Return the complete Legendre elliptic integral D.

Definition at line 837 of file sf\_ellint.tcc.

References `__ellint_rd()`.

9.3.2.43 `template<typename _Tp > _Tp std::__detail::__comp_ellint_rf ( _Tp __x, _Tp __y )`

Definition at line 238 of file `sf_ellint.tcc`.

Referenced by `__comp_ellint_1()`, and `__ellint_rf()`.

9.3.2.44 `template<typename _Tp > _Tp std::__detail::__comp_ellint_rg ( _Tp __x, _Tp __y )`

Definition at line 349 of file `sf_ellint.tcc`.

Referenced by `__ellint_rg()`.

9.3.2.45 `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

$\longleftrightarrow$ <code>__a</code>	The <i>numerator</i> parameter.
$\longleftrightarrow$ <code>__c</code>	The <i>denominator</i> parameter.
$\longleftrightarrow$ <code>__x</code>	The argument of the confluent hypergeometric function.

#### Returns

The confluent hypergeometric function.

Definition at line 283 of file `sf_hyperg.tcc`.

References `__conf_hyperg_luke()`, and `__conf_hyperg_series()`.

9.3.2.46 `template<typename _Tp > _Tp std::__detail::__conf_hyperg_lim ( _Tp __c, _Tp __x )`

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

#### Parameters

$\longleftrightarrow$ <code>__c</code>	The <i>denominator</i> parameter.
$\longleftrightarrow$ <code>__x</code>	The argument of the confluent hypergeometric limit function.

**Returns**

The confluent limit hypergeometric function.

Definition at line 111 of file sf\_hyperg.tcc.

References `__conf_hyperg_lim_series()`.

**9.3.2.47** `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**

$\longleftrightarrow$ _c	The "denominator" parameter.
$\longleftrightarrow$ _x	The argument of the confluent hypergeometric limit function.

**Returns**

The confluent hypergeometric limit function.

Definition at line 78 of file sf\_hyperg.tcc.

Referenced by `__conf_hyperg_lim()`.

**9.3.2.48** `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 178 of file sf\_hyperg.tcc.

Referenced by `__conf_hyperg()`.

**9.3.2.49** `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

$\leftrightarrow$ _a	The "numerator" parameter.
$\leftrightarrow$ _c	The "denominator" parameter.
$\leftrightarrow$ _x	The argument of the confluent hypergeometric function.

## Returns

The confluent hypergeometric function.

Definition at line 143 of file sf\_hyperg.tcc.

Referenced by \_\_conf\_hyperg().

**9.3.2.50** `template<typename _Tp> _Tp std::__detail::__cos_pi ( _Tp __x )`

Return the reperiodized cosine of argument x:

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

Definition at line 119 of file sf\_trig.tcc.

References \_\_cos\_pi().

Referenced by \_\_cyl\_bessel\_jn(), \_\_cyl\_bessel\_jn\_neg\_arg(), and \_\_log\_double\_factorial().

**9.3.2.51** `template<typename _Tp> std::complex<_Tp> std::__detail::__cos_pi ( std::complex<_Tp> __z )`

Return the reperiodized cosine of complex argument z:  $(z) = (x) (y) - i (x) (y)$

Definition at line 239 of file sf\_trig.tcc.

References \_\_sin\_pi().

Referenced by \_\_cos\_pi(), \_\_cosh\_pi(), \_\_sin\_pi(), and \_\_sinh\_pi().

**9.3.2.52** `template<typename _Tp> _Tp std::__detail::__cosh_pi ( _Tp __x )`

Return the reperiodized hyperbolic cosine of argument x:

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

Definition at line 147 of file sf\_trig.tcc.

References \_\_cosh\_pi().

9.3.2.53 `template<typename _Tp> std::complex<_Tp> std::__detail::__cosh_pi ( std::complex<_Tp> __z )`

Return the reperiodized hyperbolic cosine of complex argument  $z$ :  $(z) = (x) (y) + i (x) (y)$

Definition at line 258 of file sf\_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

Referenced by `__cosh_pi()`.

9.3.2.54 `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 556 of file sf\_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.2.55 `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 1181 of file sf\_hankel.tcc.

References `__hankel()`.

**9.3.2.56** `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 389 of file sf\_mod\_bessel.tcc.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

Referenced by `__rice_pdf()`.

**9.3.2.57** `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 416 of file sf\_bessel.tcc.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

**9.3.2.58** `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 319 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()`.

**9.3.2.59** `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 84 of file sf\_mod\_bessel.tcc.

Referenced by `__cyl_bessel_ik()`, and `__cyl_bessel_ik_steel()`.

**9.3.2.60** `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.
<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 155 of file sf\_mod\_bessel.tcc.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

**9.3.2.61** `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 565 of file sf\_bessel.tcc.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.



9.3.2.62 `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 455 of file sf\_bessel.tcc.

References `__cos_pi()`, `__cyl_bessel_jn_asymp()`, `__cyl_bessel_jn_steel()`, and `__sin_pi()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_bessel_jn_neg_arg()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

9.3.2.63 `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 82 of file sf\_bessel.tcc.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_steel()`.

9.3.2.64 `template<typename _Tp > void std::__detail::__cyl_bessel_jn_neg_arg ( _Tp __nu, _Tp __x, std::complex<_Tp> & _Jnu, std::complex<_Tp> & _Nnu, std::complex<_Tp> & _Jpnu, std::complex<_Tp> & _Npnu )`

Return the cylindrical Bessel functions and their derivatives of order  $\nu$  and argument  $x < 0$ .

Definition at line 524 of file sf\_bessel.tcc.

References `__cos_pi()`, and `__cyl_bessel_jn()`.

Referenced by `__cyl_hankel_1()`, `__cyl_hankel_2()`, and `__sph_bessel_jn_neg_arg()`.

9.3.2.65 `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.

## 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 200 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

**9.3.2.66** `template<typename _Tp> _Tp std::__detail::__cyl_bessel_k ( _Tp __nu, _Tp __x )`

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

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 427 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

**9.3.2.67** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 ( _Tp __nu, _Tp __x )`

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

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 630 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`, and `__cyl_bessel_jn_neg_arg()`.

**9.3.2.68** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 ( std::complex<_Tp> __nu, std::complex<_Tp> __z )`

Return the complex cylindrical Hankel function of the first kind.

## Parameters

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Hankel function of the first kind is evaluated.
<code>in</code>	<code>__z</code>	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 1149 of file `sf_hankel.tcc`.

References `__hankel()`.

**9.3.2.69** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 ( _Tp __nu, _Tp __x )`

Return the cylindrical Hankel function of the second kind  $H_n^{(2)}(x)$ .

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

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

## 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 669 of file sf\_bessel.tcc.

References `__cyl_bessel_jn()`, and `__cyl_bessel_jn_neg_arg()`.

**9.3.2.70** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 ( std::complex<_Tp> __nu,  
std::complex<_Tp> __z )`

Return the complex cylindrical Hankel function of the second kind.

**Parameters**

in	<code>__nu</code>	The order for which the cylindrical Hankel function of the second kind is evaluated.
in	<code>__z</code>	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 1165 of file sf\_hankel.tcc.

References `__hankel()`.

**9.3.2.71** `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_neumann ( std::complex<_Tp> __nu,  
std::complex<_Tp> __z )`

Return the complex cylindrical Neumann function.

**Parameters**

in	<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 1197 of file sf\_hankel.tcc.

References `__hankel()`.

9.3.2.72 `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 600 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

9.3.2.73 `template<typename _Tp> _Tp std::__detail::__dawson ( _Tp __x )`

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

The Dawson integral is defined by:

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

and it's derivative is:

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

## Parameters

<code>__x</code>	The argument $-\infty < x < \infty$ .
------------------	---------------------------------------

Definition at line 235 of file `sf_dawson.tcc`.

References `__dawson_cont_frac()`, and `__dawson_series()`.

9.3.2.74 `template<typename _Tp> _Tp std::__detail::__dawson_cont_frac ( _Tp __x )`

Compute the Dawson integral using a sampling theorem representation.

**Todo** this needs some compile-time construction!

Definition at line 73 of file sf\_dawson.tcc.

Referenced by \_\_dawson().

**9.3.2.75** `template<typename _Tp> _Tp std::__detail::__dawson_series ( _Tp __x )`

Compute the Dawson integral using the series expansion.

Definition at line 49 of file sf\_dawson.tcc.

Referenced by \_\_dawson().

**9.3.2.76** `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 55 of file sf\_hankel.tcc.

Referenced by \_\_hankel().

**9.3.2.77** `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 196 of file sf\_zeta.tcc.

**9.3.2.78** `template<typename _Tp> _Tp std::__detail::__dirichlet_beta ( std::complex< _Tp> __w )`

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

## Parameters

$\_w$	The complex (but on-real-axis) argument.
-------	--

## Returns

The Dirichlet Beta function of real argument.

## Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1190 of file sf\_polylog.tcc.

References `__polylog()`.

**9.3.2.79** `template<typename _Tp> _Tp std::__detail::__dirichlet_beta ( _Tp __w )`

Return the Dirichlet beta function for real argument.

## Parameters

$\_w$	The real argument.
-------	--------------------

## Returns

The Dirichlet Beta function of real argument.

Definition at line 1209 of file sf\_polylog.tcc.

References `__polylog()`.

**9.3.2.80** `template<typename _Tp> std::complex<_Tp> std::__detail::__dirichlet_eta ( std::complex<_Tp> __w )`

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

## Parameters

$\_w$	The complex (but on-real-axis) argument.
-------	--

**Returns**

The complex Dirichlet eta function.

**Exceptions**

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1153 of file sf\_polylog.tcc.

References `__polylog()`.

Referenced by `__dirichlet_lambda()`.

**9.3.2.81** `template<typename _Tp> _Tp std::__detail::__dirichlet_eta ( _Tp __w )`

Return the Dirichlet eta function for real argument.

**Parameters**

<code>__w</code>	The real argument.
------------------	--------------------

**Returns**

The Dirichlet eta function.

Definition at line 1171 of file sf\_polylog.tcc.

References `__polylog()`.

**9.3.2.82** `template<typename _Tp> _Tp std::__detail::__dirichlet_lambda ( _Tp __w )`

Return the Dirichlet lambda function for real argument.

**Parameters**

<code>__w</code>	The real argument.
------------------	--------------------

**Returns**

The Dirichlet lambda function.

Definition at line 1226 of file sf\_polylog.tcc.

References `__dirichlet_eta()`, and `__riemann_zeta()`.



9.3.2.83 `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:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } -1!! = 10!! = 1$$

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 2994 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__factorial`, `__log_double_factorial()`, `std::__detail::__Factorial_table< _Tp >::__n`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

9.3.2.84 `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 597 of file sf\_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rf()`.

Referenced by `__heuman_lambda()`.

9.3.2.85 `template<typename _Tp > _Tp std::__detail::__ellint_2 ( _Tp __k, _Tp __phi )`

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

The incomplete elliptic integral of the second kind is defined as

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

## Parameters

<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 676 of file `sf_ellint.tcc`.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

**9.3.2.86** `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 771 of file `sf_ellint.tcc`.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

**9.3.2.87** `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 925 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.88 `template<typename _Tp> _Tp std::__detail::__ellint_d ( _Tp __k, _Tp __phi )`

Return the Legendre elliptic integral D.

Definition at line 812 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

9.3.2.89 `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 853 of file `sf_ellint.tcc`.

References `__ellint_rf()`.

9.3.2.90 `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 874 of file `sf_ellint.tcc`.

References `__ellint_rd()`, and `__ellint_rf()`.

9.3.2.91 `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 899 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.92 `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**

$\leftrightarrow$ _x	The first argument.
$\leftrightarrow$ _y	The second argument.

**Returns**

The Carlson elliptic function.

Definition at line 84 of file sf\_ellint.tcc.

Referenced by `__ellint_rf()`, and `__ellint_rj()`.

**9.3.2.93** `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**

$\leftrightarrow$ _x	The first of two symmetric arguments.
$\leftrightarrow$ _y	The second of two symmetric arguments.
$\leftrightarrow$ _z	The third argument.

**Returns**

The Carlson elliptic function of the second kind.

Definition at line 166 of file sf\_ellint.tcc.

Referenced by `__comp_ellint_2()`, `__comp_ellint_d()`, `__ellint_2()`, `__ellint_d()`, `__ellint_el2()`, `__ellint_rg()`, and `__ellint_rj()`.

**9.3.2.94** `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 280 of file `sf_ellint.tcc`.

References `__comp_ellint_rf()`, and `__ellint_rc()`.

Referenced by `__comp_ellint_2()`, `__comp_ellint_3()`, `__ellint_1()`, `__ellint_2()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el1()`, `__ellint_el2()`, `__ellint_el3()`, and `__heuman_lambda()`.

**9.3.2.95** `template<typename _Tp> _Tp std::__detail::__ellint_rg ( _Tp __x, _Tp __y, _Tp __z )`

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

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

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

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 411 of file sf\_ellint.tcc.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

9.3.2.96 `template<typename _Tp> _Tp std::__detail::__ellint_rj( _Tp __x, _Tp __y, _Tp __z, _Tp __p )`

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

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

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

Based on Carlson's algorithms:

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

## Parameters

$\_x$	The first of three symmetric arguments.
$\_y$	The second of three symmetric arguments.
$\_z$	The third of three symmetric arguments.
$\_p$	The fourth argument.

## Returns

The Carlson elliptic function of the fourth kind.

Definition at line 459 of file sf\_ellint.tcc.

References `__ellint_rc()`, and `__ellint_rd()`.

Referenced by `__comp_ellint_3()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el3()`, `__heuman_lambda()`, and `__jacobi_zeta()`.

### 9.3.2.97 `template<typename _Tp> _Tp std::__detail::__ellnome ( _Tp __k )`

Return the elliptic nome given the modulus  $k$ .

Definition at line 294 of file sf\_theta.tcc.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

### 9.3.2.98 `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 280 of file sf\_theta.tcc.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

### 9.3.2.99 `template<typename _Tp> _Tp std::__detail::__ellnome_series ( _Tp __k )`

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

Definition at line 264 of file sf\_theta.tcc.

Referenced by `__ellnome()`.

### 9.3.2.100 `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 472 of file sf\_expint.tcc.

References `__expint_E1()`, and `__expint_En_recursion()`.

Referenced by `__logint()`.

**9.3.2.101** `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 512 of file sf\_expint.tcc.

References `__expint_Ei()`.

**9.3.2.102** `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 405 of file sf\_expint.tcc.

9.3.2.103 `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 374 of file sf\_expint.tcc.

References `__expint_E1_asymp()`, `__expint_E1_series()`, `__expint_Ei()`, and `__expint_En_cont_frac()`.

Referenced by `__coshint()`, `__expint()`, `__expint_Ei()`, `__expint_En_recursion()`, and `__sinhint()`.

9.3.2.104 `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 113 of file sf\_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.105 `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 76 of file sf\_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.106 `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

$\leftrightarrow$	The argument of the exponential integral function.
$x$	

## Returns

The exponential integral.

Definition at line 350 of file sf\_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asyp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

9.3.2.107 `template<typename _Tp> _Tp std::__detail::__expint_Ei_asyp ( _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

$\leftrightarrow$	The argument of the exponential integral function.
$x$	

## Returns

The exponential integral.

Definition at line 317 of file sf\_expint.tcc.

Referenced by `__expint_Ei()`.

9.3.2.108 `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 285 of file sf\_expint.tcc.

Referenced by \_\_expint\_Ei().

9.3.2.109 `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 195 of file sf\_expint.tcc.

Referenced by \_\_expint\_E1().

9.3.2.110 `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 240 of file sf\_expint.tcc.

References `__expint_E1()`.

Referenced by `__expint()`.

9.3.2.111 `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 149 of file sf\_expint.tcc.

References `__psi()`.

9.3.2.112 `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

$\leftarrow$ _n	The order of the exponential integral function.
$\leftarrow$ _x	The argument of the exponential integral function.

#### Returns

The exponential integral.

Definition at line 439 of file sf\_expint.tcc.

**9.3.2.113** `template<typename _Tp> _Tp std::__detail::__exponential_cdf ( _Tp __lambda, _Tp __x )`

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

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

Definition at line 328 of file sf\_distributions.tcc.

**9.3.2.114** `template<typename _Tp> _Tp std::__detail::__exponential_pdf ( _Tp __lambda, _Tp __x )`

Return the exponential probability density function.

The formula for the exponential probability density function is

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

Definition at line 308 of file sf\_distributions.tcc.

**9.3.2.115** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial ( unsigned int __n )`

Return the factorial of the integer n.

The factorial is:

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

Definition at line 2936 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

9.3.2.116 `template<typename _Sp, typename _Tp> _Tp std::__detail::__fermi_dirac ( _Sp __s, _Tp __x )`

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

See also

[https://en.wikipedia.org/wiki/Clausen\\_function](https://en.wikipedia.org/wiki/Clausen_function)  
<http://dlmf.nist.gov/25.12.16>

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

Parameters

<code>__s</code>	The order s > -1.
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac cosine sum F\_s(x),

Definition at line 1400 of file sf\_polylog.tcc.

References `__polylog_exp()`.

9.3.2.117 `template<typename _Tp> _Tp std::__detail::__fisher_f_cdf ( _Tp __F, unsigned int __nu1, unsigned int __nu2 )`

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

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 446 of file sf\_distributions.tcc.

References `__beta_inc()`.

9.3.2.118 `template<typename _Tp> _Tp std::__detail::__fisher_f_cdfc ( _Tp __F, unsigned int __nu1, unsigned int __nu2 )`

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

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 475 of file `sf_distributions.tcc`.

References `__beta_inc()`.

9.3.2.119 `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 583 of file `sf_mod_bessel.tcc`.

9.3.2.120 `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

$\longleftrightarrow$	The argument
$x$	

Definition at line 170 of file sf\_fresnel.tcc.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

**9.3.2.121** `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 109 of file sf\_fresnel.tcc.

Referenced by `__fresnel()`.

**9.3.2.122** `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 51 of file sf\_fresnel.tcc.

Referenced by `__fresnel()`.

**9.3.2.123** `template<typename _Tp> _Tp std::__detail::__gamma ( _Tp __x )`

Return the gamma function  $\Gamma(x)$ . The gamma function is defined by:

$$\Gamma(a) = \int_0^\infty e^{-t}t^{a-1}dt(a > 0)$$

.

#### Parameters

$\longleftrightarrow$	The argument of the gamma function.
$x$	

#### Returns

The gamma function.

Definition at line 2302 of file sf\_gamma.tcc.

References `__log_gamma()`, and `__log_gamma_sign()`.

Referenced by `__beta_gamma()`, `__gamma_cdf()`, `__gamma_cdfc()`, `__gamma_pdf()`, `__gamma_temme()`, `__hurwitz_zeta_polylog()`, `__polylog_exp_neg_even()`, `__polylog_exp_pos()`, `__riemann_zeta()`, and `std::__detail::_Airy_series<_Tp>::__S_Scorer2()`.

9.3.2.124 `template<typename _Tp> _Tp std::__detail::__gamma_cdf ( _Tp __alpha, _Tp __beta, _Tp __x )`

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

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

Definition at line 141 of file sf\_distributions.tcc.

References `__gamma()`, and `__tgamma_lower()`.

9.3.2.125 `template<typename _Tp> _Tp std::__detail::__gamma_cdfc ( _Tp __alpha, _Tp __beta, _Tp __x )`

Return the gamma complementary cumulative propability distribution function.

The formula for the gamma probability density function is:

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

Definition at line 162 of file sf\_distributions.tcc.

References `__gamma()`, and `__tgamma()`.

9.3.2.126 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac ( _Tp __a, _Tp __x )`

Return the incomplete gamma function by continued fraction.

Definition at line 2357 of file sf\_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__pgamma()`, `__qgamma()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.2.127 `template<typename _Tp> _Tp std::__detail::__gamma_pdf ( _Tp __alpha, _Tp __beta, _Tp __x )`

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

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

Definition at line 121 of file sf\_distributions.tcc.

References `__gamma()`.

9.3.2.128 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_series ( _Tp __a, _Tp __x )`

Return the incomplete gamma function by series summation.

Definition at line 2317 of file sf\_gamma.tcc.

References `__log_gamma()`, and `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__pgamma()`, `__qgamma()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.2.129 `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 166 of file sf\_bessel.tcc.

References `__gamma()`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

**9.3.2.130** `template<typename _Tp> _Tp std::__detail::__gauss ( _Tp __x )`

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

Definition at line 70 of file sf\_owens\_t.tcc.

**9.3.2.131** `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 63 of file sf\_gegenbauer.tcc.

**9.3.2.132** `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.

## Parameters

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 1086 of file `sf_hankel.tcc`.

References `__debye_region()`, `__hankel_debye()`, and `__hankel_uniform()`.

Referenced by `__cyl_bessel()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann()`, and `__sph_hankel()`.

```
9.3.2.133  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 919 of file `sf_hankel.tcc`.

References `__sin_pi()`.

Referenced by `__hankel()`.

```
9.3.2.134  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 111 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_outer()`.

9.3.2.135 `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 864 of file `sf_hankel.tcc`.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

9.3.2.136 `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 778 of file `sf_hankel.tcc`.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

9.3.2.137 `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 249 of file `sf_hankel.tcc`.

References `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

9.3.2.138 `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>	
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 325 of file sf\_hankel.tcc.

Referenced by `__hankel_uniform_olver()`.

**9.3.2.139** `template<typename _Tp> _Tp std::__detail::__harmonic_number ( unsigned int __n )`

Definition at line 2730 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__n`, `_S_harmonic_denom`, `_S_harmonic_numer`, and `_S_num_↵harmonic_numer`.

**9.3.2.140** `template<typename _Tp> _Tp std::__detail::__heuman_lambda ( _Tp __k, _Tp __phi )`

Return the Heuman lambda function.

Definition at line 983 of file sf\_ellint.tcc.

References `__comp_ellint_1()`, `__ellint_1()`, `__ellint_rf()`, `__ellint_rj()`, and `__jacobi_zeta()`.

**9.3.2.141** `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 734 of file sf\_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`.

Referenced by `__psi()`.

**9.3.2.142** `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

$\leftrightarrow$ _s	The argument $s \neq 1$
$\leftrightarrow$ _a	The scale parameter $a > -1$

Definition at line 585 of file sf\_zeta.tcc.

References \_S\_Euler\_Maclaurin\_zeta.

Referenced by \_\_hurwitz\_zeta().

9.3.2.143 `template<typename _Tp> std::complex<_Tp> std::__detail::__hurwitz_zeta_polylog ( _Tp __s, std::complex<_Tp> __a )`

Return the Hurwitz Zeta function for real s and complex a. This uses Jonquiere's identity:

$$\frac{(i2\pi)^s}{\Gamma(s)} \zeta(a, 1-s) = Li_s(e^{i2\pi a}) + (-1)^s Li_s(e^{-i2\pi a})$$

Parameters

$\leftrightarrow$ _s	The real argument
$\leftrightarrow$ _a	The complex parameter

**Todo** This \_\_hurwitz\_zeta\_polylog prefactor is prone to overflow. positive integer orders s?

Definition at line 1117 of file sf\_polylog.tcc.

References \_\_gamma(), and \_\_polylog\_exp().

9.3.2.144 `template<typename _Tp> std::complex<_Tp> std::__detail::__hydrogen ( unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi )`

Return the bound-state Coulomb wave-function.

Definition at line 49 of file sf\_hydrogen.tcc.

References \_\_assoc\_laguerre(), \_\_log\_gamma(), \_\_psi(), and \_\_sph\_legendre().

9.3.2.145 `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

$\longleftrightarrow$ __a	The first <i>numerator</i> parameter.
$\longleftrightarrow$ __b	The second <i>numerator</i> parameter.
$\longleftrightarrow$ __c	The <i>denominator</i> parameter.
$\longleftrightarrow$ __x	The argument of the confluent hypergeometric function.

#### Returns

The confluent hypergeometric function.

Definition at line 778 of file sf\_hyperg.tcc.

References `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.2.146 `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 354 of file sf\_hyperg.tcc.

Referenced by `__hyperg()`.

9.3.2.147 `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 488 of file sf\_hyperrg.tcc.

References `__hyperrg_series()`, `__log_gamma()`, `__log_gamma_sign()`, and `__psi()`.

Referenced by `__hyperrg()`.

**9.3.2.148** `template<typename _Tp> _Tp std::__detail::__hyperrg_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 323 of file sf\_hyperrg.tcc.

Referenced by `__hyperrg()`, and `__hyperrg_reflect()`.

9.3.2.149 `template<typename _Tp> _Tp std::__detail::__ibeta_cont_frac ( _Tp __a, _Tp __b, _Tp __x )`

Return the regularized incomplete beta function,  $I_x(a, b)$ , of arguments `a`, `b`, and `x`.

## Parameters

$\longleftrightarrow$ _a	The first parameter
$\longleftrightarrow$ _b	The second parameter
$\longleftrightarrow$ _x	The argument

Definition at line 203 of file sf\_beta.tcc.

Referenced by \_\_beta\_inc().

9.3.2.150 `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 416 of file sf\_theta.tcc.

9.3.2.151 `template<typename _Tp> _Tp std::__detail::__jacobi_zeta ( _Tp __k, _Tp __phi )`

Return the Jacobi zeta function.

Definition at line 946 of file sf\_ellint.tcc.

References \_\_comp\_ellint\_1(), and \_\_ellint\_rj().

Referenced by \_\_heuman\_lambda().

9.3.2.152 `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

$\longleftrightarrow$ _n	The order of the Laguerre polynomial.
$\longleftrightarrow$ _x	The argument of the Laguerre polynomial.

**Returns**

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

Definition at line 323 of file sf\_laguerre.tcc.

**9.3.2.153** `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**

$\longleftrightarrow$ <code>__l</code>	The order of the Legendre function. $l \geq 0$ .
$\longleftrightarrow$ <code>__x</code>	The argument of the Legendre function. $ x  \leq 1$ .

Definition at line 126 of file sf\_legendre.tcc.

**9.3.2.154** `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:

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

The binomial coefficients are generated by:

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

.

**Parameters**

$\longleftrightarrow$ <code>__n</code>	The first argument of the binomial coefficient.
$\longleftrightarrow$ <code>__k</code>	The second argument of the binomial coefficient.

**Returns**

The logarithm of the binomial coefficient.

Definition at line 2140 of file sf\_gamma.tcc.

References `__log_gamma()`.

Referenced by `__bincoef()`.

9.3.2.155 `template<typename _Tp> _Tp std::__detail::__log_bincoef ( _Tp __nu, unsigned int __k )`

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

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

The binomial coefficients are generated by:

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

#### Parameters

<code>__nu</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 2171 of file sf\_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table< _Tp >::__n`.

9.3.2.156 `template<typename _Tp> _Tp std::__detail::__log_bincoef_sign ( _Tp __nu, unsigned int __k )`

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 2193 of file sf\_gamma.tcc.

References `__log_gamma_sign()`, and `std::__detail::_Factorial_table< _Tp >::__n`.

Referenced by `__bincoef()`.

9.3.2.157 `template<typename _Tp > std::complex<_Tp> std::__detail::__log_bincoef_sign ( std::complex<_Tp> __nu, unsigned int __k )`

Definition at line 2208 of file `sf_gamma.tcc`.

9.3.2.158 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial ( _Tp __x )`

Definition at line 2964 of file `sf_gamma.tcc`.

References `__cos_pi()`, and `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.2.159 `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 3030 of file `sf_gamma.tcc`.

References `__log_double_factorial()`, `std::__detail::_Factorial_table<_Tp>::__log_factorial`, `std::__detail::_Factorial_table<_Tp>::__n`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

9.3.2.160 `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 2954 of file `sf_gamma.tcc`.

References `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

9.3.2.161 `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

$\leftrightarrow$ _x	The argument of the log of the gamma function.
-------------------------	--

## Returns

The logarithm of the gamma function.

Definition at line 2051 of file sf\_gamma.tcc.

References `__log_gamma1p_spouge()`, and `__sin_pi()`.

Referenced by `__beta_inc()`, `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__gamma()`, `__gamma_cont_frac()`, `__gamma_series()`, `__hydrogen()`, `__hyperg()`, `__hyperg_reflect()`, `__log_bincoef()`, `__log_double_factorial()`, `__log_factorial()`, `__log_gamma()`, `__log_pochhammer()`, `__log_pochhammer_lower()`, `__pochhammer()`, `__pochhammer_lower()`, `__poly_laguerre_large_n()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, `__polylog_exp_neg_even()`, `__polylog_exp_neg_odd()`, `__polylog_exp_pos()`, `__psi()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, and `__sph_legendre()`.

**9.3.2.162** `template<typename _Tp> std::complex<_Tp> std::__detail::__log_gamma ( std::complex<_Tp> _x )`

Return  $\log(\Gamma(x))$  for complex argument.

## Parameters

$\leftrightarrow$ _x	The complex argument of the log of the gamma function.
-------------------------	--

## Returns

The complex logarithm of the gamma function.

Definition at line 2077 of file sf\_gamma.tcc.

References `__log_gamma()`, `__log_gamma1p_spouge()`, and `__sin_pi()`.

**9.3.2.163** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma1p_lanczos ( _Tp _z )`

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

## Parameters

$\leftrightarrow$ _x	The argument of the log of the gamma function.
-------------------------	--

**Returns**

The logarithm of the gamma function.

Definition at line 2011 of file sf\_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`, and `__sin_pi()`.

**9.3.2.164** `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma1p_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 1883 of file sf\_gamma.tcc.

References `__sin_pi()`.

Referenced by `__log_gamma()`.

**9.3.2.165** `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

$\leftrightarrow$	The argument of the log of the gamma function.
$x$	

## Returns

The logarithm of the gamma function.

Definition at line 1705 of file sf\_gamma.tcc.

9.3.2.166 `template<typename _Tp> _Tp std::__detail::__log_gamma_sign ( _Tp __x )`

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

## Parameters

$\leftrightarrow$	The argument of the gamma function.
$x$	

## Returns

The sign of the gamma function.

Definition at line 2107 of file sf\_gamma.tcc.

Referenced by `__beta_inc()`, `__beta_lgamma()`, `__gamma()`, `__hyperg()`, `__hyperg_reflect()`, `__log_bincoef_sign()`, and `__pochhammer_lower()`.

9.3.2.167 `template<typename _Tp> std::complex<_Tp> std::__detail::__log_gamma_sign ( std::complex<_Tp> __x )`

Definition at line 2119 of file sf\_gamma.tcc.

9.3.2.168 `template<typename _Tp> _Tp std::__detail::__log_pochhammer ( _Tp __a, _Tp __n )`

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

Definition at line 2549 of file sf\_gamma.tcc.

References `__log_gamma()`.

**9.3.2.169** `template<typename _Tp > _Tp std::__detail::__log_pochhammer_lower ( _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,  $f[(n)_n = n!]$ . Thus this function returns

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

Many notations exist:

, 
$$a^n$$

, 
$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

Definition at line 2614 of file sf\_gamma.tcc.

References `__log_gamma()`.

**9.3.2.170** `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

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

#### Returns

The logarithmic integral.

Definition at line 533 of file sf\_expint.tcc.

References `__expint()`.

9.3.2.171 `template<typename _Tp> _Tp std::__detail::__lognormal_cdf ( _Tp __mu, _Tp __sigma, _Tp __x )`

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

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

Definition at line 287 of file `sf_distributions.tcc`.

9.3.2.172 `template<typename _Tp> _Tp std::__detail::__lognormal_pdf ( _Tp __nu, _Tp __sigma, _Tp __x )`

Return the lognormal probability density function.

The formula for the lognormal probability density function is

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

Definition at line 259 of file `sf_distributions.tcc`.

