

TR29124 C++ Special Math Functions

2.0

Generated by Doxygen 1.8.11

Contents

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

7 File Index	17
7.1 File List	17
8 Module Documentation	19
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	cyl_bessel_if(float __nu, float __x)	29
8.2.2.21	cyl_bessel_il(long double __nu, long double __x)	29
8.2.2.22	cyl_bessel_j(_Tpnu __nu, _Tp __x)	29
8.2.2.23	cyl_bessel_jf(float __nu, float __x)	30
8.2.2.24	cyl_bessel_jl(long double __nu, long double __x)	30
8.2.2.25	cyl_bessel_k(_Tpnu __nu, _Tp __x)	30
8.2.2.26	cyl_bessel_kf(float __nu, float __x)	31
8.2.2.27	cyl_bessel_kl(long double __nu, long double __x)	31
8.2.2.28	cyl_neumann(_Tpnu __nu, _Tp __x)	31
8.2.2.29	cyl_neumannf(float __nu, float __x)	32
8.2.2.30	cyl_neumannl(long double __nu, long double __x)	32
8.2.2.31	ellint_1(_Tp __k, _Tpp __phi)	32
8.2.2.32	ellint_1f(float __k, float __phi)	33
8.2.2.33	ellint_1l(long double __k, long double __phi)	33
8.2.2.34	ellint_2(_Tp __k, _Tpp __phi)	33
8.2.2.35	ellint_2f(float __k, float __phi)	34
8.2.2.36	ellint_2l(long double __k, long double __phi)	34
8.2.2.37	ellint_3(_Tp __k, _Tpn __nu, _Tpp __phi)	35
8.2.2.38	ellint_3f(float __k, float __nu, float __phi)	35
8.2.2.39	ellint_3l(long double __k, long double __nu, long double __phi)	36
8.2.2.40	expint(_Tp __x)	36
8.2.2.41	expintf(float __x)	36
8.2.2.42	expintl(long double __x)	36
8.2.2.43	hermite(unsigned int __n, _Tp __x)	37
8.2.2.44	hermitef(unsigned int __n, float __x)	37
8.2.2.45	hermitel(unsigned int __n, long double __x)	37
8.2.2.46	laguerre(unsigned int __n, _Tp __x)	38
8.2.2.47	laguerref(unsigned int __n, float __x)	38

8.2.2.48	laguerrel(unsigned int __n, long double __x)	38
8.2.2.49	legendre(unsigned int __l, _Tp __x)	39
8.2.2.50	legendref(unsigned int __l, float __x)	39
8.2.2.51	legendrel(unsigned int __l, long double __x)	39
8.2.2.52	riemann_zeta(_Tp __s)	40
8.2.2.53	riemann_zetaf(float __s)	40
8.2.2.54	riemann_zetal(long double __s)	40
8.2.2.55	sph_bessel(unsigned int __n, _Tp __x)	41
8.2.2.56	sph_besself(unsigned int __n, float __x)	41
8.2.2.57	sph_bessell(unsigned int __n, long double __x)	41
8.2.2.58	sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)	42
8.2.2.59	sph_legendref(unsigned int __l, unsigned int __m, float __theta)	42
8.2.2.60	sph_legendrel(unsigned int __l, unsigned int __m, long double __theta)	42
8.2.2.61	sph_neumann(unsigned int __n, _Tp __x)	43
8.2.2.62	sph_neumannf(unsigned int __n, float __x)	43
8.2.2.63	sph_neumannl(unsigned int __n, long double __x)	43
8.3	GNU Extended Mathematical Special Functions	44
8.3.1	Detailed Description	52
8.3.2	Enumeration Type Documentation	52
8.3.2.1	anonymous enum	52
8.3.3	Function Documentation	52
8.3.3.1	airy_ai(_Tp __x)	52
8.3.3.2	airy_aif(float __x)	53
8.3.3.3	airy_ail(long double __x)	53
8.3.3.4	airy_bi(_Tp __x)	53
8.3.3.5	airy_bif(float __x)	54
8.3.3.6	airy_bil(long double __x)	54
8.3.3.7	bernoulli(unsigned int __n)	54

8.3.3.8	bernoullif(unsigned int __n)	55
8.3.3.9	bernoullil(unsigned int __n)	55
8.3.3.10	bincoef(unsigned int __n, unsigned int __k)	55
8.3.3.11	bincoeff(unsigned int __n, unsigned int __k)	55
8.3.3.12	bincoefl(unsigned int __n, unsigned int __k)	55
8.3.3.13	chebyshev_t(unsigned int __n, _Tp __x)	55
8.3.3.14	chebyshev_tf(unsigned int __n, float __x)	56
8.3.3.15	chebyshev_tl(unsigned int __n, long double __x)	56
8.3.3.16	chebyshev_u(unsigned int __n, _Tp __x)	56
8.3.3.17	chebyshev_uf(unsigned int __n, float __x)	57
8.3.3.18	chebyshev_ul(unsigned int __n, long double __x)	57
8.3.3.19	chebyshev_v(unsigned int __n, _Tp __x)	57
8.3.3.20	chebyshev_vf(unsigned int __n, float __x)	58
8.3.3.21	chebyshev_vl(unsigned int __n, long double __x)	58
8.3.3.22	chebyshev_w(unsigned int __n, _Tp __x)	58
8.3.3.23	chebyshev_wf(unsigned int __n, float __x)	59
8.3.3.24	chebyshev_wl(unsigned int __n, long double __x)	59
8.3.3.25	clausen(unsigned int __m, _Tp __w)	59
8.3.3.26	clausen(unsigned int __m, std::complex< _Tp > __w)	60
8.3.3.27	clausen_c(unsigned int __m, _Tp __w)	60
8.3.3.28	clausen_cf(unsigned int __m, float __w)	61
8.3.3.29	clausen_cl(unsigned int __m, long double __w)	61
8.3.3.30	clausen_s(unsigned int __m, _Tp __w)	61
8.3.3.31	clausen_sf(unsigned int __m, float __w)	62
8.3.3.32	clausen_sl(unsigned int __m, long double __w)	62
8.3.3.33	clausenf(unsigned int __m, float __w)	62
8.3.3.34	clausenf(unsigned int __m, std::complex< float > __w)	63
8.3.3.35	clausenl(unsigned int __m, long double __w)	63

8.3.3.36	<code>clausenl(unsigned int __m, std::complex< long double > __w)</code>	63
8.3.3.37	<code>comp_ellint_d(_Tk __k)</code>	63
8.3.3.38	<code>comp_ellint_df(float __k)</code>	64
8.3.3.39	<code>comp_ellint_dl(long double __k)</code>	64
8.3.3.40	<code>comp_ellint_rf(float __x, float __y)</code>	64
8.3.3.41	<code>comp_ellint_rf(long double __x, long double __y)</code>	64
8.3.3.42	<code>comp_ellint_rf(_Tx __x, _Ty __y)</code>	64
8.3.3.43	<code>comp_ellint_rg(float __x, float __y)</code>	65
8.3.3.44	<code>comp_ellint_rg(long double __x, long double __y)</code>	65
8.3.3.45	<code>comp_ellint_rg(_Tx __x, _Ty __y)</code>	65
8.3.3.46	<code>conf_hyperg(_Tpa __a, _Tpc __c, _Tp __x)</code>	66
8.3.3.47	<code>conf_hyperg_lim(_Tpc __c, _Tp __x)</code>	66
8.3.3.48	<code>conf_hyperg_limf(float __c, float __x)</code>	67
8.3.3.49	<code>conf_hyperg_liml(long double __c, long double __x)</code>	67
8.3.3.50	<code>conf_hypergf(float __a, float __c, float __x)</code>	67
8.3.3.51	<code>conf_hypergl(long double __a, long double __c, long double __x)</code>	67
8.3.3.52	<code>coshint(_Tp __x)</code>	67
8.3.3.53	<code>coshintf(float __x)</code>	68
8.3.3.54	<code>coshintl(long double __x)</code>	68
8.3.3.55	<code>cosint(_Tp __x)</code>	68
8.3.3.56	<code>cosintf(float __x)</code>	69
8.3.3.57	<code>cosintl(long double __x)</code>	69
8.3.3.58	<code>cyl_hankel_1(_Tpnu __nu, _Tp __z)</code>	69
8.3.3.59	<code>cyl_hankel_1(std::complex< _Tpnu > __nu, std::complex< _Tp > __x)</code>	70
8.3.3.60	<code>cyl_hankel_1f(float __nu, float __z)</code>	70
8.3.3.61	<code>cyl_hankel_1f(std::complex< float > __nu, std::complex< float > __x)</code>	71
8.3.3.62	<code>cyl_hankel_1l(long double __nu, long double __z)</code>	71
8.3.3.63	<code>cyl_hankel_1l(std::complex< long double > __nu, std::complex< long double > __x)</code>	71

8.3.3.64	<code>cyl_hankel_2(_Tpnu __nu, _Tp __z)</code>	71
8.3.3.65	<code>cyl_hankel_2(std::complex< _Tpnu > __nu, std::complex< _Tp > __x)</code>	72
8.3.3.66	<code>cyl_hankel_2f(float __nu, float __z)</code>	72
8.3.3.67	<code>cyl_hankel_2f(std::complex< float > __nu, std::complex< float > __x)</code>	73
8.3.3.68	<code>cyl_hankel_2l(long double __nu, long double __z)</code>	73
8.3.3.69	<code>cyl_hankel_2l(std::complex< long double > __nu, std::complex< long double > __x)</code>	73
8.3.3.70	<code>dawson(_Tp __x)</code>	73
8.3.3.71	<code>dawsonf(float __x)</code>	74
8.3.3.72	<code>dawsonl(long double __x)</code>	74
8.3.3.73	<code>digamma(_Tp __z)</code>	74
8.3.3.74	<code>digammaf(float __z)</code>	74
8.3.3.75	<code>digammal(long double __z)</code>	74
8.3.3.76	<code>dilog(_Tp __x)</code>	74
8.3.3.77	<code>dilogf(float __x)</code>	75
8.3.3.78	<code>dilogl(long double __x)</code>	75
8.3.3.79	<code>dirichlet_beta(_Tp __s)</code>	75
8.3.3.80	<code>dirichlet_betaf(float __s)</code>	76
8.3.3.81	<code>dirichlet_betall(long double __s)</code>	76
8.3.3.82	<code>dirichlet_eta(_Tp __s)</code>	76
8.3.3.83	<code>dirichlet_etaf(float __s)</code>	77
8.3.3.84	<code>dirichlet_etall(long double __s)</code>	77
8.3.3.85	<code>double_factorial(int __n)</code>	77
8.3.3.86	<code>double_factorialf(int __n)</code>	77
8.3.3.87	<code>double_factoriall(int __n)</code>	77
8.3.3.88	<code>ellint_cel(_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)</code>	77
8.3.3.89	<code>ellint_celf(float __k_c, float __p, float __a, float __b)</code>	78
8.3.3.90	<code>ellint_cell(long double __k_c, long double __p, long double __a, long double __b)</code>	78
8.3.3.91	<code>ellint_d(_Tk __k, _Tphi __phi)</code>	78

8.3.3.92	<code>ellint_df(float __k, float __phi)</code>	79
8.3.3.93	<code>ellint_dl(long double __k, long double __phi)</code>	79
8.3.3.94	<code>ellint_el1(_Tp __x, _Tk __k_c)</code>	79
8.3.3.95	<code>ellint_el1f(float __x, float __k_c)</code>	80
8.3.3.96	<code>ellint_el1l(long double __x, long double __k_c)</code>	80
8.3.3.97	<code>ellint_el2(_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)</code>	80
8.3.3.98	<code>ellint_el2f(float __x, float __k_c, float __a, float __b)</code>	81
8.3.3.99	<code>ellint_el2l(long double __x, long double __k_c, long double __a, long double __b)</code>	81
8.3.3.100	<code>ellint_el3(_Tx __x, _Tk __k_c, _Tp __p)</code>	81
8.3.3.101	<code>ellint_el3f(float __x, float __k_c, float __p)</code>	82
8.3.3.102	<code>ellint_el3l(long double __x, long double __k_c, long double __p)</code>	82
8.3.3.103	<code>ellint_rc(_Tp __x, _Up __y)</code>	82
8.3.3.104	<code>ellint_rcf(float __x, float __y)</code>	83
8.3.3.105	<code>ellint_rcl(long double __x, long double __y)</code>	83
8.3.3.106	<code>ellint_rd(_Tp __x, _Up __y, _Vp __z)</code>	83
8.3.3.107	<code>ellint_rdf(float __x, float __y, float __z)</code>	84
8.3.3.108	<code>ellint_rdl(long double __x, long double __y, long double __z)</code>	84
8.3.3.109	<code>ellint_rf(_Tp __x, _Up __y, _Vp __z)</code>	84
8.3.3.110	<code>ellint_rff(float __x, float __y, float __z)</code>	85
8.3.3.111	<code>ellint_rfl(long double __x, long double __y, long double __z)</code>	85
8.3.3.112	<code>ellint_rg(_Tp __x, _Up __y, _Vp __z)</code>	85
8.3.3.113	<code>ellint_rgf(float __x, float __y, float __z)</code>	86
8.3.3.114	<code>ellint_rgl(long double __x, long double __y, long double __z)</code>	86
8.3.3.115	<code>ellint_rj(_Tp __x, _Up __y, _Vp __z, _Wp __p)</code>	86
8.3.3.116	<code>ellint_rjf(float __x, float __y, float __z, float __p)</code>	87
8.3.3.117	<code>ellint_rjl(long double __x, long double __y, long double __z, long double __p)</code>	87
8.3.3.118	<code>ellnome(_Tp __k)</code>	87
8.3.3.119	<code>ellnomef(float __k)</code>	88

8.3.3.120	<code>ellnomel(long double __k)</code>	88
8.3.3.121	<code>expint(unsigned int __n, _Tp __x)</code>	88
8.3.3.122	<code>expintf(unsigned int __n, float __x)</code>	89
8.3.3.123	<code>expintl(unsigned int __n, long double __x)</code>	89
8.3.3.124	<code>factorial(unsigned int __n)</code>	89
8.3.3.125	<code>factorialf(unsigned int __n)</code>	89
8.3.3.126	<code>factoriall(unsigned int __n)</code>	89
8.3.3.127	<code>fresnel_c(_Tp __x)</code>	89
8.3.3.128	<code>fresnel_cf(float __x)</code>	90
8.3.3.129	<code>fresnel_cl(long double __x)</code>	90
8.3.3.130	<code>fresnel_s(_Tp __x)</code>	90
8.3.3.131	<code>fresnel_sf(float __x)</code>	90
8.3.3.132	<code>fresnel_sl(long double __x)</code>	90
8.3.3.133	<code>gamma_l(_Tn __n, _Tp __x)</code>	90
8.3.3.134	<code>gamma_lf(float __n, float __x)</code>	91
8.3.3.135	<code>gamma_ll(long double __n, long double __x)</code>	91
8.3.3.136	<code>gamma_u(_Tn __n, _Tp __x)</code>	91
8.3.3.137	<code>gamma_uf(float __n, float __x)</code>	91
8.3.3.138	<code>gamma_ul(long double __n, long double __x)</code>	91
8.3.3.139	<code>gegenbauer(unsigned int __n, _Talpha __alpha, _Tp __x)</code>	91
8.3.3.140	<code>gegenbauerf(unsigned int __n, float __alpha, float __x)</code>	92
8.3.3.141	<code>gegenbauerl(unsigned int __n, long double __alpha, long double __x)</code>	92
8.3.3.142	<code>heuman_lambda(_Tk __k, _Tphi __phi)</code>	92
8.3.3.143	<code>heuman_lambdaf(float __k, float __phi)</code>	93
8.3.3.144	<code>heuman_lambdal(long double __k, long double __phi)</code>	93
8.3.3.145	<code>hurwitz_zeta(_Tp __s, _Up __a)</code>	93
8.3.3.146	<code>hurwitz_zeta(_Tp __s, std::complex< _Up > __a)</code>	93
8.3.3.147	<code>hurwitz_zetaf(float __s, float __a)</code>	93

8.3.3.148 hurwitz_zetal(long double __s, long double __a)	94
8.3.3.149 hyperg(_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)	94
8.3.3.150 hypergf(float __a, float __b, float __c, float __x)	94
8.3.3.151 hypergl(long double __a, long double __b, long double __c, long double __x)	95
8.3.3.152 ibeta(_Ta __a, _Tb __b, _Tp __x)	95
8.3.3.153 ibetac(_Ta __a, _Tb __b, _Tp __x)	95
8.3.3.154 ibetacf(float __a, float __b, float __x)	96
8.3.3.155 ibetacl(long double __a, long double __b, long double __x)	96
8.3.3.156 ibetaf(float __a, float __b, float __x)	96
8.3.3.157 ibetal(long double __a, long double __b, long double __x)	96
8.3.3.158 jacobi(unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)	96
8.3.3.159 jacobi_cn(_Kp __k, _Up __u)	97
8.3.3.160 jacobi_cnf(float __k, float __u)	97
8.3.3.161 jacobi_cnl(long double __k, long double __u)	98
8.3.3.162 jacobi_dn(_Kp __k, _Up __u)	98
8.3.3.163 jacobi_dnf(float __k, float __u)	98
8.3.3.164 jacobi_dnl(long double __k, long double __u)	99
8.3.3.165 jacobi_sn(_Kp __k, _Up __u)	99
8.3.3.166 jacobi_snf(float __k, float __u)	99
8.3.3.167 jacobi_snl(long double __k, long double __u)	100
8.3.3.168 jacobi_zeta(_Tk __k, _Tphi __phi)	100
8.3.3.169 jacobi_zetaf(float __k, float __phi)	100
8.3.3.170 jacobi_zetal(long double __k, long double __phi)	101
8.3.3.171 jacobif(unsigned __n, float __alpha, float __beta, float __x)	101
8.3.3.172 jacobil(unsigned __n, long double __alpha, long double __beta, long double __x)	101
8.3.3.173 lbincoef(unsigned int __n, unsigned int __k)	101
8.3.3.174 lbincoeff(unsigned int __n, unsigned int __k)	101
8.3.3.175 lbincoefl(unsigned int __n, unsigned int __k)	101

8.3.3.176 ldouble_factorial(int __n)	102
8.3.3.177 ldouble_factorialf(int __n)	102
8.3.3.178 ldouble_factoriall(int __n)	102
8.3.3.179 legendre_q(unsigned int __n, _Tp __x)	102
8.3.3.180 legendre_qf(unsigned int __n, float __x)	102
8.3.3.181 legendre_ql(unsigned int __n, long double __x)	102
8.3.3.182 lfactorial(unsigned int __n)	102
8.3.3.183 lfactorialf(unsigned int __n)	103
8.3.3.184 lfactoriall(unsigned int __n)	103
8.3.3.185 logint(_Tp __x)	103
8.3.3.186 logintf(float __x)	103
8.3.3.187 logintl(long double __x)	103
8.3.3.188 lpochhammer_l(_Tp __a, _Tn __n)	104
8.3.3.189 lpochhammer_lf(float __a, float __n)	104
8.3.3.190 lpochhammer_ll(long double __a, long double __n)	104
8.3.3.191 lpochhammer_u(_Tp __a, _Tn __n)	104
8.3.3.192 lpochhammer_uf(float __a, float __n)	104
8.3.3.193 lpochhammer_ul(long double __a, long double __n)	104
8.3.3.194 owens_t(_Tph __h, _Tpa __a)	104
8.3.3.195 owens_tf(float __h, float __a)	105
8.3.3.196 owens_tl(long double __h, long double __a)	105
8.3.3.197 pgamma(_Ta __a, _Tp __x)	105
8.3.3.198 pgammaf(float __a, float __x)	105
8.3.3.199 pgammal(long double __a, long double __x)	105
8.3.3.200 pochhammer_l(_Tp __a, _Tn __n)	105
8.3.3.201 pochhammer_lf(float __a, float __n)	106
8.3.3.202 pochhammer_ll(long double __a, long double __n)	106
8.3.3.203 pochhammer_u(_Tp __a, _Tn __n)	106

8.3.3.204 pochhammer_uf(float __a, float __n)	106
8.3.3.205 pochhammer_ul(long double __a, long double __n)	106
8.3.3.206 polylog(_Tp __s, _Wp __w)	106
8.3.3.207 polylog(_Tp __s, std::complex< _Tp > __w)	106
8.3.3.208 polylogf(float __s, float __w)	107
8.3.3.209 polylogf(float __s, std::complex< float > __w)	107
8.3.3.210 polylogl(long double __s, long double __w)	107
8.3.3.211 polylogl(long double __s, std::complex< long double > __w)	108
8.3.3.212 psi(_Tp __x)	108
8.3.3.213 psif(float __x)	108
8.3.3.214 psil(long double __x)	108
8.3.3.215 qgamma(_Ta __a, _Tp __x)	108
8.3.3.216 qgammaf(float __a, float __x)	108
8.3.3.217 qgammal(long double __a, long double __x)	109
8.3.3.218 radpoly(unsigned int __n, unsigned int __m, _Tp __rho)	109
8.3.3.219 radpolyf(unsigned int __n, unsigned int __m, float __rho)	109
8.3.3.220 radpolyl(unsigned int __n, unsigned int __m, long double __rho)	110
8.3.3.221 sinc(_Tp __x)	110
8.3.3.222 sinc_pi(_Tp __x)	110
8.3.3.223 sinc_pif(float __x)	111
8.3.3.224 sinc_pil(long double __x)	111
8.3.3.225 sincf(float __x)	111
8.3.3.226 sincl(long double __x)	111
8.3.3.227 sinhc(_Tp __x)	112
8.3.3.228 sinhc_pi(_Tp __x)	112
8.3.3.229 sinhc_pif(float __x)	112
8.3.3.230 sinhc_pil(long double __x)	112
8.3.3.231 sinhcf(float __x)	112

8.3.3.232 <code>sinhcl(long double __x)</code>	112
8.3.3.233 <code>sinhint(_Tp __x)</code>	112
8.3.3.234 <code>sinhintf(float __x)</code>	113
8.3.3.235 <code>sinhintl(long double __x)</code>	113
8.3.3.236 <code>sinint(_Tp __x)</code>	113
8.3.3.237 <code>sinintf(float __x)</code>	113
8.3.3.238 <code>sinintl(long double __x)</code>	114
8.3.3.239 <code>sph_bessel_i(unsigned int __n, _Tp __x)</code>	114
8.3.3.240 <code>sph_bessel_if(unsigned int __n, float __x)</code>	115
8.3.3.241 <code>sph_bessel_il(unsigned int __n, long double __x)</code>	115
8.3.3.242 <code>sph_bessel_k(unsigned int __n, _Tp __x)</code>	115
8.3.3.243 <code>sph_bessel_kf(unsigned int __n, float __x)</code>	116
8.3.3.244 <code>sph_bessel_kl(unsigned int __n, long double __x)</code>	116
8.3.3.245 <code>sph_hankel_1(unsigned int __n, _Tp __z)</code>	116
8.3.3.246 <code>sph_hankel_1(unsigned int __n, std::complex< _Tp > __x)</code>	117
8.3.3.247 <code>sph_hankel_1f(unsigned int __n, float __z)</code>	117
8.3.3.248 <code>sph_hankel_1f(unsigned int __n, std::complex< float > __x)</code>	117
8.3.3.249 <code>sph_hankel_1l(unsigned int __n, long double __z)</code>	118
8.3.3.250 <code>sph_hankel_1l(unsigned int __n, std::complex< long double > __x)</code>	118
8.3.3.251 <code>sph_hankel_2(unsigned int __n, _Tp __z)</code>	118
8.3.3.252 <code>sph_hankel_2(unsigned int __n, std::complex< _Tp > __x)</code>	119
8.3.3.253 <code>sph_hankel_2f(unsigned int __n, float __z)</code>	119
8.3.3.254 <code>sph_hankel_2f(unsigned int __n, std::complex< float > __x)</code>	119
8.3.3.255 <code>sph_hankel_2l(unsigned int __n, long double __z)</code>	120
8.3.3.256 <code>sph_hankel_2l(unsigned int __n, std::complex< long double > __x)</code>	120
8.3.3.257 <code>sph_harmonic(unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)</code>	120
8.3.3.258 <code>sph_harmonicf(unsigned int __l, int __m, float __theta, float __phi)</code>	120
8.3.3.259 <code>sph_harmonicl(unsigned int __l, int __m, long double __theta, long double __phi)</code>	121