9.3.2.173 `template<typename _Tp> _Tp std::__detail::__normal_cdf ( _Tp __mu, _Tp __sigma, _Tp __x )`

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

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

Definition at line 238 of file `sf_distributions.tcc`.

9.3.2.174 `template<typename _Tp> _Tp std::__detail::__normal_pdf ( _Tp __nu, _Tp __sigma, _Tp __x )`

Return the normal probability density function.

The formula for the normal probability density function is

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

Definition at line 210 of file `sf_distributions.tcc`.

9.3.2.175 `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()`.

**9.3.2.176** `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 2410 of file sf\_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdf()`.

**9.3.2.177** `template<typename _Tp> _Tp std::__detail::__pochhammer ( _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:

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

,

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

, and others.

Definition at line 2575 of file sf\_gamma.tcc.

References `__log_gamma()`.

9.3.2.178 `template<typename _Tp> _Tp std::__detail::__pochhammer_lower ( _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 2637 of file `sf_gamma.tcc`.

References `__log_gamma()`, and `__log_gamma_sign()`.

9.3.2.179 `template<typename _Tp> std::complex<_Tp> std::__detail::__polar_pi ( _Tp __rho, _Tp __phi_pi ) [inline]`

Reperiodized complex constructor.

Definition at line 404 of file `sf_trig.tcc`.

References `__sincos_pi()`, `__gnu_cxx::__sincos_t<_Tp>::cos_value`, and `__gnu_cxx::__sincos_t<_Tp>::sin_value`.

9.3.2.180 `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

$\leftrightarrow$ __n	The order of the Hermite polynomial.
$\leftrightarrow$ __x	The argument of the Hermite polynomial.

#### Returns

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

Definition at line 181 of file `sf_hermite.tcc`.

References `__poly_hermite_asymp()`, and `__poly_hermite_recursion()`.

9.3.2.181 `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 115 of file `sf_hermite.tcc`.

References `__airy()`.

Referenced by `__poly_hermite()`.

9.3.2.182 `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 71 of file `sf_hermite.tcc`.



Referenced by `__poly_hermite()`.

**9.3.2.183** `template<typename _Tp> _Tp std::__detail::__poly_jacobi ( unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x )`

Compute the Jacobi polynomial by recursion on  $n$  :

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

Definition at line 59 of file `sf_jacobi.tcc`.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

**9.3.2.184** `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  $\alpha$ :  $L_n^{\alpha}lpha(x)$ .

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 250 of file `sf_laguerre.tcc`.

References `__poly_laguerre_hyperg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

9.3.2.185 `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 131 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

9.3.2.186 `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 74 of file `sf_laguerre.tcc`.

References `__log_gamma()`, and `__sin_pi()`.

Referenced by `__poly_laguerre()`.

9.3.2.187 `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 189 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

**9.3.2.188** `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 76 of file `sf_legendre.tcc`.

Referenced by `__assoc_legendre_p()`, and `__sph_legendre()`.

**9.3.2.189** `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

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

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

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative  $m, n$ .

## See also

zernike for details on the Zernike polynomials.

Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

## Template Parameters

<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 144 of file sf\_jacobi.tcc.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyl()`.

**9.3.2.190** `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 1065 of file sf\_polylog.tcc.

References `__polylog_exp()`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, and `__polylog()`.

**9.3.2.191** `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

$\leftarrow$ _s	The real index.
$\leftarrow$ _w	The complex argument.

## Returns

The complex value of the polylogarithm.

Definition at line 1095 of file sf\_polylog.tcc.

References `__polylog()`, and `__polylog_exp()`.

```
9.3.2.192 template<typename _Tp , typename ArgType > __gnu_cxx::__promote_fp_t<std::complex<_Tp>, ArgType>
std::__detail::__polylog_exp ( _Tp _s, ArgType _w )
```

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

## Note

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

## Parameters

$\leftarrow$ _s	The real order.
$\leftarrow$ _w	The real or complex argument.

## Returns

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

Definition at line 1032 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()`, `__polylog()`, and `__polylog()`.

9.3.2.193 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_asyp ( _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  $Re(w) \gg 1$

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 653 of file sf\_polylog.tcc.

References `__log_gamma()`.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

9.3.2.194 `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

<code>__s</code>	a negative integer.
<code>__w</code>	an arbitrary complex number

#### Returns

the value of the polylogarithm,.

Definition at line 837 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()`.

**9.3.2.195** `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

<code>__s</code>	a negative integer.
<code>__w</code>	the argument.

#### Returns

the value of the polylogarithm.

Definition at line 882 of file sf\_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

**9.3.2.196** `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

<code>__s</code>	a positive integer.
<code>__w</code>	an arbitrary complex number.

#### Returns

The value of the polylogarithm.

Definition at line 739 of file sf\_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.



9.3.2.197 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos ( unsigned int __s, _Tp __w )`

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

#### Parameters

$\_s$	a positive integer
$\_w$	an arbitrary real argument $w$

#### Returns

the value of the polylogarithm.

Definition at line 792 of file `sf_polylog.tcc`.

References `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

9.3.2.198 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg ( _Tp __s, std::complex<_Tp> __w )`

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

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

$$A_p(w) = \sum_k \Gamma(1+k-s)/k! \sin(\pi/2(s-k)) \left(\frac{w}{2\pi}\right)^k \zeta(1+k-s)$$

#### Parameters

$\_s$	The real index
$\_w$	The complex argument

#### Returns

The value of the polylogarithm.

Definition at line 292 of file `sf_polylog.tcc`.

References `__log_gamma()`, `__riemann_zeta()`, and `__riemann_zeta_m_1()`.

Referenced by `__polylog_exp_int_neg()`, and `__polylog_exp_real_neg()`.

9.3.2.199 `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

$\leftrightarrow$ _s	the integer index s.
$\leftrightarrow$ _w	The Argument w

## Returns

The value of the Polylogarithm evaluated by a suitable function.

Definition at line 529 of file sf\_polylog.tcc.

References `__polylog_exp_neg_even()`, and `__polylog_exp_neg_odd()`.

9.3.2.200 `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 occurring 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

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

and

$$B_p(w) = -2(2\pi)^{p-1} \sum_{k=0}^{\infty} \frac{\Gamma(2+2k-p)}{(2k+1)!} (-1)^k \left(\frac{w}{2\pi}\right)^{2k+1} (\zeta(2+2k-p) - 1)$$

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)

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

## Parameters

$\leftrightarrow$ _n	the integral index $n = 4k$ .
$\leftrightarrow$ _w	The complex argument w

**Returns**

the value of the Polylogarithm.

Definition at line 406 of file sf\_polylog.tcc.

References `__gamma()`, and `__log_gamma()`.

Referenced by `__polylog_exp_neg()`.

9.3.2.201 `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 occurring 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

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

and

$$B_p(w) = 2(2\pi)^{p-1} \sum_{k=0}^{\infty} \frac{\Gamma(1+2k-p)}{(2k)!} \left(\frac{-w^2}{4\pi^2}\right)^k (\zeta(1+2k-p) - 1)$$

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)

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

**Parameters**

<code>__n</code>	the integral index $n = 4k$ .
<code>__w</code>	The complex argument $w$ .

**Returns**

The value of the Polylogarithm.

Definition at line 480 of file sf\_polylog.tcc.

References `__log_gamma()`.

Referenced by `__polylog_exp_neg()`.

9.3.2.202 `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:

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

#### 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 707 of file `sf_polylog.tcc`.

Referenced by `__polylog_exp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

9.3.2.203 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos ( unsigned int __s, std::complex<_Tp> __w )`

This function treats the cases of positive integer index  $s$ .

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

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

#### Parameters

<code>__s</code>	the index $s$ .
<code>__w</code>	the argument $w$ .

**Returns**

the value of the polylogarithm.

Definition at line 142 of file sf\_polylog.tcc.

References `__riemann_zeta()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

**9.3.2.204** `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  $\log(x)$ .

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

The radius of convergence is  $|w| < 2\pi$ . Note that this series involves a  $\log(-x)$ . 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**

<code>__s</code>	the index.
<code>__w</code>	the argument

**Returns**

the value of the Polylogarithm

Definition at line 220 of file sf\_polylog.tcc.

References `__riemann_zeta()`.

**9.3.2.205** `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos ( _Tp __s, std::complex<_Tp> __w )`

This function treats the cases of positive real index  $s$ .

The defining series is

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

with

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

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

## Parameters

$\_s$	the positive real index s.
$\_w$	The complex argument w.

## Returns

the value of the polylogarithm.

Definition at line 568 of file sf\_polylog.tcc.

References `__gamma()`, `__log_gamma()`, and `__riemann_zeta()`.

9.3.2.206 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg ( _Tp __s, std::complex<_Tp> __w )`

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

## Parameters

$\_s$	A negative real value that does not reduce to a negative integer.
$\_w$	The complex argument.

## Returns

The value of the polylogarithm.

Definition at line 977 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()`.

9.3.2.207 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg ( _Tp __s, _Tp __w )`

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

## Parameters

$\_s$	A negative real value.
$\_w$	A real argument.

## Returns

The value of the polylogarithm.

Definition at line 1006 of file sf\_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

9.3.2.208 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos ( _Tp __s, std::complex<_Tp> __w )`

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

## Parameters

$\_s$	A positive real number.
$\_w$	the complex argument.

## Returns

The value of the polylogarithm.

Definition at line 909 of file sf\_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

9.3.2.209 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos ( _Tp __s, _Tp __w )`

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

## Parameters

$\_s$	A positive real number tht does not reduce to an integer.
$\_w$	The real argument w.



## Returns

The value of the polylogarithm.

Definition at line 946 of file sf\_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

### 9.3.2.210 `template<typename _Tp> _Tp std::__detail::__psi ( unsigned int __n )`

Return the digamma function of integral argument. The digamma or  $\psi(x)$  function is defined as the logarithmic derivative of the gamma function:

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

The digamma series for integral argument is given by:

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

The latter sum is called the harmonic number,  $H_n$ .

Definition at line 2761 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__expint_En_series()`, `__hydrogen()`, `__hyperg_reflect()`, and `__psi()`.

### 9.3.2.211 `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 2844 of file sf\_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, `__psi()`, and `__psi_asymp()`.

### 9.3.2.212 `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:

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

Definition at line 2909 of file sf\_gamma.tcc.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

9.3.2.213 `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 2813 of file `sf_gamma.tcc`.

Referenced by `__psi()`.

9.3.2.214 `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 2782 of file `sf_gamma.tcc`.

9.3.2.215 `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 2443 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdfc()`.

9.3.2.216 `template<typename _Tp> _Tp std::__detail::__rice_pdf ( _Tp __nu, _Tp __sigma, _Tp __x )`

Return the Rice probability density function.

The formula for the Rice probability density function is

$$p(x|\nu, \sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2 + \nu^2}{2\sigma^2}\right) I_0\left(\frac{x\nu}{\sigma^2}\right)$$

where  $I_0(x)$  is the modified Bessel function of the first kind of order 0 and  $\nu \geq 0$  and  $\sigma > 0$ .

Definition at line 186 of file `sf_distributions.tcc`.

References `__cyl_bessel_i()`.

9.3.2.217 `template<typename _Tp> _Tp std::__detail::__riemann_zeta ( _Tp __s )`

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

The Riemann zeta function is defined by:

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

#### Parameters

<code>__s</code>	The argument
------------------	--------------

Definition at line 528 of file `sf_zeta.tcc`.

References `__gamma()`, `__log_gamma()`, `__riemann_zeta_glob()`, `__riemann_zeta_product()`, `__riemann_zeta_sum()`, and `__sin_pi()`.

Referenced by `__dirichlet_lambda()`, `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__polylog_exp_real_pos()`, and `evenzeta()`.

9.3.2.218 `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 series is:

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

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 346 of file sf\_zeta.tcc.

**9.3.2.219** `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 290 of file sf\_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

**9.3.2.220** `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 394 of file sf\_zeta.tcc.

References `__log_gamma()`, and `__sin_pi()`.

Referenced by `__riemann_zeta()`.

**9.3.2.221** `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1 ( _Tp __s )`

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

**Parameters**

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

Definition at line 702 of file sf\_zeta.tcc.

References `__riemann_zeta_m_1_sum()`, `_S_num_zetam1`, and `_S_zetam1`.

Referenced by `__polylog_exp_neg()`.

9.3.2.222 `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:

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

#### Parameters

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

Definition at line 673 of file sf\_zeta.tcc.

Referenced by `__riemann_zeta_m_1()`.

9.3.2.223 `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:

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

For  $s < 1$  use the reflection formula:

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

#### Parameters

<code>__s</code>	The argument
------------------	--------------

Definition at line 488 of file sf\_zeta.tcc.

Referenced by `__riemann_zeta()`.

9.3.2.224 `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 256 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta()`.

9.3.2.225 `template<typename _Tp> _Tp std::__detail::__sin_pi ( _Tp __x )`

Return the reperiodized sine of argument x:

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

Definition at line 69 of file `sf_trig.tcc`.

References `__sin_pi()`.

Referenced by `__cyl_bessel_jn()`, `__hankel_debye()`, `__log_gamma()`, `__log_gamma1p_lanczos()`, `__log_gamma1p←_spouge()`, `__poly_laguerre_large_n()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, and `__sinc_pi()`.

9.3.2.226 `template<typename _Tp> std::complex<_Tp> std::__detail::__sin_pi ( std::complex<_Tp> __z )`

Return the reperiodized sine of complex argument z:

$$\sin_{\pi}(z) = \sin(\pi z) = \sin_{\pi}(x) \cosh_{\pi}(y) + i \cos_{\pi}(x) \sinh_{\pi}(y)$$

Definition at line 199 of file `sf_trig.tcc`.

References `__cos_pi()`.

Referenced by `__cos_pi()`, `__cosh_pi()`, `__sin_pi()`, and `__sinh_pi()`.

9.3.2.227 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc ( _Tp __x )`

Return the sinus cardinal function

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

.

Definition at line 51 of file `sf_cardinal.tcc`.

9.3.2.228 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc_pi ( _Tp __x )`

Return the reperiodized sinus cardinal function

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

.

Definition at line 71 of file sf\_cardinal.tcc.

References `__sin_pi()`.

9.3.2.229 `template<typename _Tp> __gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos ( _Tp __x ) [inline]`

Definition at line 319 of file sf\_trig.tcc.

9.3.2.230 `template<> __gnu_cxx::__sincos_t<float> std::__detail::__sincos ( float __x ) [inline]`

Definition at line 327 of file sf\_trig.tcc.

9.3.2.231 `template<> __gnu_cxx::__sincos_t<double> std::__detail::__sincos ( double __x ) [inline]`

Definition at line 339 of file sf\_trig.tcc.

9.3.2.232 `template<> __gnu_cxx::__sincos_t<long double> std::__detail::__sincos ( long double __x ) [inline]`

Definition at line 351 of file sf\_trig.tcc.

Referenced by `__sincos_pi()`.

9.3.2.233 `template<typename _Tp> __gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos_pi ( _Tp __x )`

Reperiodized sincos.

Definition at line 363 of file sf\_trig.tcc.

References `__sincos()`, `__gnu_cxx::__sincos_t<_Tp>::cos_value`, and `__gnu_cxx::__sincos_t<_Tp>::sin_value`.

Referenced by `__polar_pi()`.

**9.3.2.234** `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 231 of file `sf_trigint.tcc`.

References `__sincosint_asyp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

**9.3.2.235** `template<typename _Tp> void std::__detail::__sincosint_asyp ( _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 166 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

**9.3.2.236** `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 58 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

**9.3.2.237** `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 101 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.



9.3.2.238 `template<typename _Tp> _Tp std::__detail::__sinh_pi ( _Tp __x )`

Return the reperiodized hyperbolic sine of argument x:

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

Definition at line 100 of file sf\_trig.tcc.

References `__sinh_pi()`.

Referenced by `__sinhc_pi()`.

9.3.2.239 `template<typename _Tp> std::complex<_Tp> std::__detail::__sinh_pi ( std::complex<_Tp> __z )`

Return the reperiodized hyperbolic sine of complex argument z:

$$\sinh_{\pi}(z) = \sinh(\pi z) = \sinh(\pi x) \cos_{\pi}(y) + i \cosh(\pi x) \sin_{\pi}(y)$$

Definition at line 220 of file sf\_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

Referenced by `__sinh_pi()`.

9.3.2.240 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc ( _Tp __x )`

Return the hyperbolic sinus cardinal function

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

.

Definition at line 96 of file sf\_cardinal.tcc.

9.3.2.241 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc_pi ( _Tp __x )`

Return the reperiodized hyperbolic sinus cardinal function

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

.

Definition at line 114 of file sf\_cardinal.tcc.

References `__sinh_pi()`.

9.3.2.242 `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 579 of file sf\_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

**9.3.2.243** `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 775 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`.

**9.3.2.244** `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 1273 of file sf\_hankel.tcc.

References `__sph_hankel()`.

9.3.2.245 `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<math>\leftrightarrow</math>_n</code>	The output derivative of the regular modified spherical Bessel function.
<code>__kp<math>\leftrightarrow</math>_n</code>	The output derivative of the irregular modified spherical Bessel function.

Definition at line 459 of file sf\_mod\_bessel.tcc.

References `__cyl_bessel_ik()`.

9.3.2.246 `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<math>\leftrightarrow</math>_n</code>	The output derivative of the spherical Bessel function.
out	<code>__np<math>\leftrightarrow</math>_n</code>	The output derivative of the spherical Neumann function.

Definition at line 708 of file sf\_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
9.3.2.247 template<typename _Tp> void std::__detail::__sph_bessel_jn_neg_arg ( unsigned int __n, _Tp __x, std::complex<_Tp>
    > & __j_n, std::complex<_Tp> & __n_n, std::complex<_Tp> & __jp_n, std::complex<_Tp> & __np_n )
```

Return the spherical Bessel functions and their derivatives of order  $\nu$  and argument  $x < 0$ .

Definition at line 733 of file sf\_bessel.tcc.

References `__cyl_bessel_jn_neg_arg()`.

Referenced by `__sph_hankel_1()`, and `__sph_hankel_2()`.

```
9.3.2.248 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 1217 of file sf\_hankel.tcc.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
9.3.2.249 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

$\leftrightarrow$ _n	The order of the spherical Neumann function.
$\leftrightarrow$ _x	The argument of the spherical Neumann function.

## Returns

The output spherical Neumann function.

Definition at line 844 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

9.3.2.250 `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 1241 of file sf\_hankel.tcc.

References `__sph_hankel()`.

9.3.2.251 `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 880 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

```
9.3.2.252 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 1257 of file sf\_hankel.tcc.

References `__sph_hankel()`.

```
9.3.2.253 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

<code>__l</code>	The order of the spherical harmonic function. $l \geq 0$ .
<code>__m</code>	The order of the spherical harmonic function. $m \leq l$ .
<code>__theta</code>	The radian polar angle argument of the spherical harmonic function.
<code>__phi</code>	The radian azimuthal angle argument of the spherical harmonic function.

Definition at line 353 of file `sf_legendre.tcc`.

References `__sph_legendre()`.

9.3.2.254 `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 256 of file `sf_legendre.tcc`.

References `__log_gamma()`, and `__poly_legendre_p()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

9.3.2.255 `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**

$\leftrightarrow$ _n	The order of the spherical Neumann function.
$\leftrightarrow$ _x	The argument of the spherical Neumann function.

**Returns**

The output spherical Neumann function.

Definition at line 812 of file sf\_bessel.tcc.

References `__sph_bessel_jn()`.

```
9.3.2.256  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 1289 of file sf\_hankel.tcc.

References `__sph_hankel()`.

```
9.3.2.257  template<typename _Tp > _Tp std::__detail::__student_t_cdf ( _Tp __t, unsigned int __nu )
```

Return the Students T probability function.

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

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



## Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 397 of file sf\_distributions.tcc.

References `__beta_inc()`.

9.3.2.258 `template<typename _Tp> _Tp std::__detail::__student_t_cdfc ( _Tp __t, unsigned int __nu )`

Return the complement of the Students T probability function.

The complement of the students T propability function is:

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

## Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 420 of file sf\_distributions.tcc.

References `__beta_inc()`.

9.3.2.259 `template<typename _Tp> _Tp std::__detail::__tan_pi ( _Tp __x )`

Return the reperiodized tangent of argument x:

$$\tan_p i(x) = \tan(\pi x)$$

Definition at line 166 of file sf\_trig.tcc.

9.3.2.260 `template<typename _Tp> std::complex<_Tp> std::__detail::__tan_pi ( std::complex<_Tp> __z )`

Return the reperiodized tangent of complex argument z:

$$\tan_\pi(z) = \tan(\pi z) = \frac{\tan_\pi(x) + i \tanh_\pi(y)}{1 - i \tan_\pi(x) \tanh_\pi(y)}$$

Definition at line 279 of file sf\_trig.tcc.

Referenced by `__tanh_pi()`.

9.3.2.261 `template<typename _Tp> _Tp std::__detail::__tanh_pi ( _Tp __x )`

Return the reperiodized hyperbolic tangent of argument x:

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

Definition at line 182 of file sf\_trig.tcc.

9.3.2.262 `template<typename _Tp> std::complex<_Tp> std::__detail::__tanh_pi ( std::complex<_Tp> __z )`

Return the reperiodized hyperbolic tangent of complex argument z:

$$\tanh_{\pi}(z) = \tanh(\pi z) = \frac{\tanh_{\pi}(x) + i \tan_{\pi}(y)}{1 + i \tanh_{\pi}(x) \tan_{\pi}(y)}$$

Definition at line 301 of file sf\_trig.tcc.

References `__tan_pi()`.

9.3.2.263 `template<typename _Tp> _Tp std::__detail::__tgamma ( _Tp __a, _Tp __x )`

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

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

.

Definition at line 2505 of file sf\_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__gamma_cdfc()`.

9.3.2.264 `template<typename _Tp> _Tp std::__detail::__tgamma_lower ( _Tp __a, _Tp __x )`

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

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

.

Definition at line 2471 of file sf\_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__gamma_cdf()`.

9.3.2.265 `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 192 of file sf\_theta.tcc.

References `__theta_2()`.

Referenced by `__theta_s()`.

9.3.2.266 `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 164 of file sf\_theta.tcc.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.

9.3.2.267 `template<typename _Tp> _Tp std::__detail::__theta_2_asymp ( _Tp __nu, _Tp __x )`

Compute and return the  $\theta_2$  function by series expansion.

Definition at line 105 of file sf\_theta.tcc.

Referenced by `__theta_2()`.

9.3.2.268 `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 51 of file sf\_theta.tcc.

Referenced by `__theta_2()`.

9.3.2.269 `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 218 of file `sf_theta.tcc`.

References `__theta_3_asyp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

9.3.2.270 `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 130 of file `sf_theta.tcc`.

Referenced by `__theta_3()`.

9.3.2.271 `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 79 of file `sf_theta.tcc`.

Referenced by `__theta_3()`.

9.3.2.272 `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 246 of file sf\_theta.tcc.

References `__theta_3()`.

Referenced by `__theta_n()`.

**9.3.2.273** `template<typename _Tp> _Tp std::__detail::__theta_c ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_c$  function

Definition at line 339 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

**9.3.2.274** `template<typename _Tp> _Tp std::__detail::__theta_d ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_d$  function

Definition at line 364 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

**9.3.2.275** `template<typename _Tp> _Tp std::__detail::__theta_n ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_n$  function

Definition at line 389 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

**9.3.2.276** `template<typename _Tp> _Tp std::__detail::__theta_s ( _Tp __k, _Tp __x )`

Return the Neville  $\theta_s$  function

Definition at line 313 of file sf\_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

9.3.2.277 `template<typename _Tp> _Tp std::__detail::__weibull_cdf ( _Tp __a, _Tp __b, _Tp __x )`

Return the Weibull cumulative probability density function.

The formula for the Weibull cumulative probability density function is

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

Definition at line 373 of file `sf_distributions.tcc`.

9.3.2.278 `template<typename _Tp> _Tp std::__detail::__weibull_pdf ( _Tp __a, _Tp __b, _Tp __x )`

Return the Weibull probability density function.

The formula for the Weibull probability density function is

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

Definition at line 352 of file `sf_distributions.tcc`.

9.3.2.279 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__zernike ( unsigned int __n, int __m, _Tp __rho, _Tp __phi )`

Return the Zernicke polynomial  $Z_n^m(\rho, \phi)$  for non-negative integral degree  $n$ , signed integral order  $m$ , and real radial argument  $\rho$  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](#)).

Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

#### Template Parameters

<code>_Tp</code>	The real type of the radial coordinate and azimuthal angle
------------------	--

## Parameters

<code>__n</code>	The non-negative integral degree.
<code>__m</code>	The integral azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 193 of file `sf_jacobi.tcc`.

References `__poly_radial_jacobi()`.

**9.3.2.280** `template<typename _Tp> _Tp std::__detail::__znorm1 ( _Tp __x )`

Definition at line 58 of file `sf_owens_t.tcc`.

Referenced by `__owens_t()`.

**9.3.2.281** `template<typename _Tp> _Tp std::__detail::__znorm2 ( _Tp __x )`

Definition at line 47 of file `sf_owens_t.tcc`.

Referenced by `__owens_t()`.

**9.3.2.282** `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 92 of file `sf_polylog.tcc`.

References `__riemann_zeta()`.

### 9.3.3 Variable Documentation

**9.3.3.1** `template<typename _Tp> constexpr int std::__detail::__max_FGH = _Airy_series<_Tp>::__N_FGH`

Definition at line 178 of file `sf_airy.tcc`.

9.3.3.2 `template<> constexpr int std::__detail::__max_FGH< double > = 79`

Definition at line 184 of file `sf_airy.tcc`.

9.3.3.3 `template<> constexpr int std::__detail::__max_FGH< float > = 15`

Definition at line 181 of file `sf_airy.tcc`.

9.3.3.4 `constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100`

Coefficients for Euler-Maclaurin summation of zeta functions.

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

where  $B_k$  are the Bernoulli numbers.

Definition at line 67 of file `sf_zeta.tcc`.

9.3.3.5 `constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]`

Definition at line 277 of file `sf_gamma.tcc`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.3.6 `constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]`

Definition at line 70 of file `sf_zeta.tcc`.

Referenced by `__hurwitz_zeta_euler_maclaurin()`, and `__riemann_zeta_euler_maclaurin()`.

9.3.3.7 `constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]`

Definition at line 87 of file `sf_gamma.tcc`.

Referenced by `__factorial()`, and `__log_factorial()`.

9.3.3.8 `constexpr unsigned long long std::__detail::_S_harmonic_denom[_S_num_harmonic_number]`

Definition at line 2696 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.



9.3.3.9 constexpr unsigned long long std::\_\_detail::\_S\_harmonic\_numer[\_S\_num\_harmonic\_numer]

Definition at line 2663 of file sf\_gamma.tcc.

Referenced by \_\_harmonic\_number().

9.3.3.10 constexpr \_Factorial\_table<long double> std::\_\_detail::\_S\_neg\_double\_factorial\_table[999]

Definition at line 598 of file sf\_gamma.tcc.

Referenced by \_\_double\_factorial(), and \_\_log\_double\_factorial().

9.3.3.11 template<typename \_Tp > constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials = 0

Definition at line 262 of file sf\_gamma.tcc.

9.3.3.12 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< double > = 301

Definition at line 267 of file sf\_gamma.tcc.

9.3.3.13 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< float > = 57

Definition at line 265 of file sf\_gamma.tcc.

9.3.3.14 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_double\_factorials< long double > = 301

Definition at line 269 of file sf\_gamma.tcc.

9.3.3.15 template<typename \_Tp > constexpr std::size\_t std::\_\_detail::\_S\_num\_factorials = 0

Definition at line 72 of file sf\_gamma.tcc.

9.3.3.16 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_factorials< double > = 171

Definition at line 77 of file sf\_gamma.tcc.

9.3.3.17 template<> constexpr std::size\_t std::\_\_detail::\_S\_num\_factorials< float > = 35

Definition at line 75 of file sf\_gamma.tcc.

9.3.3.18 `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 79 of file `sf_gamma.tcc`.

9.3.3.19 `constexpr unsigned long long std::__detail::_S_num_harmonic_numer = 29`

Definition at line 2660 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.

9.3.3.20 `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 582 of file `sf_gamma.tcc`.

9.3.3.21 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 587 of file `sf_gamma.tcc`.

9.3.3.22 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 585 of file `sf_gamma.tcc`.

9.3.3.23 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 589 of file `sf_gamma.tcc`.

9.3.3.24 `constexpr size_t std::__detail::_S_num_zetam1 = 33`

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

Definition at line 620 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

9.3.3.25 `constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]`

Definition at line 624 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

## Chapter 10

# Class Documentation

### 10.1 `__gnu_cxx::__sincos_t<_Tp>` Struct Template Reference

#### Public Attributes

- `_Tp cos_value`
- `_Tp sin_value`

#### 10.1.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__sincos_t<_Tp>
```

A struct to store a cosine and a sine value.

Definition at line 45 of file `sf_trig.tcc`.

#### 10.1.2 Member Data Documentation

10.1.2.1 `template<typename _Tp> _Tp __gnu_cxx::__sincos_t<_Tp>::cos_value`

Definition at line 48 of file `sf_trig.tcc`.

Referenced by `std::__detail::__polar_pi()`, and `std::__detail::__sincos_pi()`.

10.1.2.2 `template<typename _Tp> _Tp __gnu_cxx::__sincos_t<_Tp>::sin_value`

Definition at line 47 of file `sf_trig.tcc`.

Referenced by `std::__detail::__polar_pi()`, and `std::__detail::__sincos_pi()`.

The documentation for this struct was generated from the following file:

- `bits/sf_trig.tcc`

## 10.2 std::\_\_detail::\_Airy< \_Tp > Class Template Reference

### Public Types

- using [scalar\\_type](#) = std::\_\_detail::\_\_num\_traits\_t< [value\\_type](#) >
- using [value\\_type](#) = \_Tp

### Public Member Functions

- constexpr [\\_Airy](#) ()=default
- [\\_Airy](#) (const [\\_Airy](#) &)=default
- [\\_Airy](#) ([\\_Airy](#) &&)=default
- constexpr [\\_AiryState](#)< [value\\_type](#) > operator() ([value\\_type](#) \_\_y) const

### Public Attributes

- [scalar\\_type](#) inner\_radius { [\\_Airy\\_default\\_radII](#)<[scalar\\_type](#)>::inner\_radius}
- [scalar\\_type](#) outer\_radius { [\\_Airy\\_default\\_radII](#)<[scalar\\_type](#)>::outer\_radius}

### Static Public Attributes

- static constexpr [scalar\\_type](#) [\\_S\\_2pi\\_3](#) = [scalar\\_type](#){2} \* [\\_S\\_pi\\_3](#)
- static constexpr [scalar\\_type](#) [\\_S\\_5pi\\_6](#) = [scalar\\_type](#){5} \* [\\_S\\_pi\\_6](#)
- static constexpr auto [\\_S\\_cNaN](#) = [value\\_type](#)([\\_S\\_NaN](#), [\\_S\\_NaN](#))
- static constexpr [value\\_type](#) [\\_S\\_i](#) = [value\\_type](#){0, 1}
- static constexpr auto [\\_S\\_NaN](#) = \_\_gnu\_cxx::\_\_quiet\_NaN<[scalar\\_type](#)>()
- static constexpr [scalar\\_type](#) [\\_S\\_pi](#) = \_\_gnu\_cxx::\_\_math\_constants<[scalar\\_type](#)>::\_\_pi
- static constexpr [scalar\\_type](#) [\\_S\\_pi\\_3](#) = \_\_gnu\_cxx::\_\_math\_constants<[scalar\\_type](#)>::\_\_pi\_third
- static constexpr [scalar\\_type](#) [\\_S\\_pi\\_6](#) = [\\_S\\_pi\\_3](#) / [scalar\\_type](#){2}
- static constexpr [scalar\\_type](#) [\\_S\\_sqrt\\_pi](#) = \_\_gnu\_cxx::\_\_math\_constants<[scalar\\_type](#)>::\_\_root\_pi

### 10.2.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_Airy< _Tp >
```

Class to manage the asymptotic expansions for Airy functions. The parameters describing the various regions are adjustable.

Definition at line 2498 of file sf\_airy.tcc.

## 10.2.2 Member Typedef Documentation

10.2.2.1 `template<typename _Tp> using std::__detail::_Airy<_Tp>::scalar_type = std::__detail::__num_traits_t<value_type>`

Definition at line 2503 of file sf\_airy.tcc.

10.2.2.2 `template<typename _Tp> using std::__detail::_Airy<_Tp>::value_type = _Tp`

Definition at line 2502 of file sf\_airy.tcc.

## 10.2.3 Constructor & Destructor Documentation

10.2.3.1 `template<typename _Tp> constexpr std::__detail::_Airy<_Tp>::_Airy( ) [default]`

10.2.3.2 `template<typename _Tp> std::__detail::_Airy<_Tp>::_Airy( const _Airy<_Tp> & ) [default]`

10.2.3.3 `template<typename _Tp> std::__detail::_Airy<_Tp>::_Airy( _Airy<_Tp> && ) [default]`

## 10.2.4 Member Function Documentation

10.2.4.1 `template<typename _Tp> constexpr _AiryState<_Tp> std::__detail::_Airy<_Tp>::operator()( value_type __y ) const`

Return the Airy functions for complex argument.

Definition at line 2550 of file sf\_airy.tcc.

References `std::__detail::__beta()`, `std::__detail::_Airy_series<_Tp>::_S_Ai()`, and `std::__detail::_Airy_series<_Tp>::_S_Bi()`.

## 10.2.5 Member Data Documentation

10.2.5.1 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_2pi_3 = scalar_type{2} * _S_pi_3 [static]`

Definition at line 2510 of file sf\_airy.tcc.

10.2.5.2 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_5pi_6 = scalar_type{5} * _S_pi_6 [static]`

Definition at line 2512 of file sf\_airy.tcc.

10.2.5.3 `template<typename _Tp> constexpr auto std::__detail::_Airy<_Tp>::_S_cNaN = value_type(_S_NaN, _S_NaN)`  
`[static]`

Definition at line 2516 of file `sf_airy.tcc`.

10.2.5.4 `template<typename _Tp> constexpr _Airy<_Tp>::value_type std::__detail::_Airy<_Tp>::_S_i =`  
`value_type{0, 1} [static]`

Definition at line 2513 of file `sf_airy.tcc`.

10.2.5.5 `template<typename _Tp> constexpr auto std::__detail::_Airy<_Tp>::_S_NaN =`  
`_gnu_cxx::__quiet_NaN<scalar_type>() [static]`

Definition at line 2515 of file `sf_airy.tcc`.

10.2.5.6 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_pi =`  
`_gnu_cxx::__math_constants<scalar_type>::_pi [static]`

Definition at line 2505 of file `sf_airy.tcc`.

10.2.5.7 `template<typename _Tp> constexpr _Airy<_Tp>::scalar_type std::__detail::_Airy<_Tp>::_S_pi_3 =`  
`_gnu_cxx::__math_constants<scalar_type>::_pi_third [static]`

Definition at line 2509 of file `sf_airy.tcc`.

10.2.5.8 `template<typename _Tp> constexpr _Airy<_Tp>::scalar_type std::__detail::_Airy<_Tp>::_S_pi_6 = _S_pi_3`  
`/ scalar_type{2} [static]`

Definition at line 2511 of file `sf_airy.tcc`.

10.2.5.9 `template<typename _Tp> constexpr _Airy<_Tp>::scalar_type std::__detail::_Airy<_Tp>::_S_sqrt_pi =`  
`_gnu_cxx::__math_constants<scalar_type>::_root_pi [static]`

Definition at line 2507 of file `sf_airy.tcc`.

10.2.5.10 `template<typename _Tp> scalar_type std::__detail::_Airy<_Tp>::inner_radius`  
`{_Airy_default_radii<scalar_type>::inner_radius}`

Definition at line 2525 of file `sf_airy.tcc`.

10.2.5.11 `template<typename _Tp> scalar_type std::__detail::_Airy< _Tp >::outer_radius  
{ _Airy_default_radii<scalar_type>::outer_radius }`

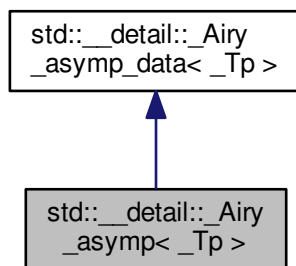
Definition at line 2526 of file `sf_airy.tcc`.

The documentation for this class was generated from the following file:

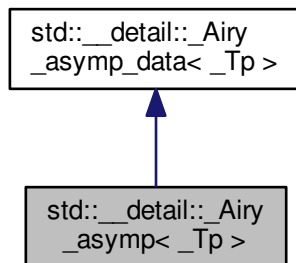
- [bits/sf\\_airy.tcc](#)

## 10.3 std::\_\_detail::\_Airy\_asymp< \_Tp > Class Template Reference

Inheritance diagram for `std::__detail::_Airy_asymp< _Tp >`:



Collaboration diagram for `std::__detail::_Airy_asymp< _Tp >`:



## Public Types

- using `__cmplx` = `std::complex< _Tp >`

## Public Member Functions

- constexpr `_Airy_asymp` ()=default
- `_AiryState`< `std::complex< _Tp >` > `_S_absarg_ge_pio3` (`std::complex< _Tp > __z`) const  
*This function evaluates  $Ai(z)$ ,  $Ai'(z)$  and  $Bi(z)$ ,  $Bi'(z)$  from their asymptotic expansions for  $|arg(z)| < 2 * \pi/3$  i.e. roughly along the negative real axis.*
- `_AiryState`< `std::complex< _Tp >` > `_S_absarg_lt_pio3` (`std::complex< _Tp > __z`) const  
*This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|arg(-z)| < \pi/3$  i.e. roughly along the negative real axis.*
- `_AiryState`< `std::complex< _Tp >` > `operator()` (`std::complex< _Tp > __t`, bool `__return_fock_airy=false`) const

### 10.3.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_Airy_asymp< _Tp >
```

A class encapsulating the asymptotic expansions of Airy functions and thier derivatives.

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 1999 of file `sf_airy.tcc`.

### 10.3.2 Member Typedef Documentation

10.3.2.1 `template<typename _Tp > using std::__detail::_Airy_asymp< _Tp >::__cmplx = std::complex<_Tp>`

Definition at line 2004 of file `sf_airy.tcc`.

### 10.3.3 Constructor & Destructor Documentation

10.3.3.1 `template<typename _Tp > constexpr std::__detail::_Airy_asymp< _Tp >::_Airy_asymp( ) [default]`

### 10.3.4 Member Function Documentation

10.3.4.1 `template<typename _Tp > _AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::__S_absarg_ge_pio3( std::complex< _Tp > __z ) const`

This function evaluates  $Ai(z)$ ,  $Ai'(z)$  and  $Bi(z)$ ,  $Bi'(z)$  from their asymptotic expansions for  $|arg(z)| < 2 * \pi/3$  i.e. roughly along the negative real axis.



## Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

## Parameters

in	<code>_z</code>	Complex argument at which $Ai(z)$ and $Bi(z)$ and their derivative are evaluated. This function assumes $ z  > 15$ and $ \arg(z)  < 2\pi/3$ .
----	-----------------	---

## Returns

A struct containing  $z$ ,  $Ai(z)$ ,  $Ai'(z)$ ,  $Bi(z)$ ,  $Bi'(z)$ .

Definition at line 2272 of file sf\_airy.tcc.

10.3.4.2 `template<typename _Tp> _AiryState< std::complex<_Tp> > > std::__detail::_Airy_asymp<_Tp> >::_S_absarg_lt_pio3 ( std::complex<_Tp> _z ) const`

This function evaluates  $Ai(z)$  and  $Ai'(z)$  from their asymptotic expansions for  $|\arg(-z)| < \pi/3$  i.e. roughly along the negative real axis.

For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined for  $|z|$ . This function assumes  $|z| > 15$  and  $|\arg(-z)| < \pi/3$ .

Note that for speed and since this function is called by another, checks for valid arguments are not made. Hence, an error return is not needed.

## Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

## Parameters

in	<code>_z</code>	The value at which the Airy function and their derivatives are evaluated.
----	-----------------	---

## Returns

A struct containing  $z$ ,  $Ai(z)$ ,  $Ai'(z)$ ,  $Bi(z)$ ,  $Bi'(z)$ .

**Todo** Revisit these numbers of terms for the Airy asymptotic expansions.

Definition at line 2302 of file sf\_airy.tcc.

10.3.4.3 `template<typename _Tp> _AiryState< std::complex< _Tp> > std::__detail::_Airy_asymp< _Tp>::operator() ( std::complex< _Tp> __t, bool __return_fock_airy = false ) const`

Return the Airy functions for a given argument using asymptotic series.

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

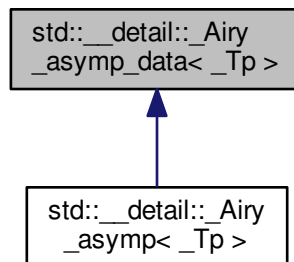
Definition at line 2030 of file `sf_airy.tcc`.

The documentation for this class was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.4 `std::__detail::_Airy_asymp_data< _Tp>` Struct Template Reference

Inheritance diagram for `std::__detail::_Airy_asymp_data< _Tp>`:



### 10.4.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Airy_asymp_data< _Tp>
```

A class encapsulating data for the asymptotic expansions of Airy functions and thier derivatives.

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 633 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.5 std::\_\_detail::\_Airy\_asymp\_data< double > Struct Template Reference

### Static Public Attributes

- static constexpr double [\\_S\\_c](#) [[\\_S\\_max\\_cd](#)]
- static constexpr double [\\_S\\_d](#) [[\\_S\\_max\\_cd](#)]
- static constexpr int [\\_S\\_max\\_cd](#) = 198

### 10.5.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< double >
```

Definition at line 740 of file sf\_airy.tcc.

### 10.5.2 Member Data Documentation

10.5.2.1 constexpr double std::\_\_detail::\_Airy\_asymp\_data< double >::\_S\_c[[\\_S\\_max\\_cd](#)] `[static]`

Definition at line 746 of file sf\_airy.tcc.

10.5.2.2 constexpr double std::\_\_detail::\_Airy\_asymp\_data< double >::\_S\_d[[\\_S\\_max\\_cd](#)] `[static]`

Definition at line 949 of file sf\_airy.tcc.

10.5.2.3 constexpr int std::\_\_detail::\_Airy\_asymp\_data< double >::\_S\_max\_cd = 198 `[static]`

Definition at line 742 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.6 `std::__detail::_Airy_asymp_data< float >` Struct Template Reference

### Static Public Attributes

- static constexpr float `_S_c` [`_S_max_cd`]
- static constexpr float `_S_d` [`_S_max_cd`]
- static constexpr int `_S_max_cd` = 43

### 10.6.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< float >
```

Definition at line 637 of file `sf_airy.tcc`.

### 10.6.2 Member Data Documentation

10.6.2.1 `constexpr float std::__detail::_Airy_asymp_data< float >::_S_c[_S_max_cd]` [static]

Definition at line 643 of file `sf_airy.tcc`.

10.6.2.2 `constexpr float std::__detail::_Airy_asymp_data< float >::_S_d[_S_max_cd]` [static]

Definition at line 691 of file `sf_airy.tcc`.

10.6.2.3 `constexpr int std::__detail::_Airy_asymp_data< float >::_S_max_cd = 43` [static]

Definition at line 639 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `bits/sf_airy.tcc`

## 10.7 `std::__detail::_Airy_asymp_data< long double >` Struct Template Reference

### Static Public Attributes

- static constexpr long double `_S_c` [`_S_max_cd`]
- static constexpr long double `_S_d` [`_S_max_cd`]
- static constexpr int `_S_max_cd` = 201

### 10.7.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< long double >
```

Definition at line 1153 of file sf\_airy.tcc.

### 10.7.2 Member Data Documentation

10.7.2.1 constexpr long double std::\_\_detail::\_Airy\_asymp\_data< long double >::\_S\_c[\_S\_max\_cd] [static]

Definition at line 1159 of file sf\_airy.tcc.

10.7.2.2 constexpr long double std::\_\_detail::\_Airy\_asymp\_data< long double >::\_S\_d[\_S\_max\_cd] [static]

Definition at line 1365 of file sf\_airy.tcc.

10.7.2.3 constexpr int std::\_\_detail::\_Airy\_asymp\_data< long double >::\_S\_max\_cd = 201 [static]

Definition at line 1155 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.8 std::\_\_detail::\_Airy\_asymp\_series<\_Sum> Class Template Reference

### Public Types

- using [scalar\\_type](#) = std::\_\_detail::\_\_num\_traits\_t< [value\\_type](#) >
- using [value\\_type](#) = typename \_Sum::value\_type

### Public Member Functions

- [\\_Airy\\_asymp\\_series](#) (\_Sum \_\_proto)
- [\\_Airy\\_asymp\\_series](#) (const [\\_Airy\\_asymp\\_series](#) &)=default
- [\\_Airy\\_asymp\\_series](#) ([\\_Airy\\_asymp\\_series](#) &&)=default
- [\\_AiryState](#)< [value\\_type](#) > [operator\(\)](#) ([value\\_type](#) \_\_y)

### Static Public Attributes

- static constexpr [scalar\\_type](#) [\\_S\\_sqrt\\_pi](#) = \_\_gnu\_cxx::\_\_math\_constants<[scalar\\_type](#)>::\_\_root\_pi

### 10.8.1 Detailed Description

```
template<typename _Sum>
class std::__detail::_Airy_asymp_series<_Sum>
```

Class to manage the asymptotic series for Airy functions.

### Template Parameters

<code>_Sum</code>	A sum type
-------------------	------------

Definition at line 2365 of file `sf_airy.tcc`.

## 10.8.2 Member Typedef Documentation

10.8.2.1 `template<typename _Sum> using std::__detail::__Airy_asymp_series< _Sum >::scalar_type = std::__detail::__num_traits_t<value_type>`

Definition at line 2370 of file `sf_airy.tcc`.

10.8.2.2 `template<typename _Sum> using std::__detail::__Airy_asymp_series< _Sum >::value_type = typename _Sum::value_type`

Definition at line 2369 of file `sf_airy.tcc`.

## 10.8.3 Constructor & Destructor Documentation

10.8.3.1 `template<typename _Sum> std::__detail::__Airy_asymp_series< _Sum >::__Airy_asymp_series ( _Sum __proto ) [inline]`

Definition at line 2374 of file `sf_airy.tcc`.

10.8.3.2 `template<typename _Sum> std::__detail::__Airy_asymp_series< _Sum >::__Airy_asymp_series ( const _Airy_asymp_series< _Sum > & ) [default]`

10.8.3.3 `template<typename _Sum> std::__detail::__Airy_asymp_series< _Sum >::__Airy_asymp_series ( _Airy_asymp_series< _Sum > && ) [default]`

## 10.8.4 Member Function Documentation

10.8.4.1 `template<typename _Sum> _AiryState< typename _Airy_asymp_series< _Sum >::value_type > std::__detail::__Airy_asymp_series< _Sum >::operator() ( value_type __y )`

Return an [\\_AiryState](#) containing, not actual Airy functions, but four asymptotic Airy components:

### Template Parameters

<code>_Sum</code>	A sum type
-------------------	------------

Definition at line 2419 of file sf\_airy.tcc.

### 10.8.5 Member Data Documentation

10.8.5.1 `template<typename _Sum> constexpr _Airy_asymp_series< _Sum >::scalar_type std::__detail::_Airy_asymp_series< _Sum >::_S_sqrt_pi = __gnu_cxx::__math_constants<scalar_type>::_root_pi`  
[static]

Definition at line 2372 of file sf\_airy.tcc.

The documentation for this class was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.9 std::\_\_detail::\_Airy\_default\_radii< \_Tp > Struct Template Reference

### 10.9.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Airy_default_radii< _Tp >
```

Definition at line 2469 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.10 std::\_\_detail::\_Airy\_default\_radii< double > Struct Template Reference

### Static Public Attributes

- static constexpr double [inner\\_radius](#) {4.0}
- static constexpr double [outer\\_radius](#) {12.0}

### 10.10.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< double >
```

Definition at line 2480 of file sf\_airy.tcc.

## 10.10.2 Member Data Documentation

10.10.2.1 `constexpr double std::__detail::_Airy_default_radii< double >::inner_radius {4.0} [static]`

Definition at line 2482 of file `sf_airy.tcc`.

10.10.2.2 `constexpr double std::__detail::_Airy_default_radii< double >::outer_radius {12.0} [static]`

Definition at line 2483 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.11 `std::__detail::_Airy_default_radii< float >` Struct Template Reference

### Static Public Attributes

- static constexpr float [inner\\_radius](#) {2.0F}
- static constexpr float [outer\\_radius](#) {6.0F}

### 10.11.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< float >
```

Definition at line 2473 of file `sf_airy.tcc`.

## 10.11.2 Member Data Documentation

10.11.2.1 `constexpr float std::__detail::_Airy_default_radii< float >::inner_radius {2.0F} [static]`

Definition at line 2475 of file `sf_airy.tcc`.

10.11.2.2 `constexpr float std::__detail::_Airy_default_radii< float >::outer_radius {6.0F} [static]`

Definition at line 2476 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)



## 10.12 std::\_\_detail::\_Airy\_default\_radii< long double > Struct Template Reference

### Static Public Attributes

- static constexpr long double [inner\\_radius](#) {5.0L}
- static constexpr long double [outer\\_radius](#) {15.0L}

### 10.12.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< long double >
```

Definition at line 2487 of file sf\_airy.tcc.

### 10.12.2 Member Data Documentation

10.12.2.1 constexpr long double std::\_\_detail::\_Airy\_default\_radii< long double >::inner\_radius {5.0L} [static]

Definition at line 2489 of file sf\_airy.tcc.

10.12.2.2 constexpr long double std::\_\_detail::\_Airy\_default\_radii< long double >::outer\_radius {15.0L} [static]

Definition at line 2490 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf\_airy.tcc

## 10.13 std::\_\_detail::\_Airy\_series< \_Tp > Class Template Reference

### Static Public Member Functions

- static std::pair< std::complex< \_Tp >, std::complex< \_Tp > > [\\_S\\_Ai](#) (std::complex< \_Tp > \_\_t)
- static [\\_AiryState](#)< std::complex< \_Tp > > [\\_S\\_Airy](#) (std::complex< \_Tp > \_\_t)
- static std::pair< std::complex< \_Tp >, std::complex< \_Tp > > [\\_S\\_Bi](#) (std::complex< \_Tp > \_\_t)
- static [\\_AiryAuxilliaryState](#)< std::complex< \_Tp > > [\\_S\\_FGH](#) (std::complex< \_Tp > \_\_t)
- static [\\_AiryState](#)< std::complex< \_Tp > > [\\_S\\_Fock](#) (std::complex< \_Tp > \_\_t)
- static [\\_AiryState](#)< std::complex< \_Tp > > [\\_S\\_Scorer](#) (std::complex< \_Tp > \_\_t)
- static [\\_AiryState](#)< std::complex< \_Tp > > [\\_S\\_Scorer2](#) (std::complex< \_Tp > \_\_t)

## Static Public Attributes

- static constexpr int `_N_FGH` = 200
- static constexpr `_Tp` `_S_Ai0` = `_Tp{3.550280538878172392600631860041831763980e-1L}`
- static constexpr `_Tp` `_S_Aip0` = `_Tp{-2.588194037928067984051835601892039634793e-1L}`
- static constexpr `_Tp` `_S_Bi0` = `_Tp{6.149266274460007351509223690936135535960e-1L}`
- static constexpr `_Tp` `_S_Bip0` = `_Tp{4.482883573538263579148237103988283908668e-1L}`
- static constexpr `_Tp` `_S_eps` = `__gnu_cxx::__epsilon(_Tp{})`
- static constexpr `_Tp` `_S_Gi0` = `_Tp{2.049755424820002450503074563645378511979e-1L}`
- static constexpr `_Tp` `_S_Gip0` = `_Tp{1.494294524512754526382745701329427969551e-1L}`
- static constexpr `_Tp` `_S_Hi0` = `_Tp{4.099510849640004901006149127290757023959e-1L}`
- static constexpr `_Tp` `_S_Hip0` = `_Tp{2.988589049025509052765491402658855939102e-1L}`
- static constexpr `__cmplx` `_S_i` { `_Tp{0}`, `_Tp{1}` }
- static constexpr `_Tp` `_S_pi` = `__gnu_cxx::__math_constants<_Tp>::__pi`
- static constexpr `_Tp` `_S_sqrt_pi` = `__gnu_cxx::__math_constants<_Tp>::__root_pi`

### 10.13.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_Airy_series< _Tp >
```

This class organizes series solutions of the Airy function.

$$f_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$g_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$h_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

This class contains tabulations of the factors appearing in the sums above.

Definition at line 108 of file `sf_airy.tcc`.

### 10.13.2 Member Function Documentation

10.13.2.1 `template<typename _Tp> std::pair< std::complex< _Tp>, std::complex< _Tp> > std::__detail::_Airy_series< _Tp>::_S_Ai( std::complex< _Tp> _t ) [static]`

Return the Airy function of the first kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$f_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$g_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the first kind is then defined by:

$$Ai(x) = Ai(0)f_{ai}(x) + Ai'(0)g_{ai}(x)$$

where  $Ai(0) = 3^{-2/3}/\Gamma(2/3)$ ,  $Ai'(0) = -3^{-1/2}Bi'(0)$  and  $Bi(0) = 3^{1/2}Ai(0)$ ,  $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

## Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 340 of file sf\_airy.tcc.

Referenced by std::\_\_detail::\_Airy<\_Tp>::operator()().

10.13.2.2 `template<typename _Tp> _AiryState< std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::S_Airy (std::complex<_Tp> __t) [static]`

Return the Fock-type Airy functions  $Ai(t)$ , and  $Bi(t)$  and their derivatives of complex argument.

## Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

## Parameters

$\leftarrow$	The complex argument
<code>__t</code>	
$\leftarrow$	
<code>__t</code>	
$t$	

Definition at line 610 of file sf\_airy.tcc.

10.13.2.3 `template<typename _Tp> std::pair< std::complex<_Tp>, std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::S_Bi ( std::complex<_Tp> __t) [static]`

Return the Airy function of the second kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the second kind is then defined by:

$$Bi(x) = Bi(0)fai(x) + Bi'(0)gai(x)$$

where  $Ai(0) = 3^{-2/3}/\Gamma(2/3)$ ,  $Ai'(0) = -3-1/2Bi'(0)$  and  $Bi(0) = 3^{1/2}Ai(0)$ ,  $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

## Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 363 of file sf\_airy.tcc.

Referenced by `std::__detail::_Airy<_Tp>::operator()()`.

10.13.2.4 `template<typename _Tp> _AiryAuxilliaryState< std::complex<_Tp> > > std::__detail::_Airy_series<_Tp>::__S_FGH( std::complex<_Tp> __t ) [static]`

Return the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 382 of file sf\_airy.tcc.

10.13.2.5 `template<typename _Tp> _AiryState< std::complex<_Tp> > > std::__detail::_Airy_series<_Tp>::__S_Fock( std::complex<_Tp> __t ) [static]`

Return the Fock-type Airy functions  $w_1(t)$ , and  $w_2(t)$  and their derivatives of complex argument.

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

#### Parameters

$\leftarrow$	The complex argument
<code>__</code> $\leftarrow$	
$\leftarrow$	
<code>__</code> $\leftarrow$	
$t$	

Definition at line 622 of file sf\_airy.tcc.

10.13.2.6 `template<typename _Tp> _AiryState< std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::_S_Scorer  
( std::complex<_Tp> __t ) [static]`

Return the Scorer functions by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

The Scorer function is then defined by:

$$Hi(x) = Hi(0) (fai(x) + gai(x) + hai(x))$$

where  $Hi(0) = 2/(3^{7/6}\Gamma(2/3))$  and  $Hi'(0) = 2/(3^{5/6}\Gamma(1/3))$ . The other Scorer function is found from the identity

$$Gi(x) + Hi(x) = Bi(x)$$

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

#### Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 464 of file sf\_airy.tcc.

10.13.2.7 `template<typename _Tp> _AiryState< std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::_S_Scorer2( std::complex<_Tp> __t ) [static]`

Return the Scorer functions by using the series expansions:

$$Hi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Hi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k-1}{3}\pi\right) \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k+1}{3}\pi\right) \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

Definition at line 501 of file sf\_airy.tcc.

References `std::__detail::_gamma()`.

### 10.13.3 Member Data Documentation

10.13.3.1 `template<typename _Tp > constexpr int std::__detail::_Airy_series<_Tp>::_N_FGH = 200` `[static]`

Definition at line 113 of file `sf_airy.tcc`.

10.13.3.2 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Ai0 =`  
`_Tp{3.550280538878172392600631860041831763980e-1L}` `[static]`

Definition at line 129 of file `sf_airy.tcc`.

10.13.3.3 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Aip0 =`  
`_Tp{-2.588194037928067984051835601892039634793e-1L}` `[static]`

Definition at line 131 of file `sf_airy.tcc`.

10.13.3.4 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Bi0 =`  
`_Tp{6.149266274460007351509223690936135535960e-1L}` `[static]`

Definition at line 133 of file `sf_airy.tcc`.

10.13.3.5 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Bip0 =`  
`_Tp{4.482883573538263579148237103988283908668e-1L}` `[static]`

Definition at line 135 of file `sf_airy.tcc`.

10.13.3.6 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_eps =`  
`_gnu_cxx::__epsilon(_Tp{})` `[static]`

Definition at line 124 of file `sf_airy.tcc`.

10.13.3.7 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gi0 =`  
`_Tp{2.049755424820002450503074563645378511979e-1L}` `[static]`

Definition at line 141 of file `sf_airy.tcc`.

10.13.3.8 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gip0 =`  
`_Tp{1.494294524512754526382745701329427969551e-1L}` `[static]`

Definition at line 143 of file `sf_airy.tcc`.

10.13.3.9 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hi0 =  
_Tp{4.099510849640004901006149127290757023959e-1L} [static]`

Definition at line 137 of file sf\_airy.tcc.

10.13.3.10 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hip0 =  
_Tp{2.988589049025509052765491402658855939102e-1L} [static]`

Definition at line 139 of file sf\_airy.tcc.

10.13.3.11 `template<typename _Tp > constexpr std::complex< _Tp > std::__detail::_Airy_series< _Tp >::_S_i {_Tp{0},  
_Tp{1}} [static]`

Definition at line 144 of file sf\_airy.tcc.

10.13.3.12 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_pi =  
__gnu_cxx::__math_constants<_Tp>::_pi [static]`

Definition at line 125 of file sf\_airy.tcc.

10.13.3.13 `template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_sqrt_pi =  
__gnu_cxx::__math_constants<_Tp>::_root_pi [static]`

Definition at line 127 of file sf\_airy.tcc.

The documentation for this class was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.14 std::\_\_detail::\_AiryAuxilliaryState< \_Tp > Struct Template Reference

### Public Types

- using [\\_Val](#) = std::\_\_detail::\_\_num\_traits\_t< \_Tp >

### Public Attributes

- [\\_Tp fai](#)
- [\\_Tp faip](#)
- [\\_Tp gai](#)
- [\\_Tp gaip](#)
- [\\_Tp hai](#)
- [\\_Tp haip](#)
- [\\_Tp z](#)

### 10.14.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_AiryAuxilliaryState< _Tp >
```

A structure containing three auxilliary Airy functions and their derivatives.

Definition at line 80 of file sf\_airy.tcc.

### 10.14.2 Member Typedef Documentation

10.14.2.1 `template<typename _Tp> using std::__detail::_AiryAuxilliaryState< _Tp >::_Val = std::__detail::_num_traits_t<_Tp>`

Definition at line 82 of file sf\_airy.tcc.

### 10.14.3 Member Data Documentation

10.14.3.1 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::_fai`

Definition at line 85 of file sf\_airy.tcc.

10.14.3.2 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::_faip`

Definition at line 86 of file sf\_airy.tcc.

10.14.3.3 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::_gai`

Definition at line 87 of file sf\_airy.tcc.

10.14.3.4 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::_gaip`

Definition at line 88 of file sf\_airy.tcc.

10.14.3.5 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::_hai`

Definition at line 89 of file sf\_airy.tcc.



10.14.3.6 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::haip`

Definition at line 90 of file `sf_airy.tcc`.

10.14.3.7 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::z`

Definition at line 84 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `bits/sf_airy.tcc`

## 10.15 std::\_\_detail::\_AiryState< \_Tp > Struct Template Reference

### Public Types

- using `_Val` = `std::__detail::__num_traits_t< _Tp >`

### Public Member Functions

- `constexpr _Tp Wronskian () const`

### Static Public Member Functions

- `static constexpr _Val true_Wronskian ()`

### Public Attributes

- `_Tp Ai`
- `_Tp Aip`
- `_Tp Bi`
- `_Tp Bip`
- `_Tp z`

#### 10.15.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_AiryState< _Tp >
```

This struct defines the Airy function state with two presumably numerically useful Airy functions and their derivatives. The data members are directly accessible. The lone method computes the Wronskian from the stored functions. A static method returns the correct Wronskian.

Definition at line 55 of file `sf_airy.tcc`.

## 10.15.2 Member Typedef Documentation

10.15.2.1 `template<typename _Tp> using std::__detail::_AiryState< _Tp >::_Val = std::__detail::_num_traits_t<_Tp>`

Definition at line 57 of file `sf_airy.tcc`.

## 10.15.3 Member Function Documentation

10.15.3.1 `template<typename _Tp> static constexpr _Val std::__detail::_AiryState< _Tp >::true_Wronskian ( )`  
`[inline], [static]`

Definition at line 70 of file `sf_airy.tcc`.

10.15.3.2 `template<typename _Tp> constexpr _Tp std::__detail::_AiryState< _Tp >::Wronskian ( ) const` `[inline]`

Definition at line 66 of file `sf_airy.tcc`.

References `std::__detail::_AiryState< _Tp >::Aip`.

## 10.15.4 Member Data Documentation

10.15.4.1 `template<typename _Tp> _Tp std::__detail::_AiryState< _Tp >::Ai`

Definition at line 60 of file `sf_airy.tcc`.

10.15.4.2 `template<typename _Tp> _Tp std::__detail::_AiryState< _Tp >::Aip`

Definition at line 61 of file `sf_airy.tcc`.

Referenced by `std::__detail::_AiryState< _Tp >::Wronskian()`.

10.15.4.3 `template<typename _Tp> _Tp std::__detail::_AiryState< _Tp >::Bi`

Definition at line 62 of file `sf_airy.tcc`.

10.15.4.4 `template<typename _Tp> _Tp std::__detail::_AiryState< _Tp >::Bip`

Definition at line 63 of file `sf_airy.tcc`.

## 10.15.4.5 template&lt;typename \_Tp&gt; \_Tp std::\_\_detail::\_AiryState&lt;\_Tp&gt;::z

Definition at line 59 of file sf\_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_airy.tcc](#)

## 10.16 std::\_\_detail::\_Factorial\_table<\_Tp> Struct Template Reference

### Public Attributes

- [\\_Tp \\_\\_factorial](#)
- [\\_Tp \\_\\_log\\_factorial](#)
- [int \\_\\_n](#)

### 10.16.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Factorial_table<_Tp>
```

Definition at line 64 of file sf\_gamma.tcc.