8.3.3.260	<code>theta_1(_Tpnu __nu, _Tp __x)</code>	121
8.3.3.261	<code>theta_1f(float __nu, float __x)</code>	121
8.3.3.262	<code>theta_1l(long double __nu, long double __x)</code>	122
8.3.3.263	<code>theta_2(_Tpnu __nu, _Tp __x)</code>	122
8.3.3.264	<code>theta_2f(float __nu, float __x)</code>	122
8.3.3.265	<code>theta_2l(long double __nu, long double __x)</code>	122
8.3.3.266	<code>theta_3(_Tpnu __nu, _Tp __x)</code>	123
8.3.3.267	<code>theta_3f(float __nu, float __x)</code>	123
8.3.3.268	<code>theta_3l(long double __nu, long double __x)</code>	123
8.3.3.269	<code>theta_4(_Tpnu __nu, _Tp __x)</code>	123
8.3.3.270	<code>theta_4f(float __nu, float __x)</code>	124
8.3.3.271	<code>theta_4l(long double __nu, long double __x)</code>	124
8.3.3.272	<code>theta_c(_Tp k, _Tp __x)</code>	124
8.3.3.273	<code>theta_cf(float __k, float __x)</code>	125
8.3.3.274	<code>theta_cl(long double __k, long double __x)</code>	125
8.3.3.275	<code>theta_d(_Tp k, _Tp __x)</code>	125
8.3.3.276	<code>theta_df(float __k, float __x)</code>	125
8.3.3.277	<code>theta_dl(long double __k, long double __x)</code>	126
8.3.3.278	<code>theta_n(_Tp k, _Tp __x)</code>	126
8.3.3.279	<code>theta_nf(float __k, float __x)</code>	126
8.3.3.280	<code>theta_nl(long double __k, long double __x)</code>	126
8.3.3.281	<code>theta_s(_Tp k, _Tp __x)</code>	127
8.3.3.282	<code>theta_sf(float __k, float __x)</code>	127
8.3.3.283	<code>theta_sl(long double __k, long double __x)</code>	127
8.3.3.284	<code>zernike(unsigned int __n, int __m, _Trho __rho, _Tphi __phi)</code>	127
8.3.3.285	<code>zernikef(unsigned int __n, int __m, float __rho, float __phi)</code>	128
8.3.3.286	<code>zernikel(unsigned int __n, int __m, long double __rho, long double __phi)</code>	128

9 Namespace Documentation	129
9.1 <code>__gnu_cxx</code> Namespace Reference	129
9.2 <code>std</code> Namespace Reference	137
9.3 <code>std::__detail</code> Namespace Reference	139
9.3.1 Enumeration Type Documentation	156
9.3.1.1 anonymous enum	156
9.3.2 Function Documentation	156
9.3.2.1 <code>__airy(_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)</code>	156
9.3.2.2 <code>__airy_ai(std::complex<_Tp> __z)</code>	157
9.3.2.3 <code>__airy_arg(std::complex<_Tp> __num2d3, std::complex<_Tp> __zeta, std::complex<_Tp> &__argp, std::complex<_Tp> &__argm)</code>	157
9.3.2.4 <code>__airy_bi(std::complex<_Tp> __z)</code>	157
9.3.2.5 <code>__assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)</code>	158
9.3.2.6 <code>__assoc_legendre_p(unsigned int __l, unsigned int __m, _Tp __x)</code>	158
9.3.2.7 <code>__bernoulli(int __n)</code>	159
9.3.2.8 <code>__bernoulli_2n(int __n)</code>	159
9.3.2.9 <code>__bernoulli_series(unsigned int __n)</code>	159
9.3.2.10 <code>__beta(_Tp __a, _Tp __b)</code>	160
9.3.2.11 <code>__beta_gamma(_Tp __a, _Tp __b)</code>	160
9.3.2.12 <code>__beta_inc(_Tp __a, _Tp __b, _Tp __x)</code>	161
9.3.2.13 <code>__beta_inc_cont_frac(_Tp __a, _Tp __b, _Tp __x)</code>	161
9.3.2.14 <code>__beta_lgamma(_Tp __a, _Tp __b)</code>	162
9.3.2.15 <code>__beta_product(_Tp __a, _Tp __b)</code>	162
9.3.2.16 <code>__bincoef(unsigned int __n, unsigned int __k)</code>	163
9.3.2.17 <code>__bose_einstein(_Tp __s, _Tp __x)</code>	163
9.3.2.18 <code>__chebyshev_recur(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)</code>	164
9.3.2.19 <code>__chebyshev_t(unsigned int __n, _Tp __x)</code>	164
9.3.2.20 <code>__chebyshev_u(unsigned int __n, _Tp __x)</code>	165

9.3.2.21	<code>__chebyshev_v(unsigned int __n, _Tp __x)</code>	165
9.3.2.22	<code>__chebyshev_w(unsigned int __n, _Tp __x)</code>	166
9.3.2.23	<code>__chshint(_Tp __x, _Tp &_Chi, _Tp &_Shi)</code>	167
9.3.2.24	<code>__chshint_cont_frac(_Tp __t, _Tp &_Chi, _Tp &_Shi)</code>	167
9.3.2.25	<code>__chshint_series(_Tp __t, _Tp &_Chi, _Tp &_Shi)</code>	167
9.3.2.26	<code>__clamp_0_m2pi(std::complex< _Tp > __w)</code>	167
9.3.2.27	<code>__clamp_pi(std::complex< _Tp > __w)</code>	167
9.3.2.28	<code>__clausen(unsigned int __m, std::complex< _Tp > __w)</code>	167
9.3.2.29	<code>__clausen(unsigned int __m, _Tp __w)</code>	168
9.3.2.30	<code>__clausen_c(unsigned int __m, std::complex< _Tp > __w)</code>	168
9.3.2.31	<code>__clausen_c(unsigned int __m, _Tp __w)</code>	169
9.3.2.32	<code>__clausen_s(unsigned int __m, std::complex< _Tp > __w)</code>	169
9.3.2.33	<code>__clausen_s(unsigned int __m, _Tp __w)</code>	170
9.3.2.34	<code>__comp_ellint_1(_Tp __k)</code>	170
9.3.2.35	<code>__comp_ellint_2(_Tp __k)</code>	171
9.3.2.36	<code>__comp_ellint_3(_Tp __k, _Tp __nu)</code>	171
9.3.2.37	<code>__comp_ellint_d(_Tp __k)</code>	172
9.3.2.38	<code>__comp_ellint_rf(_Tp __x, _Tp __y)</code>	172
9.3.2.39	<code>__comp_ellint_rg(_Tp __x, _Tp __y)</code>	172
9.3.2.40	<code>__conf_hyperg(_Tp __a, _Tp __c, _Tp __x)</code>	172
9.3.2.41	<code>__conf_hyperg_lim(_Tp __c, _Tp __x)</code>	173
9.3.2.42	<code>__conf_hyperg_lim_series(_Tp __c, _Tp __x)</code>	173
9.3.2.43	<code>__conf_hyperg_luke(_Tp __a, _Tp __c, _Tp __xin)</code>	174
9.3.2.44	<code>__conf_hyperg_series(_Tp __a, _Tp __c, _Tp __x)</code>	174
9.3.2.45	<code>__coshint(const _Tp __x)</code>	175
9.3.2.46	<code>__cyl_bessel(std::complex< _Tp > __nu, std::complex< _Tp > __z)</code>	175
9.3.2.47	<code>__cyl_bessel_i(_Tp __nu, _Tp __x)</code>	175
9.3.2.48	<code>__cyl_bessel_ij_series(_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)</code>	176

9.3.2.49	<code>__cyl_bessel_ik(_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)</code>	177
9.3.2.50	<code>__cyl_bessel_ik_asymp(_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)</code>	178
9.3.2.51	<code>__cyl_bessel_ik_steep(_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)</code>	178
9.3.2.52	<code>__cyl_bessel_j(_Tp __nu, _Tp __x)</code>	179
9.3.2.53	<code>__cyl_bessel_jn(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)</code>	179
9.3.2.54	<code>__cyl_bessel_jn_asymp(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)</code>	180
9.3.2.55	<code>__cyl_bessel_jn_steep(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)</code>	180
9.3.2.56	<code>__cyl_bessel_k(_Tp __nu, _Tp __x)</code>	180
9.3.2.57	<code>__cyl_hankel_1(_Tp __nu, _Tp __x)</code>	181
9.3.2.58	<code>__cyl_hankel_1(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	181
9.3.2.59	<code>__cyl_hankel_2(_Tp __nu, _Tp __x)</code>	182
9.3.2.60	<code>__cyl_hankel_2(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	182
9.3.2.61	<code>__cyl_neumann(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	183
9.3.2.62	<code>__cyl_neumann_n(_Tp __nu, _Tp __x)</code>	183
9.3.2.63	<code>__dawson(_Tp __x)</code>	184
9.3.2.64	<code>__dawson_cont_frac(_Tp __x)</code>	184
9.3.2.65	<code>__dawson_series(_Tp __x)</code>	184
9.3.2.66	<code>__debye_region(std::complex<_Tp> __alpha, int &__indexr, char &__aorb)</code>	184
9.3.2.67	<code>__dilog(_Tp __x)</code>	185
9.3.2.68	<code>__dirichlet_beta(std::complex<_Tp> __w)</code>	185
9.3.2.69	<code>__dirichlet_beta(_Tp __w)</code>	185
9.3.2.70	<code>__dirichlet_eta(std::complex<_Tp> __w)</code>	186
9.3.2.71	<code>__dirichlet_eta(_Tp __w)</code>	186
9.3.2.72	<code>__double_factorial(int __n)</code>	187
9.3.2.73	<code>__ellint_1(_Tp __k, _Tp __phi)</code>	187

9.3.2.74	<code>__ellint_2(_Tp __k, _Tp __phi)</code>	188
9.3.2.75	<code>__ellint_3(_Tp __k, _Tp __nu, _Tp __phi)</code>	188
9.3.2.76	<code>__ellint_cel(_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)</code>	189
9.3.2.77	<code>__ellint_d(_Tp __k, _Tp __phi)</code>	189
9.3.2.78	<code>__ellint_el1(_Tp __x, _Tp __k_c)</code>	189
9.3.2.79	<code>__ellint_el2(_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)</code>	189
9.3.2.80	<code>__ellint_el3(_Tp __x, _Tp __k_c, _Tp __p)</code>	189
9.3.2.81	<code>__ellint_rc(_Tp __x, _Tp __y)</code>	189
9.3.2.82	<code>__ellint_rd(_Tp __x, _Tp __y, _Tp __z)</code>	190
9.3.2.83	<code>__ellint_rf(_Tp __x, _Tp __y, _Tp __z)</code>	191
9.3.2.84	<code>__ellint_rg(_Tp __x, _Tp __y, _Tp __z)</code>	191
9.3.2.85	<code>__ellint_rj(_Tp __x, _Tp __y, _Tp __z, _Tp __p)</code>	192
9.3.2.86	<code>__ellnome(_Tp __k)</code>	193
9.3.2.87	<code>__ellnome_k(_Tp __k)</code>	193
9.3.2.88	<code>__ellnome_series(_Tp __k)</code>	193
9.3.2.89	<code>__expint(unsigned int __n, _Tp __x)</code>	193
9.3.2.90	<code>__expint(_Tp __x)</code>	194
9.3.2.91	<code>__expint_asymp(unsigned int __n, _Tp __x)</code>	194
9.3.2.92	<code>__expint_E1(_Tp __x)</code>	195
9.3.2.93	<code>__expint_E1_asymp(_Tp __x)</code>	195
9.3.2.94	<code>__expint_E1_series(_Tp __x)</code>	196
9.3.2.95	<code>__expint_Ei(_Tp __x)</code>	196
9.3.2.96	<code>__expint_Ei_asymp(_Tp __x)</code>	197
9.3.2.97	<code>__expint_Ei_series(_Tp __x)</code>	197
9.3.2.98	<code>__expint_En_cont_frac(unsigned int __n, _Tp __x)</code>	198
9.3.2.99	<code>__expint_En_recursion(unsigned int __n, _Tp __x)</code>	198
9.3.2.100	<code>__expint_En_series(unsigned int __n, _Tp __x)</code>	199
9.3.2.101	<code>__expint_large_n(unsigned int __n, _Tp __x)</code>	199

9.3.2.102	<code>__factorial(unsigned int __n)</code>	200
9.3.2.103	<code>__fermi_dirac(_Tp __s, _Tp __x)</code>	200
9.3.2.104	<code>__fock_airy(_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)</code>	201
9.3.2.105	<code>__fpequal(const _Tp &__a, const _Tp &__b)</code>	201
9.3.2.106	<code>__fpimag(const std::complex< _Tp > &__w)</code>	202
9.3.2.107	<code>__fpimag(const _Tp)</code>	202
9.3.2.108	<code>__fpreal(const std::complex< _Tp > &__w)</code>	202
9.3.2.109	<code>__fpreal(const _Tp)</code>	202
9.3.2.110	<code>__fresnel(const _Tp __x)</code>	203
9.3.2.111	<code>__fresnel_cont_frac(const _Tp __ax, _Tp &__Cf, _Tp &__Sf)</code>	203
9.3.2.112	<code>__fresnel_series(const _Tp __ax, _Tp &__Cf, _Tp &__Sf)</code>	203
9.3.2.113	<code>__gamma(_Tp __x)</code>	203
9.3.2.114	<code>__gamma_cont_frac(_Tp __a, _Tp __x)</code>	204
9.3.2.115	<code>__gamma_l(_Tp __a, _Tp __x)</code>	204
9.3.2.116	<code>__gamma_series(_Tp __a, _Tp __x)</code>	204
9.3.2.117	<code>__gamma_temme(_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)</code>	204
9.3.2.118	<code>__gamma_u(_Tp __a, _Tp __x)</code>	205
9.3.2.119	<code>__gauss(_Tp __x)</code>	205
9.3.2.120	<code>__gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)</code>	205
9.3.2.121	<code>__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)</code>	206
9.3.2.122	<code>__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)</code>	206
9.3.2.123	<code>__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)</code>	207

9.3.2.124	<code>__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)</code>	207
9.3.2.125	<code>__hankel_uniform_olover(std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)</code>	207
9.3.2.126	<code>__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)</code>	208
9.3.2.127	<code>__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)</code>	208
9.3.2.128	<code>__heuman_lambda(__Tp __k, __Tp __phi)</code>	209
9.3.2.129	<code>__hurwitz_zeta(__Tp __s, __Tp __a)</code>	209
9.3.2.130	<code>__hurwitz_zeta(__Tp __s, std::complex< _Tp > __a)</code>	210
9.3.2.131	<code>__hurwitz_zeta_euler_maclaurin(__Tp __s, __Tp __a)</code>	210
9.3.2.132	<code>__hydrogen(const unsigned int __n, const unsigned int __l, const unsigned int __m, const __Tp __Z, const __Tp __r, const __Tp __theta, const __Tp __phi)</code>	211
9.3.2.133	<code>__hyperg(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	211
9.3.2.134	<code>__hyperg_luke(__Tp __a, __Tp __b, __Tp __c, __Tp __xin)</code>	211
9.3.2.135	<code>__hyperg_reflect(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	212
9.3.2.136	<code>__hyperg_series(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	212
9.3.2.137	<code>__jacobi_sncndn(__Tp __k, __Tp __u)</code>	213
9.3.2.138	<code>__jacobi_zeta(__Tp __k, __Tp __phi)</code>	213
9.3.2.139	<code>__laguerre(unsigned int __n, __Tp __x)</code>	213
9.3.2.140	<code>__legendre_q(unsigned int __l, __Tp __x)</code>	214
9.3.2.141	<code>__log_bincoef(unsigned int __n, unsigned int __k)</code>	214
9.3.2.142	<code>__log_double_factorial(__Tp __x)</code>	214

9.3.2.143	<code>__log_double_factorial(int __n)</code>	. 215
9.3.2.144	<code>__log_factorial(unsigned int __n)</code>	. 215
9.3.2.145	<code>__log_gamma(_Tp __x)</code>	. 215
9.3.2.146	<code>__log_gamma_bernoulli(_Tp __x)</code>	. 216
9.3.2.147	<code>__log_gamma_lanczos(_Tp __x)</code>	. 216
9.3.2.148	<code>__log_gamma_sign(_Tp __x)</code>	. 216
9.3.2.149	<code>__log_gamma_spouge(_Tp __z)</code>	. 217
9.3.2.150	<code>__log_pochhammer_l(_Tp __a, _Tp __n)</code>	. 217
9.3.2.151	<code>__log_pochhammer_u(_Tp __a, _Tp __n)</code>	. 218
9.3.2.152	<code>__logint(const _Tp __x)</code>	. 218
9.3.2.153	<code>__owens_t(_Tp __h, _Tp __a)</code>	. 219
9.3.2.154	<code>__pgamma(_Tp __a, _Tp __x)</code>	. 219
9.3.2.155	<code>__pochhammer_l(_Tp __a, _Tp __n)</code>	. 220
9.3.2.156	<code>__pochhammer_u(_Tp __a, _Tp __n)</code>	. 220
9.3.2.157	<code>__poly_hermite(unsigned int __n, _Tp __x)</code>	. 220
9.3.2.158	<code>__poly_hermite_asymp(unsigned int __n, _Tp __x)</code>	. 221
9.3.2.159	<code>__poly_hermite_recursion(unsigned int __n, _Tp __x)</code>	. 221
9.3.2.160	<code>__poly_jacobi(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)</code>	. 222
9.3.2.161	<code>__poly_laguerre(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	. 222
9.3.2.162	<code>__poly_laguerre_hypere(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	. 223
9.3.2.163	<code>__poly_laguerre_large_n(unsigned __n, _Tpa __alpha1, _Tp __x)</code>	. 224
9.3.2.164	<code>__poly_laguerre_recursion(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	. 224
9.3.2.165	<code>__poly_legendre_p(unsigned int __l, _Tp __x)</code>	. 225
9.3.2.166	<code>__poly_radial_jacobi(unsigned int __n, unsigned int __m, _Tp __rho)</code>	. 225
9.3.2.167	<code>__polylog(_Tp __s, _Tp __x)</code>	. 226
9.3.2.168	<code>__polylog(_Tp __s, std::complex< _Tp > __w)</code>	. 227
9.3.2.169	<code>__polylog_exp(_Tp __s, ArgType __w)</code>	. 227
9.3.2.170	<code>__polylog_exp_asymp(_Tp __s, std::complex< _Tp > __w)</code>	. 228

9.3.2.171	<code>__polylog_exp_int_neg(int __s, std::complex< _Tp > __w)</code>	228
9.3.2.172	<code>__polylog_exp_int_neg(const int __s, _Tp __w)</code>	229
9.3.2.173	<code>__polylog_exp_int_pos(unsigned int __s, std::complex< _Tp > __w)</code>	229
9.3.2.174	<code>__polylog_exp_int_pos(unsigned int __s, _Tp __w)</code>	230
9.3.2.175	<code>__polylog_exp_neg(_Tp __s, std::complex< _Tp > __w)</code>	230
9.3.2.176	<code>__polylog_exp_neg(int __s, std::complex< _Tp > __w)</code>	231
9.3.2.177	<code>__polylog_exp_neg_even(unsigned int __n, std::complex< _Tp > __w)</code>	231
9.3.2.178	<code>__polylog_exp_neg_odd(unsigned int __n, std::complex< _Tp > __w)</code>	232
9.3.2.179	<code>__polylog_exp_negative_real_part(_PowTp __s, _Tp __w)</code>	233
9.3.2.180	<code>__polylog_exp_pos(unsigned int __s, std::complex< _Tp > __w)</code>	233
9.3.2.181	<code>__polylog_exp_pos(unsigned int __s, _Tp __w)</code>	234
9.3.2.182	<code>__polylog_exp_pos(_Tp __s, std::complex< _Tp > __w)</code>	235
9.3.2.183	<code>__polylog_exp_real_neg(_Tp __s, std::complex< _Tp > __w)</code>	235
9.3.2.184	<code>__polylog_exp_real_neg(_Tp __s, _Tp __w)</code>	236
9.3.2.185	<code>__polylog_exp_real_pos(_Tp __s, std::complex< _Tp > __w)</code>	236
9.3.2.186	<code>__polylog_exp_real_pos(_Tp __s, _Tp __w)</code>	237
9.3.2.187	<code>__psi(_Tp __x)</code>	237
9.3.2.188	<code>__psi(unsigned int __n, _Tp __x)</code>	237
9.3.2.189	<code>__psi_asymp(_Tp __x)</code>	238
9.3.2.190	<code>__psi_series(_Tp __x)</code>	238
9.3.2.191	<code>__qgamma(_Tp __a, _Tp __x)</code>	238
9.3.2.192	<code>__riemann_zeta(_Tp __s)</code>	238
9.3.2.193	<code>__riemann_zeta_alt(_Tp __s)</code>	239
9.3.2.194	<code>__riemann_zeta_euler_maclaurin(_Tp __s)</code>	239
9.3.2.195	<code>__riemann_zeta_glob(_Tp __s)</code>	239
9.3.2.196	<code>__riemann_zeta_m_1(_Tp __s)</code>	240
9.3.2.197	<code>__riemann_zeta_m_1_sum(_Tp __s)</code>	240
9.3.2.198	<code>__riemann_zeta_product(_Tp __s)</code>	240

9.3.2.199	<code>__riemann_zeta_sum(_Tp __s)</code>	241
9.3.2.200	<code>__sinc(_Tp __a, _Tp __x)</code>	241
9.3.2.201	<code>__sinc(_Tp __x)</code>	241
9.3.2.202	<code>__sinc_pi(_Tp __x)</code>	242
9.3.2.203	<code>__sincosint(_Tp __x)</code>	242
9.3.2.204	<code>__sincosint_asymp(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	242
9.3.2.205	<code>__sincosint_cont_frac(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	242
9.3.2.206	<code>__sincosint_series(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	243
9.3.2.207	<code>__sinhc(_Tp __a, _Tp __x)</code>	243
9.3.2.208	<code>__sinhc(_Tp __x)</code>	243
9.3.2.209	<code>__sinhc_pi(_Tp __x)</code>	243
9.3.2.210	<code>__sinhint(const _Tp __x)</code>	243
9.3.2.211	<code>__sph_bessel(unsigned int __n, _Tp __x)</code>	244
9.3.2.212	<code>__sph_bessel(unsigned int __n, std::complex<_Tp> __z)</code>	244
9.3.2.213	<code>__sph_bessel_ik(unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)</code>	245
9.3.2.214	<code>__sph_bessel_jn(unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)</code>	245
9.3.2.215	<code>__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)</code>	246
9.3.2.216	<code>__sph_hankel_1(unsigned int __n, _Tp __x)</code>	246
9.3.2.217	<code>__sph_hankel_1(unsigned int __n, std::complex<_Tp> __z)</code>	247
9.3.2.218	<code>__sph_hankel_2(unsigned int __n, _Tp __x)</code>	247
9.3.2.219	<code>__sph_hankel_2(unsigned int __n, std::complex<_Tp> __z)</code>	248
9.3.2.220	<code>__sph_harmonic(unsigned int __l, int __m, _Tp __theta, _Tp __phi)</code>	248
9.3.2.221	<code>__sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</code>	248
9.3.2.222	<code>__sph_neumann(unsigned int __n, _Tp __x)</code>	249
9.3.2.223	<code>__sph_neumann(unsigned int __n, std::complex<_Tp> __z)</code>	250
9.3.2.224	<code>__theta_1(_Tp __nu, _Tp __x)</code>	251

9.3.2.225	__theta_2(_Tp __nu, _Tp __x)	251
9.3.2.226	__theta_2_asymp(_Tp __nu, _Tp __x)	252
9.3.2.227	__theta_2_sum(_Tp __nu, _Tp __x)	252
9.3.2.228	__theta_3(_Tp __nu, _Tp __x)	252
9.3.2.229	__theta_3_asymp(_Tp __nu, _Tp __x)	253
9.3.2.230	__theta_3_sum(_Tp __nu, _Tp __x)	253
9.3.2.231	__theta_4(_Tp __nu, _Tp __x)	253
9.3.2.232	__theta_c(_Tp __k, _Tp __x)	253
9.3.2.233	__theta_d(_Tp __k, _Tp __x)	254
9.3.2.234	__theta_n(_Tp __k, _Tp __x)	254
9.3.2.235	__theta_s(_Tp __k, _Tp __x)	254
9.3.2.236	__zernike(unsigned int __n, int __m, _Tp __rho, _Tp __phi)	254
9.3.2.237	__znorm1(_Tp __x)	255
9.3.2.238	__znorm2(_Tp __x)	255
9.3.2.239	evenzeta(unsigned int __k)	255
9.3.3	Variable Documentation	255
9.3.3.1	__max_FGH	255
9.3.3.2	__max_FGH< double >	256
9.3.3.3	__max_FGH< float >	256
9.3.3.4	_Num_Euler_Maclaurin_zeta	256
9.3.3.5	_S_double_factorial_table	256
9.3.3.6	_S_Euler_Maclaurin_zeta	256
9.3.3.7	_S_factorial_table	256
9.3.3.8	_S_neg_double_factorial_table	256
9.3.3.9	_S_num_double_factorials	257
9.3.3.10	_S_num_double_factorials< double >	257
9.3.3.11	_S_num_double_factorials< float >	257
9.3.3.12	_S_num_double_factorials< long double >	257
9.3.3.13	_S_num_factorials	257
9.3.3.14	_S_num_factorials< double >	257
9.3.3.15	_S_num_factorials< float >	257
9.3.3.16	_S_num_factorials< long double >	257
9.3.3.17	_S_num_neg_double_factorials	257
9.3.3.18	_S_num_neg_double_factorials< double >	257
9.3.3.19	_S_num_neg_double_factorials< float >	258
9.3.3.20	_S_num_neg_double_factorials< long double >	258
9.3.3.21	_S_num_zetam1	258
9.3.3.22	_S_zetam1	258