### 10.16.2 Member Data Documentation

## 10.16.2.1 template&lt;typename \_Tp&gt; \_Tp std::\_\_detail::\_Factorial\_table&lt;\_Tp&gt;::\_\_factorial

Definition at line 67 of file sf\_gamma.tcc.

Referenced by std::\_\_detail::\_\_double\_factorial().

## 10.16.2.2 template&lt;typename \_Tp&gt; \_Tp std::\_\_detail::\_Factorial\_table&lt;\_Tp&gt;::\_\_log\_factorial

Definition at line 68 of file sf\_gamma.tcc.

Referenced by std::\_\_detail::\_\_log\_double\_factorial().

### 10.16.2.3 `template<typename _Tp> int std::__detail::_Factorial_table<_Tp>::_n`

Definition at line 66 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::__harmonic_number()`, `std::__detail::__log_bincoef()`, `std::__detail::__log_bincoef_sign()`, `std::__detail::__log_double_factorial()`, `std::__detail::__log_factorial()`, `std::__detail::__log_gamma1p_lanczos()`, and `std::__detail::__psi()`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)

## 10.17 `std::__detail::_GammaLanczos<_Tp>` Struct Template Reference

### 10.17.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_GammaLanczos<_Tp>
```

A struct for Lanczos algorithm Chebyshev arrays of coefficients.

Definition at line 1914 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)

## 10.18 `std::__detail::_GammaLanczos<double>` Struct Template Reference

### Static Public Attributes

- static constexpr `std::array<double, 10>` `_S_cheby`
- static constexpr `double` `_S_g` = 9.5

### 10.18.1 Detailed Description

```
template<>
struct std::__detail::_GammaLanczos<double>
```

Definition at line 1936 of file `sf_gamma.tcc`.

## 10.18.2 Member Data Documentation

10.18.2.1 constexpr std::array<double, 10> std::\_\_detail::\_GammaLanczos< double >::\_S\_cheby [static]

**Initial value:**

```
{
    5.557569219204146e+03,
    -4.248114953727554e+03,
    1.881719608233706e+03,
    -4.705537221412237e+02,
    6.325224688788239e+01,
    -4.206901076213398e+00,
    1.202512485324405e-01,
    -1.141081476816908e-03,
    2.055079676210880e-06,
    1.280568540096283e-09,
}
```

Definition at line 1941 of file sf\_gamma.tcc.

10.18.2.2 constexpr double std::\_\_detail::\_GammaLanczos< double >::\_S\_g = 9.5 [static]

Definition at line 1938 of file sf\_gamma.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)

## 10.19 std::\_\_detail::\_GammaLanczos< float > Struct Template Reference

### Static Public Attributes

- static constexpr std::array< float, 7 > [\\_S\\_cheby](#)
- static constexpr float [\\_S\\_g](#) = 6.5F

### 10.19.1 Detailed Description

```
template<>
struct std::__detail::_GammaLanczos< float >
```

Definition at line 1919 of file sf\_gamma.tcc.

## 10.19.2 Member Data Documentation

10.19.2.1 `constexpr std::array<float, 7> std::__detail::_GammaLanczos< float >::_S_cheby` `[static]`

**Initial value:**

```
{
    3.307139e+02F,
    -2.255998e+02F,
    6.989520e+01F,
    -9.058929e+00F,
    4.110107e-01F,
    -4.150391e-03F,
    -3.417969e-03F,
}
```

Definition at line 1924 of file `sf_gamma.tcc`.

10.19.2.2 `constexpr float std::__detail::_GammaLanczos< float >::_S_g = 6.5F` `[static]`

Definition at line 1921 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)

## 10.20 `std::__detail::_GammaLanczos< long double >` Struct Template Reference

### Static Public Attributes

- static `constexpr std::array< long double, 11 > _S_cheby`
- static `constexpr long double _S_g = 10.5L`

### 10.20.1 Detailed Description

```
template<>
struct std::__detail::_GammaLanczos< long double >
```

Definition at line 1956 of file `sf_gamma.tcc`.

## 10.20.2 Member Data Documentation

10.20.2.1 constexpr std::array<long double, 11> std::\_\_detail::\_GammaLanczos< long double >::\_S\_cheby [static]

Initial value:

```
{
    1.440399692024250728e+04L,
    -1.128006201837065341e+04L,
    5.384108670160999829e+03L,
    -1.536234184127325861e+03L,
    2.528551924697309561e+02L,
    -2.265389090278717887e+01L,
    1.006663776178612579e+00L,
    -1.900805731354182626e-02L,
    1.150508317664389324e-04L,
    -1.208915136885480024e-07L,
    -1.518856151960790157e-10L,
}
```

Definition at line 1961 of file sf\_gamma.tcc.

10.20.2.2 constexpr long double std::\_\_detail::\_GammaLanczos< long double >::\_S\_g = 10.5L [static]

Definition at line 1958 of file sf\_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf\_gamma.tcc

## 10.21 std::\_\_detail::\_GammaSpouge< \_Tp > Struct Template Reference

### 10.21.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_GammaSpouge< _Tp >
```

A struct for Spouge algorithm Chebyshev arrays of coefficients.

Definition at line 1731 of file sf\_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf\_gamma.tcc

## 10.22 std::\_\_detail::\_GammaSpouge< double > Struct Template Reference

### Static Public Attributes

- static constexpr std::array< double, 18 > [\\_S\\_cheby](#)

### 10.22.1 Detailed Description

```
template<>
struct std::__detail::_GammaSpouge< double >
```

Definition at line 1752 of file sf\_gamma.tcc.

### 10.22.2 Member Data Documentation

10.22.2.1 constexpr std::array<double, 18> std::\_\_detail::\_GammaSpouge< double >::\_S\_cheby [static]

#### Initial value:

```
{
    2.785716565770350e+08,
    -1.693088166941517e+09,
    4.549688586500031e+09,
    -7.121728036151557e+09,
    7.202572947273274e+09,
    -4.935548868770376e+09,
    2.338187776097503e+09,
    -7.678102458920741e+08,
    1.727524819329867e+08,
    -2.595321377008346e+07,
    2.494811203993971e+06,
    -1.437252641338402e+05,
    4.490767356961276e+03,
    -6.505596924745029e+01,
    3.362323142416327e-01,
    -3.817361443986454e-04,
    3.273137866873352e-08,
    -7.642333165976788e-15,
}
```

Definition at line 1756 of file sf\_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/[sf\\_gamma.tcc](#)

## 10.23 std::\_\_detail::\_GammaSpouge< float > Struct Template Reference

### Static Public Attributes

- static constexpr std::array< float, 7 > [\\_S\\_cheby](#)



### 10.23.1 Detailed Description

```
template<>
struct std::__detail::_GammaSpouge< float >
```

Definition at line 1736 of file sf\_gamma.tcc.

### 10.23.2 Member Data Documentation

10.23.2.1 constexpr std::array<float, 7> std::\_\_detail::\_GammaSpouge< float >::\_S\_cheby [static]

**Initial value:**

```
{
    2.901419e+03F,
    -5.929168e+03F,
    4.148274e+03F,
    -1.164761e+03F,
    1.174135e+02F,
    -2.786588e+00F,
    3.775392e-03F,
}
```

Definition at line 1740 of file sf\_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf\_gamma.tcc

## 10.24 std::\_\_detail::\_GammaSpouge< long double > Struct Template Reference

### Static Public Attributes

- static constexpr std::array< long double, 22 > \_S\_cheby

### 10.24.1 Detailed Description

```
template<>
struct std::__detail::_GammaSpouge< long double >
```

Definition at line 1779 of file sf\_gamma.tcc.

## 10.24.2 Member Data Documentation

10.24.2.1 `constexpr std::array<long double, 22> std::__detail::_GammaSpouge< long double >::_S_cheby` `[static]`

**Initial value:**

```
{
    1.681473171108908244e+10L,
    -1.269150315503303974e+11L,
    4.339449429013039995e+11L,
    -8.893680202692714895e+11L,
    1.218472425867950986e+12L,
    -1.178403473259353616e+12L,
    8.282455311246278274e+11L,
    -4.292112878930625978e+11L,
    1.646988347276488710e+11L,
    -4.661514921989111004e+10L,
    9.619972564515443397e+09L,
    -1.419382551781042824e+09L,
    1.454145470816386107e+08L,
    -9.923020719435758179e+06L,
    4.253557563919127284e+05L,
    -1.053371059784341875e+04L,
    1.332425479537961437e+02L,
    -7.118343974029489132e-01L,
    1.172051640057979518e-03L,
    -3.323940885824119041e-07L,
    4.503801674404338524e-12L,
    -5.320477002211632680e-20L,
}
```

Definition at line 1783 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf\\_gamma.tcc](#)

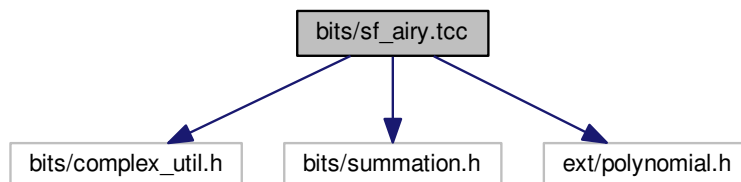
## Chapter 11

# File Documentation

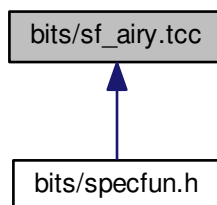
### 11.1 bits/sf\_airy.tcc File Reference

```
#include <bits/complex_util.h>
#include <bits/summation.h>
#include <ext/polynomial.h>
```

Include dependency graph for sf\_airy.tcc:



This graph shows which files directly or indirectly include this file:



## Classes

- class [std::\\_\\_detail::\\_Airy<\\_Tp>](#)
- class [std::\\_\\_detail::\\_Airy\\_asymp<\\_Tp>](#)
- struct [std::\\_\\_detail::\\_Airy\\_asymp\\_data<\\_Tp>](#)
- struct [std::\\_\\_detail::\\_Airy\\_asymp\\_data<double>](#)
- struct [std::\\_\\_detail::\\_Airy\\_asymp\\_data<float>](#)
- struct [std::\\_\\_detail::\\_Airy\\_asymp\\_data<long double>](#)
- class [std::\\_\\_detail::\\_Airy\\_asymp\\_series<\\_Sum>](#)
- struct [std::\\_\\_detail::\\_Airy\\_default\\_radii<\\_Tp>](#)
- struct [std::\\_\\_detail::\\_Airy\\_default\\_radii<double>](#)
- struct [std::\\_\\_detail::\\_Airy\\_default\\_radii<float>](#)
- struct [std::\\_\\_detail::\\_Airy\\_default\\_radii<long double>](#)
- class [std::\\_\\_detail::\\_Airy\\_series<\\_Tp>](#)
- struct [std::\\_\\_detail::\\_AiryAuxilliaryState<\\_Tp>](#)
- struct [std::\\_\\_detail::\\_AiryState<\\_Tp>](#)

## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_AIRY\\_TCC 1](#)

## Functions

- [template<typename \\_Tp>](#)  
[std::complex<\\_Tp> std::\\_\\_detail::\\_\\_airy\\_ai](#) ([std::complex<\\_Tp> \\_\\_z](#))  
*Return the complex Airy Ai function.*
- [template<typename \\_Tp>](#)  
[std::complex<\\_Tp> std::\\_\\_detail::\\_\\_airy\\_bi](#) ([std::complex<\\_Tp> \\_\\_z](#))  
*Return the complex Airy Bi function.*

## Variables

- [template<typename \\_Tp>](#)  
[constexpr int std::\\_\\_detail::\\_\\_max\\_FGH](#) = [\\_Airy\\_series<\\_Tp>::\\_N\\_FGH](#)
- [template<>](#)  
[constexpr int std::\\_\\_detail::\\_\\_max\\_FGH<double>](#) = 79
- [template<>](#)  
[constexpr int std::\\_\\_detail::\\_\\_max\\_FGH<float>](#) = 15

### 11.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

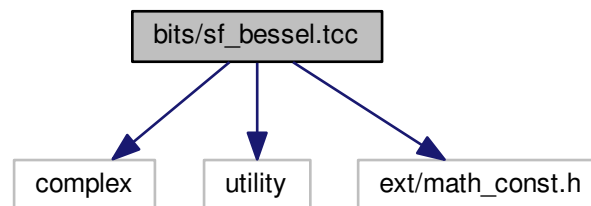
### 11.1.2 Macro Definition Documentation

#### 11.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

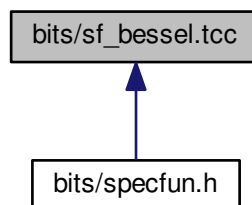
Definition at line 31 of file sf\_airy.tcc.

## 11.2 bits/sf\_bessel.tcc File Reference

```
#include <complex>
#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_neg_arg (_Tp __nu, _Tp __x, std::complex< _Tp > &_Jnu, std::complex< _Tp > &_Nnu, std::complex< _Tp > &_Jpnu, std::complex< _Tp > &_Npnu)`  
*Return the cylindrical Bessel functions and their derivatives of order  $\nu$  and argument  $x < 0$ .*
- `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  $H_n^{(2)}u(x)$ .*
- `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 >`  
`void std::__detail::__sph_bessel_jn_neg_arg` (unsigned int \_\_n, \_Tp \_\_x, std::complex<\_Tp> &\_\_j\_n, std::complex<\_Tp> &\_\_n\_n, std::complex<\_Tp> &\_\_jp\_n, std::complex<\_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 for negative arguments.*
- `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  $x$ .*

## 11.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

## 11.2.2 Macro Definition Documentation

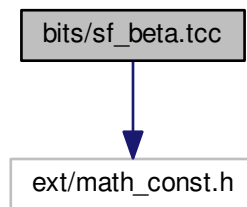
### 11.2.2.1 `#define GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file `sf_bessel.tcc`.

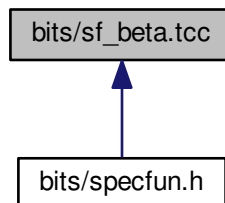
### 11.3 bits/sf\_beta.tcc File Reference

```
#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_lgamma (_Tp __a, _Tp __b)`  
*Return the beta function  $B(a, b)$  using the log gamma functions.*
- `template<typename _Tp >  
_Tp std::__detail::__beta_product (_Tp __a, _Tp __b)`  
*Return the beta function  $B(x, y)$  using the product form.*
- `template<typename _Tp >  
_Tp std::__detail::__ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

### 11.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.3.2 Macro Definition Documentation

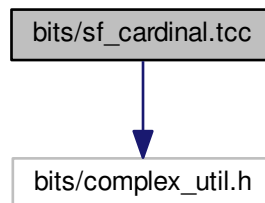
#### 11.3.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file `sf_beta.tcc`.

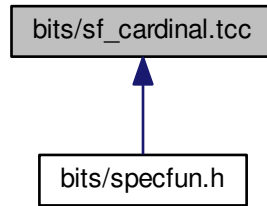
## 11.4 bits/sf\_cardinal.tcc File Reference

```
#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_fp_t< _Tp > std::__detail::__sinc (_Tp __x)`  
*Return the sinus cardinal function*  

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinc_pi (_Tp __x)`  
*Return the reperiodized sinus cardinal function*  

$$\text{sinc}_{\pi}(x) = \frac{\sin(\pi x)}{\pi x}$$
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc (_Tp __x)`  
*Return the hyperbolic sinus cardinal function*  

$$\text{sinhc}(x) = \frac{\sinh(x)}{x}$$
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)`  
*Return the reperiodized hyperbolic sinus cardinal function*  

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

### 11.4.1 Macro Definition Documentation

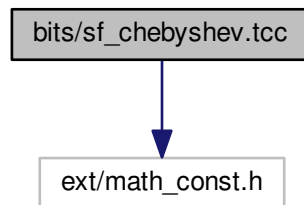
#### 11.4.1.1 `#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1`

Definition at line 30 of file sf\_cardinal.tcc.

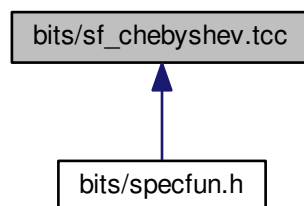
## 11.5 bits/sf\_chebyshev.tcc File Reference

```
#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_BITS_SF_CHEBYSHEV_TCC 1`

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)`
- `template<typename _Tp >`  
`_Tp std::__detail::__chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)`

### 11.5.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.5.2 Macro Definition Documentation

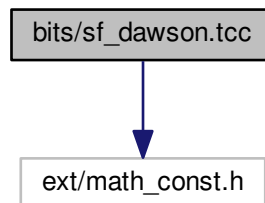
#### 11.5.2.1 `#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1`

Definition at line 31 of file `sf_chebyshev.tcc`.

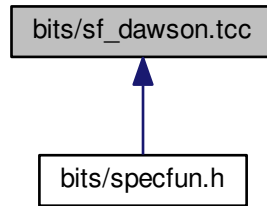
## 11.6 `bits/sf_dawson.tcc` File Reference

```
#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\\_BITS\\_SF\\_DAWSON\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_dawson \(\\_Tp \\_\\_x\)](#)  
Return the Dawson integral,  $F(x)$ , for real argument  $x$ .
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_dawson\\_cont\\_frac \(\\_Tp \\_\\_x\)](#)  
Compute the Dawson integral using a sampling theorem representation.
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_dawson\\_series \(\\_Tp \\_\\_x\)](#)  
Compute the Dawson integral using the series expansion.

### 11.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.6.2 Macro Definition Documentation

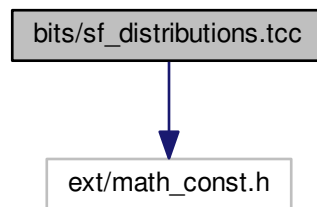
#### 11.6.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_DAWSON\\_TCC 1](#)

Definition at line 31 of file `sf_dawson.tcc`.

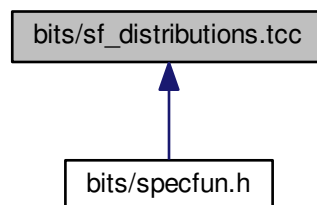
## 11.7 bits/sf\_distributions.tcc File Reference

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

Include dependency graph for sf\_distributions.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_DISTRIBUTIONS\\_TCC](#) 1

## Functions

- `template<typename _Tp >`  
`_Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the binomial cumulative distribution function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the complementary binomial cumulative distribution function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`  
*Return the binomial probability mass function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__chi_squared_pdf (_Tp __chi2, unsigned int __nu)`  
*Return the chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is less than the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__chi_squared_pdfc (_Tp __chi2, unsigned int __nu)`  
*Return the complementary chi-squared probability function. This returns the probability that the observed chi-squared for a correct model is greater than the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__exponential_cdf (_Tp __lambda, _Tp __x)`  
*Return the exponential cumulative probability density function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__exponential_pdf (_Tp __lambda, _Tp __x)`  
*Return the exponential probability density function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`  
*Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value  $\chi^2$ .*
- `template<typename _Tp >`  
`_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 >`  
`_Tp std::__detail::__gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma cumulative propability distribution function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma complementary cumulative propability distribution function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`  
*Return the gamma propability distribution function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`  
*Return the lognormal cumulative probability density function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`  
*Return the lognormal probability density function.*
- `template<typename _Tp >`  
`_Tp std::__detail::__normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`

*Return the normal cumulative probability density function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__normal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

*Return the normal probability density function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

*Return the Rice probability density function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`

*Return the Students T probability function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`

*Return the complement of the Students T probability function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__weibull_cdf (_Tp __a, _Tp __b, _Tp __x)`

*Return the Weibull cumulative probability density function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__weibull_pdf (_Tp __a, _Tp __b, _Tp __x)`

*Return the Weibull probability density function.*

### 11.7.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.7.2 Macro Definition Documentation

#### 11.7.2.1 `#define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1`

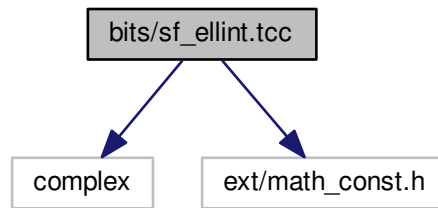
Definition at line 49 of file `sf_distributions.tcc`.

## 11.8 `bits/sf_ellint.tcc` File Reference

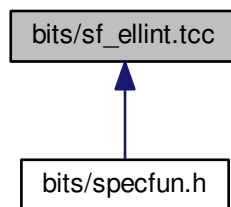
```
#include <complex>
#include <ext/math_const.h>
```



Include dependency graph for sf\_ellint.tcc:



This graph shows which files directly or indirectly include this file:



## Namespaces

- `std`
- `std::__detail`

## Macros

- `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

## Functions

- `template<typename _Tp >  
_Tp std::__detail::__comp_ellint_1 (_Tp __k)`  
*Return the complete elliptic integral of the first kind  $K(k)$  using the Carlson formulation.*

- `template<typename _Tp >`  
`_Tp std::__detail::__comp_ellint_2 (_Tp __k)`  
*Return the complete elliptic integral of the second kind  $E(k)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__comp_ellint_3 (_Tp __k, _Tp __nu)`  
*Return the complete elliptic integral of the third kind  $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__comp_ellint_d (_Tp __k)`
- `template<typename _Tp >`  
`_Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >`  
`_Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)`  
*Return the incomplete elliptic integral of the first kind  $F(k, \phi)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)`  
*Return the incomplete elliptic integral of the second kind  $E(k, \phi)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`  
*Return the incomplete elliptic integral of the third kind  $\Pi(k, \nu, \phi)$  using the Carlson formulation.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)`  
*Return the Carlson elliptic function  $R_C(x, y) = R_F(x, y, y)$  where  $R_F(x, y, z)$  is the Carlson elliptic function of the first kind.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)`  
*Return the Carlson elliptic function of the second kind  $R_D(x, y, z) = R_J(x, y, z, z)$  where  $R_J(x, y, z, p)$  is the Carlson elliptic function of the third kind.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`  
*Return the Carlson elliptic function  $R_F(x, y, z)$  of the first kind.*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`  
*Return the symmetric Carlson elliptic function of the second kind  $R_G(x, y, z)$ .*
- `template<typename _Tp >`  
`_Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`  
*Return the Carlson elliptic function  $R_J(x, y, z, p)$  of the third kind.*
- `template<typename _Tp >`  
`_Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`  
`_Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

### 11.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.8.2 Macro Definition Documentation

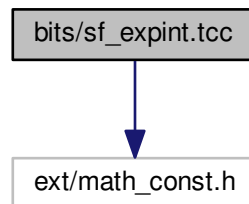
#### 11.8.2.1 `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

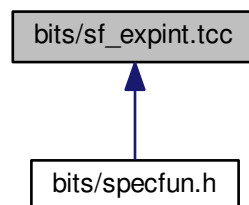
## 11.9 bits/sf\_expint.tcc File Reference

```
#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  $li(x)$ .*

### 11.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

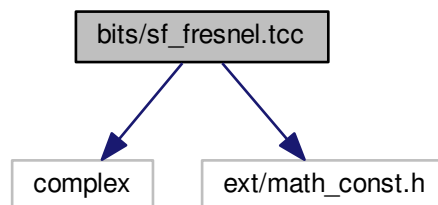
### 11.9.2 Macro Definition Documentation

#### 11.9.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

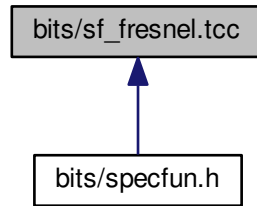
Definition at line 47 of file `sf_expint.tcc`.

## 11.10 bits/sf\_fresnel.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_FRESNEL\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_fresnel (const _Tp __x)`  
*Return the Fresnel cosine and sine integrals as a complex number  $C(x) + iS(x)$   $xf$ .*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_fresnel\_cont\_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`  
*This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_fresnel\_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`  
*This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.*

### 11.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.10.2 Macro Definition Documentation

#### 11.10.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_FRESNEL\\_TCC 1](#)

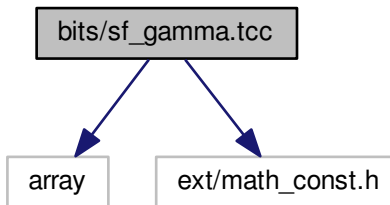
Definition at line 31 of file `sf_fresnel.tcc`.

## 11.11 bits/sf\_gamma.tcc File Reference

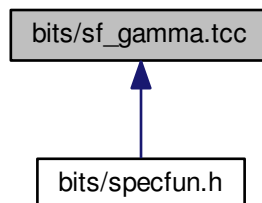
```
#include <array>
```

```
#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 >`
- struct `std::__detail::_GammaLanczos< _Tp >`
- struct `std::__detail::_GammaLanczos< double >`
- struct `std::__detail::_GammaLanczos< float >`
- struct `std::__detail::_GammaLanczos< long double >`
- struct `std::__detail::_GammaSpouge< _Tp >`
- struct `std::__detail::_GammaSpouge< double >`
- struct `std::__detail::_GammaSpouge< float >`
- struct `std::__detail::_GammaSpouge< long double >`

## 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  $B_{2n}$  at even integer arguments  $2n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::\_\_detail::\_\_bernoulli\_series (unsigned int __n)`  
*This returns Bernoulli numbers from a table or by summation for larger values.*

- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_bincoef (unsigned int __n, unsigned int __k)`  
*Return the binomial coefficient. The binomial coefficient is given by:*

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

*The binomial coefficients are generated by:*

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

- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_bincoef (_Tp __nu, unsigned int __k)`  
*Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:*

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

*The binomial coefficients are generated by:*

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

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::\_\_detail::\_\_double\_factorial (int __n)`  
*Return the double factorial of the integer  $n$ .*
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::\_\_detail::\_\_factorial (unsigned int __n)`  
*Return the factorial of the integer  $n$ .*
- `template<typename _Tp >`  
`_Tp std::\_\_detail::\_\_gamma (_Tp __x)`



Return the gamma function  $\Gamma(x)$ . The gamma function is defined by:

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

Return the incomplete gamma function by continued fraction.

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > std::__detail::__gamma_series (_Tp __a, _Tp __x)`

Return the incomplete gamma function by series summation.

- `template<typename _Tp >`  
`_Tp std::__detail::__harmonic_number (unsigned int __n)`
- `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:

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

The binomial coefficients are generated by:

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

- `template<typename _Tp >`  
`_Tp std::__detail::__log_bincoef (_Tp __nu, unsigned int __k)`

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

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

The binomial coefficients are generated by:

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

- `template<typename _Tp >`  
`_Tp std::__detail::__log_bincoef_sign (_Tp __nu, unsigned int __k)`  
Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned.
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__log_bincoef_sign (std::complex< _Tp > __nu, unsigned int __k)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`
- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`

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

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`

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

- `template<typename _Tp >`  
`_Tp std::__detail::__log_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 >`  
`std::complex< _Tp > std::__detail::__log_gamma (std::complex< _Tp > __x)`

Return  $\log(\Gamma(x))$  for complex argument.

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma1p_lanczos (_Tp __z)`

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

- `template<typename _Tp >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma1p_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 >`  
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`  
Return  $\log(\Gamma(x))$  by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

- `template<typename _Tp >`  
`_Tp std::__detail::__log_gamma_sign (_Tp __x)`  
Return the sign of  $\Gamma(x)$ . At nonpositive integers zero is returned indicating  $\Gamma(x)$  is undefined.

- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__log_gamma_sign (std::complex< _Tp > __x)`

- `template<typename _Tp >`  
`_Tp std::__detail::__log_pochhammer (_Tp __a, _Tp __n)`

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

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

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, 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 (_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::__pochhammer_lower (_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,  $\text{ff}(n)_n = n!$ .

- template<typename \_Tp >

`_Tp std::__detail::__psi (unsigned int __n)`

Return the digamma function of integral argument. The digamma or  $\psi(x)$  function is defined as the logarithmic derivative of the gamma function:

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

The digamma series for integral argument is given by:

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

The latter sum is called the harmonic number,  $H_n$ .

- template<typename \_Tp >

`_Tp std::__detail::__psi (_Tp __x)`

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

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

For negative argument the reflection formula is used:

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

.

- `template<typename _Tp >`  
`_Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`  
`_Tp std::__detail::__psi_asymp (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp std::__detail::__psi_series (_Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp std::__detail::__qgamma (_Tp __a, _Tp __x)`

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

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

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

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

*is the upper incomplete gamma function.*

- `template<typename _Tp >`  
`_Tp std::__detail::__tgamma (_Tp __a, _Tp __x)`

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

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

- `template<typename _Tp >`  
`_Tp std::__detail::__tgamma_lower (_Tp __a, _Tp __x)`

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

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

## Variables

- `constexpr _Factorial_table< long double > std::__detail::__S_double_factorial_table` [301]
- `constexpr _Factorial_table< long double > std::__detail::__S_factorial_table` [171]
- `constexpr unsigned long long std::__detail::__S_harmonic_denom` [`_S_num_harmonic_numer`]
- `constexpr unsigned long long std::__detail::__S_harmonic_numer` [`_S_num_harmonic_numer`]
- `constexpr _Factorial_table< long double > std::__detail::__S_neg_double_factorial_table` [999]
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::__S_num_double_factorials = 0`

- `template<>`  
`constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::_S_num_factorials = 0`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`
- `constexpr unsigned long long std::__detail::_S_num_harmonic_number = 29`
- `template<typename _Tp >`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`
- `template<>`  
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

### 11.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.11.2 Macro Definition Documentation

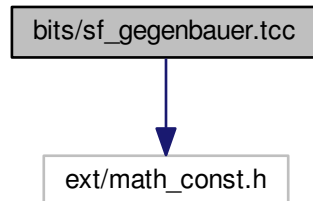
#### 11.11.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

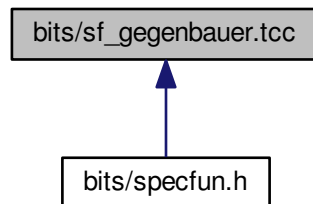
## 11.12 bits/sf\_gegenbauer.tcc File Reference

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

Include dependency graph for sf\_gegenbauer.tcc:



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_GEGENBAUER\\_TCC 1](#)

### Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_gegenbauer\\_poly \(unsigned int \\_\\_n, \\_Tp \\_\\_alpha, \\_Tp \\_\\_x\)](#)

### 11.12.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.12.2 Macro Definition Documentation

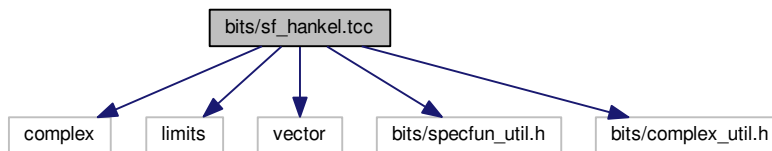
#### 11.12.2.1 `#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1`

Definition at line 31 of file `sf_gegenbauer.tcc`.

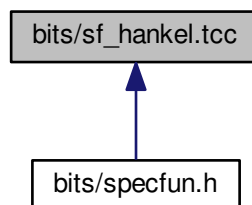
## 11.13 bits/sf\_hankel.tcc File Reference

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

Include dependency graph for `sf_hankel.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HANKEL\\_TCC 1](#)

## Functions

- `template<typename _Tp >`  
`void std::\_\_detail::\_\_airy\_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`  
*Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_cyl\_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Bessel function.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_cyl\_hankel\_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Hankel function of the first kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_cyl\_hankel\_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Hankel function of the second kind.*
- `template<typename _Tp >`  
`std::complex< _Tp > std::\_\_detail::\_\_cyl\_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`  
*Return the complex cylindrical Neumann function.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_debye\_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_hankel\_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_hankel\_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`  
*Compute parameters depending on `z` and `nu` that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.*
- `template<typename _Tp >`  
`void std::\_\_detail::\_\_hankel\_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`



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

- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`

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

- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`

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

- `template<typename _Tp >`  
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &_H1sum, std::complex< _Tp > &_H1psum, std::complex< _Tp > &_H2sum, std::complex< _Tp > &_H2psum)`

*Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `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.*

### 11.13.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

### 11.13.2 Macro Definition Documentation

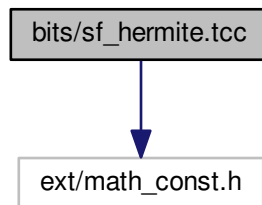
#### 11.13.2.1 `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

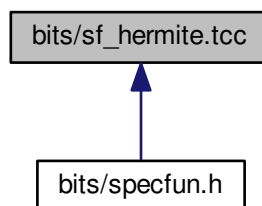
## 11.14 `bits/sf_hermite.tcc` File Reference

```
#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  $n$ .*

### 11.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.14.2 Macro Definition Documentation

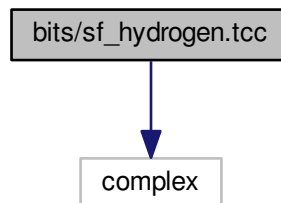
#### 11.14.2.1 `#define _GLIBCXX_BITS_SF_HERMITE_TCC 1`

Definition at line 42 of file `sf_hermite.tcc`.