10 Class Documentation	259
10.1 std::__detail::_Airy<_Tp> Class Template Reference	259
10.1.1 Detailed Description	260
10.1.2 Member Typedef Documentation	260
10.1.2.1 scalar_type	260
10.1.2.2 value_type	260
10.1.3 Constructor & Destructor Documentation	260
10.1.3.1 _Airy()=default	260
10.1.3.2 _Airy(const _Airy &)=default	260
10.1.3.3 _Airy(_Airy &&)=default	260
10.1.4 Member Function Documentation	260
10.1.4.1 operator()(value_type __y) const	260
10.1.5 Member Data Documentation	261
10.1.5.1 _S_2pi_3	261
10.1.5.2 _S_5pi_6	261
10.1.5.3 _S_cNaN	261
10.1.5.4 _S_i	261
10.1.5.5 _S_NaN	261
10.1.5.6 _S_pi	261
10.1.5.7 _S_pi_3	261
10.1.5.8 _S_pi_6	261
10.1.5.9 _S_sqrt_pi	262
10.1.5.10 inner_radius	262
10.1.5.11 outer_radius	262
10.2 std::__detail::_Airy_asymp<_Tp> Class Template Reference	262
10.2.1 Detailed Description	263
10.2.2 Member Typedef Documentation	264
10.2.2.1 __cmplx	264

10.2.3	Constructor & Destructor Documentation	264
10.2.3.1	_Airy_asymp()=default	264
10.2.4	Member Function Documentation	264
10.2.4.1	_S_absarg_ge_pio3(std::complex< _Tp > __z) const	264
10.2.4.2	_S_absarg_lt_pio3(std::complex< _Tp > __z) const	264
10.2.4.3	operator()(std::complex< _Tp > __t, bool __return_fock_airy=false) const	265
10.3	std::__detail::_Airy_asymp_data< _Tp > Struct Template Reference	266
10.3.1	Detailed Description	266
10.4	std::__detail::_Airy_asymp_data< __float128 > Struct Template Reference	266
10.4.1	Detailed Description	267
10.4.2	Member Data Documentation	267
10.4.2.1	_S_c	267
10.4.2.2	_S_d	267
10.4.2.3	_S_max_cd	267
10.5	std::__detail::_Airy_asymp_data< double > Struct Template Reference	267
10.5.1	Detailed Description	267
10.5.2	Member Data Documentation	268
10.5.2.1	_S_c	268
10.5.2.2	_S_d	268
10.5.2.3	_S_max_cd	268
10.6	std::__detail::_Airy_asymp_data< float > Struct Template Reference	268
10.6.1	Detailed Description	268
10.6.2	Member Data Documentation	268
10.6.2.1	_S_c	268
10.6.2.2	_S_d	269
10.6.2.3	_S_max_cd	269
10.7	std::__detail::_Airy_asymp_data< long double > Struct Template Reference	269
10.7.1	Detailed Description	269

10.7.2	Member Data Documentation	269
10.7.2.1	_S_c	269
10.7.2.2	_S_d	269
10.7.2.3	_S_max_cd	270
10.8	std::__detail::_Airy_asymp_series< _Sum > Class Template Reference	270
10.8.1	Detailed Description	270
10.8.2	Member Typedef Documentation	270
10.8.2.1	scalar_type	271
10.8.2.2	value_type	271
10.8.3	Constructor & Destructor Documentation	271
10.8.3.1	_Airy_asymp_series(_Sum __proto)	271
10.8.3.2	_Airy_asymp_series(const _Airy_asymp_series &)=default	271
10.8.3.3	_Airy_asymp_series(_Airy_asymp_series &&)=default	271
10.8.4	Member Function Documentation	271
10.8.4.1	operator()(value_type __y)	271
10.8.5	Member Data Documentation	271
10.8.5.1	_S_sqrt_pi	271
10.9	std::__detail::_Airy_default_radII< _Tp > Struct Template Reference	272
10.9.1	Detailed Description	272
10.10	std::__detail::_Airy_default_radII< double > Struct Template Reference	272
10.10.1	Detailed Description	272
10.10.2	Member Data Documentation	272
10.10.2.1	inner_radius	272
10.10.2.2	outer_radius	273
10.11	std::__detail::_Airy_default_radII< float > Struct Template Reference	273
10.11.1	Detailed Description	273
10.11.2	Member Data Documentation	273
10.11.2.1	inner_radius	273

10.11.2.2 outer_radius	273
10.12std::__detail::_Airy_default_radii< long double > Struct Template Reference	273
10.12.1 Detailed Description	274
10.12.2 Member Data Documentation	274
10.12.2.1 inner_radius	274
10.12.2.2 outer_radius	274
10.13std::__detail::_Airy_series< _Tp > Class Template Reference	274
10.13.1 Detailed Description	275
10.13.2 Member Function Documentation	275
10.13.2.1 _S_Ai(std::complex< _Tp > __t)	275
10.13.2.2 _S_Airy(std::complex< _Tp > __t)	276
10.13.2.3 _S_Bi(std::complex< _Tp > __t)	276
10.13.2.4 _S_FGH(std::complex< _Tp > __t)	277
10.13.2.5 _S_Fock(std::complex< _Tp > __t)	277
10.13.2.6 _S_Scorer(std::complex< _Tp > __t)	277
10.13.2.7 _S_Scorer2(std::complex< _Tp > __t)	278
10.13.3 Member Data Documentation	278
10.13.3.1 _N_FGH	278
10.13.3.2 _S_Ai0	279
10.13.3.3 _S_Aip0	279
10.13.3.4 _S_Bi0	279
10.13.3.5 _S_Bip0	279
10.13.3.6 _S_eps	279
10.13.3.7 _S_Gi0	279
10.13.3.8 _S_Gip0	279
10.13.3.9 _S_Hi0	279
10.13.3.10_S_Hip0	280
10.13.3.11_S_i	280

10.13.3.12 <code>_S_log10min</code>	280
10.13.3.13 <code>_S_pi</code>	280
10.13.3.14 <code>_S_sqrt_pi</code>	280
10.14 <code>std::__detail::_AiryAuxilliaryState< _Tp ></code> Struct Template Reference	280
10.14.1 Detailed Description	281
10.14.2 Member Typedef Documentation	281
10.14.2.1 <code>_Val</code>	281
10.14.3 Member Data Documentation	281
10.14.3.1 <code>fai</code>	281
10.14.3.2 <code>faip</code>	281
10.14.3.3 <code>gai</code>	281
10.14.3.4 <code>gaip</code>	281
10.14.3.5 <code>hai</code>	281
10.14.3.6 <code>haip</code>	282
10.14.3.7 <code>z</code>	282
10.15 <code>std::__detail::_AiryState< _Tp ></code> Struct Template Reference	282
10.15.1 Detailed Description	282
10.15.2 Member Typedef Documentation	283
10.15.2.1 <code>_Val</code>	283
10.15.3 Member Function Documentation	283
10.15.3.1 <code>true_Wronskian()</code>	283
10.15.3.2 <code>Wronskian() const</code>	283
10.15.4 Member Data Documentation	283
10.15.4.1 <code>Ai</code>	283
10.15.4.2 <code>Aip</code>	283
10.15.4.3 <code>Bi</code>	283
10.15.4.4 <code>Bip</code>	283
10.15.4.5 <code>z</code>	284
10.16 <code>std::__detail::_Factorial_table< _Tp ></code> Struct Template Reference	284
10.16.1 Detailed Description	284
10.16.2 Member Data Documentation	284
10.16.2.1 <code>__factorial</code>	284
10.16.2.2 <code>__log_factorial</code>	284
10.16.2.3 <code>__n</code>	284

11 File Documentation	285
11.1 bits/sf_airy.tcc File Reference	285
11.1.1 Detailed Description	287
11.1.2 Macro Definition Documentation	287
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC	287
11.2 bits/sf_bessel.tcc File Reference	287
11.2.1 Detailed Description	289
11.2.2 Macro Definition Documentation	289
11.2.2.1 _GLIBCXX_BITS_SF_BESSEL_TCC	289
11.3 bits/sf_beta.tcc File Reference	289
11.3.1 Detailed Description	290
11.3.2 Macro Definition Documentation	291
11.3.2.1 _GLIBCXX_BITS_SF_BETA_TCC	291
11.4 bits/sf_cardinal.tcc File Reference	291
11.4.1 Macro Definition Documentation	292
11.4.1.1 _GLIBCXX_BITS_SF_CARDINAL_TCC	292
11.5 bits/sf_chebyshev.tcc File Reference	293
11.5.1 Detailed Description	294
11.5.2 Macro Definition Documentation	294
11.5.2.1 _GLIBCXX_BITS_SF_CHEBYSHEV_TCC	294
11.6 bits/sf_dawson.tcc File Reference	294
11.6.1 Detailed Description	295
11.6.2 Macro Definition Documentation	295
11.6.2.1 _GLIBCXX_BITS_SF_DAWSON_TCC	295
11.7 bits/sf_ellint.tcc File Reference	296
11.7.1 Detailed Description	298
11.7.2 Macro Definition Documentation	298
11.7.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC	298

11.8 bits/sf_expint.tcc File Reference	298
11.8.1 Detailed Description	300
11.8.2 Macro Definition Documentation	300
11.8.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC	300
11.9 bits/sf_fresnel.tcc File Reference	301
11.9.1 Detailed Description	302
11.9.2 Macro Definition Documentation	302
11.9.2.1 _GLIBCXX_BITS_SF_FRESNEL_TCC	302
11.10bits/sf_gamma.tcc File Reference	302
11.10.1 Detailed Description	307
11.10.2 Macro Definition Documentation	308
11.10.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC	308
11.11bits/sf_gegenbauer.tcc File Reference	308
11.11.1 Detailed Description	309
11.11.2 Macro Definition Documentation	309
11.11.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC	309
11.12bits/sf_hankel.tcc File Reference	309
11.12.1 Detailed Description	312
11.12.2 Macro Definition Documentation	312
11.12.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC	312
11.13bits/sf_hermite.tcc File Reference	312
11.13.1 Detailed Description	313
11.13.2 Macro Definition Documentation	313
11.13.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC	313
11.14bits/sf_hydrogen.tcc File Reference	314
11.14.1 Detailed Description	315
11.14.2 Macro Definition Documentation	315
11.14.2.1 _GLIBCXX_BITS_SF_HYDROGEN_TCC	315

11.15bits/sf_hyperg.tcc File Reference	315
11.15.1 Detailed Description	316
11.15.2 Macro Definition Documentation	317
11.15.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC	317
11.16bits/sf_hypint.tcc File Reference	317
11.16.1 Detailed Description	318
11.16.2 Macro Definition Documentation	318
11.16.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC	318
11.17bits/sf_jacobi.tcc File Reference	318
11.17.1 Detailed Description	319
11.17.2 Macro Definition Documentation	319
11.17.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC	319
11.18bits/sf_laguerre.tcc File Reference	320
11.18.1 Detailed Description	321
11.18.2 Macro Definition Documentation	321
11.18.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC	321
11.19bits/sf_legendre.tcc File Reference	321
11.19.1 Detailed Description	323
11.19.2 Macro Definition Documentation	323
11.19.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC	323
11.20bits/sf_mod_bessel.tcc File Reference	323
11.20.1 Detailed Description	325
11.20.2 Macro Definition Documentation	325
11.20.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC	325
11.21bits/sf_owens_t.tcc File Reference	325
11.21.1 Detailed Description	326
11.21.2 Macro Definition Documentation	326
11.21.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC	326

11.22bits/sf_polylog.tcc File Reference	326
11.22.1 Detailed Description	329
11.22.2 Macro Definition Documentation	329
11.22.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC	329
11.23bits/sf_theta.tcc File Reference	329
11.23.1 Detailed Description	331
11.23.2 Macro Definition Documentation	331
11.23.2.1 _GLIBCXX_BITS_SF_THETA_TCC	331
11.24bits/sf_trigint.tcc File Reference	331
11.24.1 Detailed Description	333
11.24.2 Macro Definition Documentation	333
11.24.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC	333
11.25bits/sf_zeta.tcc File Reference	333
11.25.1 Detailed Description	335
11.25.2 Macro Definition Documentation	335
11.25.2.1 _GLIBCXX_BITS_SF_ZETA_TCC	335
11.26bits/specfun.h File Reference	336
11.26.1 Detailed Description	346
11.26.2 Macro Definition Documentation	346
11.26.2.1 __cpp_lib_math_special_functions	346
11.26.2.2 __STDCPP_MATH_SPEC_FUNCS__	346
Index	347

Chapter 1

Mathematical Special Functions

1.1 Introduction and History

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

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

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

1.2 Contents

The following functions are implemented in namespace `std`:

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

- [ellint_2](#) - Incomplete elliptic functions of the second kind
- [ellint_3](#) - Incomplete elliptic functions of the third kind
- [expint](#) - The exponential integral
- [hermite](#) - Hermite polynomials
- [laguerre](#) - Laguerre functions
- [legendre](#) - Legendre polynomials
- [riemann_zeta](#) - The Riemann zeta function
- [sph_bessel](#) - Spherical Bessel functions
- [sph_legendre](#) - Spherical Legendre functions
- [sph_neumann](#) - Spherical Neumann functions

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

- [conf_hyperg](#) - Confluent hypergeometric functions
- [hyperg](#) - Hypergeometric functions

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

- [airy_ai](#) - Airy functions of the first kind
- [airy_bi](#) - Airy functions of the second kind
- [bincoef](#) - Binomial coefficients
- [chebyshev_t](#) - Chebyshev polynomials of the first kind
- [chebyshev_u](#) - Chebyshev polynomials of the second kind
- [chebyshev_v](#) - Chebyshev polynomials of the third kind
- [chebyshev_w](#) - Chebyshev polynomials of the fourth kind
- [clausen](#) - Clausen integrals
- [clausen_c](#) - Clausen cosine integrals
- [clausen_s](#) - Clausen sine integrals
- [comp_ellint_d](#) - Incomplete Legendre D elliptic integral
- [conf_hyperg_lim](#) - Confluent hypergeometric limit functions
- [coshint](#) - Hyperbolic cosine integral
- [cosint](#) - Cosine integral
- [cyl_hankel_1](#) - Cylindrical Hankel functions of the first kind
- [cyl_hankel_2](#) - Cylindrical Hankel functions of the second kind

- [dawson](#) - Dawson integrals
- [dilog](#) - Dilogarithm functions
- [dirichlet_beta](#) - Dirichlet beta function
- [dirichlet_eta](#) - Dirichlet beta function
- [dirichlet_lambda](#) - Dirichlet lambda function
- [double_factorial](#) -
- [ellint_d](#) - Legendre D elliptic integrals
- [ellint_rc](#) - Carlson elliptic functions R_C
- [ellint_rd](#) - Carlson elliptic functions R_D
- [ellint_rf](#) - Carlson elliptic functions R_F
- [ellint_rg](#) - Carlson elliptic functions R_G
- [ellint_rj](#) - Carlson elliptic functions R_J
- [expint](#) - Exponential integrals
- [factorial](#) - Factorials
- [fresnel_c](#) - Fresnel cosine integrals
- [fresnel_s](#) - Fresnel sine integrals
- [gamma_l](#) - Lower incomplete gamma functions
- [pgamma](#) - Regularized lower incomplete gamma functions
- [qgamma](#) - Regularized upper incomplete gamma functions
- [gamma_u](#) - upper incomplete gamma functions
- [gegenbauer](#) - Gegenbauer polynomials
- [heuman_lambda](#) - Heuman lambda functions
- [hurwitz_zeta](#) - Hurwitz zeta functions
- [ibeta](#) - Regularized incomplete beta functions
- [jacobi](#) - Jacobi polynomials
- [jacobi_sn](#) - Jacobi sine amplitude functions
- [jacobi_cn](#) - Jacobi cosine amplitude functions
- [jacobi_dn](#) - Jacobi delta amplitude functions
- [jacobi_zeta](#) - Jacobi zeta functions
- [lbincoef](#) - Log binomial coefficients
- [ldouble_factorial](#) - Log double factorials
- [legendre_q](#) - Legendre functions of the second kind
- [lfactorial](#) - Log factorials

- [lpochhammer_l](#) - Log lower Pochhammer functions
- [lpochhammer_u](#) - Log upper Pochhammer functions
- [owens_t](#) - Owens T functions
- [pochhammer_l](#) - Lower Pochhammer functions
- [pochhammer_u](#) - Upper Pochhammer functions
- [psi](#) - Psi of digamma function
- [radpoly](#) - Radial polynomials
- [sinhc](#) - Hyperbolic sinus cardinal function
- [sinhc_pi](#) -
- [sinc](#) - Normalized sinus cardinal 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 second kind
- [sph_harmonic](#) - Spherical
- [zernike](#) - Zernike polynomials

1.3 General Features

1.3.1 Argument Promotion

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

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

1.3.2 NaN Arguments

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

1.4 Implementation

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

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

1.5 Testing

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

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

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

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

1.6 General Bibliography

See also

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

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

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

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

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

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

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

Chapter 2

Todo List

page [Mathematical Special Functions](#)

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

Member [std::__detail::__dawson_cont_frac](#) ([_Tp __x](#))

this needs some compile-time construction!

Member [std::__detail::__expint_E1](#) ([_Tp __x](#))

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

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

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

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

This [__hurwitz_zeta](#) prefactor is prone to overflow. positive integer orders s ?

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	19
C++17/IS29124 Mathematical Special Functions	20
GNU Extended Mathematical Special Functions	44

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

__gnu_cxx	129
std	137
std::__detail	139

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

std::__detail::_Airy<_Tp>	259
std::__detail::_Airy_asymp_data<_Tp>	266
std::__detail::_Airy_asymp<_Tp>	262
std::__detail::_Airy_asymp_data<__float128>	266
std::__detail::_Airy_asymp_data<double>	267
std::__detail::_Airy_asymp_data<float>	268
std::__detail::_Airy_asymp_data<long double>	269
std::__detail::_Airy_asymp_series<_Sum>	270
std::__detail::_Airy_default_radii<_Tp>	272
std::__detail::_Airy_default_radii<double>	272
std::__detail::_Airy_default_radii<float>	273
std::__detail::_Airy_default_radii<long double>	273
std::__detail::_Airy_series<_Tp>	274
std::__detail::_AiryAuxilliaryState<_Tp>	280
std::__detail::_AiryState<_Tp>	282
std::__detail::_Factorial_table<_Tp>	284

Chapter 6

Class Index

6.1 Class List

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

std::__detail::_Airy<_Tp>	259
std::__detail::_Airy_asymp<_Tp>	262
std::__detail::_Airy_asymp_data<_Tp>	266
std::__detail::_Airy_asymp_data<__float128>	266
std::__detail::_Airy_asymp_data<double>	267
std::__detail::_Airy_asymp_data<float>	268
std::__detail::_Airy_asymp_data<long double>	269
std::__detail::_Airy_asymp_series<_Sum>	270
std::__detail::_Airy_default_radii<_Tp>	272
std::__detail::_Airy_default_radii<double>	272
std::__detail::_Airy_default_radii<float>	273
std::__detail::_Airy_default_radii<long double>	273
std::__detail::_Airy_series<_Tp>	274
std::__detail::_AiryAuxilliaryState<_Tp>	280
std::__detail::_AiryState<_Tp>	282
std::__detail::_Factorial_table<_Tp>	284

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

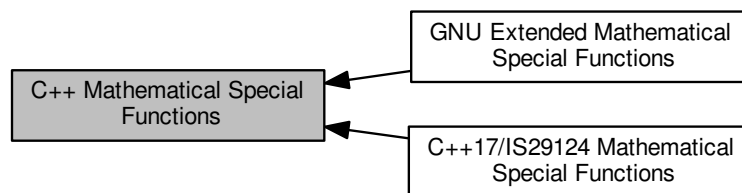
bits/sf_airy.tcc	285
bits/sf_bessel.tcc	287
bits/sf_beta.tcc	289
bits/sf_cardinal.tcc	291
bits/sf_chebyshev.tcc	293
bits/sf_dawson.tcc	294
bits/sf_ellint.tcc	296
bits/sf_expint.tcc	298
bits/sf_fresnel.tcc	301
bits/sf_gamma.tcc	302
bits/sf_gegenbauer.tcc	308
bits/sf_hankel.tcc	309
bits/sf_hermite.tcc	312
bits/sf_hydrogen.tcc	314
bits/sf_hyperg.tcc	315
bits/sf_hypint.tcc	317
bits/sf_jacobi.tcc	318
bits/sf_laguerre.tcc	320
bits/sf_legendre.tcc	321
bits/sf_mod_bessel.tcc	323
bits/sf_owens_t.tcc	325
bits/sf_polylog.tcc	326
bits/sf_theta.tcc	329
bits/sf_trigint.tcc	331
bits/sf_zeta.tcc	333
bits/specfun.h	336

Chapter 8

Module Documentation

8.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

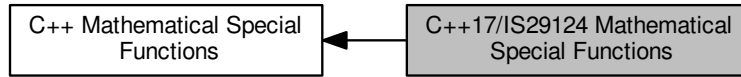
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

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 α , $L_n^\alpha(x)$, is defined by

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

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

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

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

where the Laguerre polynomial is defined by:

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

and $x \geq 0$.

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 372 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 324 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 335 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 >= 0</code> .
<code>__m</code>	The order <code>__m <= 1</code> .
<code>__x</code>	The argument, <code>abs (__x) <= 1</code> .

Exceptions

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

Definition at line 420 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 387 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 398 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 > 0</code> .
<code>__b</code>	The second argument of the beta function, <code>__b > 0</code> .

Exceptions

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

Definition at line 465 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 434 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 444 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) <= 1</code>
------------------	---

Exceptions

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

Definition at line 513 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 480 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 490 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) <= 1</code>
------------------	---

Exceptions

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

Definition at line 560 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 528 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 538 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) <= 1</code>
<code>__nu</code>	The argument

Exceptions

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

Definition at line 611 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 575 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 585 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 ν 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 >= 0</code>

Exceptions

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

Definition at line 657 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_i](#) for setails.

Definition at line 626 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_i](#) for setails.

Definition at line 636 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 ν 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 >= 0</code>

Exceptions

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

Definition at line 703 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_j](#) for setails.

Definition at line 672 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_j](#) for setails.

Definition at line 682 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 ν 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 >= 0</code>

Exceptions

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

Definition at line 755 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_k](#) for details.

Definition at line 718 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 ν and argument $x \geq 0$.

See also

[cyl_bessel_k](#) for details.

Definition at line 728 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 ν 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 >= 0</code>

Exceptions

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

Definition at line 803 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 ν and argument x .

See also

[cyl_neumann](#) for setails.

Definition at line 770 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 ν and argument x .

See also

[cyl_neumann](#) for setails.

Definition at line 780 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 ϕ .

The incomplete elliptic integral of the first kind is defined as

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

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

See also

[comp_ellint_1](#).

Template Parameters

<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) <= 1</code>
<code>__phi</code>	The integral limit argument in radians

Exceptions

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

Definition at line 851 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 ϕ .

See also

[ellint_1](#) for details.

Definition at line 818 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 ϕ .

See also

[ellint_1](#) for details.

Definition at line 828 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) <= 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) > 1</code> .
--------------------------------	------------------------------------

Definition at line 899 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 866 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 876 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) <= 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) > 1</code> .
--------------------------------	------------------------------------

Definition at line 952 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 914 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 924 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 992 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 966 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 976 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 1040 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 1007 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 1017 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> ≥ 0

Exceptions

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

Definition at line 1084 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 1055 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 1065 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 1129 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 1099 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 1109 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 1180 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 1144 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 1154 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 < 0</code> .
--------------------------------	------------------------------

Definition at line 1224 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 1195 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 1205 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 θ 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> ≥ 0
<code>__m</code>	The degree <code>__m</code> ≥ 0 and <code>__m</code> \leq <code>__l</code>
<code>__theta</code>	The radian polar angle argument

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

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 θ in radians.

See also

[sph_legendre](#) for details.

Definition at line 1239 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 θ in radians.

See also

[sph_legendre](#) for details.

Definition at line 1250 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> ≥ 0

Exceptions

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

Definition at line 1315 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 1286 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` $x \geq 0$.

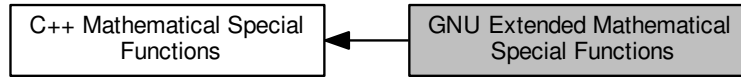
See also

[sph_neumann](#) for details.

Definition at line 1296 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](#) }

Functions

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

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

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

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

- float [__gnu_cxx::fresnel_sf](#) (float __x)
- long double [__gnu_cxx::fresnel_sl](#) (long double __x)
- template<typename _Tn, typename _Tp >
[__gnu_cxx::__promote_num_t<_Tn, _Tp>](#) [__gnu_cxx::gamma_l](#) (_Tn __n, _Tp __x)
- float [__gnu_cxx::gamma_lf](#) (float __n, float __x)
- long double [__gnu_cxx::gamma_ll](#) (long double __n, long double __x)
- template<typename _Tn, typename _Tp >
[__gnu_cxx::__promote_num_t<_Tn, _Tp>](#) [__gnu_cxx::gamma_u](#) (_Tn __n, _Tp __x)
- float [__gnu_cxx::gamma_uf](#) (float __n, float __x)
- long double [__gnu_cxx::gamma_ul](#) (long double __n, long double __x)
- template<typename _Talpha, typename _Tp >
[__gnu_cxx::__promote_num_t<_Talpha, _Tp>](#) [__gnu_cxx::gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tk, typename _Tphi >
[__gnu_cxx::__promote_num_t<_Tk, _Tphi>](#) [__gnu_cxx::heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::heuman_lambdaf](#) (float __k, float __phi)
- long double [__gnu_cxx::heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
[__gnu_cxx::__promote_num_t<_Tp, _Up>](#) [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, _Up __a)
- template<typename _Tp, typename _Up >
std::complex<_Tp> [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, std::complex<_Up> __a)
- float [__gnu_cxx::hurwitz_zetaf](#) (float __s, float __a)
- long double [__gnu_cxx::hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
[__gnu_cxx::__promote_4<_Tpa, _Tpb, _Tpc, _Tp>::type](#) [__gnu_cxx::hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [__gnu_cxx::hypergf](#) (float __a, float __b, float __c, float __x)
- long double [__gnu_cxx::hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [__gnu_cxx::ibetacf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetacl](#) (long double __a, long double __b, long double __x)
- float [__gnu_cxx::ibetaf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetal](#) (long double __a, long double __b, long double __x)
- template<typename _Talpha, typename _Tbeta, typename _Tp >
[__gnu_cxx::__promote_num_t<_Talpha, _Tbeta, _Tp>](#) [__gnu_cxx::jacobi](#) (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_num_t<_Kp, _Up>](#) [__gnu_cxx::jacobi_cn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_cnf](#) (float __k, float __u)
- long double [__gnu_cxx::jacobi_cnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_num_t<_Kp, _Up>](#) [__gnu_cxx::jacobi_dn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_dnf](#) (float __k, float __u)
- long double [__gnu_cxx::jacobi_dnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_num_t<_Kp, _Up>](#) [__gnu_cxx::jacobi_sn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_snf](#) (float __k, float __u)

- long double [__gnu_cxx::jacobi_snl](#) (long double __k, long double __u)
- template<typename _Tk, typename _Tphi >
[__gnu_cxx::__promote_num_t<_Tk, _Tphi > __gnu_cxx::jacobi_zeta](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::jacobi_zetaf](#) (float __k, float __phi)
- long double [__gnu_cxx::jacobi_zetal](#) (long double __k, long double __phi)
- float [__gnu_cxx::jacobif](#) (unsigned __n, float __alpha, float __beta, float __x)
- long double [__gnu_cxx::jacobil](#) (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::lbincoef](#) (unsigned int __n, unsigned int __k)
- float [__gnu_cxx::lbincoeff](#) (unsigned int __n, unsigned int __k)
- long double [__gnu_cxx::lbincoefl](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::ldouble_factorial](#) (int __n)
- float [__gnu_cxx::ldouble_factorialf](#) (int __n)
- long double [__gnu_cxx::ldouble_factoriall](#) (int __n)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::legendre_q](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::legendre_qf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::legendre_ql](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::lfactorial](#) (unsigned int __n)
- float [__gnu_cxx::lfactorialf](#) (unsigned int __n)
- long double [__gnu_cxx::lfactoriall](#) (unsigned int __n)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::logint](#) (_Tp __x)
- float [__gnu_cxx::logintf](#) (float __x)
- long double [__gnu_cxx::logintl](#) (long double __x)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::lpochhammer_l](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::lpochhammer_lf](#) (float __a, float __n)
- long double [__gnu_cxx::lpochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::lpochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::lpochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::lpochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tph, typename _Tpa >
[__gnu_cxx::__promote_num_t<_Tph, _Tpa > __gnu_cxx::owens_t](#) (_Tph __h, _Tpa __a)
- float [__gnu_cxx::owens_tf](#) (float __h, float __a)
- long double [__gnu_cxx::owens_tl](#) (long double __h, long double __a)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::__promote_num_t<_Ta, _Tp > __gnu_cxx::pgamma](#) (_Ta __a, _Tp __x)
- float [__gnu_cxx::pgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::pgammal](#) (long double __a, long double __x)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_l](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_lf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ul](#) (long double __a, long double __n)

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

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

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

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 1763 of file specfun.h.

8.3.3 Function Documentation

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2705 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2670 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2684 of file specfun.h.

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

\longleftrightarrow __x	The argument
------------------------------	--------------

Definition at line 2757 of file specfun.h.

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

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

See also

[airy_bi](#) for details.

Definition at line 2721 of file specfun.h.

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

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

See also

[airy_bi](#) for details.

Definition at line 2735 of file specfun.h.

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

Return the Bernoulli number of integer order n .

The Bernoulli numbers are defined by

Parameters

\longleftrightarrow __n	The order.
------------------------------	------------

Definition at line 3744 of file specfun.h.

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

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

See also

[bernoulli](#) for details.

Definition at line 3719 of file `specfun.h`.

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

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

See also

[bernoulli](#) for details.

Definition at line 3729 of file `specfun.h`.

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

Definition at line 3684 of file `specfun.h`.

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

Definition at line 3672 of file `specfun.h`.

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

Definition at line 3676 of file `specfun.h`.

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

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

The Chebyshev polynomial of the first kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 1964 of file `specfun.h`.

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

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

See also

[chebyshev_t](#) for details.

Definition at line 1935 of file `specfun.h`.

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

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

See also

[chebyshev_t](#) for details.

Definition at line 1945 of file `specfun.h`.

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

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

The Chebyshev polynomial of the second kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2008 of file specfun.h.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 1979 of file specfun.h.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 1989 of file specfun.h.

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

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

The Chebyshev polynomial of the third kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2053 of file `specfun.h`.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2023 of file `specfun.h`.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2033 of file `specfun.h`.

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

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2098 of file `specfun.h`.

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2068 of file `specfun.h`.

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2078 of file `specfun.h`.

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

Return the Clausen function $Cl_n(w)$ 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

<code>_↵ _m</code>	The integral order
<code>_↵ _w</code>	The complex argument

Definition at line 4700 of file specfun.h.

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

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

<code>_Tp</code>	The real type of the complex components
------------------	---

Parameters

<code>_↵ _m</code>	The integral order
<code>_↵ _w</code>	The complex argument

Definition at line 4744 of file specfun.h.

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

Return the Clausen cosine function $C_n(w)$ 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

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

Parameters

<code>__m</code>	The unsigned integer order
<code>__w</code>	The real argument

Definition at line 4656 of file `specfun.h`.

8.3.3.28 `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 4628 of file `specfun.h`.

8.3.3.29 `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 4638 of file `specfun.h`.

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

Return the Clausen sine function $S_n(w)$ 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

<code>__m</code>	The unsigned integer order
<code>__w</code>	The real argument

Definition at line 4613 of file specfun.h.

8.3.3.31 `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 4585 of file specfun.h.

8.3.3.32 `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 4595 of file specfun.h.

8.3.3.33 `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 4671 of file specfun.h.

8.3.3.34 `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 4715 of file `specfun.h`.

8.3.3.35 `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 4681 of file `specfun.h`.

8.3.3.36 `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 4725 of file `specfun.h`.

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

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

The complete Legendre elliptic integral D is defined by

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

Template Parameters

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

Parameters

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

Definition at line 3930 of file specfun.h.

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 3903 of file specfun.h.

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 3913 of file specfun.h.

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 2878 of file specfun.h.

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 2888 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 2906 of file specfun.h.

8.3.3.43 `float __gnu_cxx::comp_ellint_rg (float _x, float _y) [inline]`

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3111 of file specfun.h.

8.3.3.44 `long double __gnu_cxx::comp_ellint_rg (long double _x, long double _y) [inline]`

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3120 of file specfun.h.

8.3.3.45 `template<typename _Tx, typename _Ty> __gnu_cxx::__promote_num_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rg (_Tx _x, _Ty _y) [inline]`

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

The complete Carlson elliptic function is defined by:

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

Parameters

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

Definition at line 3139 of file specfun.h.

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

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

The confluent hypergeometric function is defined by

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

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

Parameters

\hookleftarrow __a	The numeratorial parameter
\hookleftarrow __c	The denominatorial parameter
\hookleftarrow __x	The argument

Definition at line 1378 of file specfun.h.

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

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

The confluent hypergeometric limit function is defined by

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

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

Parameters

\hookleftarrow __c	The denominatorial parameter
\hookleftarrow __x	The argument

Definition at line 1474 of file specfun.h.

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

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

See also

[conf_hyperg_lim](#) for details.

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

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

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

See also

[conf_hyperg_lim](#) for details.

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

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

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

See also

[conf_hyperg](#) for details.

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

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

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

See also

[conf_hyperg](#) for details.

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

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

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

The hyperbolic cosine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1756 of file `specfun.h`.

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

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

See also

[coshint](#) for details.

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

8.3.3.54 `long double __gnu_cxx::coshintl (long double __x) [inline]`

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

See also

[coshint](#) for details.

Definition at line 1738 of file `specfun.h`.

8.3.3.55 `template<typename _Tp> __gnu_cxx::_promote_num_t<_Tp> __gnu_cxx::cosint (_Tp __x) [inline]`

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

The cosine integral is defined by

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

Parameters

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

Definition at line 1673 of file specfun.h.

8.3.3.56 `float __gnu_cxx::cosintf (float __x) [inline]`

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

See also

[cosint](#) for details.

Definition at line 1647 of file specfun.h.

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

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

See also

[cosint](#) for details.

Definition at line 1657 of file specfun.h.

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

Return the cylindrical Hankel function of the first kind $H_n^{(1)}(x)$ of real order ν 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 2407 of file `specfun.h`.

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

Return the complex cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$ of complex order ν 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 4207 of file `specfun.h`.

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2374 of file `specfun.h`.

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

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

See also

[cyl_hankel_1](#) for more details.

Definition at line 4176 of file specfun.h.

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2385 of file specfun.h.

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

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

See also

[cyl_hankel_1](#) for more details.

Definition at line 4187 of file specfun.h.

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

Return the cylindrical Hankel function of the second kind $H_n^{(2)}(x)$ of real order ν 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 2456 of file specfun.h.

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

Return the complex cylindrical Hankel function of the second kind $H_\nu^{(2)}(x)$ of complex order ν 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 4254 of file specfun.h.

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

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

See also

[cyl_hankel_2](#) for details.

Definition at line 2423 of file specfun.h.

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4223 of file `specfun.h`.

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

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

See also

[cyl_hankel_2](#) for details.

Definition at line 2434 of file `specfun.h`.

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4234 of file `specfun.h`.

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

\leftrightarrow	The argument $-inf < x < inf$.
x	

Definition at line 3449 of file specfun.h.

8.3.3.71 `float __gnu_cxx::dawsonf (float __x) [inline]`

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

See also

[dawson](#) for details.

Definition at line 3421 of file specfun.h.

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

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

See also

[dawson](#) for details.

Definition at line 3430 of file specfun.h.

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

Definition at line 2822 of file specfun.h.

8.3.3.74 `float __gnu_cxx::digammaf (float __z) [inline]`

Definition at line 2810 of file specfun.h.

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

Definition at line 2814 of file specfun.h.

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

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

The dilogarithm is defined by:

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

Parameters

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

Definition at line 2863 of file specfun.h.

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

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

See also

[dilog](#) for details.

Definition at line 2837 of file specfun.h.

8.3.3.78 `long double __gnu_cxx::dilogl (long double __x) [inline]`

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

See also

[dilog](#) for details.

Definition at line 2847 of file specfun.h.

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

Return the Dirichlet beta function of real argument s .

The Dirichlet beta function is defined by:

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

An important reflection formula is:

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

Parameters

<code>_↔</code>	
<code>_s</code>	

Definition at line 4570 of file specfun.h.

8.3.3.80 `float __gnu_cxx::dirichlet_betal(float __s) [inline]`

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 4541 of file specfun.h.

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

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 4550 of file specfun.h.

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

Return the Dirichlet eta function of real argument s .

The Dirichlet eta function is defined by

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

An important reflection formula is:

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

Parameters

\leftarrow	
s	

Definition at line 4527 of file specfun.h.

8.3.3.83 `float __gnu_cxx::dirichlet_eta(float __s) [inline]`

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 4497 of file specfun.h.

8.3.3.84 `long double __gnu_cxx::dirichlet_eta(long double __s) [inline]`

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 4506 of file specfun.h.

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

Definition at line 3621 of file specfun.h.

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

Definition at line 3609 of file specfun.h.

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

Definition at line 3613 of file specfun.h.

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

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

The Bulirsch complete elliptic integral is defined by

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

Parameters

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

Definition at line 4160 of file specfun.h.

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

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

See also

[ellint_cel](#) for details.

Definition at line 4128 of file specfun.h.

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

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

See also

[ellint_cel](#) for details.

Definition at line 4137 of file specfun.h.

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

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

The Legendre elliptic integral D is defined by

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

Parameters

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

Definition at line 3973 of file `specfun.h`.

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

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

See also

[ellint_d](#) for details.

Definition at line 3945 of file `specfun.h`.

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

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

See also

[ellint_d](#) for details.

Definition at line 3955 of file `specfun.h`.

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

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

The Bulirsch elliptic integral of the first kind is defined by

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

Parameters

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

Definition at line 4019 of file specfun.h.

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

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

See also

[ellint_el1](#) for details.

Definition at line 3989 of file specfun.h.

8.3.3.96 `long double __gnu_cxx::ellint_el1(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 4000 of file specfun.h.

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

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

The Bulirsch elliptic integral of the second kind is defined by

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

Parameters

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

Definition at line 4065 of file specfun.h.

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

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

See also

[ellint_el2](#) for details.

Definition at line 4034 of file specfun.h.

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

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

See also

[ellint_el2](#) for details.

Definition at line 4044 of file specfun.h.

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

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

The Bulirsch elliptic integral of the third kind is defined by

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

Parameters

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

Definition at line 4112 of file specfun.h.

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

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

See also

[ellint_el3](#) for details.

Definition at line 4081 of file `specfun.h`.

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

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

See also

[ellint_el3](#) for details.

Definition at line 4092 of file `specfun.h`.

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 2998 of file specfun.h.

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

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

See also

[ellint_rc](#) for details.

Definition at line 2964 of file specfun.h.

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

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

See also

[ellint_rc](#) for details.

Definition at line 2973 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3097 of file specfun.h.

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

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

See also

[ellint_rd](#) for details.

Definition at line 3061 of file specfun.h.

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

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

See also

[ellint_rd](#) for details.

Definition at line 3070 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 2950 of file specfun.h.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2921 of file `specfun.h`.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2931 of file `specfun.h`.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3188 of file specfun.h.

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

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

See also

[ellint_rg](#) for details.

Definition at line 3153 of file specfun.h.

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

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

See also

[ellint_rg](#) for details.

Definition at line 3162 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3047 of file specfun.h.

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

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

See also

[ellint_rj](#) for details.

Definition at line 3012 of file specfun.h.

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

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

See also

[ellint_rj](#) for details.

Definition at line 3021 of file specfun.h.

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

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

The elliptic nome function is defined by

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

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 4958 of file specfun.h.

8.3.3.119 `float __gnu_cxx::ellnomef (float __k) [inline]`

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

See also

[ellnome](#) for details.

Definition at line 4931 of file specfun.h.

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

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

See also

[ellnome](#) for details.

Definition at line 4941 of file specfun.h.

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

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

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

In particular

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

Template Parameters

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

Parameters

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

Definition at line 3495 of file specfun.h.

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

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

See also

[expint](#) for details.

Definition at line 3464 of file `specfun.h`.

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

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

See also

[expint](#) for details.

Definition at line 3474 of file `specfun.h`.

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

Definition at line 3600 of file `specfun.h`.

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

Definition at line 3588 of file `specfun.h`.

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

Definition at line 3592 of file `specfun.h`.

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

Return the Fresnel cosine integral of argument x .

The Fresnel cosine integral is defined by

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

Parameters

\leftrightarrow	The argument
x	

Definition at line 3407 of file specfun.h.

8.3.3.128 `float __gnu_cxx::fresnel_cf (float x)` `[inline]`

Definition at line 3388 of file specfun.h.

8.3.3.129 `long double __gnu_cxx::fresnel_cl (long double x)` `[inline]`

Definition at line 3392 of file specfun.h.

8.3.3.130 `template<typename T_p > __gnu_cxx::__promote_num_t< T_p > __gnu_cxx::fresnel_s (T_p 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 3379 of file specfun.h.

8.3.3.131 `float __gnu_cxx::fresnel_sf (float x)` `[inline]`

Definition at line 3360 of file specfun.h.

8.3.3.132 `long double __gnu_cxx::fresnel_sl (long double x)` `[inline]`

Definition at line 3364 of file specfun.h.

8.3.3.133 `template<typename T_n , typename T_p > __gnu_cxx::__promote_num_t< T_n , T_p > __gnu_cxx::gamma_l (T_n n , T_p x)` `[inline]`

Definition at line 2801 of file specfun.h.

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

Definition at line 2789 of file specfun.h.

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

Definition at line 2793 of file specfun.h.

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

Definition at line 2780 of file specfun.h.

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

Definition at line 2768 of file specfun.h.

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

Definition at line 2772 of file specfun.h.

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

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

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

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

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

Template Parameters

<code>_Talpha</code>	The real type of the order
<code>_Tp</code>	The real type of the argument

Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__x</code>	The real argument

Definition at line 2206 of file specfun.h.

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

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

See also

[gegenbauer](#) for details.

Definition at line 2173 of file specfun.h.

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

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

See also

[gegenbauer](#) for details.

Definition at line 2184 of file specfun.h.

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

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

The complete Heuman lambda function is defined by

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

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

Template Parameters

<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 3888 of file specfun.h.

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

Definition at line 3862 of file specfun.h.

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

Definition at line 3866 of file specfun.h.

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

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

The the Hurwitz zeta function is defined by

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

Parameters

<code>↔ _s</code>	The argument
<code>↔ _a</code>	The parameter

Definition at line 3229 of file specfun.h.

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

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3243 of file specfun.h.

8.3.3.147 `float __gnu_cxx::hurwitz_zetaf (float __s, float __a) [inline]`

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3203 of file specfun.h.

8.3.3.148 `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a) [inline]`

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3213 of file `specfun.h`.

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

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

The hypergeometric function is defined by

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

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

Parameters

<code>__a</code>	The first numeratorial parameter
<code>__b</code>	The second numeratorial parameter
<code>__c</code>	The denominatorial parameter
<code>__x</code>	The argument

Definition at line 1427 of file `specfun.h`.

8.3.3.150 `float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x) [inline]`

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

See also

[hyperg](#) for details.

Definition at line 1394 of file `specfun.h`.

8.3.3.151 `long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x) [inline]`

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

See also

[hyperg](#) for details.

Definition at line 1405 of file `specfun.h`.

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

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

The regularized incomplete beta function is defined by

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

where

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

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

Parameters

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 3320 of file `specfun.h`.

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

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

The regularized complementary incomplete beta function is defined by

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

Parameters

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

Definition at line 3351 of file specfun.h.

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

Definition at line 3329 of file specfun.h.

References `__gnu_cxx::ibetaf()`.

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

Definition at line 3333 of file specfun.h.

References `__gnu_cxx::ibetal()`.

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

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

See `ibeta` for details.

Definition at line 3286 of file specfun.h.

Referenced by `__gnu_cxx::ibetacf()`.

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

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

See `ibeta` for details.

Definition at line 3296 of file specfun.h.

Referenced by `__gnu_cxx::ibetacl()`.

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

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

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

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

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

Template Parameters

<code>_Talpha</code>	The real type of the order α
<code>_Tbeta</code>	The real type of the order β
<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 2158 of file `specfun.h`.

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

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

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

The Jacobi elliptic `cn` integral is defined by

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

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

Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1867 of file `specfun.h`.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1831 of file specfun.h.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1844 of file specfun.h.

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

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

The Jacobi elliptic dn integral is defined by

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

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

Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1919 of file specfun.h.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1883 of file specfun.h.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1896 of file specfun.h.

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

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

The Jacobi elliptic sn integral is defined by

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

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

Template Parameters

<code>_Kp</code>	The type of the real modulus
<code>_Up</code>	The type of the real argument

Parameters

<code>__k</code>	The real modulus
<code>__u</code>	The real argument

Definition at line 1815 of file specfun.h.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1779 of file specfun.h.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1792 of file specfun.h.

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

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

The Jacobi zeta function is defined by

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

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

Template Parameters

<code>_Tk</code>	the real type of the modulus
<code>_Tphi</code>	the real type of the angle limit

Parameters

<code>__k</code>	The modulus
<code>__phi</code>	The angle

Definition at line 3853 of file specfun.h.

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

Definition at line 3828 of file specfun.h.

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

Definition at line 3832 of file specfun.h.

8.3.3.171 `float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)` `[inline]`

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

See also

[jacobi](#) for details.

Definition at line 2114 of file specfun.h.

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

8.3.3.172 `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
`[inline]`

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

See also

[jacobi](#) for details.

Definition at line 2125 of file specfun.h.

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

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

Definition at line 3705 of file specfun.h.

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

Definition at line 3693 of file specfun.h.

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

Definition at line 3697 of file specfun.h.

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

Definition at line 3663 of file specfun.h.

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

Definition at line 3651 of file specfun.h.

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

Definition at line 3655 of file specfun.h.

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

Definition at line 3777 of file specfun.h.

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

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

See also

[legendre_q](#) for details.

Definition at line 3759 of file specfun.h.

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

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

See also

[legendre_q](#) for details.

Definition at line 3769 of file specfun.h.

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

Definition at line 3642 of file specfun.h.

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

Definition at line 3630 of file specfun.h.

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

Definition at line 3634 of file specfun.h.

8.3.3.185 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::logint (_Tp __x) [inline]`

Return the logarithmic integral of argument x .

The logarithmic integral is defined by

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

Parameters

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

Definition at line 1594 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1570 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1579 of file specfun.h.

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

Definition at line 3537 of file specfun.h.

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

Definition at line 3525 of file specfun.h.

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

Definition at line 3529 of file specfun.h.

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

Definition at line 3516 of file specfun.h.

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

Definition at line 3504 of file specfun.h.

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

Definition at line 3508 of file specfun.h.

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

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

The Owens T function is defined by

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

Parameters

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

Definition at line 5169 of file specfun.h.

8.3.3.195 `float __gnu_cxx::owens_tf (float __h, float __a) [inline]`

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

See also

[owens_t](#) for details.

Definition at line 5141 of file specfun.h.

8.3.3.196 `long double __gnu_cxx::owens_tl (long double __h, long double __a) [inline]`

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

See also

[owens_t](#) for details.

Definition at line 5151 of file specfun.h.

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

Definition at line 3798 of file specfun.h.

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

Definition at line 3786 of file specfun.h.

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

Definition at line 3790 of file specfun.h.

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

Definition at line 3579 of file specfun.h.

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

Definition at line 3567 of file specfun.h.

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

Definition at line 3571 of file specfun.h.

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

Definition at line 3558 of file specfun.h.

8.3.3.204 `float __gnu_cxx::pochhammer_uf (float __a, float __n) [inline]`

Definition at line 3546 of file specfun.h.

8.3.3.205 `long double __gnu_cxx::pochhammer_ul (long double __a, long double __n) [inline]`

Definition at line 3550 of file specfun.h.

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

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

The polylogarithm function is defined by

Parameters

\leftarrow _s	
\leftarrow _w	

Definition at line 4443 of file specfun.h.

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

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

The polylogarithm function is defined by

Parameters

\leftarrow __s	
\leftarrow __w	

Definition at line 4483 of file specfun.h.

8.3.3.208 `float __gnu_cxx::polylogf (float __s, float __w) [inline]`

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

See also

[polylog](#) for details.

Definition at line 4416 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4456 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4426 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4466 of file specfun.h.

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

Return the psi or digamma function of argument x .

The the psi or digamma function is defined by

$$\psi(x) =$$

Parameters

\leftarrow __x	The parameter
---------------------	---------------

Definition at line 3271 of file specfun.h.

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

Definition at line 3252 of file specfun.h.

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

Definition at line 3256 of file specfun.h.

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

Definition at line 3819 of file specfun.h.

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

Definition at line 3807 of file specfun.h.

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

Definition at line 3811 of file `specfun.h`.

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

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

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 2316 of file `specfun.h`.

8.3.3.219 `float __gnu_cxx::radpolyf(unsigned int __n, unsigned int __m, float __rho) [inline]`

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

See also

[radpoly](#) for details.

Definition at line 2277 of file `specfun.h`.

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

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

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

See also

[radpoly](#) for details.

Definition at line 2288 of file specfun.h.

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

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

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

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

Template Parameters

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

Parameters

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

Definition at line 1556 of file specfun.h.

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

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

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

Template Parameters

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

Parameters

\leftrightarrow	The argument
x	

Definition at line 1515 of file specfun.h.

8.3.3.223 `float __gnu_cxx::sinc_pif (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 1489 of file specfun.h.