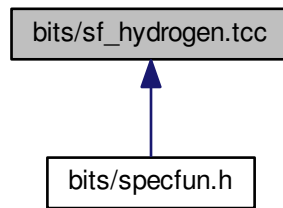
## 11.15 bits/sf\_hydrogen.tcc File Reference

```
#include <complex>
```

Include dependency graph for `sf_hydrogen.tcc`:



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 \(unsigned int \\_\\_n, unsigned int \\_\\_l, unsigned int \\_\\_m, \\_Tp \\_\\_Z, \\_Tp \\_\\_r, \\_Tp \\_\\_theta, \\_Tp \\_\\_phi\)](#)

### 11.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

### 11.15.2 Macro Definition Documentation

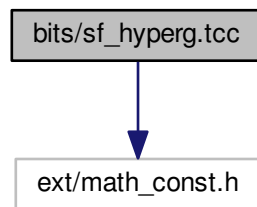
#### 11.15.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_HYDROGEN\\_TCC 1](#)

Definition at line 31 of file `sf_hydrogen.tcc`.

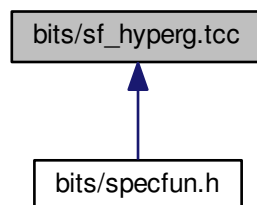
## 11.16 bits/sf\_hyperg.tcc File Reference

```
#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.*

### 11.16.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

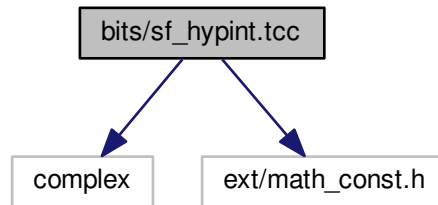
### 11.16.2 Macro Definition Documentation

#### 11.16.2.1 `#define _GLIBCXX_BITS_SF_HYPERG_TCC 1`

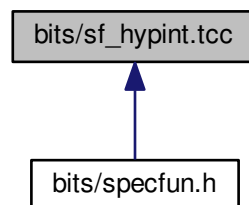
Definition at line 44 of file `sf_hyperg.tcc`.

## 11.17 bits/sf\_hypint.tcc File Reference

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



This graph shows which files directly or indirectly include this file:



### Namespaces

- [std](#)
- [std::\\_\\_detail](#)

### Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_HYPINT\\_TCC](#) 1

## Functions

- `template<typename _Tp >`  
`std::pair< _Tp, _Tp > std::__detail::__chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`  
*This function returns the hyperbolic cosine  $Chi(x)$  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.*

### 11.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.17.2 Macro Definition Documentation

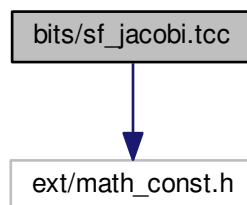
#### 11.17.2.1 `#define _GLIBCXX_BITS_SF_HYPINT_TCC 1`

Definition at line 31 of file `sf_hypint.tcc`.

## 11.18 bits/sf\_jacobi.tcc File Reference

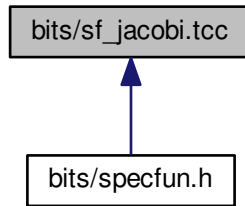
`#include <ext/math_const.h>`

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\\_BITS\\_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\\_fp\\_t< \\_Tp > std::\\_\\_detail::\\_\\_zernike \(unsigned int \\_\\_n, int \\_\\_m, \\_Tp \\_\\_rho, \\_Tp \\_\\_phi\)](#)

### 11.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.18.2 Macro Definition Documentation

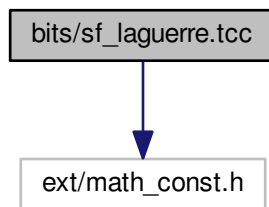
#### 11.18.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_JACOBI\\_TCC 1](#)

Definition at line 31 of file `sf_jacobi.tcc`.

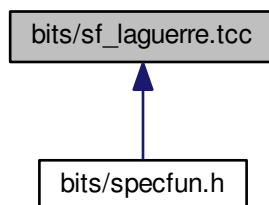
## 11.19 bits/sf\_laguerre.tcc File Reference

```
#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.*

### 11.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.19.2 Macro Definition Documentation

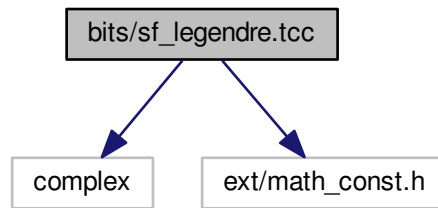
#### 11.19.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

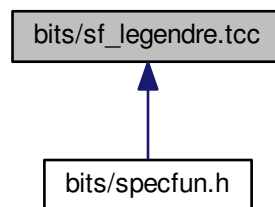
## 11.20 bits/sf\_legendre.tcc File Reference

```
#include <complex>
#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.*

### 11.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

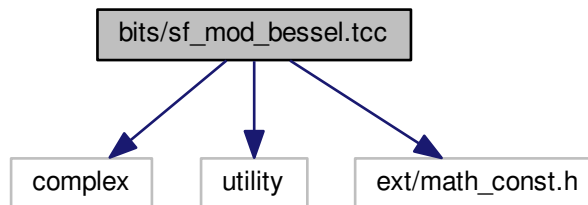
### 11.20.2 Macro Definition Documentation

#### 11.20.2.1 `#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1`

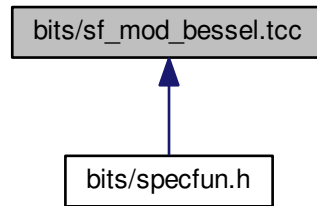
Definition at line 47 of file `sf_legendre.tcc`.

## 11.21 bits/sf\_mod\_bessel.tcc File Reference

```
#include <complex>
#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.

### 11.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

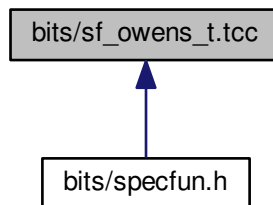
### 11.21.2 Macro Definition Documentation

#### 11.21.2.1 `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

## 11.22 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 \(\\_Tp \\_\\_x\)](#)

### 11.22.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

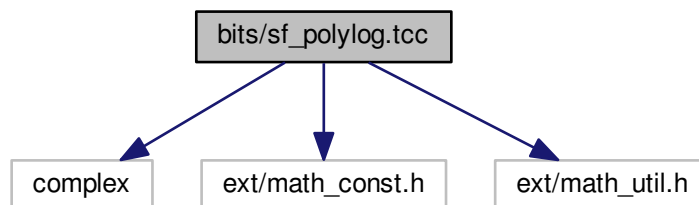
### 11.22.2 Macro Definition Documentation

#### 11.22.2.1 [#define \\_GLIBCXX\\_BITS\\_SF\\_OWENS\\_T\\_TCC 1](#)

Definition at line 31 of file `sf_owens_t.tcc`.

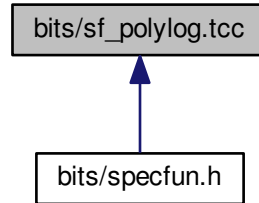
## 11.23 bits/sf\_polylog.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
#include <ext/math_util.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 \\_Sp, typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_bose\\_einstein \(\\_Sp \\_\\_s, \\_Tp \\_\\_x\)](#)
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_clamp\\_0\\_m2pi \(std::complex< \\_Tp > \\_\\_w\)](#)
- [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::__dirichlet_lambda (_Tp __w)`
- `template<typename _Sp, typename _Tp >`  
`_Tp std::__detail::__fermi_dirac (_Sp __s, _Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__hurwitz_zeta_polylog (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`  
`_Tp std::__detail::__polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`  
`__gnu_cxx::__promote_fp_t< std::complex< _Tp >, ArgType > std::__detail::__polylog_exp (_Tp __s, ArgType __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__polylog_exp_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)`

### 11.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.23.2 Macro Definition Documentation

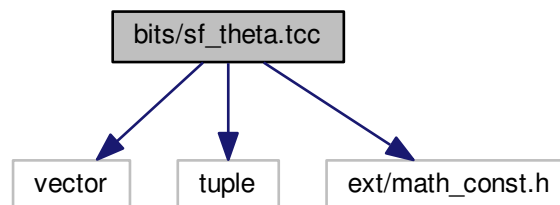
#### 11.23.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

Definition at line 41 of file `sf_polylog.tcc`.