8.3.3.224 `long double __gnu_cxx::sinc_pil (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 1499 of file specfun.h.

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

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

See also

[sinc](#) for details.

Definition at line 1530 of file specfun.h.

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

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

See also

[sinc](#) for details.

Definition at line 1540 of file specfun.h.

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

Definition at line 2358 of file `specfun.h`.

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

Definition at line 2337 of file `specfun.h`.

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

Definition at line 2325 of file `specfun.h`.

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

Definition at line 2329 of file `specfun.h`.

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

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

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

Definition at line 2350 of file `specfun.h`.

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

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

The hyperbolic sine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1714 of file specfun.h.

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

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

See also

[sinhint](#) for details.

Definition at line 1687 of file specfun.h.

8.3.3.235 `long double __gnu_cxx::sinhintl (long double __x) [inline]`

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

See also

[sinhint](#) for details.

Definition at line 1697 of file specfun.h.

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

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

The sine integral is defined by

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

Parameters

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

Definition at line 1633 of file specfun.h.

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

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

See also

[sinint](#) for details.

Definition at line 1608 of file specfun.h.

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

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

See also

[sinint](#) for details.

Definition at line 1618 of file specfun.h.

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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

<code>↵ _n</code>	The integral order $n \geq 0$
<code>↵ _x</code>	The real argument $x \geq 0$

Exceptions

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

Definition at line 2596 of file specfun.h.

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

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

See also

[sph_bessel_i](#) for details.

Definition at line 2557 of file `specfun.h`.

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

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

See also

[sph_bessel_i](#) for details.

Definition at line 2572 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 2653 of file specfun.h.

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

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

See also

[sph_bessel_k](#) for more details.

Definition at line 2614 of file specfun.h.

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

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

See also

[sph_bessel_k](#) for more details.

Definition at line 2629 of file specfun.h.

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

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$ of nonnegative order n and real argument $x \geq 0$.

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

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

Template Parameters

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

Parameters

<code>↔ _n</code>	The non-negative order
<code>↔ _z</code>	The real argument

Definition at line 2499 of file specfun.h.

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

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

The spherical Hankel function of the first kind is defined by

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

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

Parameters

\leftrightarrow __n	The integral order ≥ 0
\leftrightarrow __x	The complex argument

Definition at line 4302 of file specfun.h.

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

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$ of nonnegative order n and `float` argument $x \geq 0$.

See also

[sph_hankel_1](#) for details.

Definition at line 2471 of file specfun.h.

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

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

See also

[sph_hankel_1](#) for more details.

Definition at line 4270 of file specfun.h.

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

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$ of nonnegative order n and `long double` argument $x \geq 0$.

See also

[sph_hankel_1](#) for details.

Definition at line 2481 of file `specfun.h`.

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

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

See also

[sph_hankel_1](#) for more details.

Definition at line 4281 of file `specfun.h`.

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

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$ of nonnegative order n and real argument $x \geq 0$.

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

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

Template Parameters

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

Parameters

<code>__n</code>	The non-negative order
<code>__z</code>	The real argument

Definition at line 2542 of file `specfun.h`.

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

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

The spherical Hankel function of the second kind is defined by

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

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

Parameters

<code>__n</code>	The integral order ≥ 0
<code>__x</code>	The complex argument

Definition at line 4350 of file `specfun.h`.

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

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

See also

[sph_hankel_2](#) for details.

Definition at line 2514 of file `specfun.h`.

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

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

See also

[sph_hankel_2](#) for more details.

Definition at line 4318 of file `specfun.h`.

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

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$ of nonnegative order n and `long double` argument $x \geq 0$.

See also

[sph_hankel_2](#) for details.

Definition at line 2524 of file `specfun.h`.

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

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

See also

[sph_hankel_2](#) for more details.

Definition at line 4329 of file `specfun.h`.

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

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

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 4401 of file `specfun.h`.

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

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

See also

[sph_harmonic](#) for details.

Definition at line 4365 of file specfun.h.

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

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

See also

[sph_harmonic](#) for details.

Definition at line 4377 of file specfun.h.

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

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

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 4787 of file specfun.h.

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

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

See also

[theta_1](#) for details.

Definition at line 4759 of file specfun.h.

8.3.3.262 `long double __gnu_cxx::theta_1l (long double __nu, long double __x) [inline]`

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

See also

[theta_1](#) for details.

Definition at line 4769 of file specfun.h.

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

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

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 4830 of file specfun.h.

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

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

See also

[theta_2](#) for details.

Definition at line 4802 of file specfun.h.

8.3.3.265 `long double __gnu_cxx::theta_2l (long double __nu, long double __x) [inline]`

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

See also

[theta_2](#) for details.

Definition at line 4812 of file specfun.h.

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

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

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 4873 of file specfun.h.

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

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

See also

[theta_3](#) for details.

Definition at line 4845 of file specfun.h.

8.3.3.268 `long double __gnu_cxx::theta_3l (long double __nu, long double __x) [inline]`

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

See also

[theta_3](#) for details.

Definition at line 4855 of file specfun.h.

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

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

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j + 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 4916 of file specfun.h.

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

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

See also

[theta_4](#) for details.

Definition at line 4888 of file specfun.h.

8.3.3.271 `long double __gnu_cxx::theta_4l (long double __nu, long double __x) [inline]`

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

See also

[theta_4](#) for details.

Definition at line 4898 of file specfun.h.

8.3.3.272 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_num_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

<code>__k</code>	The modulus $-1 \leq k \leq +1$
<code>__x</code>	The argument

Definition at line 5042 of file specfun.h.

8.3.3.273 `float __gnu_cxx::theta_cf (float __k, float __x) [inline]`

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 5015 of file specfun.h.

8.3.3.274 `long double __gnu_cxx::theta_cl (long double __k, long double __x) [inline]`

Return the Neville theta-c function $\theta_c(k, x)$ of long double modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 5025 of file specfun.h.

8.3.3.275 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_num_t<_Tp, _Tp> __gnu_cxx::theta_d (_Tp __k, _Tp __x) [inline]`

Return the Neville theta-d function $\theta_d(k, x)$ of modulus k and argument x .

The Neville theta-d function is defined by

$$\theta_d(k, x) =$$

Parameters

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

Definition at line 5084 of file specfun.h.

8.3.3.276 `float __gnu_cxx::theta_df (float __k, float __x) [inline]`

Return the Neville theta-d function $\theta_d(k, x)$ of modulus k and argument x .

See also

[theta_d](#) for details.

Definition at line 5057 of file specfun.h.

8.3.3.277 `long double __gnu_cxx::theta_dl (long double __k, long double __x)` `[inline]`

Return the Neville theta-d function $\theta_d(k, x)$ of `long double` modulus `k` and argument `x`.

See also

[theta_d](#) for details.

Definition at line 5067 of file `specfun.h`.

8.3.3.278 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_n (_Tpk __k, _Tp __x)` `[inline]`

Return the Neville theta-n function $\theta_n(k, x)$ of modulus `k` and argument `x`.

The Neville theta-n function is defined by

$$\theta_n(k, x) =$$

Parameters

\leftrightarrow <code>__k</code>	The modulus $-1 \leq k \leq +1$
\leftrightarrow <code>__x</code>	The argument

Definition at line 5126 of file `specfun.h`.

8.3.3.279 `float __gnu_cxx::theta_nf (float __k, float __x)` `[inline]`

Return the Neville theta-n function $\theta_n(k, x)$ of modulus `k` and argument `x`.

See also

[theta_n](#) for details.

Definition at line 5099 of file `specfun.h`.

8.3.3.280 `long double __gnu_cxx::theta_nl (long double __k, long double __x)` `[inline]`

Return the Neville theta-n function $\theta_n(k, x)$ of `long double` modulus `k` and argument `x`.

See also

[theta_n](#) for details.

Definition at line 5109 of file `specfun.h`.

8.3.3.281 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_s (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

The Neville theta-s function is defined by

Parameters

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

Definition at line 5000 of file specfun.h.

8.3.3.282 `float __gnu_cxx::theta_sf (float __k, float __x) [inline]`

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 4973 of file specfun.h.

8.3.3.283 `long double __gnu_cxx::theta_sl (long double __k, long double __x) [inline]`

Return the Neville theta-s function $\theta_s(k, x)$ of long double modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 4983 of file specfun.h.

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

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ 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 2261 of file specfun.h.

```
8.3.3.285 float __gnu_cxx::zernikef( unsigned int __n, int __m, float __rho, float __phi ) [inline]
```

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

See also

[zernike](#) for details.

Definition at line 2222 of file specfun.h.

```
8.3.3.286 long double __gnu_cxx::zernikel( unsigned int __n, int __m, long double __rho, long double __phi ) [inline]
```

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

See also

[zernike](#) for details.

Definition at line 2233 of file specfun.h.

Chapter 9

Namespace Documentation

9.1 `__gnu_cxx` Namespace Reference

Enumerations

- enum { `_GLIBCXX_JACOBI_SN`, `_GLIBCXX_JACOBI_CN`, `_GLIBCXX_JACOBI_DN` }

Functions

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

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

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

- long double [ellint_d1](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tk >
__gnu_cxx::__promote_num_t< _Tp, _Tk > [ellint_el1](#) (_Tp __x, _Tk __k_c)
- float [ellint_el1f](#) (float __x, float __k_c)
- long double [ellint_el1l](#) (long double __x, long double __k_c)
- template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >
__gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > [ellint_el2](#) (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float [ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
__gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > [ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_num_t< _Tp, _Up > [ellint_rc](#) (_Tp __x, _Up __y)
- float [ellint_rcf](#) (float __x, float __y)
- long double [ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rdf](#) (float __x, float __y, float __z)
- long double [ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rff](#) (float __x, float __y, float __z)
- long double [ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rgf](#) (float __x, float __y, float __z)
- long double [ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > [ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
_Tp [ellnome](#) (_Tp __k)
- float [ellnomef](#) (float __k)
- long double [ellnomel](#) (long double __k)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [expint](#) (unsigned int __n, _Tp __x)
- float [expintf](#) (unsigned int __n, float __x)
- long double [expintl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [factorial](#) (unsigned int __n)
- float [factorialf](#) (unsigned int __n)
- long double [factoriall](#) (unsigned int __n)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [fresnel_c](#) (_Tp __x)
- float [fresnel_cf](#) (float __x)
- long double [fresnel_cl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [fresnel_s](#) (_Tp __x)

- float [fresnel_sf](#) (float __x)
- long double [fresnel_sl](#) (long double __x)
- template<typename _Tn, typename _Tp >
__gnu_cxx::__promote_num_t<_Tn, _Tp > [gamma_l](#) (_Tn __n, _Tp __x)
- float [gamma_lf](#) (float __n, float __x)
- long double [gamma_ll](#) (long double __n, long double __x)
- template<typename _Tn, typename _Tp >
__gnu_cxx::__promote_num_t<_Tn, _Tp > [gamma_u](#) (_Tn __n, _Tp __x)
- float [gamma_uf](#) (float __n, float __x)
- long double [gamma_ul](#) (long double __n, long double __x)
- template<typename _Talpha, typename _Tp >
__gnu_cxx::__promote_num_t<_Talpha, _Tp > [gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tk, typename _Tphi >
__gnu_cxx::__promote_num_t<_Tk, _Tphi > [heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [heuman_lambdaf](#) (float __k, float __phi)
- long double [heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_num_t<_Tp, _Up > [hurwitz_zeta](#) (_Tp __s, _Up __a)
- template<typename _Tp, typename _Up >
std::complex<_Tp > [hurwitz_zeta](#) (_Tp __s, std::complex<_Up > __a)
- float [hurwitz_zetaf](#) (float __s, float __a)
- long double [hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
__gnu_cxx::__promote_4<_Tpa, _Tpb, _Tpc, _Tp >::__type [hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [hypergf](#) (float __a, float __b, float __c, float __x)
- long double [hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp > [ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp > [ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [ibetacf](#) (float __a, float __b, float __x)
- long double [ibetacdl](#) (long double __a, long double __b, long double __x)
- float [ibetacf](#) (float __a, float __b, float __x)
- long double [ibetacdl](#) (long double __a, long double __b, long double __x)
- template<typename _Talpha, typename _Tbeta, typename _Tp >
__gnu_cxx::__promote_num_t<_Talpha, _Tbeta, _Tp > [jacobi](#) (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp, typename _Up >
__gnu_cxx::__promote_num_t<_Kp, _Up > [jacobi_cn](#) (_Kp __k, _Up __u)
- float [jacobi_cnf](#) (float __k, float __u)
- long double [jacobi_cnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
__gnu_cxx::__promote_num_t<_Kp, _Up > [jacobi_dn](#) (_Kp __k, _Up __u)
- float [jacobi_dnf](#) (float __k, float __u)
- long double [jacobi_dnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
__gnu_cxx::__promote_num_t<_Kp, _Up > [jacobi_sn](#) (_Kp __k, _Up __u)
- float [jacobi_snf](#) (float __k, float __u)
- long double [jacobi_snl](#) (long double __k, long double __u)

- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float jacobi_zetaf (float __k, float __phi)`
- `long double jacobi_zetal (long double __k, long double __phi)`
- `float jacobiif (unsigned __n, float __alpha, float __beta, float __x)`
- `long double jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > lbincoef (unsigned int __n, unsigned int __k)`
- `float lbincoeff (unsigned int __n, unsigned int __k)`
- `long double lbincoefl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > ldouble_factorial (int __n)`
- `float ldouble_factorialf (int __n)`
- `long double ldouble_factoriall (int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > legendre_q (unsigned int __n, _Tp __x)`
- `float legendre_qf (unsigned int __n, float __x)`
- `long double legendre_ql (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > lfactorial (unsigned int __n)`
- `float lfactorialf (unsigned int __n)`
- `long double lfactoriall (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > logint (_Tp __x)`
- `float logintf (float __x)`
- `long double logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_l (_Tp __a, _Tn __n)`
- `float lpochhammer_lf (float __a, float __n)`
- `long double lpochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_u (_Tp __a, _Tn __n)`
- `float lpochhammer_uf (float __a, float __n)`
- `long double lpochhammer_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > owens_t (_Tph __h, _Tpa __a)`
- `float owens_tf (float __h, float __a)`
- `long double owens_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tp > pgamma (_Ta __a, _Tp __x)`
- `float pgammaf (float __a, float __x)`
- `long double pgammal (long double __a, long double __x)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_l (_Tp __a, _Tn __n)`
- `float pochhammer_lf (float __a, float __n)`
- `long double pochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_u (_Tp __a, _Tn __n)`
- `float pochhammer_uf (float __a, float __n)`
- `long double pochhammer_ul (long double __a, long double __n)`
- `template<typename _Tp, typename _Wp >`
`__gnu_cxx::__promote_num_t< _Tp, _Wp > polylog (_Tp __s, _Wp __w)`

- `template<typename _Tp, typename _Wp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp, _Wp > > polylog (_Tp __s, std::complex< _Tp > __w)`
- `float polylogf (float __s, float __w)`
- `std::complex< float > polylogf (float __s, std::complex< float > __w)`
- `long double polylogl (long double __s, long double __w)`
- `std::complex< long double > polylogl (long double __s, std::complex< long double > __w)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > psi (_Tp __x)`
- `float psif (float __x)`
- `long double psil (long double __x)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tp > qgamma (_Ta __a, _Tp __x)`
- `float qgammaf (float __a, float __x)`
- `long double qgammal (long double __a, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > radpoly (unsigned int __n, unsigned int __m, _Tp __rho)`
- `float radpolyf (unsigned int __n, unsigned int __m, float __rho)`
- `long double radpolyl (unsigned int __n, unsigned int __m, long double __rho)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinc_pi (_Tp __x)`
- `float sinc_pif (float __x)`
- `long double sinc_pil (long double __x)`
- `float sincf (float __x)`
- `long double sincl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinhc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinhc_pi (_Tp __x)`
- `float sinhc_pif (float __x)`
- `long double sinhc_pil (long double __x)`
- `float sinhcf (float __x)`
- `long double sinhcl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinhnt (_Tp __x)`
- `float sinhntf (float __x)`
- `long double sinhntl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sinint (_Tp __x)`
- `float sinintf (float __x)`
- `long double sinintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sph_bessel_i (unsigned int __n, _Tp __x)`
- `float sph_bessel_if (unsigned int __n, float __x)`
- `long double sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > sph_bessel_k (unsigned int __n, _Tp __x)`
- `float sph_bessel_kf (unsigned int __n, float __x)`
- `long double sph_bessel_kl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)`

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

- long double [theta_sl](#) (long double __k, long double __x)
- template<typename _Trho, typename _Tphi >
__gnu_cxx::__promote_num_t< _Trho, _Tphi > [zernike](#) (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float [zernikef](#) (unsigned int __n, int __m, float __rho, float __phi)
- long double [zernikel](#) (unsigned int __n, int __m, long double __rho, long double __phi)

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](#)< __float128 >
- 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](#)

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_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)`
This returns Bernoulli numbers from a table or by summation for larger values.
- `template<typename _Tp >`
`_Tp __beta (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$.
- `template<typename _Tp >`
`_Tp __beta_gamma (_Tp __a, _Tp __b)`
Return the beta function: $B(a, b)$.
- `template<typename _Tp >`
`_Tp __beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp __beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp __beta_lgamma (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$ using the log gamma functions.
- `template<typename _Tp >`
`_Tp __beta_product (_Tp __a, _Tp __b)`
Return the beta function $B(x, y)$ using the product form.
- `template<typename _Tp >`
`_Tp __bincoef (unsigned int __n, unsigned int __k)`
Return the binomial coefficient. The binomial coefficient is given by:

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

- `template<typename _Tp >`
`_Tp __bose_einstein (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`
`_Tp __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_v (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_Tp __conf_hyperg_lim_series (_Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric limit function by series expansion.
- `template<typename _Tp >`
`_Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp __conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric function by series expansion.
- `template<typename _Tp >`
`_Tp __coshint (const _Tp __x)`
Return the hyperbolic cosine integral $li(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`_Tp __cyl_bessel_i (_Tp __nu, _Tp __x)`
Return the regular modified Bessel function of order ν : $I_\nu(x)$.
- `template<typename _Tp >`
`_Tp __cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`
This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`void __cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. 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 ν : $J_\nu(x)$.
- `template<typename _Tp >`
`void __cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void __cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`void __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 >`
`_Tp __cyl_bessel_k (_Tp __nu, _Tp __x)`
Return the irregular modified Bessel function $K_\nu(x)$ of order ν .

- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_1 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `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 ν : $N_\nu(x)$.
- `template<typename _Tp >`
`_Tp __dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .
- `template<typename _Tp >`
`_Tp __dawson_cont_frac (_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`
`_Tp __dawson_series (_Tp __x)`
Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`
`void __debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`_Tp __dilog (_Tp __x)`
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.
- `template<typename _Tp >`
`_Tp __dirichlet_beta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)`
Return the double factorial of the integer n .
- `template<typename _Tp >`
`_Tp __ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.

- `template<typename _Tp >`
`_Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`
`_Tp __ellint_rc (_Tp __x, _Tp __y)`
Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.
- `template<typename _Tp >`
`_Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)`
Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.
- `template<typename _Tp >`
`_Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`
Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.
- `template<typename _Tp >`
`_Tp __ellnome (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_k (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_series (_Tp __k)`
- `template<typename _Tp >`
`_Tp __expint (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$.
- `template<typename _Tp >`
`_Tp __expint (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp __expint_asymp (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large argument.
- `template<typename _Tp >`
`_Tp __expint_E1 (_Tp __x)`
Return the exponential integral $E_1(x)$.
- `template<typename _Tp >`
`_Tp __expint_E1_asymp (_Tp __x)`

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

- `template<typename _Tp >`
`_Tp __expint_E1_series (_Tp __x)`

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

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

Return the exponential integral $Ei(x)$.

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

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

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

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

- `template<typename _Tp >`
`_Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_Tp __expint_En_recursion (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_Tp __expint_En_series (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_Tp __expint_large_n (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)`

Return the factorial of the integer n .

- `template<typename _Tp >`
`_Tp __fermi_dirac (_Tp __s, _Tp __x)`

- `template<typename _Tp >`
`void __fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`

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

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

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

- `template<typename _Tp >`
`bool __fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`
`bool __fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpimag (const _Tp)`
- `template<typename _Tp >`
`bool __fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpreal (const _Tp)`
- `template<typename _Tp >`
`std::complex< _Tp > __fresnel (const _Tp __x)`

Return the Fresnel cosine and sine integrals as a complex number $C(x) + iS(x)$.

- `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 $\Gamma(x)$.

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`

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

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

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

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)`
- `template<typename _Tp >`
`void __gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`

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

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

and

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

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

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

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

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

- `template<typename _Tp >`
`_Tp __gauss (_Tp __x)`
- `template<typename _Tp >`
`_Tp __gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`
- `template<typename _Tp >`
`void __hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`
`void __hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`

- template<typename _Tp >
 void [__hankel_params](#) (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)
Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- template<typename _Tp >
 void [__hankel_uniform](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)
This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- template<typename _Tp >
 void [__hankel_uniform_olver](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)
Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.
- template<typename _Tp >
 void [__hankel_uniform_outer](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)
Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.
- template<typename _Tp >
 void [__hankel_uniform_sum](#) (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > &__num2, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)
Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- template<typename _Tp >
 _Tp [__heuman_lambda](#) (_Tp __k, _Tp __phi)
- template<typename _Tp >
 _Tp [__hurwitz_zeta](#) (_Tp __s, _Tp __a)
Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- template<typename _Tp >
 std::complex< _Tp > [__hurwitz_zeta](#) (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
 _Tp [__hurwitz_zeta_euler_maclaurin](#) (_Tp __s, _Tp __a)
Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- template<typename _Tp >
 std::complex< _Tp > [__hydrogen](#) (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)
- template<typename _Tp >
 _Tp [__hyperg](#) (_Tp __a, _Tp __b, _Tp __c, _Tp __x)

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

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

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

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

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.

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

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

- `template<typename _Tp >`
`std::tuple< _Tp, _Tp, _Tp > __jacobi_sncndn (_Tp __k, _Tp __u)`

- `template<typename _Tp >`
`_Tp __jacobi_zeta (_Tp __k, _Tp __phi)`

- `template<typename _Tp >`
`_Tp __laguerre (unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`
`_Tp __legendre_q (unsigned int __l, _Tp __x)`

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

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

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

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)`

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)`

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)`

Return the logarithm of the factorial of the integer n .

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

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)`

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_lanczos (_Tp __x)`

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

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

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

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_spouge (_Tp __z)`

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

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

where

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

and the error is bounded by

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

- `template<typename _Tp >`
`_Tp __log_pochhammer_l (_Tp __a, _Tp __n)`

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

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

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

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

Many notations exist:

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

,

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

, and others.

- `template<typename _Tp >`
`_Tp __log_pochhammer_u (_Tp __a, _Tp __n)`

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

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

Thus this function returns

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

Many notations exist:

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

,

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

, and others.

- `template<typename _Tp >`
`_Tp __logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

- `template<typename _Tp >`
`_Tp __owens_t (_Tp __h, _Tp __a)`

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

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

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

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

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

is the lower incomplete gamma function.

- `template<typename _Tp >`

`_Tp __pochhammer_l(_Tp __a, _Tp __n)`

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

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

In particular, $(n)_n = n!$.

- `template<typename _Tp >`

`_Tp __pochhammer_u(_Tp __a, _Tp __n)`

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

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

Many notations exist:

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

,

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

, and others.

- `template<typename _Tp >`

`_Tp __poly_hermite(unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`

`_Tp __poly_hermite_asymp(unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`

`_Tp __poly_hermite_recursion(unsigned int __n, _Tp __x)`

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

- `template<typename _Tp >`

`_Tp __poly_jacobi(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

- `template<typename _Tpa, typename _Tp >`

`_Tp __poly_laguerre(unsigned int __n, _Tpa __alpha1, _Tp __x)`

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

- `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 α : $L_n^\alpha(x)$ by recursion.

- `template<typename _Tp >`
`_Tp __poly_legendre_p (unsigned int __l, _Tp __x)`
Return the Legendre polynomial by upward recursion on order l .
- `template<typename _Tp >`
`_Tp __poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`
- `template<typename _Tp >`
`_Tp __polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > __polylog_exp (_Tp __s, ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_neg (const int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma >`
`std::complex< _Tp > __polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma >`
`std::complex< _Tp > __polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp __polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`_Tp __psi (_Tp __x)`
Return the digamma function. The digamma or $\psi(x)$ function is defined by

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

For negative argument the reflection formula is used:

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

- `template<typename _Tp >`
`_Tp __psi (unsigned int __n, _Tp __x)`
Return the polygamma function $\psi^{(n)}(x)$.

- `template<typename _Tp >`
`_Tp __psi_asymp (_Tp __x)`
Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

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

- `template<typename _Tp >`
`_Tp __psi_series (_Tp __x)`
Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

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

- `template<typename _Tp >`
`_Tp __qgamma (_Tp __a, _Tp __x)`
Return the regularized upper incomplete gamma function. The regularized upper incomplete gamma function is defined by

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

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

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

is the upper incomplete gamma function.

- `template<typename _Tp >`
`_Tp __riemann_zeta (_Tp __s)`
Return the Riemann zeta function $\zeta(s)$.
- `template<typename _Tp >`
`_Tp __riemann_zeta_alt (_Tp __s)`
Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.
- `template<typename _Tp >`
`_Tp __riemann_zeta_euler_maclaurin (_Tp __s)`
Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.
- `template<typename _Tp >`
`_Tp __riemann_zeta_glob (_Tp __s)`
Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.
- `template<typename _Tp >`
`_Tp __riemann_zeta_m_1 (_Tp __s)`
Return the Riemann zeta function $\zeta(s) - 1$.
- `template<typename _Tp >`
`_Tp __riemann_zeta_m_1_sum (_Tp __s)`
Return the Riemann zeta function $\zeta(s) - 1$ by summation for $s > 1$. This is a small remainder for large s .

- template<typename _Tp >
_Tp [__riemann_zeta_product](#) (_Tp __s)
Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.
- template<typename _Tp >
_Tp [__riemann_zeta_sum](#) (_Tp __s)
Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinc](#) (_Tp __a, _Tp __x)
Return the generalized sinus cardinal function
$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinc](#) (_Tp __x)
Return the normalized sinus cardinal function
$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinc_pi](#) (_Tp __x)
Return the unnormalized sinus cardinal function
$$\text{sinc}_\pi(x) = \frac{\sin(x)}{x}$$
- template<typename _Tp >
std::pair<_Tp, _Tp> [__sincosint](#) (_Tp __x)
This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.
- template<typename _Tp >
void [__sincosint_asymp](#) (_Tp __t, _Tp &_Si, _Tp &_Ci)
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- template<typename _Tp >
void [__sincosint_cont_frac](#) (_Tp __t, _Tp &_Si, _Tp &_Ci)
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- template<typename _Tp >
void [__sincosint_series](#) (_Tp __t, _Tp &_Si, _Tp &_Ci)
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinhc](#) (_Tp __a, _Tp __x)
Return the generalized hyperbolic sinus cardinal function
$$\text{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinhc](#) (_Tp __x)
Return the normalized hyperbolic sinus cardinal function
$$\text{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinhc_pi](#) (_Tp __x)

Return the unnormalized hyperbolic sinus cardinal function

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

- `template<typename _Tp >`
`_Tp __sinhint (const _Tp __x)`
Return the hyperbolic sine integral $li(x)$.
- `template<typename _Tp >`
`_Tp __sph_bessel (unsigned int __n, _Tp __x)`
Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .
- `template<typename _Tp >`
`std::complex< _Tp > __sph_bessel (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Bessel function.
- `template<typename _Tp >`
`void __sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`
Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.
- `template<typename _Tp >`
`void __sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`
Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.
- `template<typename _Tp >`
`void __sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)`
Helper to compute complex spherical Hankel functions and their derivatives.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`
Return the spherical harmonic function.
- `template<typename _Tp >`
`_Tp __sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`
Return the spherical associated Legendre function.
- `template<typename _Tp >`
`_Tp __sph_neumann (unsigned int __n, _Tp __x)`
Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .
- `template<typename _Tp >`
`std::complex< _Tp > __sph_neumann (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Neumann function.

- `template<typename _Tp >`
`_Tp __theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_s (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __znorm1 (_Tp __x)`
- `template<typename _Tp >`
`_Tp __znorm2 (_Tp __x)`
- `template<typename _Tp = double>`
`_Tp evenzeta (unsigned int __k)`

Variables

- `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 _Factorial_table< long double > _S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< double > = 301`

- `template<>`
`constexpr std::size_t _S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t _S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t _S_num_factorials< long double > = 171`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< long double > = 999`
- `constexpr size_t _S_num_zetam1 = 33`
- `constexpr long double _S_zetam1 [_S_num_zetam1]`

9.3.1 Enumeration Type Documentation

9.3.1.1 anonymous enum

Enumerator

SININT

COSINT

Definition at line 43 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

<code>__z</code>	The argument of the Airy functions.
<code>_Ai</code>	The output Airy function of the first kind.
<code>_Bi</code>	The output Airy function of the second kind.
<code>_Aip</code>	The output derivative of the Airy function of the first kind.
<code>_Bip</code>	The output derivative of the Airy function of the second kind.

Definition at line 498 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

Referenced by `__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 3872 of file sf_airy.tcc.

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 3884 of file sf_airy.tcc.

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 301 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

<code>__l</code>	The order of the associated Legendre function. $l \geq 0$.
<code>__m</code>	The order of the associated Legendre function. $m \leq l$.
<code>__x</code>	The argument of the associated Legendre function. $ x \leq 1$.

Definition at line 176 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

<code>__n</code>	the order n of the Bernoulli number.
------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1673 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_n .

Parameters

<code>__n</code>	the order n of the Bernoulli number.
------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1685 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

\leftrightarrow _n	the order n of the Bernoulli number.
-------------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1608 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

\leftrightarrow _a	The first argument of the beta function.
\leftrightarrow _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 173 of file sf_beta.tcc.

References __beta_lgamma().

Referenced by __poly_jacobi(), __gnu_cxx::jacobi(), __gnu_cxx::jacobiif(), __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

\leftrightarrow _a	The first argument of the beta function.
\leftrightarrow _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 75 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

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

Definition at line 262 of file sf_beta.tcc.

References `__beta_inc_cont_frac()`.

9.3.2.13 `template<typename _Tp> _Tp std::__detail::__beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

Parameters

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

Definition at line 193 of file sf_beta.tcc.

Referenced by __beta_inc().

9.3.2.14 `template<typename _Tp> _Tp std::__detail::__beta_lgamma (_Tp __a, _Tp __b)`

Return the beta function $B(a, b)$ using the log gamma functions.

The beta function is defined by

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

Parameters

\leftrightarrow _a	The first argument of the beta function.
\leftrightarrow _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 109 of file sf_beta.tcc.

References __log_gamma().

Referenced by __beta().

9.3.2.15 `template<typename _Tp> _Tp std::__detail::__beta_product (_Tp __a, _Tp __b)`

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

The beta function is defined by

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

Here, we employ the product form:

$$B(a, b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1 + (a+b)/k}{(1 + a/k)(1 + b/k)}$$

Parameters

\leftrightarrow _a	The first argument of the beta function.
\leftrightarrow _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 140 of file sf_beta.tcc.

9.3.2.16 `template<typename _Tp> _Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)`

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

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

Parameters

\leftrightarrow _n	The first argument of the binomial coefficient.
\leftrightarrow _k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 1888 of file sf_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`.

9.3.2.17 `template<typename _Tp> _Tp std::__detail::__bose_einstein (_Tp __s, _Tp __x)`

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

See also

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

Parameters

\leftrightarrow _s	The order $s \geq 0$.
\leftrightarrow _x	The real argument.

Returns

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

Definition at line 1402 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.18 `template<typename _Tp> _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`

Return a Chebyshev polynomial of non-negative order n and real argument x by the recursion

$$C_n(x) = 2xC_{n-1} - C_{n-2}$$

Template Parameters

_Tp	The real type of the argument
-----	-------------------------------

Parameters

\leftrightarrow _n	The non-negative integral order
\leftrightarrow _x	The real argument $-1 \leq x \leq +1$
_C0	The value of the zeroth-order Chebyshev polynomial at x
_C1	The value of the first-order Chebyshev polynomial at x

Definition at line 57 of file sf_chebyshev.tcc.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

9.3.2.19 `template<typename _Tp> _Tp std::__detail::__chebyshev_t (unsigned int __n, _Tp __x)`

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

The Chebyshev polynomial of the first kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 85 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.20 `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

<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 114 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.21 `template<typename _Tp> _Tp std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)`

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

The Chebyshev polynomial of the third kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 144 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.22 `template<typename _Tp> _Tp std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)`

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 174 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.23 `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 164 of file sf_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

9.3.2.24 `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 51 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.25 `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 94 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.26 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi (std::complex<_Tp> __w)`

Definition at line 137 of file sf_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↔real_pos()`.

9.3.2.27 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi (std::complex<_Tp> __w)`

Definition at line 124 of file sf_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↔real_pos()`.

9.3.2.28 `template<typename _Tp> std::complex<_Tp> std::__detail::__clausen (unsigned int __m, std::complex<_Tp> __w)`

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

Parameters

\leftrightarrow _m	The non-negative integral order.
\leftrightarrow _w	The complex argument.

Returns

The complex Clausen function.

Definition at line 1231 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.29 `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 1255 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.30 `template<typename _Tp> _Tp std::__detail::__clausen_c (unsigned int __m, std::complex<_Tp> __w)`

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

See also

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

Parameters

\leftrightarrow _m	The integer order $m \geq 1$.
\leftrightarrow _w	The real argument.

Returns

The Clausen cosine sum $Cl_m(w)$,

Definition at line 1330 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.31 `template<typename _Tp> _Tp std::__detail::__clausen_c(unsigned int __m, _Tp __w)`

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

See also

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

Parameters

\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 1355 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.32 `template<typename _Tp> _Tp std::__detail::__clausen_s(unsigned int __m, std::complex<_Tp> __w)`

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

See also

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

Parameters

\leftrightarrow _m	The integer order $m \geq 1$.
\leftrightarrow _w	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1280 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.33 `template<typename _Tp> _Tp std::__detail::__clausen_s (unsigned int _m, _Tp _w)`

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

See also

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

Parameters

\leftrightarrow _m	The integer order $m \geq 1$.
\leftrightarrow _w	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1305 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.34 `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 566 of file sf_ellint.tcc.

References `__comp_ellint_rf()`.

Referenced by `__ellint_1()`, `__ellnome_k()`, `__jacobi_zeta()`, `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

9.3.2.35 `template<typename _Tp> _Tp std::__detail::__comp_ellint_2(_Tp _k)`

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

The complete elliptic integral of the second kind is defined as

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

Parameters

\leftrightarrow _k	The modulus of the complete elliptic function.
-------------------------	--

Returns

The complete elliptic function of the second kind.

Definition at line 639 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

9.3.2.36 `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 728 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

9.3.2.37 `template<typename _Tp> _Tp std::__detail::__comp_ellint_d (_Tp __k)`

Return the complete Legendre elliptic integral D.

Definition at line 833 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

9.3.2.38 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`

Definition at line 236 of file `sf_ellint.tcc`.

Referenced by `__comp_ellint_1()`, and `__ellint_rf()`.

9.3.2.39 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`

Definition at line 347 of file `sf_ellint.tcc`.

Referenced by `__ellint_rg()`.

9.3.2.40 `template<typename _Tp> _Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`

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

Parameters

<code>__↔ _a</code>	The <i>numerator</i> parameter.
<code>__↔ _c</code>	The <i>denominator</i> parameter.
<code>__↔ _x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 281 of file sf_hyperg.tcc.

References `__conf_hyperg_luke()`, and `__conf_hyperg_series()`.

9.3.2.41 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)`

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

Parameters

<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.

Returns

The confluent limit hypergeometric function.

Definition at line 109 of file sf_hyperg.tcc.

References `__conf_hyperg_lim_series()`.

9.3.2.42 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)`

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

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

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

Parameters

<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.

Returns

The confluent hypergeometric limit function.

Definition at line 76 of file sf_hyperg.tcc.

Referenced by __conf_hyperg_lim().

9.3.2.43 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`

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

Like the case of the $2F_1$ rational approximations, these are probably guaranteed to converge for $x < 0$, barring gross numerical instability in the pre-asymptotic regime.

Definition at line 176 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

9.3.2.44 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`

This routine returns the confluent hypergeometric function by series expansion.

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

Parameters

<code>__a</code>	The "numerator" parameter.
<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 141 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

9.3.2.45 `template<typename _Tp> _Tp std::__detail::__coshint (const _Tp __x)`

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

The hyperbolic cosine integral is given by

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

Parameters

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

Returns

The hyperbolic cosine integral.

Definition at line 554 of file `sf_expint.tcc`.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.2.46 `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 1178 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.2.47 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_i (_Tp __nu, _Tp __x)`

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

The regular modified cylindrical Bessel function is:

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

Parameters

<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 387 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

9.3.2.48 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`

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

The modified cylindrical Bessel function is:

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

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

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

Parameters

<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 414 of file `sf_bessel.tcc`.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

9.3.2.49 `template<typename _Tp > void std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

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

Parameters

<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 317 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()`.

9.3.2.50 `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 ν : $I_\nu(x)$, $N_\nu(x)$. 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 82 of file `sf_mod_bessel.tcc`.

Referenced by `__cyl_bessel_ik()`, and `__cyl_bessel_ik_steel()`.

9.3.2.51 `template<typename _Tp> void std::__detail::__cyl_bessel_ik_steel (_Tp __nu, _Tp __x, _Tp & _Inu, _Tp & _Knu, _Tp & _Ipnu, _Tp & _Kpnu)`

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

Parameters

<code>__nu</code>	The order of the Bessel functions.
-------------------	------------------------------------

Parameters

<code>__x</code>	The argument of the Bessel functions.
<code>_Inu</code>	The output regular modified Bessel function.
<code>_Knu</code>	The output irregular modified Bessel function.
<code>_Ipnu</code>	The output derivative of the regular modified Bessel function.
<code>_Kpnu</code>	The output derivative of the irregular modified Bessel function.

Definition at line 153 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

9.3.2.52 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_j(_Tp __nu, _Tp __x)`

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

The cylindrical Bessel function is:

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

Parameters

<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 534 of file `sf_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

9.3.2.53 `template<typename _Tp> void std::__detail::__cyl_bessel_jn(_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`

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

Definition at line 453 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__cyl_bessel_jn_stepped()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

9.3.2.54 `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 ν : $J_\nu(x)$, $N_\nu(x)$. 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 80 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_stepped()`.

9.3.2.55 `template<typename _Tp > void std::__detail::__cyl_bessel_jn_stepped (_Tp __nu, _Tp __x, _Tp & _Jnu, _Tp & _Nnu, _Tp & _Jpnu, _Tp & _Npnu)`

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

Parameters

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

Definition at line 198 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

9.3.2.56 `template<typename _Tp > _Tp std::__detail::__cyl_bessel_k (_Tp __nu, _Tp __x)`

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

The irregular modified Bessel function is defined by:

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

where for integral $\nu = n$ a limit is taken: $\lim_{\nu \rightarrow n}$. 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 425 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

9.3.2.57 `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 599 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

9.3.2.58 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Return the complex cylindrical Hankel function of the first kind.

Parameters

in	<code>__nu</code>	The order for which the cylindrical Hankel function of the first kind is evaluated.
in	<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 1146 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.2.59 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)`

Return the cylindrical Hankel function of the second kind $H_n^{(2)}u(x)$.

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

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

Parameters

<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 634 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

9.3.2.60 `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 1162 of file sf_hankel.tcc.

References `__hankel()`.

9.3.2.61 `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 1194 of file sf_hankel.tcc.

References `__hankel()`.

9.3.2.62 `template<typename _Tp > _Tp std::__detail::__cyl_neumann_n (_Tp __nu, _Tp __x)`

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

The Neumann function is defined by:

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

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

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 569 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

9.3.2.63 `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 $-inf < x < inf$.
------------------	---------------------------------

Definition at line 233 of file `sf_dawson.tcc`.

References `__dawson_cont_frac()`, and `__dawson_series()`.

9.3.2.64 `template<typename _Tp> _Tp std::__detail::__dawson_cont_frac (_Tp __x)`

Compute the Dawson integral using a sampling theorem representation.

Todo this needs some compile-time construction!

Definition at line 71 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

9.3.2.65 `template<typename _Tp> _Tp std::__detail::__dawson_series (_Tp __x)`

Compute the Dawson integral using the series expansion.

Definition at line 47 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

9.3.2.66 `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 56 of file `sf_hankel.tcc`.

Referenced by `__hankel()`.

9.3.2.67 `template<typename _Tp> _Tp std::__detail::__dilog (_Tp __x)`

Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.

The Riemann zeta function is defined by:

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

For $|x|$ near 1 use the reflection formulae:

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

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

For $x < 1$ use the reflection formula:

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

Definition at line 194 of file `sf_zeta.tcc`.

9.3.2.68 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta (std::complex<_Tp> __w)`

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

Parameters

<code>__w</code>	The complex (but on-real-axis) argument.
------------------	--

Returns

The Dirichlet Beta function of real argument.

Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1193 of file `sf_polylog.tcc`.

References `__fpequal()`, and `__polylog()`.

9.3.2.69 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta (_Tp __w)`

Return the Dirichlet beta function for real argument.

Parameters

\leftrightarrow	The real argument.
w	

Returns

The Dirichlet Beta function of real argument.

Definition at line 1212 of file sf_polylog.tcc.

References `__polylog()`.

9.3.2.70 `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

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

Returns

The complex Dirichlet eta function.

Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1156 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

9.3.2.71 `template<typename _Tp> _Tp std::__detail::__dirichlet_eta (_Tp __w)`

Return the Dirichlet eta function for real argument.

Parameters

\leftrightarrow	The real argument.
w	

Returns

The Dirichlet eta function.

Definition at line 1174 of file sf_polylog.tcc.

References `__polylog()`.

9.3.2.72 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`

Return the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } n-1!! = 10!! = 1$$

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

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

for $n = -2m - 1$.

Definition at line 2482 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`.

9.3.2.73 `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 595 of file sf_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rf()`.

9.3.2.74 `template<typename _Tp> _Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)`

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

The incomplete elliptic integral of the second kind is defined as

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

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the second kind.

Definition at line 674 of file `sf_ellint.tcc`.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

9.3.2.75 `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 769 of file `sf_ellint.tcc`.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.76 `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 921 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.77 `template<typename _Tp> _Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`

Return the Legendre elliptic integral D.

Definition at line 810 of file sf_ellint.tcc.

References `__ellint_rd()`.

9.3.2.78 `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 849 of file sf_ellint.tcc.

References `__ellint_rf()`.

9.3.2.79 `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 870 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

9.3.2.80 `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 895 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.81 `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 82 of file sf_ellint.tcc.

Referenced by `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.82 `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 164 of file sf_ellint.tcc.

Referenced by `__comp_ellint_2()`, `__comp_ellint_d()`, `__ellint_2()`, `__ellint_d()`, `__ellint_el2()`, `__ellint_rg()`, and `__ellint_rj()`.

9.3.2.83 `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 278 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()`.

9.3.2.84 `template<typename _Tp> _Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

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

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

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

Based on Carlson's algorithms:

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

Parameters

\leftrightarrow _x	The first of three symmetric arguments.
\leftrightarrow _y	The second of three symmetric arguments.
\leftrightarrow _z	The third of three symmetric arguments.

Returns

The Carlson symmetric elliptic function of the second kind.

Definition at line 409 of file sf_ellint.tcc.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

9.3.2.85 `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

\leftrightarrow _x	The first of three symmetric arguments.
\leftrightarrow _y	The second of three symmetric arguments.
\leftrightarrow _z	The third of three symmetric arguments.
\leftrightarrow _p	The fourth argument.

Returns

The Carlson elliptic function of the fourth kind.

Definition at line 457 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()`.

9.3.2.86 `template<typename _Tp> _Tp std::__detail::__ellnome (_Tp __k)`

Return the elliptic nome given the modulus k .

Definition at line 292 of file sf_theta.tcc.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

9.3.2.87 `template<typename _Tp> _Tp std::__detail::__ellnome_k (_Tp __k)`

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

Definition at line 278 of file sf_theta.tcc.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

9.3.2.88 `template<typename _Tp> _Tp std::__detail::__ellnome_series (_Tp __k)`

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

Definition at line 262 of file sf_theta.tcc.

Referenced by `__ellnome()`.

9.3.2.89 `template<typename _Tp> _Tp std::__detail::__expint (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$.

The exponential integral is given by

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

Parameters

\leftrightarrow _n	The order of the exponential integral function.
\leftrightarrow _x	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 470 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_En_recursion()`.

Referenced by `__logint()`.

9.3.2.90 `template<typename _Tp> _Tp std::__detail::__expint (_Tp __x)`

Return the exponential integral $Ei(x)$.

The exponential integral is given by

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

Parameters

\leftrightarrow _x	The argument of the exponential integral function.
-------------------------	--

Returns

The exponential integral.

Definition at line 510 of file sf_expint.tcc.

References `__expint_Ei()`.

9.3.2.91 `template<typename _Tp> _Tp std::__detail::__expint_asymp (unsigned int __n, _Tp __x)`

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

The exponential integral is given by

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

This is something of an extension.

Parameters

\leftrightarrow _n	The order of the exponential integral function.
\leftrightarrow _x	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 403 of file sf_expint.tcc.

9.3.2.92 `template<typename _Tp> _Tp std::__detail::__expint_E1 (_Tp _x)`

Return the exponential integral $E_1(x)$.

The exponential integral is given by

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

Parameters

\leftrightarrow _x	The argument of the exponential integral function.
-------------------------	--

Returns

The exponential integral.

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

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

Definition at line 372 of file sf_expint.tcc.

References `__expint_E1_asyp()`, `__expint_E1_series()`, `__expint_Ei()`, and `__expint_En_cont_frac()`.

Referenced by `__coshint()`, `__expint()`, `__expint_Ei()`, `__expint_En_recursion()`, and `__sinhint()`.

9.3.2.93 `template<typename _Tp> _Tp std::__detail::__expint_E1_asyp (_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.
<code>_X</code>	

Returns

The exponential integral.

Definition at line 111 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.94 `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.
<code>_X</code>	

Returns

The exponential integral.

Definition at line 74 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.95 `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 348 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asymp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

9.3.2.96 `template<typename _Tp> _Tp std::__detail::__expint_Ei_asymp (_Tp x)`

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

The exponential integral is given by

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

Parameters

\leftrightarrow	The argument of the exponential integral function.
x	

Returns

The exponential integral.

Definition at line 315 of file sf_expint.tcc.

Referenced by `__expint_Ei()`.

9.3.2.97 `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

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

Returns

The exponential integral.

Definition at line 283 of file sf_expint.tcc.

Referenced by `__expint_Ei()`.

9.3.2.98 `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

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

Returns

The exponential integral.

Definition at line 193 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.99 `template<typename _Tp> _Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`

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

The exponential integral is given by

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

Parameters

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

Returns

The exponential integral.

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

Definition at line 238 of file sf_expint.tcc.

References `__expint_E1()`.

Referenced by `__expint()`.

9.3.2.100 `template<typename _Tp > _Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`

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

The exponential integral is given by

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

Parameters

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

Returns

The exponential integral.

Definition at line 147 of file sf_expint.tcc.

References `__psi()`.

9.3.2.101 `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

\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 437 of file sf_expint.tcc.

9.3.2.102 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n.

The factorial is:

$$n! = 12\dots(n-1)n, 0! = 1$$

Definition at line 2424 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__n`, and `_S_factorial_table`.

9.3.2.103 `template<typename _Tp > _Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)`

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

See also

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

Parameters

\leftrightarrow _s	The order $s \geq 0$.
\leftrightarrow _x	The real argument.

Returns

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

Definition at line 1380 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.104 `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 581 of file sf_mod_bessel.tcc.

9.3.2.105 `template<typename _Tp> bool std::__detail::__fpequal (const _Tp & __a, const _Tp & __b)`

A function to reliably compare two floating point numbers.

Parameters

<code>__↔ __a</code>	the left hand side.
<code>__↔ __b</code>	the right hand side

Returns

returns true if a and b are equal to zero or differ only by $\max(a, b) * 5 * \epsilon$

Definition at line 63 of file sf_polylog.tcc.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, `__fpimag()`, `__fpreal()`, `__polylog()`, `__polylog_exp_asymp()`, `__↔polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_neg_even()`, `__polylog_exp_↔neg_odd()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__polylog_exp_real_pos()`.

9.3.2.106 `template<typename _Tp> bool std::__detail::__fpimag (const std::complex<_Tp> & __w)`

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

Parameters

$_w$	The complex argument.
-------	-----------------------

Returns

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

Definition at line 108 of file `sf_polylog.tcc`.

References `__fpequal()`.

9.3.2.107 `template<typename _Tp> bool std::__detail::__fpimag (const _Tp)`

Definition at line 118 of file `sf_polylog.tcc`.

9.3.2.108 `template<typename _Tp> bool std::__detail::__fpreal (const std::complex<_Tp> & __w)`

A function to reliably test a complex number for realness.

Parameters

$_w$	The complex argument.
-------	-----------------------

Returns

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

Definition at line 85 of file `sf_polylog.tcc`.

References `__fpequal()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

9.3.2.109 `template<typename _Tp> bool std::__detail::__fpreal (const _Tp)`

Definition at line 95 of file `sf_polylog.tcc`.

9.3.2.110 `template<typename _Tp> std::complex<_Tp> std::__detail::__fresnel (const _Tp __x)`

Return the Fresnel cosine and sine integrals as a complex number $C(x) + iS(x)$.

The Fresnel cosine integral is defined by:

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

The Fresnel sine integral is defined by:

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

Parameters

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

Definition at line 168 of file `sf_fresnel.tcc`.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

9.3.2.111 `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 107 of file `sf_fresnel.tcc`.

Referenced by `__fresnel()`.

9.3.2.112 `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 49 of file `sf_fresnel.tcc`.

Referenced by `__fresnel()`.