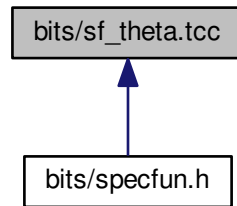
## 11.24 bits/sf\_theta.tcc File Reference

```
#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\\_BITS\\_SF\\_THETA\\_TCC 1](#)

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome\\_k \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_ellnome\\_series \(\\_Tp \\_\\_k\)](#)
- [template<typename \\_Tp > 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)`

### 11.24.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.24.2 Macro Definition Documentation

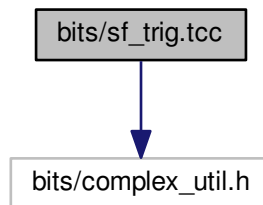
#### 11.24.2.1 `#define _GLIBCXX_BITS_SF_THETA_TCC 1`

Definition at line 31 of file `sf_theta.tcc`.

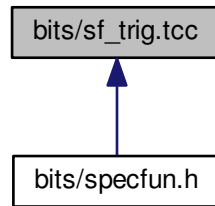
## 11.25 bits/sf\_trig.tcc File Reference

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

Include dependency graph for `sf_trig.tcc`:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t](#)< [\\_Tp](#) >

## Namespaces

- [\\_\\_gnu\\_cxx](#)
- [std](#)
- [std::\\_\\_detail](#)

## Macros

- [#define \\_GLIBCXX\\_BITS\\_SF\\_TRIG\\_TCC](#) 1

## Functions

- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_cos\\_pi](#) ([\\_Tp \\_\\_x](#))
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_cos\\_pi](#) ([std::complex< \\_Tp > \\_\\_z](#))
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_cosh\\_pi](#) ([\\_Tp \\_\\_x](#))
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_cosh\\_pi](#) ([std::complex< \\_Tp > \\_\\_z](#))
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_polar\\_pi](#) ([\\_Tp \\_\\_rho](#), [\\_Tp \\_\\_phi\\_pi](#))
- [template<typename \\_Tp > \\_Tp std::\\_\\_detail::\\_\\_sin\\_pi](#) ([\\_Tp \\_\\_x](#))
- [template<typename \\_Tp > std::complex< \\_Tp > std::\\_\\_detail::\\_\\_sin\\_pi](#) ([std::complex< \\_Tp > \\_\\_z](#))

- `template<typename _Tp >`  
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos (_Tp __x)`
- `template<>`  
`__gnu_cxx::__sincos_t< float > std::__detail::__sincos (float __x)`
- `template<>`  
`__gnu_cxx::__sincos_t< double > std::__detail::__sincos (double __x)`
- `template<>`  
`__gnu_cxx::__sincos_t< long double > std::__detail::__sincos (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos_pi (_Tp __x)`
- `template<typename _Tp >`  
`_Tp std::__detail::__sinh_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__sinh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`_Tp std::__detail::__tan_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__tan_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`  
`_Tp std::__detail::__tanh_pi (_Tp __x)`
- `template<typename _Tp >`  
`std::complex< _Tp > std::__detail::__tanh_pi (std::complex< _Tp > __z)`

### 11.25.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

### 11.25.2 Macro Definition Documentation

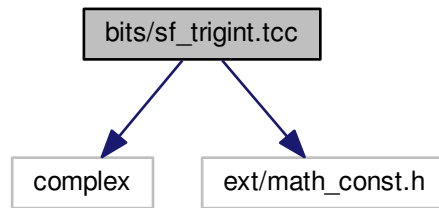
#### 11.25.2.1 `#define _GLIBCXX_BITS_SF_TRIG_TCC 1`

Definition at line 31 of file sf\_trig.tcc.

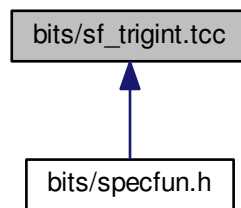
## 11.26 bits/sf\_trigint.tcc File Reference

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

Include dependency graph for `sf_trigint.tcc`:



This graph shows which files directly or indirectly include this file:



## Namespaces

- [std](#)
- [std::\\_\\_detail](#)

## Macros

- `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

## 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.*

### 11.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.26.2 Macro Definition Documentation

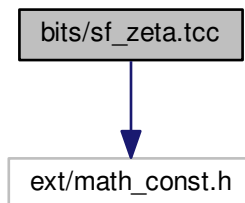
#### 11.26.2.1 `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

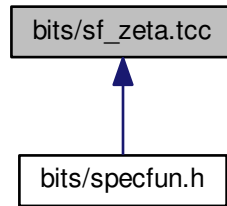
## 11.27 bits/sf\_zeta.tcc File Reference

`#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]`

### 11.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.27.2 Macro Definition Documentation

#### 11.27.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

## 11.28 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_trig.tcc>
#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\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::airy\\_ai](#) (\_Tp \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::airy\\_ai](#) (std::complex< \_Tp > \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_aif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_ail](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::airy\\_bi](#) (\_Tp \_\_x)
- template<typename \_Tp >  
std::complex< [\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::airy\\_bi](#) (std::complex< \_Tp > \_\_x)
- float [\\_\\_gnu\\_cxx::airy\\_bif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::airy\\_bil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote<\\_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\\_fp\\_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\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::bincoef](#) (unsigned int \_\_n, unsigned int \_\_k)
- float [\\_\\_gnu\\_cxx::bincoefficient](#) (unsigned int \_\_n, unsigned int \_\_k)
- long double [\\_\\_gnu\\_cxx::bincoefficientl](#) (unsigned int \_\_n, unsigned int \_\_k)
- template<typename \_Tps, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tps, \\_Tp>](#) [\\_\\_gnu\\_cxx::bose\\_einstein](#) (\_Tps \_\_s, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::bose\\_einsteinf](#) (float \_\_s, float \_\_x)
- long double [\\_\\_gnu\\_cxx::bose\\_einsteinl](#) (long double \_\_s, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::chebyshev\\_t](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_tf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_tl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) [\\_\\_gnu\\_cxx::chebyshev\\_u](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::chebyshev\\_uf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::chebyshev\\_ul](#) (unsigned int \_\_n, long double \_\_x)

- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_wl (unsigned int __n, long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_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_fp_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_fp_t< _Tk > __gnu_cxx::comp_ellint_d (_Tk __k)`
- `float __gnu_cxx::comp_ellint_df (float __k)`
- `long double __gnu_cxx::comp_ellint_dl (long double __k)`
- `float __gnu_cxx::comp_ellint_rf (float __x, float __y)`
- `long double __gnu_cxx::comp_ellint_rf (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf (_Tx __x, _Ty __y)`
- `float __gnu_cxx::comp_ellint_rg (float __x, float __y)`
- `long double __gnu_cxx::comp_ellint_rg (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`  
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)`

- `template<typename _Tpa, typename _Tpc, typename _Tp >`  
`__gnu_cxx::__promote_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<_Tp>::__type __gnu_cxx::cos_pi (_Tp __x)`
- `float __gnu_cxx::cos_pif (float __x)`
- `long double __gnu_cxx::cos_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote<_Tp>::__type __gnu_cxx::cosh_pi (_Tp __x)`
- `float __gnu_cxx::cosh_pif (float __x)`
- `long double __gnu_cxx::cosh_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::coshint (_Tp __x)`
- `float __gnu_cxx::coshintf (float __x)`
- `long double __gnu_cxx::coshintl (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cosint (_Tp __x)`
- `float __gnu_cxx::cosintf (float __x)`
- `long double __gnu_cxx::cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_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_fp_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_1 (std::complex<_Tpnu > __nu, std::complex<_Tp> __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> > __gnu_cxx::cyl_hankel_2 (std::complex<_Tpnu > __nu, std::complex<_Tp> __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`

- `std::complex< float > __gnu_cxx::cyl_hankel_2f` (`std::complex< float > __nu`, `std::complex< float > __x`)
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l` (`long double __nu`, `long double __z`)
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l` (`std::complex< long double > __nu`, `std::complex< long double > __x`)
- `template<typename _Tpnu, typename _Tp >`  
`__gnu_cxx::__promote_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_fp_t< _Tp > __gnu_cxx::dawson` (`_Tp __x`)
- `float __gnu_cxx::dawsonf` (`float __x`)
- `long double __gnu_cxx::dawsonl` (`long double __x`)
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::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 >`  
`_Tp __gnu_cxx::dirichlet_lambda` (`_Tp __s`)
- `float __gnu_cxx::dirichlet_lambdaf` (`float __s`)
- `long double __gnu_cxx::dirichlet_lambdall` (`long double __s`)
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_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_fp_t< _Tk, _Tp, _Ta, _Tb > \_\_gnu\_cxx::ellint\_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)`
- `float \_\_gnu\_cxx::ellint\_celf (float __k_c, float __p, float __a, float __b)`
- `long double \_\_gnu\_cxx::ellint\_cell (long double __k_c, long double __p, long double __a, long double __b)`
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > \_\_gnu\_cxx::ellint\_d (_Tk __k, _Tphi __phi)`
- `float \_\_gnu\_cxx::ellint\_df (float __k, float __phi)`
- `long double \_\_gnu\_cxx::ellint\_dl (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tk >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Tk > \_\_gnu\_cxx::ellint\_el1 (_Tp __x, _Tk __k_c)`
- `float \_\_gnu\_cxx::ellint\_el1f (float __x, float __k_c)`
- `long double \_\_gnu\_cxx::ellint\_el1l (long double __x, long double __k_c)`
- `template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Tk, _Ta, _Tb > \_\_gnu\_cxx::ellint\_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)`
- `float \_\_gnu\_cxx::ellint\_el2f (float __x, float __k_c, float __a, float __b)`
- `long double \_\_gnu\_cxx::ellint\_el2l (long double __x, long double __k_c, long double __a, long double __b)`
- `template<typename _Tx, typename _Tk, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tx, _Tk, _Tp > \_\_gnu\_cxx::ellint\_el3 (_Tx __x, _Tk __k_c, _Tp __p)`
- `float \_\_gnu\_cxx::ellint\_el3f (float __x, float __k_c, float __p)`
- `long double \_\_gnu\_cxx::ellint\_el3l (long double __x, long double __k_c, long double __p)`
- `template<typename _Tp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up > \_\_gnu\_cxx::ellint\_rc (_Tp __x, _Up __y)`
- `float \_\_gnu\_cxx::ellint\_rcf (float __x, float __y)`
- `long double \_\_gnu\_cxx::ellint\_rcl (long double __x, long double __y)`
- `template<typename _Tp, typename _Up, typename _Vp >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > \_\_gnu\_cxx::ellint\_rd (_Tp __x, _Up __y, _Vp __z)`
- `float \_\_gnu\_cxx::ellint\_rdf (float __x, float __y, float __z)`
- `long double \_\_gnu\_cxx::ellint\_rdl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > \_\_gnu\_cxx::ellint\_rf (_Tp __x, _Up __y, _Vp __z)`
- `float \_\_gnu\_cxx::ellint\_rff (float __x, float __y, float __z)`
- `long double \_\_gnu\_cxx::ellint\_rfl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > \_\_gnu\_cxx::ellint\_rg (_Tp __x, _Up __y, _Vp __z)`
- `float \_\_gnu\_cxx::ellint\_rgf (float __x, float __y, float __z)`
- `long double \_\_gnu\_cxx::ellint\_rgl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp, typename _Wp >`  
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp, _Wp > \_\_gnu\_cxx::ellint\_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)`
- `float \_\_gnu\_cxx::ellint\_rjf (float __x, float __y, float __z, float __p)`
- `long double \_\_gnu\_cxx::ellint\_rjl (long double __x, long double __y, long double __z, long double __p)`
- `template<typename _Tp >`  
`_Tp \_\_gnu\_cxx::ellnome (_Tp __k)`
- `float \_\_gnu\_cxx::ellnomef (float __k)`
- `long double \_\_gnu\_cxx::ellnomel (long double __k)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::type std::expint (_Tp __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > \_\_gnu\_cxx::expint (unsigned int __n, _Tp __x)`
- `float std::expintf (float __x)`
- `float \_\_gnu\_cxx::expintf (unsigned int __n, float __x)`

- long double [std::expintl](#) (long double \_\_x)
- long double [\\_\\_gnu\\_cxx::expintl](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_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 \_Tps, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tps, \\_Tp>](#) > [\\_\\_gnu\\_cxx::fermi\\_dirac](#) (\_Tps \_\_s, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::fermi\\_diracf](#) (float \_\_s, float \_\_x)
- long double [\\_\\_gnu\\_cxx::fermi\\_diracl](#) (long double \_\_s, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::fresnel\\_c](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::fresnel\\_cf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::fresnel\\_cl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp>](#) > [\\_\\_gnu\\_cxx::fresnel\\_s](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::fresnel\\_sf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::fresnel\\_sl](#) (long double \_\_x)
- template<typename \_Talpha, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Talpha, \\_Tp>](#) > [\\_\\_gnu\\_cxx::gegenbauer](#) (unsigned int \_\_n, \_Talpha \_\_alpha, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::gegenbauerf](#) (unsigned int \_\_n, float \_\_alpha, float \_\_x)
- long double [\\_\\_gnu\\_cxx::gegenbauerl](#) (unsigned int \_\_n, long double \_\_alpha, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote<\\_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\\_fp\\_t<\\_Tk, \\_Tphi>](#) > [\\_\\_gnu\\_cxx::heuman\\_lambda](#) (\_Tk \_\_k, \_Tphi \_\_phi)
- float [\\_\\_gnu\\_cxx::heuman\\_lambdaf](#) (float \_\_k, float \_\_phi)
- long double [\\_\\_gnu\\_cxx::heuman\\_lambdal](#) (long double \_\_k, long double \_\_phi)
- template<typename \_Tp, typename \_Up >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Tp, \\_Up>](#) > [\\_\\_gnu\\_cxx::hurwitz\\_zeta](#) (\_Tp \_\_s, \_Up \_\_a)
- template<typename \_Tp, typename \_Up >  
[std::complex<\\_Tp>](#) > [\\_\\_gnu\\_cxx::hurwitz\\_zeta](#) (\_Tp \_\_s, [std::complex<\\_Up>](#) \_\_a)
- float [\\_\\_gnu\\_cxx::hurwitz\\_zetaf](#) (float \_\_s, float \_\_a)
- long double [\\_\\_gnu\\_cxx::hurwitz\\_zetal](#) (long double \_\_s, long double \_\_a)
- template<typename \_Tpa, typename \_Tpb, typename \_Tpc, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_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\\_fp\\_t<\\_Ta, \\_Tb, \\_Tp>](#) > [\\_\\_gnu\\_cxx::ibeta](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- template<typename \_Ta, typename \_Tb, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t<\\_Ta, \\_Tb, \\_Tp>](#) > [\\_\\_gnu\\_cxx::ibetac](#) (\_Ta \_\_a, \_Tb \_\_b, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::ibetaacf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [\\_\\_gnu\\_cxx::ibetaacl](#) (long double \_\_a, long double \_\_b, long double \_\_x)
- float [\\_\\_gnu\\_cxx::ibetaf](#) (float \_\_a, float \_\_b, float \_\_x)
- long double [\\_\\_gnu\\_cxx::ibetal](#) (long double \_\_a, long double \_\_b, long double \_\_x)

- `template<typename _Talpha, typename _Tbeta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi` (unsigned \_\_n, \_Talpha \_\_alpha, \_Tbeta \_\_beta, \_Tp \_\_x)
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_cn` (\_Kp \_\_k, \_Up \_\_u)
- `float __gnu_cxx::jacobi_cnf` (float \_\_k, float \_\_u)
- `long double __gnu_cxx::jacobi_cnl` (long double \_\_k, long double \_\_u)
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_dn` (\_Kp \_\_k, \_Up \_\_u)
- `float __gnu_cxx::jacobi_dnf` (float \_\_k, float \_\_u)
- `long double __gnu_cxx::jacobi_dnl` (long double \_\_k, long double \_\_u)
- `template<typename _Kp, typename _Up >`  
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_sn` (\_Kp \_\_k, \_Up \_\_u)
- `float __gnu_cxx::jacobi_snf` (float \_\_k, float \_\_u)
- `long double __gnu_cxx::jacobi_snl` (long double \_\_k, long double \_\_u)
- `template<typename _Tk, typename _Tphi >`  
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta` (\_Tk \_\_k, \_Tphi \_\_phi)
- `float __gnu_cxx::jacobi_zetaf` (float \_\_k, float \_\_phi)
- `long double __gnu_cxx::jacobi_zetal` (long double \_\_k, long double \_\_phi)
- `float __gnu_cxx::jacobif` (unsigned \_\_n, float \_\_alpha, float \_\_beta, float \_\_x)
- `long double __gnu_cxx::jacobil` (unsigned \_\_n, long double \_\_alpha, long double \_\_beta, long double \_\_x)
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _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_fp_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_fp_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_fp_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_fp_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 _Ta >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::lgamma` (std::complex< \_Ta > \_\_a)
- `std::complex< float > __gnu_cxx::lgammaf` (std::complex< float > \_\_a)
- `std::complex< long double > __gnu_cxx::lgammal` (std::complex< long double > \_\_a)
- `template<typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::logint` (\_Tp \_\_x)
- `float __gnu_cxx::logintf` (float \_\_x)

- long double [\\_\\_gnu\\_cxx::logintl](#) (long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Tn > \\_\\_gnu\\_cxx::lpochhammer](#) (\_Tp \_\_a, \_Tn \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Tn > \\_\\_gnu\\_cxx::lpochhammer\\_lower](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::lpochhammer\\_lowerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::lpochhammer\\_lowerl](#) (long double \_\_a, long double \_\_n)
- float [\\_\\_gnu\\_cxx::lpochhammerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::lpochhammerl](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tph, typename \_Tpa >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tph, \\_Tpa > \\_\\_gnu\\_cxx::owens\\_t](#) (\_Tph \_\_h, \_Tpa \_\_a)
- float [\\_\\_gnu\\_cxx::owens\\_tf](#) (float \_\_h, float \_\_a)
- long double [\\_\\_gnu\\_cxx::owens\\_tl](#) (long double \_\_h, long double \_\_a)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Ta, \\_Tp > \\_\\_gnu\\_cxx::pgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::pgammaf](#) (float \_\_a, float \_\_x)
- long double [\\_\\_gnu\\_cxx::pgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Tn > \\_\\_gnu\\_cxx::pochhammer](#) (\_Tp \_\_a, \_Tn \_\_n)
- template<typename \_Tp, typename \_Tn >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Tn > \\_\\_gnu\\_cxx::pochhammer\\_lower](#) (\_Tp \_\_a, \_Tn \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammer\\_lowerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammer\\_lowerl](#) (long double \_\_a, long double \_\_n)
- float [\\_\\_gnu\\_cxx::pochhammerf](#) (float \_\_a, float \_\_n)
- long double [\\_\\_gnu\\_cxx::pochhammerl](#) (long double \_\_a, long double \_\_n)
- template<typename \_Tp, typename \_Wp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Wp > \\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, \_Wp \_\_w)
- template<typename \_Tp, typename \_Wp >  
[std::complex< \\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp, \\_Wp > > \\_\\_gnu\\_cxx::polylog](#) (\_Tp \_\_s, std::complex< \_Tp > \_\_w)
- float [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, float \_\_w)
- std::complex< float > [\\_\\_gnu\\_cxx::polylogf](#) (float \_\_s, std::complex< float > \_\_w)
- long double [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, long double \_\_w)
- std::complex< long double > [\\_\\_gnu\\_cxx::polylogl](#) (long double \_\_s, std::complex< long double > \_\_w)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::psi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::psif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::psil](#) (long double \_\_x)
- template<typename \_Ta, typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Ta, \\_Tp > \\_\\_gnu\\_cxx::qgamma](#) (\_Ta \_\_a, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::qgammaf](#) (float \_\_a, float \_\_x)
- long double [\\_\\_gnu\\_cxx::qgammal](#) (long double \_\_a, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::radpoly](#) (unsigned int \_\_n, unsigned int \_\_m, \_Tp \_\_rho)
- float [\\_\\_gnu\\_cxx::radpolyf](#) (unsigned int \_\_n, unsigned int \_\_m, float \_\_rho)
- long double [\\_\\_gnu\\_cxx::radpolyl](#) (unsigned int \_\_n, unsigned int \_\_m, long double \_\_rho)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote< \\_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 \\_\\_gnu\\_cxx::sin\\_pi](#) (\_Tp \_\_x)

- float [\\_\\_gnu\\_cxx::sin\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sin\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sincf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sincl](#) (long double \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< double > \\_\\_gnu\\_cxx::sincos](#) (double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< \\_Tp > \\_\\_gnu\\_cxx::sincos](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< \\_Tp > \\_\\_gnu\\_cxx::sincos\\_pi](#) (\_Tp \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< float > \\_\\_gnu\\_cxx::sincos\\_pif](#) (float \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< long double > \\_\\_gnu\\_cxx::sincos\\_pil](#) (long double \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< float > \\_\\_gnu\\_cxx::sincosf](#) (float \_\_x)
- [\\_\\_gnu\\_cxx::\\_\\_sincos\\_t< long double > \\_\\_gnu\\_cxx::sincosl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote< \\_Tp >::\\_\\_type \\_\\_gnu\\_cxx::sinh\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinh\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinh\\_pil](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinhc](#) (\_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinhc\\_pi](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinhc\\_pif](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhc\\_pil](#) (long double \_\_x)
- float [\\_\\_gnu\\_cxx::sinhcf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhcl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinhint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinhintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinhintl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sinint](#) (\_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sinintf](#) (float \_\_x)
- long double [\\_\\_gnu\\_cxx::sinintl](#) (long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote< \\_Tp >::\\_\\_type std::sph\\_bessel](#) (unsigned int \_\_n, \_Tp \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sph\\_bessel\\_i](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sph\\_bessel\\_if](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::sph\\_bessel\\_il](#) (unsigned int \_\_n, long double \_\_x)
- template<typename \_Tp >  
[\\_\\_gnu\\_cxx::\\_\\_promote\\_fp\\_t< \\_Tp > \\_\\_gnu\\_cxx::sph\\_bessel\\_k](#) (unsigned int \_\_n, \_Tp \_\_x)
- float [\\_\\_gnu\\_cxx::sph\\_bessel\\_kf](#) (unsigned int \_\_n, float \_\_x)
- long double [\\_\\_gnu\\_cxx::sph\\_bessel\\_kl](#) (unsigned int \_\_n, long double \_\_x)
- float [std::sph\\_besself](#) (unsigned int \_\_n, float \_\_x)
- long double [std::sph\\_bessell](#) (unsigned int \_\_n, long double \_\_x)

- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Ttheta, typename _Tphi >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)`
- `std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)`
- `std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _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 _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type __gnu_cxx::tan_pi (_Tp __x)`
- `float __gnu_cxx::tan_pif (float __x)`
- `long double __gnu_cxx::tan_pil (long double __x)`
- `template<typename _Tp >`  
`__gnu_cxx::__promote< _Tp >::__type __gnu_cxx::tanh_pi (_Tp __x)`
- `float __gnu_cxx::tanh_pif (float __x)`
- `long double __gnu_cxx::tanh_pil (long double __x)`
- `template<typename _Ta >`  
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma (std::complex< _Ta > __a)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma (_Ta __a, _Tp __x)`
- `template<typename _Ta, typename _Tp >`  
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower (_Ta __a, _Tp __x)`
- `float __gnu_cxx::tgamma_lowerf (float __a, float __x)`
- `long double __gnu_cxx::tgamma_lowerl (long double __a, long double __x)`
- `std::complex< float > __gnu_cxx::tgammaf (std::complex< float > __a)`
- `float __gnu_cxx::tgammaf (float __a, float __x)`
- `std::complex< long double > __gnu_cxx::tgammal (std::complex< long double > __a)`
- `long double __gnu_cxx::tgammal (long double __a, long double __x)`

- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp> __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_1f (float __nu, float __x)`
- `long double __gnu_cxx::theta_1l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp> __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_2f (float __nu, float __x)`
- `long double __gnu_cxx::theta_2l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp> __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_3f (float __nu, float __x)`
- `long double __gnu_cxx::theta_3l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp>`  
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp> __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_4f (float __nu, float __x)`
- `long double __gnu_cxx::theta_4l (long double __nu, long double __x)`
- `template<typename _Tp_k, typename _Tp>`  
`__gnu_cxx::__promote_fp_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_fp_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_fp_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_fp_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_fp_t< _Trho, _Tphi> __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)`
- `float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)`
- `long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)`

### 11.28.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

### 11.28.2 Macro Definition Documentation

#### 11.28.2.1 `#define __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

#### 11.28.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file `specfun.h`.





# Index

`_Airy`  
    `std::__detail::_Airy`, [293](#)  
`_Airy_asymp`  
    `std::__detail::_Airy_asymp`, [296](#)  
`_Airy_asymp_series`  
    `std::__detail::_Airy_asymp_series`, [302](#)  
`_GLIBCXX_BITS_SF_AIRY_TCC`  
    `sf_airy.tcc`, [325](#)  
`_GLIBCXX_BITS_SF_BESSEL_TCC`  
    `sf_bessel.tcc`, [327](#)  
`_GLIBCXX_BITS_SF_BETA_TCC`  
    `sf_beta.tcc`, [329](#)  
`_GLIBCXX_BITS_SF_CARDINAL_TCC`  
    `sf_cardinal.tcc`, [331](#)  
`_GLIBCXX_BITS_SF_CHEBYSHEV_TCC`  
    `sf_chebyshev.tcc`, [332](#)  
`_GLIBCXX_BITS_SF_DAWSON_TCC`  
    `sf_dawson.tcc`, [333](#)  
`_GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC`  
    `sf_distributions.tcc`, [336](#)  
`_GLIBCXX_BITS_SF_ELLINT_TCC`  
    `sf_ellint.tcc`, [339](#)  
`_GLIBCXX_BITS_SF_EXPINT_TCC`  
    `sf_expint.tcc`, [341](#)  
`_GLIBCXX_BITS_SF_FRESNEL_TCC`  
    `sf_fresnel.tcc`, [342](#)  
`_GLIBCXX_BITS_SF_GAMMA_TCC`  
    `sf_gamma.tcc`, [349](#)  
`_GLIBCXX_BITS_SF_GEGENBAUER_TCC`  
    `sf_gegenbauer.tcc`, [351](#)  
`_GLIBCXX_BITS_SF_HANKEL_TCC`  
    `sf_hankel.tcc`, [354](#)  
`_GLIBCXX_BITS_SF_HERMITE_TCC`  
    `sf_hermite.tcc`, [355](#)  
`_GLIBCXX_BITS_SF_HYDROGEN_TCC`  
    `sf_hydrogen.tcc`, [356](#)  
`_GLIBCXX_BITS_SF_HYPERG_TCC`  
    `sf_hyperg.tcc`, [358](#)  
`_GLIBCXX_BITS_SF_HYPINT_TCC`  
    `sf_hypint.tcc`, [360](#)  
`_GLIBCXX_BITS_SF_JACOBI_TCC`  
    `sf_jacobi.tcc`, [361](#)  
`_GLIBCXX_BITS_SF_LAGUERRE_TCC`  
    `sf_laguerre.tcc`, [363](#)  
`_GLIBCXX_BITS_SF_LEGENDRE_TCC`  
    `sf_legendre.tcc`, [365](#)  
`_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`  
    `sf_mod_bessel.tcc`, [367](#)  
`_GLIBCXX_BITS_SF_OWENS_T_TCC`  
    `sf_owens.t.tcc`, [368](#)  
`_GLIBCXX_BITS_SF_POLYLOG_TCC`  
    `sf_polylog.tcc`, [371](#)  
`_GLIBCXX_BITS_SF_THETA_TCC`  
    `sf_theta.tcc`, [373](#)  
`_GLIBCXX_BITS_SF_TRIGINT_TCC`  
    `sf_trigint.tcc`, [377](#)  
`_GLIBCXX_BITS_SF_TRIG_TCC`  
    `sf_trig.tcc`, [375](#)  
`_GLIBCXX_BITS_SF_ZETA_TCC`  
    `sf_zeta.tcc`, [379](#)  
`_GLIBCXX_JACOBI_CN`  
    GNU Extended Mathematical Special Functions, [53](#)  
`_GLIBCXX_JACOBI_DN`  
    GNU Extended Mathematical Special Functions, [53](#)  
`_GLIBCXX_JACOBI_SN`  
    GNU Extended Mathematical Special Functions, [53](#)  
`_N_FGH`  
    `std::__detail::_Airy_series`, [310](#)  
`_Num_Euler_Maclaurin_zeta`  
    `std::__detail`, [288](#)  
`_S_2pi_3`  
    `std::__detail::_Airy`, [293](#)  
`_S_5pi_6`  
    `std::__detail::_Airy`, [293](#)  
`_S_Ai`  
    `std::__detail::_Airy_series`, [306](#)  
`_S_Ai0`  
    `std::__detail::_Airy_series`, [310](#)  
`_S_Aip0`  
    `std::__detail::_Airy_series`, [310](#)  
`_S_Airy`  
    `std::__detail::_Airy_series`, [307](#)  
`_S_Bi`  
    `std::__detail::_Airy_series`, [307](#)  
`_S_Bi0`  
    `std::__detail::_Airy_series`, [310](#)  
`_S_Bip0`  
    `std::__detail::_Airy_series`, [310](#)  
`_S_Euler_Maclaurin_zeta`  
    `std::__detail`, [288](#)

- `_S_FGH`
  - `std::__detail::Airy_series`, 308
- `_S_Fock`
  - `std::__detail::Airy_series`, 308
- `_S_Gi0`
  - `std::__detail::Airy_series`, 310
- `_S_Gip0`
  - `std::__detail::Airy_series`, 310
- `_S_Hi0`
  - `std::__detail::Airy_series`, 310
- `_S_Hip0`
  - `std::__detail::Airy_series`, 311
- `_S_NaN`
  - `std::__detail::Airy`, 294
- `_S_Scorer`
  - `std::__detail::Airy_series`, 308
- `_S_Scorer2`
  - `std::__detail::Airy_series`, 309
- `_S_absarg_ge_pio3`
  - `std::__detail::Airy_asymp`, 296
- `_S_absarg_lt_pio3`
  - `std::__detail::Airy_asymp`, 297
- `_S_c`
  - `std::__detail::Airy_asymp_data< double >`, 299
  - `std::__detail::Airy_asymp_data< float >`, 300
  - `std::__detail::Airy_asymp_data< long double >`, 301
- `_S_cNaN`
  - `std::__detail::Airy`, 293
- `_S_cheby`
  - `std::__detail::GammaLanczos< double >`, 317
  - `std::__detail::GammaLanczos< float >`, 318
  - `std::__detail::GammaLanczos< long double >`, 319
  - `std::__detail::GammaSpouge< double >`, 320
  - `std::__detail::GammaSpouge< float >`, 321
  - `std::__detail::GammaSpouge< long double >`, 322
- `_S_d`
  - `std::__detail::Airy_asymp_data< double >`, 299
  - `std::__detail::Airy_asymp_data< float >`, 300
  - `std::__detail::Airy_asymp_data< long double >`, 301
- `_S_double_factorial_table`
  - `std::__detail`, 288
- `_S_eps`
  - `std::__detail::Airy_series`, 310
- `_S_factorial_table`
  - `std::__detail`, 288
- `_S_g`
  - `std::__detail::GammaLanczos< double >`, 317
  - `std::__detail::GammaLanczos< float >`, 318
  - `std::__detail::GammaLanczos< long double >`, 319
- `_S_harmonic_denom`
  - `std::__detail`, 288
- `_S_harmonic_numer`
  - `std::__detail`, 288
- `_S_i`
  - `std::__detail::Airy`, 294
  - `std::__detail::Airy_series`, 311
- `_S_max_cd`
  - `std::__detail::Airy_asymp_data< double >`, 299
  - `std::__detail::Airy_asymp_data< float >`, 300
  - `std::__detail::Airy_asymp_data< long double >`, 301
- `_S_neg_double_factorial_table`
  - `std::__detail`, 289
- `_S_num_double_factorials`
  - `std::__detail`, 289
- `_S_num_double_factorials< double >`
  - `std::__detail`, 289
- `_S_num_double_factorials< float >`
  - `std::__detail`, 289