9.3.2.113 `template<typename _Tp> _Tp std::__detail::__gamma (_Tp __x)`

Return $\Gamma(x)$.

Parameters

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

Returns

The gamma function.

Definition at line 1918 of file sf_gamma.tcc.

References `__log_gamma()`.

Referenced by `__beta_gamma()`, and `__riemann_zeta()`.

9.3.2.114 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

Definition at line 1965 of file sf_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma_l()`, `__gamma_u()`, `__pgamma()`, and `__qgamma()`.

9.3.2.115 `template<typename _Tp> _Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)`

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

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

.

Definition at line 2072 of file sf_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.116 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp __a, _Tp __x)`

Definition at line 1930 of file sf_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma_l()`, `__gamma_u()`, `__pgamma()`, and `__qgamma()`.

9.3.2.117 `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 ν . 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 164 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

9.3.2.118 `template<typename _Tp> _Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)`

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

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2104 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.119 `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`.

9.3.2.120 `template<typename _Tp> _Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`

Return the Gegenbauer polynomial $C_n^\alpha(x)$ of degree n and real order α and argument x .

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^\alpha(x) = \frac{1}{n} [2x(n + \alpha - 1)C_{n-1}^\alpha(x) - (n + 2\alpha - 2)C_{n-2}^\alpha(x)]$$

and $C_0^\alpha(x) = 1$, $C_1^\alpha(x) = 2\alpha x$.

Template Parameters

<code>_Talpha</code>	The real type of the order
<code>_Tp</code>	The real type of the argument

Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__x</code>	The real argument

Definition at line 61 of file `sf_gegenbauer.tcc`.

```
9.3.2.121  template<typename _Tp> void std::__detail::__hankel ( std::complex< _Tp> __nu, std::complex< _Tp> __z,
std::complex< _Tp> & _H1, std::complex< _Tp> & _H2, std::complex< _Tp> & _H1p, std::complex< _Tp> & _H2p
)
```

Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 1083 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()`.

```
9.3.2.122  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 915 of file sf_hankel.tcc.

Referenced by `__hankel()`.

```
9.3.2.123 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 112 of file sf_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

```
9.3.2.124 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 860 of file sf_hankel.tcc.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

```
9.3.2.125 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 774 of file `sf_hankel.tcc`.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

```
9.3.2.126 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()`.

```
9.3.2.127 template<typename _Tp> void std::__detail::__hankel_uniform_sum ( std::complex<_Tp> __p, std::complex<_Tp> __p2, std::complex<_Tp> __num2, std::complex<_Tp> __zetam3hf, std::complex<_Tp> __Aip, std::complex<_Tp> __o4dp, std::complex<_Tp> __Aim, std::complex<_Tp> __o4dm, std::complex<_Tp> __od2p, std::complex<_Tp> __od0dp, std::complex<_Tp> __od2m, std::complex<_Tp> __od0dm, _Tp __eps, std::complex<_Tp> & __H1sum, std::complex<_Tp> & __H1psum, std::complex<_Tp> & __H2sum, std::complex<_Tp> & __H2psum )
```

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `nterms` (less than 5) to achieve relative error `eps`.

Parameters

in	<code>__p</code>	
in	<code>__p2</code>	
in	<code>__num2</code>	

Parameters

in	<code>__zetam3hf</code>	
in	<code>_Aip</code>	The Airy function value $Ai()$.
in	<code>__o4dp</code>	
in	<code>_Aim</code>	The Airy function value $Ai()$.
in	<code>__o4dm</code>	
in	<code>__od2p</code>	
in	<code>__od0dp</code>	
in	<code>__od2m</code>	
in	<code>__od0dm</code>	
in	<code>__eps</code>	The error tolerance
out	<code>_H1sum</code>	The Hankel function of the first kind.
out	<code>_H1psum</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2sum</code>	The Hankel function of the second kind.
out	<code>_H2psum</code>	The derivative of the Hankel function of the second kind.

Definition at line 325 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_olver()`.

9.3.2.128 `template<typename _Tp> _Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`

Return the Heuman lambda function.

Definition at line 942 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.129 `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 703 of file sf_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`.

9.3.2.130 `template<typename _Tp> std::complex<_Tp> std::__detail::__hurwitz_zeta (_Tp __s, std::complex<_Tp> __a)`

Return the Hurwitz Zeta function for real s and complex a .

Parameters

<code>__s</code>	The real argument
<code>__a</code>	The complex parameter

Todo This `__hurwitz_zeta` prefactor is prone to overflow. positive integer orders s ?

Definition at line 1120 of file sf_polylog.tcc.

References `__polylog_exp()`.

Referenced by `__psi()`.

9.3.2.131 `template<typename _Tp> _Tp std::__detail::__hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)`

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.

See also

An efficient algorithm for accelerating the convergence of oscillatory series, useful for computing the polylogarithm and Hurwitz zeta functions, Linas Vep

Parameters

<code>__s</code>	The argument $s \neq 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 561 of file sf_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

Referenced by `__hurwitz_zeta()`.

9.3.2.132 `template<typename _Tp> std::complex<_Tp> std::__detail::__hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)`

Definition at line 46 of file sf_hydrogen.tcc.

References `__assoc_laguerre()`, `__psi()`, and `__sph_legendre()`.

9.3.2.133 `template<typename _Tp> _Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Parameters

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 776 of file sf_hyperg.tcc.

References `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.2.134 `template<typename _Tp> _Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Definition at line 352 of file sf_hyperg.tcc.

Referenced by `__hyperg()`.

9.3.2.135 `template<typename _Tp> _Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

The reflection formula for nonintegral $d = c - a - b$ is:

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)\Gamma(d)}{\Gamma(c-a)\Gamma(c-b)} {}_2F_1(a, b; 1-d; 1-x) + \frac{\Gamma(c)\Gamma(-d)}{\Gamma(a)\Gamma(b)} {}_2F_1(c-a, c-b; 1+d; 1-x)$$

The reflection formula for integral $m = c - a - b$ is:

$${}_2F_1(a, b; a+b+m; x) = \frac{\Gamma(m)\Gamma(a+b+m)}{\Gamma(a+m)\Gamma(b+m)} \sum_{k=0}^{m-1} \frac{(m+a)_k(m+b)_k}{k!(1-m)_k} -$$

Definition at line 486 of file `sf_hyperg.tcc`.

References `__hyperg_series()`, `__log_gamma()`, `__log_gamma_sign()`, and `__psi()`.

Referenced by `__hyperg()`.

9.3.2.136 `template<typename _Tp> _Tp std::__detail::__hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

This works and it's pretty fast.

Parameters

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 321 of file sf_hyperg.tcc.

Referenced by __hyperg(), and __hyperg_reflect().

9.3.2.137 `template<typename _Tp> std::tuple<_Tp, _Tp, _Tp> std::__detail::__jacobi_sncndn (_Tp __k, _Tp __u)`

Return a tuple of the three primary Jacobi elliptic functions: $sn(k, u)$, $cn(k, u)$, $dn(k, u)$.

Definition at line 414 of file sf_theta.tcc.

9.3.2.138 `template<typename _Tp> _Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

Return the Jacobi zeta function.

Definition at line 972 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rj().

9.3.2.139 `template<typename _Tp> _Tp std::__detail::__laguerre (unsigned int __n, _Tp __x)`

This routine returns the Laguerre polynomial of order n: $L_n(x)$.

The Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Parameters

<code>__n</code>	The order of the Laguerre polynomial.
<code>__x</code>	The argument of the Laguerre polynomial.

Returns

The value of the Laguerre polynomial of order n and argument x.

Definition at line 321 of file sf_laguerre.tcc.

9.3.2.140 `template<typename _Tp> _Tp std::__detail::__legendre_q (unsigned int __l, _Tp __x)`

Return the Legendre function of the second kind by upward recursion on order l .

The Legendre function of the second kind of order l and argument x , $Q_l(x)$, is defined by:

$$Q_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l$$

Parameters

<code>__l</code>	The order of the Legendre function. $l \geq 0$.
<code>__x</code>	The argument of the Legendre function. $ x \leq 1$.

Definition at line 124 of file `sf_legendre.tcc`.

9.3.2.141 `template<typename _Tp> _Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)`

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Parameters

<code>__n</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Definition at line 1862 of file `sf_gamma.tcc`.

9.3.2.142 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`

Definition at line 2452 of file `sf_gamma.tcc`.

References `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.2.143 `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 2518 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`.

9.3.2.144 `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 2442 of file `sf_gamma.tcc`.

References `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

9.3.2.145 `template<typename _Tp> _Tp std::__detail::__log_gamma (_Tp __x)`

Return $\log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use `__log_gamma_sign`.

Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1800 of file `sf_gamma.tcc`.

References `__log_gamma_lanczos()`.

Referenced by `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__gamma()`, `__hyperg()`, `__hyperg_reflect()`, `__log_↵double_factorial()`, `__log_factorial()`, `__log_pochhammer_u()`, `__poly_laguerre_large_n()`, `__psi()`, `__riemann_zeta()`, `↵__riemann_zeta_glob()`, and `__sph_legendre()`.

9.3.2.146 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

Parameters

<code>↵__x</code>	The argument of the log of the gamma function.
-------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1699 of file `sf_gamma.tcc`.

9.3.2.147 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)`

Return $\log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

Parameters

<code>↵__x</code>	The argument of the log of the gamma function.
-------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1755 of file `sf_gamma.tcc`.

Referenced by `__log_gamma()`.

9.3.2.148 `template<typename _Tp > _Tp std::__detail::__log_gamma_sign (_Tp __x)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

Parameters

<code>↵__x</code>	The argument of the gamma function.
-------------------	-------------------------------------

Returns

The sign of the gamma function.

Definition at line 1831 of file sf_gamma.tcc.

Referenced by __hyperg(), __hyperg_reflect(), and __pochhammer_l().

9.3.2.149 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)`

Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

.

See also

Spouge, J.L., Computation of the gamma, digamma, and trigamma functions. SIAM Journal on Numerical Analysis 31, 3 (1994), pp. 931-944

Parameters

<code>__z</code>	The argument of the gamma function.
------------------	-------------------------------------

Returns

The the gamma function.

Definition at line 1739 of file sf_gamma.tcc.

9.3.2.150 `template<typename _Tp> _Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $\ln[(a)_n] = \ln(n!) - \ln(a)$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

Definition at line 2211 of file `sf_gamma.tcc`.

9.3.2.151 `template<typename _Tp> _Tp std::__detail::__log_pochhammer_u(_Tp __a, _Tp __n)`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(a), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 2146 of file `sf_gamma.tcc`.

References `__log_gamma()`.

9.3.2.152 `template<typename _Tp> _Tp std::__detail::__logint(const _Tp __x)`

Return the logarithmic integral $li(x)$.

The logarithmic integral is given by

$$li(x) = Ei(\log(x))$$

Parameters

$_x$	The argument of the logarithmic integral function.
-------	--

Returns

The logarithmic integral.

Definition at line 531 of file sf_expint.tcc.

References `__expint()`.

9.3.2.153 `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 <code>__h</code>	The scale parameter.
in	\leftrightarrow <code>__a</code>	The integration limit.

Returns

The owens T function.

Definition at line 92 of file sf_owens_t.tcc.

References `__znorm1()`, and `__znorm2()`.

9.3.2.154 `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 2015 of file sf_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.155 `template<typename _Tp> _Tp std::__detail::__pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $(n)_n = n!$.

Definition at line 2234 of file `sf_gamma.tcc`.

References `__log_gamma_sign()`.

9.3.2.156 `template<typename _Tp> _Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

Definition at line 2172 of file `sf_gamma.tcc`.

9.3.2.157 `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

\leftarrow <code>__n</code>	The order of the Hermite polynomial.
\leftarrow <code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 179 of file sf_hermite.tcc.

References `__poly_hermite_asymp()`, and `__poly_hermite_recursion()`.

9.3.2.158 `template<typename _Tp> _Tp std::__detail::__poly_hermite_asymp (unsigned int __n, _Tp __x)`

This routine returns the Hermite polynomial of large order n: $H_n(x)$. We assume here that $x \geq 0$.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

see "Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, arXiv:math/0601078v1 [math.CA] 4 Jan 2006

Parameters

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 113 of file sf_hermite.tcc.

References `__airy()`.

Referenced by `__poly_hermite()`.

9.3.2.159 `template<typename _Tp> _Tp std::__detail::__poly_hermite_recursion (unsigned int __n, _Tp __x)`

This routine returns the Hermite polynomial of order n: $H_n(x)$ by recursion on n.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

Parameters

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x .

Definition at line 69 of file `sf_hermite.tcc`.

Referenced by `__poly_hermite()`.

9.3.2.160 `template<typename _Tp> _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

Compute the Jacobi polynomial by recursion on n :

$$2n(\alpha+\beta+n)(\alpha+\beta+2n-2)P_n^{(\alpha,\beta)}(x) = (\alpha+\beta+2n-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2n-2)(\alpha+\beta+2n))P_{n-1}^{(\alpha,\beta)}(x) - 2(\alpha+n-1)(\beta+n-1)(\alpha+$$

Definition at line 57 of file `sf_jacobi.tcc`.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

9.3.2.161 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^{alpha}(x)$.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Template Parameters

<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 α , and argument x .

Definition at line 248 of file `sf_laguerre.tcc`.

References `__poly_laguerre_hyperg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

9.3.2.162 `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 α , and argument x .

Definition at line 129 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

9.3.2.163 `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 α , and argument x .

This is from the GNU Scientific Library.

Definition at line 72 of file `sf_laguerre.tcc`.

References `__log_gamma()`.

Referenced by `__poly_laguerre()`.

9.3.2.164 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_recursion (unsigned int __n, _Tpa __alpha1, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$ by recursion.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Template Parameters

<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 α , and argument x .

Definition at line 187 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

9.3.2.165 `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 74 of file `sf_legendre.tcc`.

Referenced by `__assoc_legendre_p()`, and `__sph_legendre()`.

9.3.2.166 `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 ρ .

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k! (\frac{n+m}{2} - k)! (\frac{n-m}{2} - k)!} \rho^{n-2k}$$

for $n - m$ even and identically 0 for $n - m$ odd. The radial polynomials can be related to the Zernike polynomials:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative m, n .

See also

[zernike](#) for details on the Zernike polynomials.

Template Parameters

<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 139 of file `sf_jacobi.tcc`.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyl()`.

9.3.2.167 `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 1073 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog_exp()`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, and `__polylog()`.

9.3.2.168 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog (_Tp __s, std::complex<_Tp> __w)`

Return the polylog in those cases where we can calculate it.

Parameters

<code>__s</code>	The real index.
<code>__w</code>	The complex argument.

Returns

The complex value of the polylogarithm.

Definition at line 1103 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog()`, and `__polylog_exp()`.

9.3.2.169 `template<typename _Tp, typename ArgType> __gnu_cxx::__promote_num_t<std::complex<_Tp>, ArgType>
std::__detail::__polylog_exp (_Tp __s, ArgType __w)`

This is the frontend function which calculates $Li_s(e^w)$. First we branch into different parts depending on the properties of s . This function is the same irrespective of a real or complex w , hence the template parameter `ArgType`.

Note

: I *really* wish we could return a `variant<Tp, std::complex<Tp>>`.

Parameters

<code>__s</code>	The real order.
<code>__w</code>	The real or complex argument.

Returns

The real or complex value of $Li_s(e^w)$.

Definition at line 1040 of file sf_polylog.tcc.

References `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_↵_real_neg()`, and `__polylog_exp_real_pos()`.

Referenced by `__bose_einstein()`, `__clausen()`, `__clausen_c()`, `__clausen_s()`, `__fermi_dirac()`, `__hurwitz_zeta()`, and `__polylog()`.

9.3.2.170 `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 $Re(w) >> 1$

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 687 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_↵_real_pos()`.

9.3.2.171 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg (int __s, std::complex<_Tp> ↵ __w)`

This treats the case where s is a negative integer.

Parameters

$_s$	a negative integer.
$_w$	an arbitrary complex number

Returns

the value of the polylogarithm.

Definition at line 857 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

Referenced by `__polylog_exp()`.

9.3.2.172 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)`

This treats the case where `s` is a negative integer and `w` is a real.

Parameters

$_s$	a negative integer.
$_w$	the argument.

Returns

the value of the polylogarithm.

Definition at line 899 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

9.3.2.173 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex<_Tp> __w)`

Here `s` is a positive integer and the function descends into the different kernels depending on `w`.

Parameters

$_s$	a positive integer.
$_w$	an arbitrary complex number.

Returns

The value of the polylogarithm.

Definition at line 768 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_↵negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

9.3.2.174 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos (unsigned int __s, _Tp __w)`

Here s is a positive integer and the function descends into the different kernels depending on w.

Parameters

<code>__↵ __s</code>	a positive integer
<code>__↵ __w</code>	an arbitrary real argument w

Returns

the value of the polylogarithm.

Definition at line 816 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__↵riemann_zeta()`.

9.3.2.175 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg (_Tp __s, std::complex<_Tp> __w)`

This function treats the cases of negative real index s. Theoretical convergence is present for $|w| < 2\pi$. We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^{(s-1)} + (2\pi)^{(s-1)}/\pi A_p(w)$$

$$A_p(w) = \sum_k \Gamma(1+k-s)/k! \sin(\pi/2 * (s-k))(w/2/\pi)^k \zeta(1+k-s)$$

Parameters

<code>__↵ __s</code>	The real index
<code>__↵ __w</code>	The complex argument

Returns

The value of the polylogarithm.

Definition at line 347 of file sf_polylog.tcc.

References `__fpequal()`, `__riemann_zeta()`, and `__riemann_zeta_m_1()`.

Referenced by `__polylog_exp_int_neg()`, and `__polylog_exp_real_neg()`.

9.3.2.176 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg (int __s, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s and branches accordingly

Parameters

<code>__s</code>	the integer index s.
<code>__w</code>	The Argument w

Returns

The value of the Polylogarithm evaluated by a suitable function.

Definition at line 565 of file sf_polylog.tcc.

References `__polylog_exp_neg_even()`, and `__polylog_exp_neg_odd()`.

9.3.2.177 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are multiples of two.

In that case the sine occuring in the expansion occasionally takes on the value zero. We use that to provide an optimized series for $p = 2n$:

In the template parameter sigma we transport whether $p = 4k$ ($\sigma = 1$) or $p = 4k + 2$ ($\sigma = -1$)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}(-p)!/(2\pi)^{(p-1)/2}(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi/w))$$

and

$$B_p(w) = -2(2\pi)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} (\zeta(2+2k-p) - 1)$$

This is suitable for $|w| < 2\pi$ The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - \sigma(2\pi)^p/\pi \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} \zeta(2+2k-p)$$

Parameters

\leftrightarrow _n	the integral index $n = 4k$.
\leftrightarrow _w	The complex argument w

Returns

the value of the Polylogarithm.

Definition at line 451 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

9.3.2.178 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are odd.

In that case the sine occuring in the expansion occasionally vanishes. We use that to provide an optimized series for $p = 1 + 2k$: In the template parameter sigma we transport whether $p = 1 + 4k$ ($\sigma = 1$) or $p = 3 + 4k$ ($\sigma = -1$)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} + \sigma * A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}\Gamma(1-p)(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi i/w))$$

and

$$B_p(w) = 2(2\pi i)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-w^2/4/\pi^2)^k (\zeta(1+2k-p) - T p1)$$

This is suitable for $|w| < 2\pi$. The use of evenzeta gives a speedup of about 50 The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p) * (-w)^{p-1} - \sigma 2(2\pi)^{(p-1)} * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-1)^k (w/2/\pi)^{(2k)} \zeta(1+2k-p)$$

Parameters

\leftrightarrow _n	the integral index n = 4k.
\leftrightarrow _w	The complex argument w.

Returns

The value of the Polylogarithm.

Definition at line 518 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

9.3.2.179 `template<typename _PowTp, typename _Tp> _Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`

Theoretical convergence for $\text{Re}(w) < 0$.

Seems to beat the other expansions for $\text{Re}(w) < -\pi/2 - \pi/5$. Note that this is an implementation of the basic series:

$$Li_s(e^z) = \sum_{k=1} e^{(k * z) * k^{(s-1)}}$$

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 738 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

9.3.2.180 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, std::complex<_Tp> __w)`

This function treats the cases of positive integer index s .

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{(s-1)} / (s-1)!$$

The radius of convergence is $|w| < 2\pi i$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{i\pi}) = +i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{i\pi}) = -i\pi$

Parameters

\leftarrow _s	the index s.
\leftarrow _w	the argument w.

Returns

the value of the polylogarithm.

Definition at line 207 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

9.3.2.181 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`

This function treats the cases of positive integer index s for real w.

This specialization is worthwhile to catch the differing behaviour of $\log(x)$.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{s-1} / (s-1)!$$

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. The use of `evenzeta` yields a speedup of about 2.5. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{i\pi}) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{i\pi}) = + i\pi$

Parameters

\leftarrow _s	the index.
\leftarrow _w	the argument

Returns

the value of the Polylogarithm

Definition at line 280 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

9.3.2.182 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (_Tp __s, std::complex<_Tp> __w)`

This function treats the cases of positive real index s.

The defining series is

$$Li_s(e^w) = A_s(w) + B_s(w) + \Gamma(1-s)(-w)^{(s-1)}$$

with

$$A_s(w) = \sum_{k=0}^m \zeta(s-k) w^k / k!$$

$$B_s(w) = \sum_{k=m+1}^{\infty} \sin(\pi/2(s-k)) \Gamma(1-s+k) \zeta(1-s+k) (w/2/\pi)^k / k!$$

Parameters

<code>__s</code>	the positive real index s.
<code>__w</code>	The complex argument w.

Returns

the value of the polylogarithm.

Definition at line 604 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

9.3.2.183 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex<_Tp> __w)`

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

Parameters

<code>__s</code>	A negative real value that does not reduce to a negative integer.
<code>__w</code>	The complex argument.

Returns

The value of the polylogarithm.

Definition at line 986 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()`.

9.3.2.184 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)`

Return the polylog where `s` is a negative real value and for real argument. Now we branch depending on the properties of `w` in the specific functions.

Parameters

<code>__↔ _s</code>	A negative real value.
<code>__↔ _w</code>	A real argument.

Returns

The value of the polylogarithm.

Definition at line 1014 of file `sf_polylog.tcc`.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

9.3.2.185 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex<_Tp> __w)`

Return the polylog where `s` is a positive real value and for complex argument.

Parameters

<code>__↔ _s</code>	A positive real number.
<code>__↔ _w</code>	the complex argument.

Returns

The value of the polylogarithm.

Definition at line 923 of file `sf_polylog.tcc`.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_↔negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

9.3.2.186 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)`

Return the polylog where s is a positive real value and the argument is real.

Parameters

<code>__s</code>	A positive real number tht does not reduce to an integer.
<code>__w</code>	The real argument w.

Returns

The value of the polylogarithm.

Definition at line 957 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

9.3.2.187 `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 2332 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, and `__psi_asymp()`.

Referenced by `__expint_En_series()`, `__hydrogen()`, `__hyperg_reflect()`, and `__psi()`.

9.3.2.188 `template<typename _Tp> _Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(n)}(x) = (-1)^{n+1} n! \zeta(m+1, x)$$

Definition at line 2397 of file sf_gamma.tcc.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

9.3.2.189 `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 2301 of file `sf_gamma.tcc`.

Referenced by `__psi()`.

9.3.2.190 `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 2270 of file `sf_gamma.tcc`.

9.3.2.191 `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 2046 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.192 `template<typename _Tp> _Tp std::__detail::__riemann_zeta (_Tp __s)`

Return the Riemann zeta function $\zeta(s)$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } s > 1 \quad \frac{(2\pi)^s}{\pi} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } s < 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Parameters

\leftrightarrow	The argument
$_s$	

Definition at line 506 of file sf_zeta.tcc.

References `__gamma()`, `__log_gamma()`, `__riemann_zeta_glob()`, `__riemann_zeta_product()`, and `__riemann_zeta_ \leftrightarrow sum()`.

Referenced by `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__polylog_exp_real_pos()`, and `evenzeta()`.

9.3.2.193 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_alt (_Tp __s)`

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 330 of file sf_zeta.tcc.

9.3.2.194 `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 283 of file sf_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

9.3.2.195 `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 375 of file sf_zeta.tcc.

References `__log_gamma()`.

Referenced by `__riemann_zeta()`.

9.3.2.196 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$.

Parameters

\leftarrow _s	The argument $s! = 1$
--------------------	-----------------------

Definition at line 673 of file sf_zeta.tcc.

References `__riemann_zeta_m_1_sum()`, `_S_num_zetam1`, and `_S_zetam1`.

Referenced by `__polylog_exp_neg()`.

9.3.2.197 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$ by summation for $s > 1$. This is a small remainder for large s .

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

Parameters

\leftarrow _s	The argument $s! = 1$
--------------------	-----------------------

Definition at line 646 of file sf_zeta.tcc.

Referenced by `__riemann_zeta_m_1()`.

9.3.2.198 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_product (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

$$\zeta(s) = \prod_{i=1}^{\infty} \frac{1}{1 - p_i^{-s}}$$

where p_i are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Parameters

\leftrightarrow	The argument
<code>_s</code>	

Definition at line 464 of file sf_zeta.tcc.

Referenced by `__riemann_zeta()`.

9.3.2.199 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_sum (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 254 of file sf_zeta.tcc.

Referenced by `__riemann_zeta()`.

9.3.2.200 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp __a, _Tp __x)`

Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

.

Definition at line 51 of file sf_cardinal.tcc.

9.3.2.201 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp __x)`

Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

.

Definition at line 98 of file sf_cardinal.tcc.

9.3.2.202 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc_pi (_Tp __x)`

Return the unnormalized sinus cardinal function

$$\text{sinc}_{\pi}(x) = \frac{\sin(x)}{x}$$

.

Definition at line 78 of file `sf_cardinal.tcc`.

9.3.2.203 `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 229 of file `sf_trigint.tcc`.

References `__sincosint_asyp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

9.3.2.204 `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 164 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

9.3.2.205 `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 56 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

9.3.2.206 `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 99 of file sf_trigint.tcc.

Referenced by `__sincosint()`.

9.3.2.207 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __a, _Tp __x)`

Return the generalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

.

Definition at line 124 of file sf_cardinal.tcc.

9.3.2.208 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __x)`

Return the normalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

.

Definition at line 167 of file sf_cardinal.tcc.

9.3.2.209 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc_pi (_Tp __x)`

Return the unnormalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_\pi(x) = \frac{\sinh(x)}{x}$$

.

Definition at line 149 of file sf_cardinal.tcc.

9.3.2.210 `template<typename _Tp> _Tp std::__detail::__sinhint (const _Tp __x)`

Return the hyperbolic sine integral $li(x)$.

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) - E_1(x))/2$$

Parameters

\leftrightarrow _x	The argument of the hyperbolic sine integral function.
-------------------------	--

Returns

The hyperbolic sine integral.

Definition at line 577 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.2.211 `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 704 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.2.212 `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 1270 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.213 `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\leftrightarrow_n</code>	The output derivative of the regular modified spherical Bessel function.
<code>__kp\leftrightarrow_n</code>	The output derivative of the irregular modified spherical Bessel function.

Definition at line 457 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`.

9.3.2.214 `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\leftrightarrow_n</code>	The output derivative of the spherical Bessel function.
out	<code>__np\leftrightarrow_n</code>	The output derivative of the spherical Neumann function.

Definition at line 669 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
9.3.2.215  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 1214 of file sf_hankel.tcc.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
9.3.2.216  template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_1 ( unsigned int __n, _Tp __x )
```

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = j_n(x) + in_n(x)$$

Parameters

<code>__n</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 773 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.2.217 `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	\leftarrow __n	The order for which the spherical Hankel function of the first kind is evaluated.
in	\leftarrow __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 1238 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.218 `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

\leftarrow __n	The non-negative integral order
\leftarrow __x	The non-negative real argument

Returns

The output spherical Neumann function.