- `_S_num_double_factorials< long double >`
  - `std::__detail`, 289
- `_S_num_factorials`
  - `std::__detail`, 289
- `_S_num_factorials< double >`
  - `std::__detail`, 289
- `_S_num_factorials< float >`
  - `std::__detail`, 289
- `_S_num_factorials< long double >`
  - `std::__detail`, 289
- `_S_num_harmonic_numer`
  - `std::__detail`, 290
- `_S_num_neg_double_factorials`
  - `std::__detail`, 290
- `_S_num_neg_double_factorials< double >`
  - `std::__detail`, 290
- `_S_num_neg_double_factorials< float >`
  - `std::__detail`, 290
- `_S_num_neg_double_factorials< long double >`
  - `std::__detail`, 290
- `_S_num_zetam1`
  - `std::__detail`, 290
- `_S_pi`
  - `std::__detail::Airy`, 294
  - `std::__detail::Airy_series`, 311
- `_S_pi_3`
  - `std::__detail::Airy`, 294
- `_S_pi_6`
  - `std::__detail::Airy`, 294
- `_S_sqrt_pi`
  - `std::__detail::Airy`, 294
  - `std::__detail::Airy_asymp_series`, 303
  - `std::__detail::Airy_series`, 311
- `_S_zetam1`
  - `std::__detail`, 290
- `_Val`
  - `std::__detail::AiryAuxilliaryState`, 312

- std::\_\_detail::\_\_AiryState, 314
- \_\_STDCPP\_MATH\_SPEC\_FUNCS\_\_
  - specfun.h, 391
- \_\_airy
  - std::\_\_detail, 174
- \_\_airy\_ai
  - std::\_\_detail, 175
- \_\_airy\_arg
  - std::\_\_detail, 175
- \_\_airy\_bi
  - std::\_\_detail, 175
- \_\_assoc\_laguerre
  - std::\_\_detail, 176
- \_\_assoc\_legendre\_p
  - std::\_\_detail, 176
- \_\_bernoulli
  - std::\_\_detail, 178
- \_\_bernoulli\_2n
  - std::\_\_detail, 178
- \_\_bernoulli\_series
  - std::\_\_detail, 178
- \_\_beta
  - std::\_\_detail, 179
- \_\_beta\_gamma
  - std::\_\_detail, 179
- \_\_beta\_inc
  - std::\_\_detail, 180
- \_\_beta\_lgamma
  - std::\_\_detail, 181
- \_\_beta\_product
  - std::\_\_detail, 181
- \_\_bincoef
  - std::\_\_detail, 182
- \_\_binomial\_cdf
  - std::\_\_detail, 183
- \_\_binomial\_cdfc
  - std::\_\_detail, 183
- \_\_binomial\_pdf
  - std::\_\_detail, 184
- \_\_bose\_einstein
  - std::\_\_detail, 184
- \_\_chebyshev\_recur
  - std::\_\_detail, 185
- \_\_chebyshev\_t
  - std::\_\_detail, 185
- \_\_chebyshev\_u
  - std::\_\_detail, 186
- \_\_chebyshev\_v
  - std::\_\_detail, 186
- \_\_chebyshev\_w
  - std::\_\_detail, 187
- \_\_chi\_squared\_pdf
  - std::\_\_detail, 187
- \_\_chi\_squared\_pdfc
  - std::\_\_detail, 188
- \_\_chshint
  - std::\_\_detail, 188
- \_\_chshint\_cont\_frac
  - std::\_\_detail, 188
- \_\_chshint\_series
  - std::\_\_detail, 188
- \_\_clamp\_0\_m2pi
  - std::\_\_detail, 189
- \_\_clamp\_pi
  - std::\_\_detail, 189
- \_\_clausen
  - std::\_\_detail, 189
- \_\_clausen\_c
  - std::\_\_detail, 190
- \_\_clausen\_s
  - std::\_\_detail, 191
- \_\_cmplx
  - std::\_\_detail::\_\_Airy\_asymp, 296
- \_\_comp\_ellint\_1
  - std::\_\_detail, 192
- \_\_comp\_ellint\_2
  - std::\_\_detail, 192
- \_\_comp\_ellint\_3
  - std::\_\_detail, 193
- \_\_comp\_ellint\_d
  - std::\_\_detail, 193
- \_\_comp\_ellint\_rf
  - std::\_\_detail, 193
- \_\_comp\_ellint\_rg
  - std::\_\_detail, 194
- \_\_conf\_hyperg
  - std::\_\_detail, 194
- \_\_conf\_hyperg\_lim
  - std::\_\_detail, 194
- \_\_conf\_hyperg\_lim\_series
  - std::\_\_detail, 195
- \_\_conf\_hyperg\_luke
  - std::\_\_detail, 195
- \_\_conf\_hyperg\_series
  - std::\_\_detail, 195
- \_\_cos\_pi
  - std::\_\_detail, 196
- \_\_cosh\_pi
  - std::\_\_detail, 196
- \_\_coshint
  - std::\_\_detail, 197
- \_\_cpp\_lib\_math\_special\_functions
  - specfun.h, 391
- \_\_cyl\_bessel
  - std::\_\_detail, 197
- \_\_cyl\_bessel\_i
  - std::\_\_detail, 198
- \_\_cyl\_bessel\_ij\_series

- `std::__detail`, 198
- `__cyl_bessel_ik`
  - `std::__detail`, 199
- `__cyl_bessel_ik_asymp`
  - `std::__detail`, 199
- `__cyl_bessel_ik_steel`
  - `std::__detail`, 200
- `__cyl_bessel_j`
  - `std::__detail`, 200
- `__cyl_bessel_jn`
  - `std::__detail`, 200
- `__cyl_bessel_jn_asymp`
  - `std::__detail`, 201
- `__cyl_bessel_jn_neg_arg`
  - `std::__detail`, 201
- `__cyl_bessel_jn_steel`
  - `std::__detail`, 201
- `__cyl_bessel_k`
  - `std::__detail`, 202
- `__cyl_hankel_1`
  - `std::__detail`, 202, 203
- `__cyl_hankel_2`
  - `std::__detail`, 203, 204
- `__cyl_neumann`
  - `std::__detail`, 204
- `__cyl_neumann_n`
  - `std::__detail`, 204
- `__dawson`
  - `std::__detail`, 205
- `__dawson_cont_frac`
  - `std::__detail`, 205
- `__dawson_series`
  - `std::__detail`, 206
- `__debye_region`
  - `std::__detail`, 206
- `__dilog`
  - `std::__detail`, 206
- `__dirichlet_beta`
  - `std::__detail`, 206, 207
- `__dirichlet_eta`
  - `std::__detail`, 207, 208
- `__dirichlet_lambda`
  - `std::__detail`, 208
- `__double_factorial`
  - `std::__detail`, 208
- `__ellint_1`
  - `std::__detail`, 209
- `__ellint_2`
  - `std::__detail`, 209
- `__ellint_3`
  - `std::__detail`, 210
- `__ellint_cel`
  - `std::__detail`, 210
- `__ellint_d`
  - `std::__detail`, 210
- `__ellint_el1`
  - `std::__detail`, 211
- `__ellint_el2`
  - `std::__detail`, 211
- `__ellint_el3`
  - `std::__detail`, 211
- `__ellint_rc`
  - `std::__detail`, 211
- `__ellint_rd`
  - `std::__detail`, 212
- `__ellint_rf`
  - `std::__detail`, 213
- `__ellint_rg`
  - `std::__detail`, 213
- `__ellint_rj`
  - `std::__detail`, 214
- `__ellnome`
  - `std::__detail`, 215
- `__ellnome_k`
  - `std::__detail`, 215
- `__ellnome_series`
  - `std::__detail`, 215
- `__expint`
  - `std::__detail`, 215, 216
- `__expint_E1`
  - `std::__detail`, 217
- `__expint_E1_asymp`
  - `std::__detail`, 217
- `__expint_E1_series`
  - `std::__detail`, 218
- `__expint_Ei`
  - `std::__detail`, 218
- `__expint_Ei_asymp`
  - `std::__detail`, 219
- `__expint_Ei_series`
  - `std::__detail`, 219
- `__expint_En_cont_frac`
  - `std::__detail`, 220
- `__expint_En_recursion`
  - `std::__detail`, 220
- `__expint_En_series`
  - `std::__detail`, 221
- `__expint_asymp`
  - `std::__detail`, 216
- `__expint_large_n`
  - `std::__detail`, 221
- `__exponential_cdf`
  - `std::__detail`, 222
- `__exponential_pdf`
  - `std::__detail`, 222
- `__factorial`
  - `std::__detail`, 222
  - `std::__detail::Factorial_table`, 315

- `__fermi_dirac`
  - `std::__detail`, 222
- `__fisher_f_cdf`
  - `std::__detail`, 223
- `__fisher_f_cdfc`
  - `std::__detail`, 223
- `__fock_airy`
  - `std::__detail`, 224
- `__fresnel`
  - `std::__detail`, 224
- `__fresnel_cont_frac`
  - `std::__detail`, 225
- `__fresnel_series`
  - `std::__detail`, 225
- `__gamma`
  - `std::__detail`, 225
- `__gamma_cdf`
  - `std::__detail`, 226
- `__gamma_cdfc`
  - `std::__detail`, 226
- `__gamma_cont_frac`
  - `std::__detail`, 226
- `__gamma_pdf`
  - `std::__detail`, 226
- `__gamma_series`
  - `std::__detail`, 227
- `__gamma_temme`
  - `std::__detail`, 227
- `__gauss`
  - `std::__detail`, 228
- `__gegenbauer_poly`
  - `std::__detail`, 228
- `__gnu_cxx`, 143
- `__gnu_cxx::__sincos_t`
  - `cos_value`, 291
  - `sin_value`, 291
- `__gnu_cxx::__sincos_t< _Tp >`, 291
- `__hankel`
  - `std::__detail`, 228
- `__hankel_debye`
  - `std::__detail`, 229
- `__hankel_params`
  - `std::__detail`, 229
- `__hankel_uniform`
  - `std::__detail`, 229
- `__hankel_uniform_olver`
  - `std::__detail`, 230
- `__hankel_uniform_outer`
  - `std::__detail`, 230
- `__hankel_uniform_sum`
  - `std::__detail`, 231
- `__harmonic_number`
  - `std::__detail`, 232
- `__heuman_lambda`
  - `std::__detail`, 232
- `__hurwitz_zeta`
  - `std::__detail`, 232
- `__hurwitz_zeta_euler_maclaurin`
  - `std::__detail`, 232
- `__hurwitz_zeta_polylog`
  - `std::__detail`, 233
- `__hydrogen`
  - `std::__detail`, 233
- `__hyperg`
  - `std::__detail`, 233
- `__hyperg_luke`
  - `std::__detail`, 234
- `__hyperg_reflect`
  - `std::__detail`, 234
- `__hyperg_series`
  - `std::__detail`, 235
- `__ibeta_cont_frac`
  - `std::__detail`, 235
- `__jacobi_sncndn`
  - `std::__detail`, 237
- `__jacobi_zeta`
  - `std::__detail`, 237
- `__laguerre`
  - `std::__detail`, 237
- `__legendre_q`
  - `std::__detail`, 238
- `__log_bincoef`
  - `std::__detail`, 238, 239
- `__log_bincoef_sign`
  - `std::__detail`, 239
- `__log_double_factorial`
  - `std::__detail`, 240
- `__log_factorial`
  - `std::__detail`, 240
  - `std::__detail::Factorial_table`, 315
- `__log_gamma`
  - `std::__detail`, 240, 241
- `__log_gamma1p_lanczos`
  - `std::__detail`, 241
- `__log_gamma1p_spouge`
  - `std::__detail`, 242
- `__log_gamma_bernoulli`
  - `std::__detail`, 242
- `__log_gamma_sign`
  - `std::__detail`, 243
- `__log_pochhammer`
  - `std::__detail`, 243
- `__log_pochhammer_lower`
  - `std::__detail`, 244
- `__logint`
  - `std::__detail`, 244
- `__lognormal_cdf`
  - `std::__detail`, 245

- `__lognormal_pdf`
  - `std::__detail`, 245
- `__max_FGH`
  - `std::__detail`, 287
- `__max_FGH< double >`
  - `std::__detail`, 287
- `__max_FGH< float >`
  - `std::__detail`, 288
- `__n`
  - `std::__detail::Factorial_table`, 315
- `__normal_cdf`
  - `std::__detail`, 245
- `__normal_pdf`
  - `std::__detail`, 245
- `__owens_t`
  - `std::__detail`, 245
- `__pgamma`
  - `std::__detail`, 246
- `__pochhammer`
  - `std::__detail`, 246
- `__pochhammer_lower`
  - `std::__detail`, 246
- `__polar_pi`
  - `std::__detail`, 247
- `__poly_hermite`
  - `std::__detail`, 247
- `__poly_hermite_asymp`
  - `std::__detail`, 247
- `__poly_hermite_recursion`
  - `std::__detail`, 248
- `__poly_jacobi`
  - `std::__detail`, 249
- `__poly_laguerre`
  - `std::__detail`, 249
- `__poly_laguerre_hyperg`
  - `std::__detail`, 250
- `__poly_laguerre_large_n`
  - `std::__detail`, 250
- `__poly_laguerre_recursion`
  - `std::__detail`, 251
- `__poly_legendre_p`
  - `std::__detail`, 252
- `__poly_radial_jacobi`
  - `std::__detail`, 252
- `__polylog`
  - `std::__detail`, 253
- `__polylog_exp`
  - `std::__detail`, 254
- `__polylog_exp_asymp`
  - `std::__detail`, 254
- `__polylog_exp_int_neg`
  - `std::__detail`, 255, 256
- `__polylog_exp_int_pos`
  - `std::__detail`, 256
- `__polylog_exp_neg`
  - `std::__detail`, 257
- `__polylog_exp_neg_even`
  - `std::__detail`, 259
- `__polylog_exp_neg_odd`
  - `std::__detail`, 260
- `__polylog_exp_negative_real_part`
  - `std::__detail`, 260
- `__polylog_exp_pos`
  - `std::__detail`, 261, 262
- `__polylog_exp_real_neg`
  - `std::__detail`, 263
- `__polylog_exp_real_pos`
  - `std::__detail`, 264
- `__psi`
  - `std::__detail`, 265
- `__psi_asymp`
  - `std::__detail`, 265
- `__psi_series`
  - `std::__detail`, 266
- `__qgamma`
  - `std::__detail`, 266
- `__rice_pdf`
  - `std::__detail`, 266
- `__riemann_zeta`
  - `std::__detail`, 267
- `__riemann_zeta_alt`
  - `std::__detail`, 267
- `__riemann_zeta_euler_maclaurin`
  - `std::__detail`, 268
- `__riemann_zeta_glob`
  - `std::__detail`, 268
- `__riemann_zeta_m_1`
  - `std::__detail`, 268
- `__riemann_zeta_m_1_sum`
  - `std::__detail`, 269
- `__riemann_zeta_product`
  - `std::__detail`, 269
- `__riemann_zeta_sum`
  - `std::__detail`, 269
- `__sin_pi`
  - `std::__detail`, 270
- `__sinc`
  - `std::__detail`, 270
- `__sinc_pi`
  - `std::__detail`, 270
- `__sincos`
  - `std::__detail`, 271
- `__sincos_pi`
  - `std::__detail`, 271
- `__sincosint`
  - `std::__detail`, 271
- `__sincosint_asymp`
  - `std::__detail`, 272

- `__sincosint_cont_frac`
  - `std::__detail`, 272
- `__sincosint_series`
  - `std::__detail`, 272
- `__sinh_pi`
  - `std::__detail`, 272, 273
- `__sinhc`
  - `std::__detail`, 273
- `__sinhc_pi`
  - `std::__detail`, 273
- `__sinhint`
  - `std::__detail`, 273
- `__sph_bessel`
  - `std::__detail`, 274
- `__sph_bessel_ik`
  - `std::__detail`, 275
- `__sph_bessel_jn`
  - `std::__detail`, 275
- `__sph_bessel_jn_neg_arg`
  - `std::__detail`, 276
- `__sph_hankel`
  - `std::__detail`, 276
- `__sph_hankel_1`
  - `std::__detail`, 276, 277
- `__sph_hankel_2`
  - `std::__detail`, 277, 278
- `__sph_harmonic`
  - `std::__detail`, 278
- `__sph_legendre`
  - `std::__detail`, 279
- `__sph_neumann`
  - `std::__detail`, 279, 280
- `__student_t_cdf`
  - `std::__detail`, 280
- `__student_t_cdfc`
  - `std::__detail`, 281
- `__tan_pi`
  - `std::__detail`, 281
- `__tanh_pi`
  - `std::__detail`, 281, 282
- `__tgamma`
  - `std::__detail`, 282
- `__tgamma_lower`
  - `std::__detail`, 282
- `__theta_1`
  - `std::__detail`, 282
- `__theta_2`
  - `std::__detail`, 283
- `__theta_2_asymp`
  - `std::__detail`, 283
- `__theta_2_sum`
  - `std::__detail`, 283
- `__theta_3`
  - `std::__detail`, 283
- `__theta_3_asymp`
  - `std::__detail`, 284
- `__theta_3_sum`
  - `std::__detail`, 284
- `__theta_4`
  - `std::__detail`, 284
- `__theta_c`
  - `std::__detail`, 285
- `__theta_d`
  - `std::__detail`, 285
- `__theta_n`
  - `std::__detail`, 285
- `__theta_s`
  - `std::__detail`, 285
- `__weibull_cdf`
  - `std::__detail`, 285
- `__weibull_pdf`
  - `std::__detail`, 286
- `__zernike`
  - `std::__detail`, 286
- `__znorm1`
  - `std::__detail`, 287
- `__znorm2`
  - `std::__detail`, 287
- `Ai`
  - `std::__detail::__AiryState`, 314
- `Aip`
  - `std::__detail::__AiryState`, 314
- `airy_ai`
  - GNU Extended Mathematical Special Functions, 53, 54
- `airy_aif`
  - GNU Extended Mathematical Special Functions, 54
- `airy_al`
  - GNU Extended Mathematical Special Functions, 54
- `airy_bi`
  - GNU Extended Mathematical Special Functions, 54, 55
- `airy_bif`
  - GNU Extended Mathematical Special Functions, 55
- `airy_bil`
  - GNU Extended Mathematical Special Functions, 56
- `assoc_laguerre`
  - C++17/IS29124 Mathematical Special Functions, 22
- `assoc_laguerref`
  - C++17/IS29124 Mathematical Special Functions, 23
- `assoc_laguerrel`
  - C++17/IS29124 Mathematical Special Functions, 23
- `assoc_legendre`
  - C++17/IS29124 Mathematical Special Functions, 23
- `assoc_legendref`
  - C++17/IS29124 Mathematical Special Functions, 24
- `assoc_legendrel`

- C++17/IS29124 Mathematical Special Functions, [24](#)
- bernoulli
  - GNU Extended Mathematical Special Functions, [56](#)
- bernoullif
  - GNU Extended Mathematical Special Functions, [56](#)
- bernoullil
  - GNU Extended Mathematical Special Functions, [56](#)
- beta
  - C++17/IS29124 Mathematical Special Functions, [24](#)
- betaf
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- betal
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- Bi
  - std::\_\_detail::\_AiryState, [314](#)
- bincoef
  - GNU Extended Mathematical Special Functions, [56](#)
- bincoeff
  - GNU Extended Mathematical Special Functions, [57](#)
- bincoefl
  - GNU Extended Mathematical Special Functions, [57](#)
- Bip
  - std::\_\_detail::\_AiryState, [314](#)
- bits/sf\_airy.tcc, [323](#)
- bits/sf\_bessel.tcc, [325](#)
- bits/sf\_beta.tcc, [328](#)
- bits/sf\_cardinal.tcc, [329](#)
- bits/sf\_chebyshev.tcc, [331](#)
- bits/sf\_dawson.tcc, [332](#)
- bits/sf\_distributions.tcc, [334](#)
- bits/sf\_ellint.tcc, [336](#)
- bits/sf\_expint.tcc, [339](#)
- bits/sf\_fresnel.tcc, [341](#)
- bits/sf\_gamma.tcc, [343](#)
- bits/sf\_gegenbauer.tcc, [350](#)
- bits/sf\_hankel.tcc, [351](#)
- bits/sf\_hermite.tcc, [354](#)
- bits/sf\_hydrogen.tcc, [355](#)
- bits/sf\_hyperg.tcc, [357](#)
- bits/sf\_hypint.tcc, [359](#)
- bits/sf\_jacobi.tcc, [360](#)
- bits/sf\_laguerre.tcc, [362](#)
- bits/sf\_legendre.tcc, [363](#)
- bits/sf\_mod\_bessel.tcc, [365](#)
- bits/sf\_owens\_t.tcc, [367](#)
- bits/sf\_polylog.tcc, [368](#)
- bits/sf\_theta.tcc, [371](#)
- bits/sf\_trig.tcc, [373](#)
- bits/sf\_trigint.tcc, [375](#)
- bits/sf\_zeta.tcc, [377](#)
- bits/specfun.h, [380](#)
- bose\_einstein
  - GNU Extended Mathematical Special Functions, [57](#)
- bose\_einsteinl
  - GNU Extended Mathematical Special Functions, [57](#)
- C++ Mathematical Special Functions, [19](#)
- C++17/IS29124 Mathematical Special Functions, [20](#)
  - assoc\_laguerre, [22](#)
  - assoc\_laguerref, [23](#)
  - assoc\_laguerrel, [23](#)
  - assoc\_legendre, [23](#)
  - assoc\_legendref, [24](#)
  - assoc\_legendrel, [24](#)
  - beta, [24](#)
  - betaf, [25](#)
  - betal, [25](#)
  - comp\_ellint\_1, [25](#)
  - comp\_ellint\_1f, [26](#)
  - comp\_ellint\_1l, [26](#)
  - comp\_ellint\_2, [26](#)
  - comp\_ellint\_2f, [27](#)
  - comp\_ellint\_2l, [27](#)
  - comp\_ellint\_3, [27](#)
  - comp\_ellint\_3f, [28](#)
  - comp\_ellint\_3l, [28](#)
  - cyl\_bessel\_i, [28](#)
  - cyl\_bessel\_if, [29](#)
  - cyl\_bessel\_il, [29](#)
  - cyl\_bessel\_j, [29](#)
  - cyl\_bessel\_jf, [30](#)
  - cyl\_bessel\_jl, [30](#)
  - cyl\_bessel\_k, [30](#)
  - cyl\_bessel\_kf, [31](#)
  - cyl\_bessel\_kl, [31](#)
  - cyl\_neumann, [31](#)
  - cyl\_neumannf, [32](#)
  - cyl\_neumannl, [32](#)
  - ellint\_1, [32](#)
  - ellint\_1f, [33](#)
  - ellint\_1l, [33](#)
  - ellint\_2, [33](#)
  - ellint\_2f, [34](#)
  - ellint\_2l, [34](#)
  - ellint\_3, [34](#)
  - ellint\_3f, [35](#)
  - ellint\_3l, [35](#)
  - expint, [36](#)
  - expintf, [36](#)
  - expintl, [36](#)
  - hermite, [36](#)
  - hermitef, [37](#)
  - hermitel, [37](#)
  - laguerre, [37](#)
  - laguerref, [38](#)



- laguerrel, [38](#)
- legendre, [38](#)
- legendref, [39](#)
- legendrel, [39](#)
- riemann\_zeta, [39](#)
- riemann\_zetaf, [40](#)
- riemann\_zetal, [40](#)
- sph\_bessel, [40](#)
- sph\_besself, [41](#)
- sph\_bessell, [41](#)
- sph\_legendre, [41](#)
- sph\_legendref, [42](#)
- sph\_legendrel, [42](#)
- sph\_neumann, [42](#)
- sph\_neumannf, [43](#)
- sph\_neumannl, [43](#)
- COSINT
  - std::\_\_detail, [174](#)
- chebyshev\_t
  - GNU Extended Mathematical Special Functions, [57](#)
- chebyshev\_tf
  - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev\_tl
  - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev\_u
  - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev\_uf
  - GNU Extended Mathematical Special Functions, [59](#)
- chebyshev\_ul
  - GNU Extended Mathematical Special Functions, [59](#)
- chebyshev\_v
  - GNU Extended Mathematical Special Functions, [59](#)
- chebyshev\_vf
  - GNU Extended Mathematical Special Functions, [60](#)
- chebyshev\_vl
  - GNU Extended Mathematical Special Functions, [60](#)
- chebyshev\_w
  - GNU Extended Mathematical Special Functions, [60](#)
- chebyshev\_wf
  - GNU Extended Mathematical Special Functions, [61](#)
- chebyshev\_wl
  - GNU Extended Mathematical Special Functions, [61](#)
- clausen
  - GNU Extended Mathematical Special Functions, [61](#), [62](#)
- clausen\_c
  - GNU Extended Mathematical Special Functions, [62](#)
- clausen\_cf
  - GNU Extended Mathematical Special Functions, [63](#)
- clausen\_cl
  - GNU Extended Mathematical Special Functions, [63](#)
- clausen\_s
  - GNU Extended Mathematical Special Functions, [63](#)
- clausen\_sf
  - GNU Extended Mathematical Special Functions, [64](#)
- clausen\_sl
  - GNU Extended Mathematical Special Functions, [64](#)
- clausenf
  - GNU Extended Mathematical Special Functions, [64](#)
- clausenl
  - GNU Extended Mathematical Special Functions, [65](#)
- comp\_ellint\_1
  - C++17/IS29124 Mathematical Special Functions, [25](#)
- comp\_ellint\_1f
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp\_ellint\_1l
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp\_ellint\_2
  - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp\_ellint\_2f
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- comp\_ellint\_2l
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- comp\_ellint\_3
  - C++17/IS29124 Mathematical Special Functions, [27](#)
- comp\_ellint\_3f
  - C++17/IS29124 Mathematical Special Functions, [28](#)
- comp\_ellint\_3l
  - C++17/IS29124 Mathematical Special Functions, [28](#)
- comp\_ellint\_d
  - GNU Extended Mathematical Special Functions, [65](#)
- comp\_ellint\_df
  - GNU Extended Mathematical Special Functions, [66](#)
- comp\_ellint\_dl
  - GNU Extended Mathematical Special Functions, [66](#)
- comp\_ellint\_rf
  - GNU Extended Mathematical Special Functions, [66](#)
- comp\_ellint\_rg
  - GNU Extended Mathematical Special Functions, [67](#)
- conf\_hyperg
  - GNU Extended Mathematical Special Functions, [68](#)
- conf\_hyperg\_lim
  - GNU Extended Mathematical Special Functions, [68](#)
- conf\_hyperg\_limf
  - GNU Extended Mathematical Special Functions, [68](#)
- conf\_hyperg\_liml
  - GNU Extended Mathematical Special Functions, [69](#)
- conf\_hypergf
  - GNU Extended Mathematical Special Functions, [69](#)
- conf\_hypergl
  - GNU Extended Mathematical Special Functions, [69](#)
- cos\_pi
  - GNU Extended Mathematical Special Functions, [69](#)
- cos\_pif
  - GNU Extended Mathematical Special Functions, [70](#)
- cos\_pil
  - GNU Extended Mathematical Special Functions, [70](#)
- cos\_value

- `__gnu_cxx::__sincos_t`, 291
- `cosh_pi`
  - GNU Extended Mathematical Special Functions, 70
- `cosh_pif`
  - GNU Extended Mathematical Special Functions, 71
- `cosh_pil`
  - GNU Extended Mathematical Special Functions, 71
- `coshint`
  - GNU Extended Mathematical Special Functions, 71
- `coshintf`
  - GNU Extended Mathematical Special Functions, 72
- `coshintl`
  - GNU Extended Mathematical Special Functions, 72
- `cosint`
  - GNU Extended Mathematical Special Functions, 72
- `cosintf`
  - GNU Extended Mathematical Special Functions, 72
- `cosintl`
  - GNU Extended Mathematical Special Functions, 73
- `cyl_bessel_i`
  - C++17/IS29124 Mathematical Special Functions, 28
- `cyl_bessel_if`
  - C++17/IS29124 Mathematical Special Functions, 29
- `cyl_bessel_il`
  - C++17/IS29124 Mathematical Special Functions, 29
- `cyl_bessel_j`
  - C++17/IS29124 Mathematical Special Functions, 29
- `cyl_bessel_jf`
  - C++17/IS29124 Mathematical Special Functions, 30
- `cyl_bessel_jl`
  - C++17/IS29124 Mathematical Special Functions, 30
- `cyl_bessel_k`
  - C++17/IS29124 Mathematical Special Functions, 30
- `cyl_bessel_kf`
  - C++17/IS29124 Mathematical Special Functions, 31
- `cyl_bessel_kl`
  - C++17/IS29124 Mathematical Special Functions, 31
- `cyl_hankel_1`
  - GNU Extended Mathematical Special Functions, 73, 74
- `cyl_hankel_1f`
  - GNU Extended Mathematical Special Functions, 74
- `cyl_hankel_1l`
  - GNU Extended Mathematical Special Functions, 74, 75
- `cyl_hankel_2`
  - GNU Extended Mathematical Special Functions, 75, 76
- `cyl_hankel_2f`
  - GNU Extended Mathematical Special Functions, 76
- `cyl_hankel_2l`
  - GNU Extended Mathematical Special Functions, 76, 77
- `cyl_neumann`
  - C++17/IS29124 Mathematical Special Functions, 31
- `cyl_neumannf`
  - C++17/IS29124 Mathematical Special Functions, 32
- `cyl_neumannl`
  - C++17/IS29124 Mathematical Special Functions, 32
- `dawson`
  - GNU Extended Mathematical Special Functions, 77
- `dawsonf`
  - GNU Extended Mathematical Special Functions, 77
- `dawsonl`
  - GNU Extended Mathematical Special Functions, 78
- `dilog`
  - GNU Extended Mathematical Special Functions, 78
- `dilogf`
  - GNU Extended Mathematical Special Functions, 78
- `dilogl`
  - GNU Extended Mathematical Special Functions, 78
- `dirichlet_beta`
  - GNU Extended Mathematical Special Functions, 79
- `dirichlet_betaf`
  - GNU Extended Mathematical Special Functions, 79
- `dirichlet_betall`
  - GNU Extended Mathematical Special Functions, 79
- `dirichlet_eta`
  - GNU Extended Mathematical Special Functions, 79
- `dirichlet_etaf`
  - GNU Extended Mathematical Special Functions, 80
- `dirichlet_etall`
  - GNU Extended Mathematical Special Functions, 80
- `dirichlet_lambda`
  - GNU Extended Mathematical Special Functions, 80
- `dirichlet_lambdaf`
  - GNU Extended Mathematical Special Functions, 81
- `dirichlet_lambdal`
  - GNU Extended Mathematical Special Functions, 81
- `double_factorial`
  - GNU Extended Mathematical Special Functions, 81
- `double_factorialf`
  - GNU Extended Mathematical Special Functions, 81
- `double_factoriall`
  - GNU Extended Mathematical Special Functions, 81
- `ellint_1`
  - C++17/IS29124 Mathematical Special Functions, 32
- `ellint_1f`
  - C++17/IS29124 Mathematical Special Functions, 33
- `ellint_1l`
  - C++17/IS29124 Mathematical Special Functions, 33
- `ellint_2`
  - C++17/IS29124 Mathematical Special Functions, 33
- `ellint_2f`
  - C++17/IS29124 Mathematical Special Functions, 34
- `ellint_2l`
  - C++17/IS29124 Mathematical Special Functions, 34

- ellint\_3
  - C++17/IS29124 Mathematical Special Functions, [34](#)
- ellint\_3f
  - C++17/IS29124 Mathematical Special Functions, [35](#)
- ellint\_3l
  - C++17/IS29124 Mathematical Special Functions, [35](#)
- ellint\_cel
  - GNU Extended Mathematical Special Functions, [81](#)
- ellint\_celf
  - GNU Extended Mathematical Special Functions, [82](#)
- ellint\_cell
  - GNU Extended Mathematical Special Functions, [82](#)
- ellint\_d
  - GNU Extended Mathematical Special Functions, [82](#)
- ellint\_df
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_dl
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_el1
  - GNU Extended Mathematical Special Functions, [83](#)
- ellint\_el1f
  - GNU Extended Mathematical Special Functions, [84](#)
- ellint\_el1l
  - GNU Extended Mathematical Special Functions, [84](#)
- ellint\_el2
  - GNU Extended Mathematical Special Functions, [84](#)
- ellint\_el2f
  - GNU Extended Mathematical Special Functions, [84](#)
- ellint\_el2l
  - GNU Extended Mathematical Special Functions, [85](#)
- ellint\_el3
  - GNU Extended Mathematical Special Functions, [85](#)
- ellint\_el3f
  - GNU Extended Mathematical Special Functions, [85](#)
- ellint\_el3l
  - GNU Extended Mathematical Special Functions, [86](#)
- ellint\_rc
  - GNU Extended Mathematical Special Functions, [86](#)
- ellint\_rcf
  - GNU Extended Mathematical Special Functions, [87](#)
- ellint\_rcl
  - GNU Extended Mathematical Special Functions, [87](#)
- ellint\_rd
  - GNU Extended Mathematical Special Functions, [87](#)
- ellint\_rdf
  - GNU Extended Mathematical Special Functions, [88](#)
- ellint\_rdl
  - GNU Extended Mathematical Special Functions, [88](#)
- ellint\_rf
  - GNU Extended Mathematical Special Functions, [88](#)
- ellint\_rff
  - GNU Extended Mathematical Special Functions, [88](#)
- ellint\_rfl
  - GNU Extended Mathematical Special Functions, [89](#)
- ellint\_rg
  - GNU Extended Mathematical Special Functions, [89](#)
- ellint\_rgf
  - GNU Extended Mathematical Special Functions, [90](#)
- ellint\_rgl
  - GNU Extended Mathematical Special Functions, [90](#)
- ellint\_rj
  - GNU Extended Mathematical Special Functions, [90](#)
- ellint\_rjf
  - GNU Extended Mathematical Special Functions, [91](#)
- ellint\_rjl
  - GNU Extended Mathematical Special Functions, [91](#)
- ellnome
  - GNU Extended Mathematical Special Functions, [91](#)
- ellnomef
  - GNU Extended Mathematical Special Functions, [91](#)
- ellnomel
  - GNU Extended Mathematical Special Functions, [92](#)
- evenzeta
  - std::\_\_detail, [287](#)
- expint
  - C++17/IS29124 Mathematical Special Functions, [36](#)
  - GNU Extended Mathematical Special Functions, [92](#)
- expintf
  - C++17/IS29124 Mathematical Special Functions, [36](#)
  - GNU Extended Mathematical Special Functions, [92](#)
- expintl
  - C++17/IS29124 Mathematical Special Functions, [36](#)
  - GNU Extended Mathematical Special Functions, [93](#)
- factorial
  - GNU Extended Mathematical Special Functions, [93](#)
- factorialf
  - GNU Extended Mathematical Special Functions, [93](#)
- factoriall
  - GNU Extended Mathematical Special Functions, [93](#)
- fai
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- faip
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- fermi\_dirac
  - GNU Extended Mathematical Special Functions, [93](#)
- fermi\_diracf
  - GNU Extended Mathematical Special Functions, [93](#)
- fermi\_diracl
  - GNU Extended Mathematical Special Functions, [93](#)
- fresnel\_c
  - GNU Extended Mathematical Special Functions, [93](#)
- fresnel\_cf
  - GNU Extended Mathematical Special Functions, [94](#)
- fresnel\_cl
  - GNU Extended Mathematical Special Functions, [94](#)
- fresnel\_s
  - GNU Extended Mathematical Special Functions, [94](#)

- fresnel\_sf
  - GNU Extended Mathematical Special Functions, [94](#)
- fresnel\_sl
  - GNU Extended Mathematical Special Functions, [94](#)
- GNU Extended Mathematical Special Functions, [44](#)
  - \_GLIBCXX\_JACOBI\_CN, [53](#)
  - \_GLIBCXX\_JACOBI\_DN, [53](#)
  - \_GLIBCXX\_JACOBI\_SN, [53](#)
  - airy\_ai, [53](#), [54](#)
  - airy\_aif, [54](#)
  - airy\_ail, [54](#)
  - airy\_bi, [54](#), [55](#)
  - airy\_bif, [55](#)
  - airy\_bil, [56](#)
  - bernoulli, [56](#)
  - bernoullif, [56](#)
  - bernoullil, [56](#)
  - bincoef, [56](#)
  - bincoeff, [57](#)
  - bincoefl, [57](#)
  - bose\_einstein, [57](#)
  - bose\_einsteinf, [57](#)
  - bose\_einsteinl, [57](#)
  - chebyshev\_t, [57](#)
  - chebyshev\_tf, [58](#)
  - chebyshev\_tl, [58](#)
  - chebyshev\_u, [58](#)
  - chebyshev\_uf, [59](#)
  - chebyshev\_ul, [59](#)
  - chebyshev\_v, [59](#)
  - chebyshev\_vf, [60](#)
  - chebyshev\_vl, [60](#)
  - chebyshev\_w, [60](#)
  - chebyshev\_wf, [61](#)
  - chebyshev\_wl, [61](#)
  - clausen, [61](#), [62](#)
  - clausen\_c, [62](#)
  - clausen\_cf, [63](#)
  - clausen\_cl, [63](#)
  - clausen\_s, [63](#)
  - clausen\_sf, [64](#)
  - clausen\_sl, [64](#)
  - clausenf, [64](#)
  - clausenl, [65](#)
  - comp\_ellint\_d, [65](#)
  - comp\_ellint\_df, [66](#)
  - comp\_ellint\_dl, [66](#)
  - comp\_ellint\_rf, [66](#)
  - comp\_ellint\_rg, [67](#)
  - conf\_hyperg, [68](#)
  - conf\_hyperg\_lim, [68](#)
  - conf\_hyperg\_limf, [68](#)
  - conf\_hyperg\_liml, [69](#)
  - conf\_hypergf, [69](#)
  - conf\_hypergl, [69](#)
  - cos\_pi, [69](#)
  - cos\_pif, [70](#)
  - cos\_pil, [70](#)
  - cosh\_pi, [70](#)
  - cosh\_pif, [71](#)
  - cosh\_pil, [71](#)
  - coshint, [71](#)
  - coshintf, [72](#)
  - coshintl, [72](#)
  - cosint, [72](#)
  - cosintf, [72](#)
  - cosintl, [73](#)
  - cyl\_hankel\_1, [73](#), [74](#)
  - cyl\_hankel\_1f, [74](#)
  - cyl\_hankel\_1l, [74](#), [75](#)
  - cyl\_hankel\_2, [75](#), [76](#)
  - cyl\_hankel\_2f, [76](#)
  - cyl\_hankel\_2l, [76](#), [77](#)
  - dawson, [77](#)
  - dawsonf, [77](#)
  - dawsonl, [78](#)
  - dilog, [78](#)
  - dilogf, [78](#)
  - dilogl, [78](#)
  - dirichlet\_beta, [79](#)
  - dirichlet\_betaf, [79](#)
  - dirichlet\_betall, [79](#)
  - dirichlet\_eta, [79](#)
  - dirichlet\_etaf, [80](#)
  - dirichlet\_etall, [80](#)
  - dirichlet\_lambda, [80](#)
  - dirichlet\_lambdaf, [81](#)
  - dirichlet\_lambdal, [81](#)
  - double\_factorial, [81](#)
  - double\_factorialf, [81](#)
  - double\_factoriall, [81](#)
  - ellint\_cel, [81](#)
  - ellint\_celf, [82](#)
  - ellint\_cell, [82](#)
  - ellint\_d, [82](#)
  - ellint\_df, [83](#)
  - ellint\_dl, [83](#)
  - ellint\_el1, [83](#)
  - ellint\_el1f, [84](#)
  - ellint\_el1l, [84](#)
  - ellint\_el2, [84](#)
  - ellint\_el2f, [84](#)
  - ellint\_el2l, [85](#)
  - ellint\_el3, [85](#)
  - ellint\_el3f, [85](#)
  - ellint\_el3l, [86](#)
  - ellint\_rc, [86](#)

ellint\_rcf, [87](#)  
ellint\_rcl, [87](#)  
ellint\_rd, [87](#)  
ellint\_rdf, [88](#)  
ellint\_rdl, [88](#)  
ellint\_rf, [88](#)  
ellint\_rff, [88](#)  
ellint\_rfl, [89](#)  
ellint\_rg, [89](#)  
ellint\_rgf, [90](#)  
ellint\_rgl, [90](#)  
ellint\_rj, [90](#)  
ellint\_rjf, [91](#)  
ellint\_rjl, [91](#)  
ellnome, [91](#)  
ellnomef, [91](#)  
ellnomei, [92](#)  
expint, [92](#)  
expintf, [92](#)  
expintl, [93](#)  
factorial, [93](#)  
factorialf, [93](#)  
factoriall, [93](#)  
fermi\_dirac, [93](#)  
fermi\_diracf, [93](#)  
fermi\_diracl, [93](#)  
fresnel\_c, [93](#)  
fresnel\_cf, [94](#)  
fresnel\_cl, [94](#)  
fresnel\_s, [94](#)  
fresnel\_sf, [94](#)  
fresnel\_sl, [94](#)  
gegenbauer, [94](#)  
gegenbauerf, [95](#)  
gegenbauerl, [95](#)  
heuman\_lambda, [95](#)  
heuman\_lambdaf, [96](#)  
heuman\_lambdal, [96](#)  
hurwitz\_zeta, [96](#), [97](#)  
hurwitz\_zetaf, [97](#)  
hurwitz\_zetal, [97](#)  
hyperg, [97](#)  
hypergf, [98](#)  
hypergl, [98](#)  
ibeta, [98](#)  
ibetac, [99](#)  
ibetacf, [99](#)  
ibetacl, [99](#)  
ibetaf, [99](#)  
ibetal, [100](#)  
jacobi, [100](#)  
jacobi\_cn, [100](#)  
jacobi\_cnf, [101](#)  
jacobi\_cnl, [101](#)  
jacobi\_dn, [101](#)  
jacobi\_dnf, [102](#)  
jacobi\_dnl, [102](#)  
jacobi\_sn, [102](#)  
jacobi\_snf, [103](#)  
jacobi\_snl, [103](#)  
jacobi\_zeta, [103](#)  
jacobi\_zetaf, [104](#)  
jacobi\_zetal, [104](#)  
jacobif, [104](#)  
jacobil, [104](#)  
lbincoef, [105](#)  
lbincoeff, [105](#)  
lbincoefl, [105](#)  
ldouble\_factorial, [105](#)  
ldouble\_factorialf, [105](#)  
ldouble\_factoriall, [105](#)  
legendre\_q, [105](#)  
legendre\_qf, [106](#)  
legendre\_ql, [106](#)  
lfactorial, [106](#)  
lfactorialf, [106](#)  
lfactoriall, [107](#)  
lgamma, [107](#)  
lgammaf, [107](#)  
lgammal, [107](#)  
logint, [107](#)  
logintf, [108](#)  
logintl, [108](#)  
lpochhammer, [108](#)  
lpochhammer\_lower, [108](#)  
lpochhammer\_lowerf, [108](#)  
lpochhammer\_lowerl, [108](#)  
lpochhammerf, [109](#)  
lpochhammerl, [109](#)  
owens\_t, [109](#)  
owens\_tf, [109](#)  
owens\_tl, [109](#)  
pgamma, [110](#)  
pgammaf, [110](#)  
pgammal, [110](#)  
pochhammer, [110](#)  
pochhammer\_lower, [110](#)  
pochhammer\_lowerf, [110](#)  
pochhammer\_lowerl, [110](#)  
pochhammerf, [110](#)  
pochhammerl, [111](#)  
polylog, [111](#)  
polylogf, [111](#), [112](#)  
polylogl, [112](#)  
psi, [112](#)  
psif, [113](#)  
psil, [113](#)  
qgamma, [113](#)

- qgammaf, [113](#)
- qgammal, [113](#)
- radpoly, [113](#)
- radpolyf, [114](#)
- radpolyl, [114](#)
- sin\_pi, [115](#)
- sin\_pif, [115](#)
- sin\_pil, [115](#)
- sinc, [116](#)
- sinc\_pi, [116](#)
- sinc\_pif, [117](#)
- sinc\_pil, [117](#)
- sincf, [117](#)
- sincl, [117](#)
- sincos, [117](#), [118](#)
- sincos\_pi, [118](#)
- sincos\_pif, [118](#)
- sincos\_pil, [118](#)
- sincosf, [118](#)
- sincosl, [119](#)
- sinh\_pi, [119](#)
- sinh\_pif, [119](#)
- sinh\_pil, [119](#)
- sinhc, [120](#)
- sinhc\_pi, [120](#)
- sinhc\_pif, [121](#)
- sinhc\_pil, [121](#)
- sinhcf, [121](#)
- sinhcl, [121](#)
- sinhint, [121](#)
- sinhintf, [122](#)
- sinhintl, [122](#)
- sinint, [122](#)
- sinintf, [123](#)
- sinintl, [123](#)
- sph\_bessel\_i, [123](#)
- sph\_bessel\_if, [124](#)
- sph\_bessel\_il, [124](#)
- sph\_bessel\_k, [124](#)
- sph\_bessel\_kf, [125](#)
- sph\_bessel\_kl, [125](#)
- sph\_hankel\_1, [125](#), [126](#)
- sph\_hankel\_1f, [126](#)
- sph\_hankel\_1l, [127](#)
- sph\_hankel\_2, [127](#), [128](#)
- sph\_hankel\_2f, [128](#)
- sph\_hankel\_2l, [129](#)
- sph\_harmonic, [129](#)
- sph\_harmonicf, [130](#)
- sph\_harmonicl, [130](#)
- tan\_pi, [130](#)
- tan\_pif, [131](#)
- tan\_pil, [131](#)
- tanh\_pi, [131](#)
- tanh\_pif, [132](#)
- tanh\_pil, [132](#)
- tgamma, [132](#)
- tgamma\_lower, [132](#)
- tgamma\_lowerf, [133](#)
- tgamma\_lowerl, [133](#)
- tgammaf, [133](#)
- tgammal, [134](#)
- theta\_1, [134](#)
- theta\_1f, [135](#)
- theta\_1l, [135](#)
- theta\_2, [135](#)
- theta\_2f, [135](#)
- theta\_2l, [136](#)
- theta\_3, [136](#)
- theta\_3f, [136](#)
- theta\_3l, [136](#)
- theta\_4, [137](#)
- theta\_4f, [137](#)
- theta\_4l, [137](#)
- theta\_c, [137](#)
- theta\_cf, [138](#)
- theta\_cl, [138](#)
- theta\_d, [138](#)
- theta\_df, [139](#)
- theta\_dl, [139](#)
- theta\_n, [139](#)
- theta\_nf, [140](#)
- theta\_nl, [140](#)
- theta\_s, [140](#)
- theta\_sf, [140](#)
- theta\_sl, [141](#)
- zernike, [141](#)
- zernikef, [142](#)
- zernikel, [142](#)
- gai
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- gaip
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- gegenbauer
  - GNU Extended Mathematical Special Functions, [94](#)
- gegenbauerf
  - GNU Extended Mathematical Special Functions, [95](#)
- gegenbauerl
  - GNU Extended Mathematical Special Functions, [95](#)
- hai
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- haip
  - std::\_\_detail::\_AiryAuxilliaryState, [312](#)
- hermite
  - C++17/IS29124 Mathematical Special Functions, [36](#)
- hermitef
  - C++17/IS29124 Mathematical Special Functions, [37](#)

- hermitel
  - C++17/IS29124 Mathematical Special Functions, [37](#)
- heuman\_lambda
  - GNU Extended Mathematical Special Functions, [95](#)
- heuman\_lambdaf
  - GNU Extended Mathematical Special Functions, [96](#)
- heuman\_lambdal
  - GNU Extended Mathematical Special Functions, [96](#)
- hurwitz\_zeta
  - GNU Extended Mathematical Special Functions, [96](#), [97](#)
- hurwitz\_zetaf
  - GNU Extended Mathematical Special Functions, [97](#)
- hurwitz\_zetal
  - GNU Extended Mathematical Special Functions, [97](#)
- hyperg
  - GNU Extended Mathematical Special Functions, [97](#)
- hypergf
  - GNU Extended Mathematical Special Functions, [98](#)
- hypergl
  - GNU Extended Mathematical Special Functions, [98](#)
- ibeta
  - GNU Extended Mathematical Special Functions, [98](#)
- ibetac
  - GNU Extended Mathematical Special Functions, [99](#)
- ibetacf
  - GNU Extended Mathematical Special Functions, [99](#)
- ibetacl
  - GNU Extended Mathematical Special Functions, [99](#)
- ibetaf
  - GNU Extended Mathematical Special Functions, [99](#)
- ibetal
  - GNU Extended Mathematical Special Functions, [100](#)
- inner\_radius
  - std::detail::Airy, [294](#)
  - std::detail::Airy\_default\_radii< double >, [304](#)
  - std::detail::Airy\_default\_radii< float >, [304](#)
  - std::detail::Airy\_default\_radii< long double >, [305](#)
- jacobi
  - GNU Extended Mathematical Special Functions, [100](#)
- jacobi\_cn
  - GNU Extended Mathematical Special Functions, [100](#)
- jacobi\_cnf
  - GNU Extended Mathematical Special Functions, [101](#)
- jacobi\_cnl
  - GNU Extended Mathematical Special Functions, [101](#)
- jacobi\_dn
  - GNU Extended Mathematical Special Functions, [101](#)
- jacobi\_dnf
  - GNU Extended Mathematical Special Functions, [102](#)
- jacobi\_dnl
  - GNU Extended Mathematical Special Functions, [102](#)
- jacobi\_sn
  - GNU Extended Mathematical Special Functions, [102](#)
- jacobi\_snf
  - GNU Extended Mathematical Special Functions, [103](#)
- jacobi\_snl
  - GNU Extended Mathematical Special Functions, [103](#)
- jacobi\_zeta
  - GNU Extended Mathematical Special Functions, [103](#)
- jacobi\_zetaf
  - GNU Extended Mathematical Special Functions, [104](#)
- jacobi\_zetal
  - GNU Extended Mathematical Special Functions, [104](#)
- jacobif
  - GNU Extended Mathematical Special Functions, [104](#)
- jacobil
  - GNU Extended Mathematical Special Functions, [104](#)
- laguerre
  - C++17/IS29124 Mathematical Special Functions, [37](#)
- laguerref
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- laguerrel
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- lbincoef
  - GNU Extended Mathematical Special Functions, [105](#)
- lbincoefficient
  - GNU Extended Mathematical Special Functions, [105](#)
- lbincoefficientl
  - GNU Extended Mathematical Special Functions, [105](#)
- ldouble\_factorial
  - GNU Extended Mathematical Special Functions, [105](#)
- ldouble\_factorialf
  - GNU Extended Mathematical Special Functions, [105](#)
- ldouble\_factoriall
  - GNU Extended Mathematical Special Functions, [105](#)
- legendre
  - C++17/IS29124 Mathematical Special Functions, [38](#)
- legendre\_q
  - GNU Extended Mathematical Special Functions, [105](#)
- legendre\_qf
  - GNU Extended Mathematical Special Functions, [106](#)
- legendre\_ql
  - GNU Extended Mathematical Special Functions, [106](#)
- legendref
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- legendrel
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- lfactorial
  - GNU Extended Mathematical Special Functions, [106](#)
- lfactorialf
  - GNU Extended Mathematical Special Functions, [106](#)
- lfactoriall
  - GNU Extended Mathematical Special Functions, [107](#)
- lgamma



- GNU Extended Mathematical Special Functions, [107](#)
- lgammaf
  - GNU Extended Mathematical Special Functions, [107](#)
- lgammal
  - GNU Extended Mathematical Special Functions, [107](#)
- logint
  - GNU Extended Mathematical Special Functions, [107](#)
- logintf
  - GNU Extended Mathematical Special Functions, [108](#)
- logintl
  - GNU Extended Mathematical Special Functions, [108](#)
- lpochhammer
  - GNU Extended Mathematical Special Functions, [108](#)
- lpochhammer\_lower
  - GNU Extended Mathematical Special Functions, [108](#)
- lpochhammer\_lowerf
  - GNU Extended Mathematical Special Functions, [108](#)
- lpochhammer\_lowerl
  - GNU Extended Mathematical Special Functions, [108](#)
- lpochhammerf
  - GNU Extended Mathematical Special Functions, [109](#)
- lpochhammerl
  - GNU Extended Mathematical Special Functions, [109](#)
- operator()
  - std::\_\_detail::\_\_Airy, [293](#)
  - std::\_\_detail::\_\_Airy\_asymp, [297](#)
  - std::\_\_detail::\_\_Airy\_asymp\_series, [302](#)
- outer\_radius
  - std::\_\_detail::\_\_Airy, [294](#)
  - std::\_\_detail::\_\_Airy\_default\_radII< double >, [304](#)
  - std::\_\_detail::\_\_Airy\_default\_radII< float >, [304](#)
  - std::\_\_detail::\_\_Airy\_default\_radII< long double >, [305](#)
- owens\_t
  - GNU Extended Mathematical Special Functions, [109](#)
- owens\_tf
  - GNU Extended Mathematical Special Functions, [109](#)
- owens\_tl
  - GNU Extended Mathematical Special Functions, [109](#)
- pgamma
  - GNU Extended Mathematical Special Functions, [110](#)
- pgammaf
  - GNU Extended Mathematical Special Functions, [110](#)
- pgammal
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammer
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammer\_lower
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammer\_lowerf
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammer\_lowerl
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammerf
  - GNU Extended Mathematical Special Functions, [110](#)
- pochhammerl
  - GNU Extended Mathematical Special Functions, [111](#)
- polylog
  - GNU Extended Mathematical Special Functions, [111](#)
- polylogf
  - GNU Extended Mathematical Special Functions, [111](#), [112](#)
- polylogl
  - GNU Extended Mathematical Special Functions, [112](#)
- psi
  - GNU Extended Mathematical Special Functions, [112](#)
- psif
  - GNU Extended Mathematical Special Functions, [113](#)
- psil
  - GNU Extended Mathematical Special Functions, [113](#)
- qgamma
  - GNU Extended Mathematical Special Functions, [113](#)
- qgammaf
  - GNU Extended Mathematical Special Functions, [113](#)
- qgammal
  - GNU Extended Mathematical Special Functions, [113](#)
- radpoly
  - GNU Extended Mathematical Special Functions, [113](#)
- radpolyf
  - GNU Extended Mathematical Special Functions, [114](#)
- radpolyl
  - GNU Extended Mathematical Special Functions, [114](#)
- riemann\_zeta
  - C++17/IS29124 Mathematical Special Functions, [39](#)
- riemann\_zetaf
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- riemann\_zetal
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- SININT
  - std::\_\_detail, [174](#)
- scalar\_type
  - std::\_\_detail::\_\_Airy, [293](#)
  - std::\_\_detail::\_\_Airy\_asymp\_series, [302](#)
- sf\_airy.tcc
  - \_GLIBCXX\_BITS\_SF\_AIRY\_TCC, [325](#)
- sf\_bessel.tcc
  - \_GLIBCXX\_BITS\_SF\_BESSEL\_TCC, [327](#)
- sf\_beta.tcc
  - \_GLIBCXX\_BITS\_SF\_BETA\_TCC, [329](#)
- sf\_cardinal.tcc
  - \_GLIBCXX\_BITS\_SF\_CARDINAL\_TCC, [331](#)
- sf\_chebyshev.tcc
  - \_GLIBCXX\_BITS\_SF\_CHEBYSHEV\_TCC, [332](#)
- sf\_dawson.tcc
  - \_GLIBCXX\_BITS\_SF\_DAWSON\_TCC, [333](#)



- sf\_distributions.tcc
  - \_GLIBCXX\_BITS\_SF\_DISTRIBUTIONS\_TCC, [336](#)
- sf\_ellint.tcc
  - \_GLIBCXX\_BITS\_SF\_ELLINT\_TCC, [339](#)
- sf\_expint.tcc
  - \_GLIBCXX\_BITS\_SF\_EXPINT\_TCC, [341](#)
- sf\_fresnel.tcc
  - \_GLIBCXX\_BITS\_SF\_FRESNEL\_TCC, [342](#)
- sf\_gamma.tcc
  - \_GLIBCXX\_BITS\_SF\_GAMMA\_TCC, [349](#)
- sf\_gegenbauer.tcc
  - \_GLIBCXX\_BITS\_SF\_GEGENBAUER\_TCC, [351](#)
- sf\_hankel.tcc
  - \_GLIBCXX\_BITS\_SF\_HANKEL\_TCC, [354](#)
- sf\_hermite.tcc
  - \_GLIBCXX\_BITS\_SF\_HERMITE\_TCC, [355](#)
- sf\_hydrogen.tcc
  - \_GLIBCXX\_BITS\_SF\_HYDROGEN\_TCC, [356](#)
- sf\_hyperg.tcc
  - \_GLIBCXX\_BITS\_SF\_HYPERG\_TCC, [358](#)
- sf\_hypint.tcc
  - \_GLIBCXX\_BITS\_SF\_HYPINT\_TCC, [360](#)
- sf\_jacobi.tcc
  - \_GLIBCXX\_BITS\_SF\_JACOBI\_TCC, [361](#)
- sf\_laguerre.tcc
  - \_GLIBCXX\_BITS\_SF\_LAGUERRE\_TCC, [363](#)
- sf\_legendre.tcc
  - \_GLIBCXX\_BITS\_SF\_LEGENDRE\_TCC, [365](#)
- sf\_mod\_bessel.tcc
  - \_GLIBCXX\_BITS\_SF\_MOD\_BESSEL\_TCC, [367](#)
- sf\_owens\_t.tcc
  - \_GLIBCXX\_BITS\_SF\_OWENS\_T\_TCC, [368](#)
- sf\_polylog.tcc
  - \_GLIBCXX\_BITS\_SF\_POLYLOG\_TCC, [371](#)
- sf\_theta.tcc
  - \_GLIBCXX\_BITS\_SF\_THETA\_TCC, [373](#)
- sf\_trig.tcc
  - \_GLIBCXX\_BITS\_SF\_TRIG\_TCC, [375](#)
- sf\_trigint.tcc
  - \_GLIBCXX\_BITS\_SF\_TRIGINT\_TCC, [377](#)
- sf\_zeta.tcc
  - \_GLIBCXX\_BITS\_SF\_ZETA\_TCC, [379](#)
- sin\_pi
  - GNU Extended Mathematical Special Functions, [115](#)
- sin\_pif
  - GNU Extended Mathematical Special Functions, [115](#)
- sin\_pil
  - GNU Extended Mathematical Special Functions, [115](#)
- sin\_value
  - \_\_gnu\_cxx::\_\_sincos\_t, [291](#)
- sinc
  - GNU Extended Mathematical Special Functions, [116](#)
- sinc\_pi
  - GNU Extended Mathematical Special Functions, [116](#)
- sinc\_pif
  - GNU Extended Mathematical Special Functions, [117](#)
- sinc\_pil
  - GNU Extended Mathematical Special Functions, [117](#)
- sincf
  - GNU Extended Mathematical Special Functions, [117](#)
- sincl
  - GNU Extended Mathematical Special Functions, [117](#)
- sincos
  - GNU Extended Mathematical Special Functions, [117](#), [118](#)
- sincos\_pi
  - GNU Extended Mathematical Special Functions, [118](#)
- sincos\_pif
  - GNU Extended Mathematical Special Functions, [118](#)
- sincos\_pil
  - GNU Extended Mathematical Special Functions, [118](#)
- sincosf
  - GNU Extended Mathematical Special Functions, [118](#)
- sincosl
  - GNU Extended Mathematical Special Functions, [119](#)
- sinh\_pi
  - GNU Extended Mathematical Special Functions, [119](#)
- sinh\_pif
  - GNU Extended Mathematical Special Functions, [119](#)
- sinh\_pil
  - GNU Extended Mathematical Special Functions, [119](#)
- sinhc
  - GNU Extended Mathematical Special Functions, [120](#)
- sinhc\_pi
  - GNU Extended Mathematical Special Functions, [120](#)
- sinhc\_pif
  - GNU Extended Mathematical Special Functions, [121](#)
- sinhc\_pil
  - GNU Extended Mathematical Special Functions, [121](#)
- sinhcf
  - GNU Extended Mathematical Special Functions, [121](#)
- sinhcl
  - GNU Extended Mathematical Special Functions, [121](#)
- sinhint
  - GNU Extended Mathematical Special Functions, [121](#)
- sinhintf
  - GNU Extended Mathematical Special Functions, [122](#)
- sinhintl
  - GNU Extended Mathematical Special Functions, [122](#)
- sinint
  - GNU Extended Mathematical Special Functions, [122](#)
- sinintf
  - GNU Extended Mathematical Special Functions, [123](#)
- sinintl
  - GNU Extended Mathematical Special Functions, [123](#)
- specfun.h
  - \_\_STDCPP\_MATH\_SPEC\_FUNCS\_\_, [391](#)
  - \_\_cpp\_lib\_math\_special\_functions, [391](#)

- sph\_bessel
  - C++17/IS29124 Mathematical Special Functions, [40](#)
- sph\_bessel\_i
  - GNU Extended Mathematical Special Functions, [123](#)
- sph\_bessel\_if
  - GNU Extended Mathematical Special Functions, [124](#)
- sph\_bessel\_il
  - GNU Extended Mathematical Special Functions, [124](#)
- sph\_bessel\_k
  - GNU Extended Mathematical Special Functions, [124](#)
- sph\_bessel\_kf
  - GNU Extended Mathematical Special Functions, [125](#)
- sph\_bessel\_kl
  - GNU Extended Mathematical Special Functions, [125](#)
- sph\_besself
  - C++17/IS29124 Mathematical Special Functions, [41](#)
- sph\_bessell
  - C++17/IS29124 Mathematical Special Functions, [41](#)
- sph\_hankel\_1
  - GNU Extended Mathematical Special Functions, [125](#), [126](#)
- sph\_hankel\_1f
  - GNU Extended Mathematical Special Functions, [126](#)
- sph\_hankel\_1l
  - GNU Extended Mathematical Special Functions, [127](#)
- sph\_hankel\_2
  - GNU Extended Mathematical Special Functions, [127](#), [128](#)
- sph\_hankel\_2f
  - GNU Extended Mathematical Special Functions, [128](#)
- sph\_hankel\_2l
  - GNU Extended Mathematical Special Functions, [129](#)
- sph\_harmonic
  - GNU Extended Mathematical Special Functions, [129](#)
- sph\_harmonicf
  - GNU Extended Mathematical Special Functions, [130](#)
- sph\_harmonicl
  - GNU Extended Mathematical Special Functions, [130](#)
- sph\_legendre
  - C++17/IS29124 Mathematical Special Functions, [41](#)
- sph\_legendref
  - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph\_legendrel
  - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph\_neumann
  - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph\_neumannf
  - C++17/IS29124 Mathematical Special Functions, [43](#)
- sph\_neumannl
  - C++17/IS29124 Mathematical Special Functions, [43](#)
- std, [152](#)
- std::\_\_detail, [154](#)
  - \_Num\_Euler\_Maclaurin\_zeta, [288](#)
  - \_S\_Euler\_Maclaurin\_zeta, [288](#)
  - \_S\_double\_factorial\_table, [288](#)
  - \_S\_factorial\_table, [288](#)
  - \_S\_harmonic\_denom, [288](#)
  - \_S\_harmonic\_numer, [288](#)
  - \_S\_neg\_double\_factorial\_table, [289](#)
  - \_S\_num\_double\_factorials, [289](#)
  - \_S\_num\_double\_factorials< double >, [289](#)
  - \_S\_num\_double\_factorials< float >, [289](#)
  - \_S\_num\_double\_factorials< long double >, [289](#)
  - \_S\_num\_factorials, [289](#)
  - \_S\_num\_factorials< double >, [289](#)
  - \_S\_num\_factorials< float >, [289](#)
  - \_S\_num\_factorials< long double >, [289](#)
  - \_S\_num\_harmonic\_numer, [290](#)
  - \_S\_num\_neg\_double\_factorials, [290](#)
  - \_S\_num\_neg\_double\_factorials< double >, [290](#)
  - \_S\_num\_neg\_double\_factorials< float >, [290](#)
  - \_S\_num\_neg\_double\_factorials< long double >, [290](#)
  - \_S\_num\_zetam1, [290](#)
  - \_S\_zetam1, [290](#)
  - \_\_airy, [174](#)
  - \_\_airy\_ai, [175](#)
  - \_\_airy\_arg, [175](#)
  - \_\_airy\_bi, [175](#)
  - \_\_assoc\_laguerre, [176](#)
  - \_\_assoc\_legendre\_p, [176](#)
  - \_\_bernoulli, [178](#)
  - \_\_bernoulli\_2n, [178](#)
  - \_\_bernoulli\_series, [178](#)
  - \_\_beta, [179](#)
  - \_\_beta\_gamma, [179](#)
  - \_\_beta\_inc, [180](#)
  - \_\_beta\_lgamma, [181](#)
  - \_\_beta\_product, [181](#)
  - \_\_bincoef, [182](#)
  - \_\_binomial\_cdf, [183](#)
  - \_\_binomial\_cdfc, [183](#)
  - \_\_binomial\_pdf, [184](#)
  - \_\_bose\_einstein, [184](#)
  - \_\_chebyshev\_recur, [185](#)
  - \_\_chebyshev\_t, [185](#)
  - \_\_chebyshev\_u, [186](#)
  - \_\_chebyshev\_v, [186](#)
  - \_\_chebyshev\_w, [187](#)
  - \_\_chi\_squared\_pdf, [187](#)
  - \_\_chi\_squared\_pdfc, [188](#)
  - \_\_chshint, [188](#)
  - \_\_chshint\_cont\_frac, [188](#)
  - \_\_chshint\_series, [188](#)
  - \_\_clamp\_0\_m2pi, [189](#)
  - \_\_clamp\_pi, [189](#)
  - \_\_clausen, [189](#)
  - \_\_clausen\_c, [190](#)
  - \_\_clausen\_s, [191](#)

- [\\_\\_comp\\_ellint\\_1](#), 192
- [\\_\\_comp\\_ellint\\_2](#), 192
- [\\_\\_comp\\_ellint\\_3](#), 193
- [\\_\\_comp\\_ellint\\_d](#), 193
- [\\_\\_comp\\_ellint\\_rf](#), 193
- [\\_\\_comp\\_ellint\\_rg](#), 194
- [\\_\\_conf\\_hyperg](#), 194
- [\\_\\_conf\\_hyperg\\_lim](#), 194
- [\\_\\_conf\\_hyperg\\_lim\\_series](#), 195
- [\\_\\_conf\\_hyperg\\_luke](#), 195
- [\\_\\_conf\\_hyperg\\_series](#), 195
- [\\_\\_cos\\_pi](#), 196
- [\\_\\_cosh\\_pi](#), 196
- [\\_\\_coshint](#), 197
- [\\_\\_cyl\\_bessel](#), 197
- [\\_\\_cyl\\_bessel\\_i](#), 198
- [\\_\\_cyl\\_bessel\\_ij\\_series](#), 198
- [\\_\\_cyl\\_bessel\\_ik](#), 199
- [\\_\\_cyl\\_bessel\\_ik\\_asymp](#), 199
- [\\_\\_cyl\\_bessel\\_ik\\_steel](#), 200
- [\\_\\_cyl\\_bessel\\_j](#), 200
- [\\_\\_cyl\\_bessel\\_jn](#), 200
- [\\_\\_cyl\\_bessel\\_jn\\_asymp](#), 201
- [\\_\\_cyl\\_bessel\\_jn\\_neg\\_arg](#), 201
- [\\_\\_cyl\\_bessel\\_jn\\_steel](#), 201
- [\\_\\_cyl\\_bessel\\_k](#), 202
- [\\_\\_cyl\\_hankel\\_1](#), 202, 203
- [\\_\\_cyl\\_hankel\\_2](#), 203, 204
- [\\_\\_cyl\\_neumann](#), 204
- [\\_\\_cyl\\_neumann\\_n](#), 204
- [\\_\\_dawson](#), 205
- [\\_\\_dawson\\_cont\\_frac](#), 205
- [\\_\\_dawson\\_series](#), 206
- [\\_\\_debye\\_region](#), 206
- [\\_\\_dilog](#), 206
- [\\_\\_dirichlet\\_beta](#), 206, 207
- [\\_\\_dirichlet\\_eta](#), 207, 208
- [\\_\\_dirichlet\\_lambda](#), 208
- [\\_\\_double\\_factorial](#), 208
- [\\_\\_ellint\\_1](#), 209
- [\\_\\_ellint\\_2](#), 209
- [\\_\\_ellint\\_3](#), 210
- [\\_\\_ellint\\_cel](#), 210
- [\\_\\_ellint\\_d](#), 210
- [\\_\\_ellint\\_el1](#), 211
- [\\_\\_ellint\\_el2](#), 211
- [\\_\\_ellint\\_el3](#), 211
- [\\_\\_ellint\\_rc](#), 211
- [\\_\\_ellint\\_rd](#), 212
- [\\_\\_ellint\\_rf](#), 213
- [\\_\\_ellint\\_rg](#), 213
- [\\_\\_ellint\\_rj](#), 214
- [\\_\\_ellnome](#), 215
- [\\_\\_ellnome\\_k](#), 215
- [\\_\\_ellnome\\_series](#), 215
- [\\_\\_expint](#), 215, 216
- [\\_\\_expint\\_E1](#), 217
- [\\_\\_expint\\_E1\\_asymp](#), 217
- [\\_\\_expint\\_E1\\_series](#), 218
- [\\_\\_expint\\_Ei](#), 218
- [\\_\\_expint\\_Ei\\_asymp](#), 219
- [\\_\\_expint\\_Ei\\_series](#), 219
- [\\_\\_expint\\_En\\_cont\\_frac](#), 220
- [\\_\\_expint\\_En\\_recursion](#), 220
- [\\_\\_expint\\_En\\_series](#), 221
- [\\_\\_expint\\_asymp](#), 216
- [\\_\\_expint\\_large\\_n](#), 221
- [\\_\\_exponential\\_cdf](#), 222
- [\\_\\_exponential\\_pdf](#), 222
- [\\_\\_factorial](#), 222
- [\\_\\_fermi\\_dirac](#), 222
- [\\_\\_fisher\\_f\\_cdf](#), 223
- [\\_\\_fisher\\_f\\_cdfc](#), 223
- [\\_\\_fock\\_airy](#), 224
- [\\_\\_fresnel](#), 224
- [\\_\\_fresnel\\_cont\\_frac](#), 225
- [\\_\\_fresnel\\_series](#), 225
- [\\_\\_gamma](#), 225
- [\\_\\_gamma\\_cdf](#), 226
- [\\_\\_gamma\\_cdfc](#), 226
- [\\_\\_gamma\\_cont\\_frac](#), 226
- [\\_\\_gamma\\_pdf](#), 226
- [\\_\\_gamma\\_series](#), 227
- [\\_\\_gamma\\_temme](#), 227
- [\\_\\_gauss](#), 228
- [\\_\\_gegenbauer\\_poly](#), 228
- [\\_\\_hankel](#), 228
- [\\_\\_hankel\\_debye](#), 229
- [\\_\\_hankel\\_params](#), 229
- [\\_\\_hankel\\_uniform](#), 229
- [\\_\\_hankel\\_uniform\\_olver](#), 230
- [\\_\\_hankel\\_uniform\\_outer](#), 230
- [\\_\\_hankel\\_uniform\\_sum](#), 231
- [\\_\\_harmonic\\_number](#), 232
- [\\_\\_heuman\\_lambda](#), 232
- [\\_\\_hurwitz\\_zeta](#), 232
- [\\_\\_hurwitz\\_zeta\\_euler\\_maclaurin](#), 232
- [\\_\\_hurwitz\\_zeta\\_polylog](#), 233
- [\\_\\_hydrogen](#), 233
- [\\_\\_hyperg](#), 233
- [\\_\\_hyperg\\_luke](#), 234
- [\\_\\_hyperg\\_reflect](#), 234
- [\\_\\_hyperg\\_series](#), 235
- [\\_\\_ibeta\\_cont\\_frac](#), 235
- [\\_\\_jacobi\\_sncndn](#), 237
- [\\_\\_jacobi\\_zeta](#), 237
- [\\_\\_laguerre](#), 237
- [\\_\\_legendre\\_q](#), 238

- \_\_log\_bincoef, 238, 239
- \_\_log\_bincoef\_sign, 239
- \_\_log\_double\_factorial, 240
- \_\_log\_factorial, 240
- \_\_log\_gamma, 240, 241
- \_\_log\_gamma1p\_lanczos, 241
- \_\_log\_gamma1p\_spouge, 242
- \_\_log\_gamma\_bernoulli, 242
- \_\_log\_gamma\_sign, 243
- \_\_log\_pochhammer, 243
- \_\_log\_pochhammer\_lower, 244
- \_\_logint, 244
- \_\_lognormal\_cdf, 245
- \_\_lognormal\_pdf, 245
- \_\_max\_FGH, 287
- \_\_max\_FGH< double >, 287
- \_\_max\_FGH< float >, 288
- \_\_normal\_cdf, 245
- \_\_normal\_pdf, 245
- \_\_owens\_t, 245
- \_\_pgamma, 246
- \_\_pochhammer, 246
- \_\_pochhammer\_lower, 246
- \_\_polar\_pi, 247
- \_\_poly\_hermite, 247
- \_\_poly\_hermite\_asymp, 247
- \_\_poly\_hermite\_recursion, 248
- \_\_poly\_jacobi, 249
- \_\_poly\_laguerre, 249
- \_\_poly\_laguerre\_hyperg, 250
- \_\_poly\_laguerre\_large\_n, 250
- \_\_poly\_laguerre\_recursion, 251
- \_\_poly\_legendre\_p, 252
- \_\_poly\_radial\_jacobi, 252
- \_\_polylog, 253
- \_\_polylog\_exp, 254
- \_\_polylog\_exp\_asymp, 254
- \_\_polylog\_exp\_int\_neg, 255, 256
- \_\_polylog\_exp\_int\_pos, 256
- \_\_polylog\_exp\_neg, 257
- \_\_polylog\_exp\_neg\_even, 259
- \_\_polylog\_exp\_neg\_odd, 260
- \_\_polylog\_exp\_negative\_real\_part, 260
- \_\_polylog\_exp\_pos, 261, 262
- \_\_polylog\_exp\_real\_neg, 263
- \_\_polylog\_exp\_real\_pos, 264
- \_\_psi, 265
- \_\_psi\_asymp, 265
- \_\_psi\_series, 266
- \_\_qgamma, 266
- \_\_rice\_pdf, 266
- \_\_riemann\_zeta, 267
- \_\_riemann\_zeta\_alt, 267
- \_\_riemann\_zeta\_euler\_maclaurin, 268
- \_\_riemann\_zeta\_glob, 268
- \_\_riemann\_zeta\_m\_1, 268
- \_\_riemann\_zeta\_m\_1\_sum, 269
- \_\_riemann\_zeta\_product, 269
- \_\_riemann\_zeta\_sum, 269
- \_\_sin\_pi, 270
- \_\_sinc, 270
- \_\_sinc\_pi, 270
- \_\_sincos, 271
- \_\_sincos\_pi, 271
- \_\_sincosint, 271
- \_\_sincosint\_asymp, 272
- \_\_sincosint\_cont\_frac, 272
- \_\_sincosint\_series, 272
- \_\_sinh\_pi, 272, 273
- \_\_sinhc, 273
- \_\_sinhc\_pi, 273
- \_\_sinhint, 273
- \_\_sph\_bessel, 274
- \_\_sph\_bessel\_ik, 275
- \_\_sph\_bessel\_jn, 275
- \_\_sph\_bessel\_jn\_neg\_arg, 276
- \_\_sph\_hankel, 276
- \_\_sph\_hankel\_1, 276, 277
- \_\_sph\_hankel\_2, 277, 278
- \_\_sph\_harmonic, 278
- \_\_sph\_legendre, 279
- \_\_sph\_neumann, 279, 280
- \_\_student\_t\_cdf, 280
- \_\_student\_t\_cdfc, 281
- \_\_tan\_pi, 281
- \_\_tanh\_pi, 281, 282
- \_\_tgamma, 282
- \_\_tgamma\_lower, 282
- \_\_theta\_1, 282
- \_\_theta\_2, 283
- \_\_theta\_2\_asymp, 283
- \_\_theta\_2\_sum, 283
- \_\_theta\_3, 283
- \_\_theta\_3\_asymp, 284
- \_\_theta\_3\_sum, 284
- \_\_theta\_4, 284
- \_\_theta\_c, 285
- \_\_theta\_d, 285
- \_\_theta\_n, 285
- \_\_theta\_s, 285
- \_\_weibull\_cdf, 285
- \_\_weibull\_pdf, 286
- \_\_zernike, 286
- \_\_znorm1, 287
- \_\_znorm2, 287
- COSINT, 174
- evenzeta, 287
- SININT, 174

std::\_\_detail::\_Airy  
   \_Airy, 293  
   \_S\_2pi\_3, 293  
   \_S\_5pi\_6, 293  
   \_S\_NaN, 294  
   \_S\_cNaN, 293  
   \_S\_i, 294  
   \_S\_pi, 294  
   \_S\_pi\_3, 294  
   \_S\_pi\_6, 294  
   \_S\_sqrt\_pi, 294  
   inner\_radius, 294  
   operator(), 293  
   outer\_radius, 294  
   scalar\_type, 293  
   value\_type, 293  
 std::\_\_detail::\_Airy< \_Tp >, 292  
 std::\_\_detail::\_Airy\_asymp  
   \_Airy\_asymp, 296  
   \_S\_absarg\_ge\_pio3, 296  
   \_S\_absarg\_lt\_pio3, 297  
   \_cmplx, 296  
   operator(), 297  
 std::\_\_detail::\_Airy\_asymp< \_Tp >, 295  
 std::\_\_detail::\_Airy\_asymp\_data< \_Tp >, 298  
 std::\_\_detail::\_Airy\_asymp\_data< double >, 299  
   \_S\_c, 299  
   \_S\_d, 299  
   \_S\_max\_cd, 299  
 std::\_\_detail::\_Airy\_asymp\_data< float >, 300  
   \_S\_c, 300  
   \_S\_d, 300  
   \_S\_max\_cd, 300  
 std::\_\_detail::\_Airy\_asymp\_data< long double >, 300  
   \_S\_c, 301  
   \_S\_d, 301  
   \_S\_max\_cd, 301  
 std::\_\_detail::\_Airy\_asymp\_series  
   \_Airy\_asymp\_series, 302  
   \_S\_sqrt\_pi, 303  
   operator(), 302  
   scalar\_type, 302  
   value\_type, 302  
 std::\_\_detail::\_Airy\_asymp\_series< \_Sum >, 301  
 std::\_\_detail::\_Airy\_default\_radII< \_Tp >, 303  
 std::\_\_detail::\_Airy\_default\_radII< double >, 303  
   inner\_radius, 304  
   outer\_radius, 304  
 std::\_\_detail::\_Airy\_default\_radII< float >, 304  
   inner\_radius, 304  
   outer\_radius, 304  
 std::\_\_detail::\_Airy\_default\_radII< long double >, 305  
   inner\_radius, 305  
   outer\_radius, 305  
 std::\_\_detail::\_Airy\_series  
   \_N\_FGH, 310  
   \_S\_Ai, 306  
   \_S\_Ai0, 310  
   \_S\_Aip0, 310  
   \_S\_Airy, 307  
   \_S\_Bi, 307  
   \_S\_Bi0, 310  
   \_S\_Bip0, 310  
   \_S\_FGH, 308  
   \_S\_Fock, 308  
   \_S\_Gi0, 310  
   \_S\_Gip0, 310  
   \_S\_Hi0, 310  
   \_S\_Hip0, 311  
   \_S\_Scorer, 308  
   \_S\_Scorer2, 309  
   \_S\_eps, 310  
   \_S\_i, 311  
   \_S\_pi, 311  
   \_S\_sqrt\_pi, 311  
 std::\_\_detail::\_Airy\_series< \_Tp >, 305  
 std::\_\_detail::\_AiryAuxilliaryState  
   \_Val, 312  
   fai, 312  
   faip, 312  
   gai, 312  
   gaip, 312  
   hai, 312  
   haip, 312  
   z, 313  
 std::\_\_detail::\_AiryAuxilliaryState< \_Tp >, 311  
 std::\_\_detail::\_AiryState  
   \_Val, 314  
   Ai, 314  
   Aip, 314  
   Bi, 314  
   Bip, 314  
   true\_Wronskian, 314  
   Wronskian, 314  
   z, 314  
 std::\_\_detail::\_AiryState< \_Tp >, 313  
 std::\_\_detail::\_Factorial\_table  
   \_factorial, 315  
   \_log\_factorial, 315  
   \_n, 315  
 std::\_\_detail::\_Factorial\_table< \_Tp >, 315  
 std::\_\_detail::\_GammaLanczos< \_Tp >, 316  
 std::\_\_detail::\_GammaLanczos< double >, 316  
   \_S\_cheby, 317  
   \_S\_g, 317  
 std::\_\_detail::\_GammaLanczos< float >, 317  
   \_S\_cheby, 318  
   \_S\_g, 318

- std::\_\_detail::\_\_GammaLanczos< long double >, 318
  - \_S\_cheby, 319
  - \_S\_g, 319
- std::\_\_detail::\_\_GammaSpouge< \_Tp >, 319
- std::\_\_detail::\_\_GammaSpouge< double >, 320
  - \_S\_cheby, 320
- std::\_\_detail::\_\_GammaSpouge< float >, 320
  - \_S\_cheby, 321
- std::\_\_detail::\_\_GammaSpouge< long double >, 321
  - \_S\_cheby, 322
- tan\_pi
  - GNU Extended Mathematical Special Functions, 130
- tan\_pif
  - GNU Extended Mathematical Special Functions, 131
- tan\_pil
  - GNU Extended Mathematical Special Functions, 131
- tanh\_pi
  - GNU Extended Mathematical Special Functions, 131
- tanh\_pif
  - GNU Extended Mathematical Special Functions, 132
- tanh\_pil
  - GNU Extended Mathematical Special Functions, 132
- tgamma
  - GNU Extended Mathematical Special Functions, 132
- tgamma\_lower
  - GNU Extended Mathematical Special Functions, 132
- tgamma\_lowerf
  - GNU Extended Mathematical Special Functions, 133
- tgamma\_lowerl
  - GNU Extended Mathematical Special Functions, 133
- tgammaf
  - GNU Extended Mathematical Special Functions, 133
- tgammaf
  - GNU Extended Mathematical Special Functions, 134
- theta\_1
  - GNU Extended Mathematical Special Functions, 134
- theta\_1f
  - GNU Extended Mathematical Special Functions, 135
- theta\_1l
  - GNU Extended Mathematical Special Functions, 135
- theta\_2
  - GNU Extended Mathematical Special Functions, 135
- theta\_2f
  - GNU Extended Mathematical Special Functions, 135
- theta\_2l
  - GNU Extended Mathematical Special Functions, 136
- theta\_3
  - GNU Extended Mathematical Special Functions, 136
- theta\_3f
  - GNU Extended Mathematical Special Functions, 136
- theta\_3l
  - GNU Extended Mathematical Special Functions, 136
- theta\_4
  - GNU Extended Mathematical Special Functions, 137
- theta\_4f
  - GNU Extended Mathematical Special Functions, 137
- theta\_4l
  - GNU Extended Mathematical Special Functions, 137
- theta\_c
  - GNU Extended Mathematical Special Functions, 137
- theta\_cf
  - GNU Extended Mathematical Special Functions, 138
- theta\_cl
  - GNU Extended Mathematical Special Functions, 138
- theta\_d
  - GNU Extended Mathematical Special Functions, 138
- theta\_df
  - GNU Extended Mathematical Special Functions, 139
- theta\_dl
  - GNU Extended Mathematical Special Functions, 139
- theta\_n
  - GNU Extended Mathematical Special Functions, 139
- theta\_nf
  - GNU Extended Mathematical Special Functions, 140
- theta\_nl
  - GNU Extended Mathematical Special Functions, 140
- theta\_s
  - GNU Extended Mathematical Special Functions, 140
- theta\_sf
  - GNU Extended Mathematical Special Functions, 140
- theta\_sl
  - GNU Extended Mathematical Special Functions, 141
- true\_Wronskian
  - std::\_\_detail::\_\_AiryState, 314
- value\_type
  - std::\_\_detail::\_\_Airy, 293
  - std::\_\_detail::\_\_Airy\_asymp\_series, 302
- Wronskian
  - std::\_\_detail::\_\_AiryState, 314
- z
  - std::\_\_detail::\_\_AiryAuxilliaryState, 313
  - std::\_\_detail::\_\_AiryState, 314
- zernike
  - GNU Extended Mathematical Special Functions, 141
- zernikef
  - GNU Extended Mathematical Special Functions, 142
- zernikel
  - GNU Extended Mathematical Special Functions, 142