Definition at line 805 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.2.219 `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 1254 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.220 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`

Return the spherical harmonic function.

The spherical harmonic function of l , m , and θ , ϕ is defined by:

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^{|m|}(\cos \theta) \exp^{im\phi}$$

Parameters

__l	The order of the spherical harmonic function. $l \geq 0$.
__m	The order of the spherical harmonic function. $m \leq l$.
__theta	The radian polar angle argument of the spherical harmonic function.
__phi	The radian azimuthal angle argument of the spherical harmonic function.

Definition at line 351 of file sf_legendre.tcc.

References `__sph_legendre()`.

9.3.2.221 `template<typename _Tp> _Tp std::__detail::__sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

Return the spherical associated Legendre function.

The spherical associated Legendre function of l , m , and θ is defined as $Y_l^m(\theta, 0)$ where

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^m(\cos \theta) \exp^{im\phi}$$

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 254 of file `sf_legendre.tcc`.

References `__log_gamma()`, and `__poly_legendre_p()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

9.3.2.222 `template<typename _Tp> _Tp std::__detail::__sph_neumann (unsigned int __n, _Tp __x)`

Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .

The spherical Neumann function is defined by:

$$n_n(x) = \left(\frac{\pi}{2x} \right)^{1/2} N_{n+1/2}(x)$$

Parameters

<code>__n</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 741 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

9.3.2.223 `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 1286 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.224 `template<typename _Tp> _Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)`

Return the exponential theta-1 function of period `nu` and argument `x`.

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

__nu	The periodic (period = 2) argument
__x	The argument

Definition at line 190 of file sf_theta.tcc.

References `__theta_2()`.

Referenced by `__theta_s()`.

9.3.2.225 `template<typename _Tp> _Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 162 of file `sf_theta.tcc`.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.

9.3.2.226 `template<typename _Tp> _Tp std::__detail::__theta_2_asyp (_Tp __nu, _Tp __x)`

Compute and return the θ_2 function by series expansion.

Definition at line 103 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

9.3.2.227 `template<typename _Tp> _Tp std::__detail::__theta_2_sum (_Tp __nu, _Tp __x)`

Compute and return the θ_1 function by series expansion.

Definition at line 49 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

9.3.2.228 `template<typename _Tp> _Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)`

Return the exponential theta-3 function of period `nu` and argument `x`.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 216 of file `sf_theta.tcc`.

References `__theta_3_asymp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

9.3.2.229 `template<typename _Tp> _Tp std::__detail::__theta_3_asymp (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by asymptotic series expansion.

Definition at line 128 of file `sf_theta.tcc`.

Referenced by `__theta_3()`.

9.3.2.230 `template<typename _Tp> _Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by series expansion.

Definition at line 77 of file `sf_theta.tcc`.

Referenced by `__theta_3()`.

9.3.2.231 `template<typename _Tp> _Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 244 of file `sf_theta.tcc`.

References `__theta_3()`.

Referenced by `__theta_n()`.

9.3.2.232 `template<typename _Tp> _Tp std::__detail::__theta_c (_Tp __k, _Tp __x)`

Return the Neville θ_c function

Definition at line 337 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

9.3.2.233 `template<typename _Tp> _Tp std::__detail::__theta_d (_Tp __k, _Tp __x)`

Return the Neville θ_d function

Definition at line 362 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

9.3.2.234 `template<typename _Tp> _Tp std::__detail::__theta_n (_Tp __k, _Tp __x)`

Return the Neville θ_n function

Definition at line 387 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

9.3.2.235 `template<typename _Tp> _Tp std::__detail::__theta_s (_Tp __k, _Tp __x)`

Return the Neville θ_s function

Definition at line 311 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

9.3.2.236 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

[__poly_radial_jacobi](#)).

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate and azimuthal angle
------------------	--

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 184 of file sf_jacobi.tcc.

References `__poly_radial_jacobi()`.

9.3.2.237 `template<typename _Tp> _Tp std::__detail::__znorm1 (_Tp __x)`

Definition at line 58 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.2.238 `template<typename _Tp> _Tp std::__detail::__znorm2 (_Tp __x)`

Definition at line 47 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.2.239 `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 157 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 1427 of file sf_airy.tcc.

9.3.3.2 `template<> constexpr int std::__detail::__max_FGH< double > = 79`

Definition at line 1433 of file `sf_airy.tcc`.

9.3.3.3 `template<> constexpr int std::__detail::__max_FGH< float > = 15`

Definition at line 1430 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 65 of file `sf_zeta.tcc`.

9.3.3.5 `constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]`

Definition at line 274 of file `sf_gamma.tcc`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.3.6 `constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]`

Definition at line 68 of file `sf_zeta.tcc`.

Referenced by `__hurwitz_zeta_euler_maclaurin()`, and `__riemann_zeta_euler_maclaurin()`.

9.3.3.7 `constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]`

Definition at line 84 of file `sf_gamma.tcc`.

Referenced by `__factorial()`, and `__log_factorial()`.

9.3.3.8 `constexpr _Factorial_table<long double> std::__detail::_S_neg_double_factorial_table[999]`

Definition at line 595 of file `sf_gamma.tcc`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.3.9 `template<typename _Tp> constexpr std::size_t std::__detail::_S_num_double_factorials = 0`

Definition at line 259 of file `sf_gamma.tcc`.

9.3.3.10 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301`

Definition at line 264 of file `sf_gamma.tcc`.

9.3.3.11 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57`

Definition at line 262 of file `sf_gamma.tcc`.

9.3.3.12 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301`

Definition at line 266 of file `sf_gamma.tcc`.

9.3.3.13 `template<typename _Tp> constexpr std::size_t std::__detail::_S_num_factorials = 0`

Definition at line 69 of file `sf_gamma.tcc`.

9.3.3.14 `template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`

Definition at line 74 of file `sf_gamma.tcc`.

9.3.3.15 `template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`

Definition at line 72 of file `sf_gamma.tcc`.

9.3.3.16 `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 76 of file `sf_gamma.tcc`.

9.3.3.17 `template<typename _Tp> constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 579 of file `sf_gamma.tcc`.

9.3.3.18 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 584 of file `sf_gamma.tcc`.

9.3.3.19 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 582 of file `sf_gamma.tcc`.

9.3.3.20 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 586 of file `sf_gamma.tcc`.

9.3.3.21 `constexpr size_t std::__detail::_S_num_zetam1 = 33`

Table of $\zeta(n) - 1$ from 2 - 32. MPFR - 128 bits.

Definition at line 593 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

9.3.3.22 `constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]`

Definition at line 597 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

Chapter 10

Class Documentation

10.1 `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`

10.1.1 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 3729 of file sf_airy.tcc.

10.1.2 Member Typedef Documentation

10.1.2.1 `template<typename _Tp> using std::__detail::_Airy< _Tp >::scalar_type = std::__detail::__num_traits_t<value_type>`

Definition at line 3734 of file sf_airy.tcc.

10.1.2.2 `template<typename _Tp> using std::__detail::_Airy< _Tp >::value_type = _Tp`

Definition at line 3733 of file sf_airy.tcc.

10.1.3 Constructor & Destructor Documentation

10.1.3.1 `template<typename _Tp> constexpr std::__detail::_Airy< _Tp >::_Airy () [default]`

10.1.3.2 `template<typename _Tp> std::__detail::_Airy< _Tp >::_Airy (const _Airy< _Tp > &) [default]`

10.1.3.3 `template<typename _Tp> std::__detail::_Airy< _Tp >::_Airy (_Airy< _Tp > &&) [default]`

10.1.4 Member Function Documentation

10.1.4.1 `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 3781 of file sf_airy.tcc.

References `std::__detail::__beta()`, `std::__detail::_Airy_series< _Tp >::_S_Ai()`, and `std::__detail::_Airy_series< _Tp >::_S_Bi()`.

10.1.5 Member Data Documentation

10.1.5.1 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_2pi_3 = scalar_type{2} *
_S_pi_3 [static]`

Definition at line 3741 of file sf_airy.tcc.

10.1.5.2 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_5pi_6 = scalar_type{5} *
_S_pi_6 [static]`

Definition at line 3743 of file sf_airy.tcc.

10.1.5.3 `template<typename _Tp> constexpr auto std::__detail::_Airy<_Tp>::_S_cNaN = value_type(_S_NaN, _S_NaN)
[static]`

Definition at line 3747 of file sf_airy.tcc.

10.1.5.4 `template<typename _Tp> constexpr _Airy<_Tp>::value_type std::__detail::_Airy<_Tp>::_S_i =
value_type{0, 1} [static]`

Definition at line 3744 of file sf_airy.tcc.

10.1.5.5 `template<typename _Tp> constexpr auto std::__detail::_Airy<_Tp>::_S_NaN =
_gnu_cxx::__quiet_NaN<scalar_type>() [static]`

Definition at line 3746 of file sf_airy.tcc.

10.1.5.6 `template<typename _Tp> constexpr scalar_type std::__detail::_Airy<_Tp>::_S_pi =
_gnu_cxx::__math_constants<scalar_type>::_pi [static]`

Definition at line 3736 of file sf_airy.tcc.

10.1.5.7 `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 3740 of file sf_airy.tcc.

10.1.5.8 `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 3742 of file sf_airy.tcc.

```
10.1.5.9  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 3738 of file sf_airy.tcc.

```
10.1.5.10 template<typename _Tp> scalar_type std::__detail::_Airy<_Tp>::inner_radius
        { _Airy_default_radII<scalar_type>::inner_radius }
```

Definition at line 3756 of file sf_airy.tcc.

```
10.1.5.11 template<typename _Tp> scalar_type std::__detail::_Airy<_Tp>::outer_radius
        { _Airy_default_radII<scalar_type>::outer_radius }
```

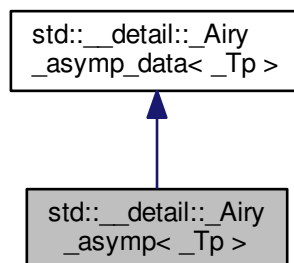
Definition at line 3757 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

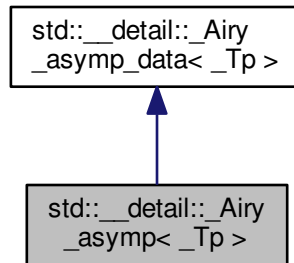
- [bits/sf_airy.tcc](#)

10.2 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 `__cplx` = 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

10.2.1 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 3231 of file sf_airy.tcc.

10.2.2 Member Typedef Documentation

10.2.2.1 `template<typename _Tp > using std::__detail::_Airy_asymp<_Tp>::__cmplx = std::complex<_Tp>`

Definition at line 3236 of file sf_airy.tcc.

10.2.3 Constructor & Destructor Documentation

10.2.3.1 `template<typename _Tp > constexpr std::__detail::_Airy_asymp<_Tp>::__Airy_asymp() [default]`

10.2.4 Member Function Documentation

10.2.4.1 `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 3503 of file sf_airy.tcc.

10.2.4.2 `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>_↔</code> <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 3533 of file sf_airy.tcc.

```
10.2.4.3 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
------------------	-------------

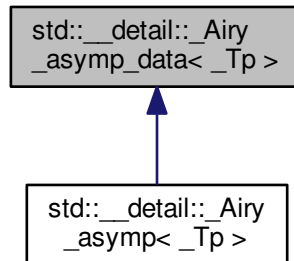
Definition at line 3262 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- [bits/sf_airy.tcc](#)

10.3 `std::__detail::_Airy_asymp_data<_Tp>` Struct Template Reference

Inheritance diagram for `std::__detail::_Airy_asymp_data<_Tp>`:



10.3.1 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 1867 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.4 `std::__detail::_Airy_asymp_data<__float128>` Struct Template Reference

Static Public Attributes

- static constexpr `__float128 _S_c` [`_S_max_cd`]
- static constexpr `__float128 _S_d` [`_S_max_cd`]
- static constexpr `int _S_max_cd` = 201

10.4.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< __float128 >
```

Definition at line 2806 of file sf_airy.tcc.

10.4.2 Member Data Documentation

10.4.2.1 `constexpr __float128 std::__detail::_Airy_asymp_data< __float128 >::_S_c[_S_max_cd]` `[static]`

Definition at line 2812 of file sf_airy.tcc.

10.4.2.2 `constexpr __float128 std::__detail::_Airy_asymp_data< __float128 >::_S_d[_S_max_cd]` `[static]`

Definition at line 3018 of file sf_airy.tcc.

10.4.2.3 `constexpr int std::__detail::_Airy_asymp_data< __float128 >::_S_max_cd = 201` `[static]`

Definition at line 2808 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 1974 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 1980 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 2183 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 1976 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 1871 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 1877 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 1925 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 1873 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 2387 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 2393 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 2599 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 2389 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 3596 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 3601 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 3600 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 3605 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 3650 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 3603 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 3700 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 3711 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 3713 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 3714 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 3704 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 3706 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 3707 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 3718 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 3720 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 3721 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-1Q}`
- static `constexpr _Tp _S_Aip0 = _Tp{-2.588194037928067984051835601892039634793e-1Q}`
- static `constexpr _Tp _S_Bi0 = _Tp{6.149266274460007351509223690936135535960e-1Q}`
- static `constexpr _Tp _S_Bip0 = _Tp{4.482883573538263579148237103988283908668e-1Q}`
- static `constexpr _Tp _S_eps = __gnu_cxx::__epsilon(_Tp{})`
- static `constexpr _Tp _S_Gi0 = _Tp{2.049755424820002450503074563645378511979e-1Q}`
- static `constexpr _Tp _S_Gip0 = _Tp{1.494294524512754526382745701329427969551e-1Q}`
- static `constexpr _Tp _S_Hi0 = _Tp{4.099510849640004901006149127290757023959e-1Q}`
- static `constexpr _Tp _S_Hip0 = _Tp{2.988589049025509052765491402658855939102e-1Q}`
- static `constexpr __cmplx _S_i { _Tp{0}, _Tp{1}}`
- static `constexpr _Tp _S_log10min = __gnu_cxx::__log10_min(_Tp{})`
- 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.

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

This class contains tabulations of the factors appearing in the sums above.

Definition at line 108 of file sf_airy.tcc.

10.13.2 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:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the first kind is then defined by:

$$Ai(x) = Ai(0)fai(x) + Ai'(0)gai(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3^{-1/2}/2\Gamma(1/3)$ 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 1590 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

\leftrightarrow	The complex argument
<code>__t</code>	
\leftrightarrow	
<code>__t</code>	
<code>t</code>	

Definition at line 1844 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 1613 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 1632 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

<code>↩</code>	The complex argument
<code>_↩</code>	
<code>↩</code>	
<code>_↩</code>	
<code>t</code>	

Definition at line 1856 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 1706 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 1743 of file sf_airy.tcc.

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-1Q} [static]`

Definition at line 1353 of file sf_airy.tcc.

10.13.3.3 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Aip0 =
_Tp{-2.588194037928067984051835601892039634793e-1Q} [static]`

Definition at line 1355 of file sf_airy.tcc.

10.13.3.4 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Bi0 =
_Tp{6.149266274460007351509223690936135535960e-1Q} [static]`

Definition at line 1357 of file sf_airy.tcc.

10.13.3.5 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Bip0 =
_Tp{4.482883573538263579148237103988283908668e-1Q} [static]`

Definition at line 1359 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 1348 of file sf_airy.tcc.

10.13.3.7 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gi0 =
_Tp{2.049755424820002450503074563645378511979e-1Q} [static]`

Definition at line 1365 of file sf_airy.tcc.

10.13.3.8 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gip0 =
_Tp{1.494294524512754526382745701329427969551e-1Q} [static]`

Definition at line 1367 of file sf_airy.tcc.

10.13.3.9 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Hi0 =
_Tp{4.099510849640004901006149127290757023959e-1Q} [static]`

Definition at line 1361 of file sf_airy.tcc.

```
10.13.3.10 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hip0 =
        _Tp{2.988589049025509052765491402658855939102e-1Q} [static]
```

Definition at line 1363 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 1368 of file sf_airy.tcc.

```
10.13.3.12 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_log10min =
        __gnu_cxx::__log10_min(_Tp{}) [static]
```

Definition at line 1369 of file sf_airy.tcc.

```
10.13.3.13 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_pi =
        __gnu_cxx::__math_constants<_Tp>::_pi [static]
```

Definition at line 1349 of file sf_airy.tcc.

```
10.13.3.14 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_sqrt_pi =
        __gnu_cxx::__math_constants<_Tp>::_root_pi [static]
```

Definition at line 1351 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<typename _Tp> _Tp std::__detail::__AiryState<_Tp>::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`
- `unsigned int __n`

10.16.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Factorial_table<_Tp>
```

Definition at line 61 of file `sf_gamma.tcc`.

10.16.2 Member Data Documentation

10.16.2.1 `template<typename _Tp> _Tp std::__detail::_Factorial_table<_Tp>::__factorial`

Definition at line 64 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__double_factorial()`.

10.16.2.2 `template<typename _Tp> _Tp std::__detail::_Factorial_table<_Tp>::__log_factorial`

Definition at line 65 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__log_double_factorial()`.

10.16.2.3 `template<typename _Tp> unsigned int std::__detail::_Factorial_table<_Tp>::__n`

Definition at line 63 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__bernoulli()`, `std::__detail::__bernoulli_2n()`, `std::__detail::__bernoulli_series()`, `std::__detail::__bincoef()`, `std::__detail::__double_factorial()`, `std::__detail::__factorial()`, `std::__detail::__gamma_cont_frac()`, `std::__detail::__gamma_series()`, `std::__detail::__log_double_factorial()`, `std::__detail::__log_factorial()`, and `std::__detail::__psi()`.

The documentation for this struct was generated from the following file:

- [bits/sf_gamma.tcc](#)

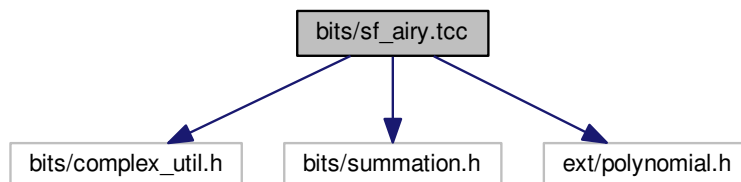
Chapter 11

File Documentation

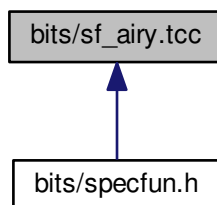
11.1 bits/sf_airy.tcc File Reference

```
#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<__float128>](#)
- 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.

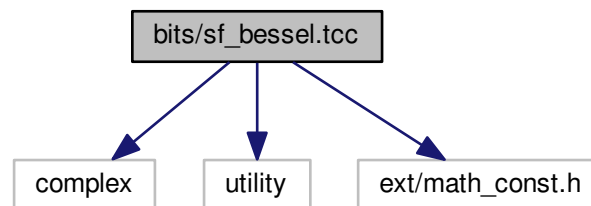
11.1.2 Macro Definition Documentation

11.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

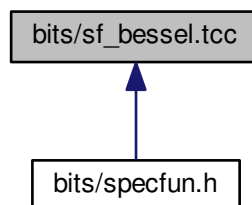
Definition at line 31 of file sf_airy.tcc.

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 ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`
Return the Bessel function of order ν : $J_\nu(x)$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg nu^2 + 1$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn_steel (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the second kind $H_\nu^{(2)}(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_neumann_n (_Tp __nu, _Tp __x)`
Return the Neumann function of order ν : $N_\nu(x)$.
- `template<typename _Tp >`
`void std::__detail::__gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`
Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

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 ν . The values of $\Gamma(1+\mu)$ and $\Gamma(1-\mu)$ are returned as well.

- `template<typename _Tp >`
`_Tp std::__detail::__sph_bessel (unsigned int __n, _Tp __x)`
Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .
- `template<typename _Tp >`
`void std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`
Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- `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

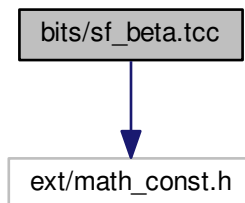
11.2.2.1 `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file `sf_bessel.tcc`.

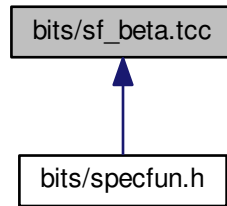
11.3 bits/sf_beta.tcc File Reference

`#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_inc_cont_frac \(_Tp __a, _Tp __b, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__beta_lgamma \(_Tp __a, _Tp __b\)](#)
Return the beta function $B(a, b)$ using the log gamma functions.
- [template<typename _Tp > _Tp std::__detail::__beta_product \(_Tp __a, _Tp __b\)](#)
Return the beta function $B(x, y)$ using the product form.

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

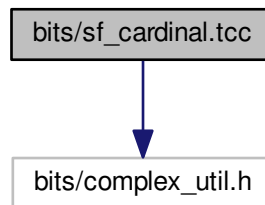
11.3.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file sf_beta.tcc.

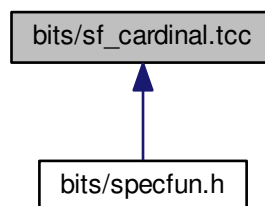
11.4 bits/sf_cardinal.tcc File Reference

```
#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_num_t< _Tp > std::__detail::__sinc (_Tp __a, _Tp __x)`
Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinc (_Tp __x)`
Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinc_pi (_Tp __x)`
Return the unnormalized sinus cardinal function

$$\text{sinc}_\pi(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __a, _Tp __x)`
Return the generalized hyperbolic sinus cardinal function

$$\text{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __x)`
Return the normalized hyperbolic sinus cardinal function

$$\text{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)`
Return the unnormalized hyperbolic sinus cardinal function

$$\text{sinhc}_\pi(x) = \frac{\sinh(x)}{x}$$

11.4.1 Macro Definition Documentation

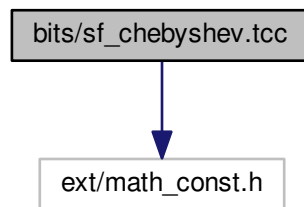
11.4.1.1 `#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1`

Definition at line 30 of file `sf_cardinal.tcc`.

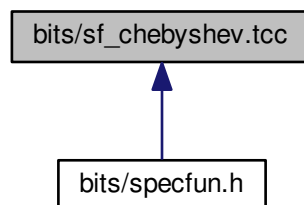
11.5 bits/sf_chebyshev.tcc File Reference

```
#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

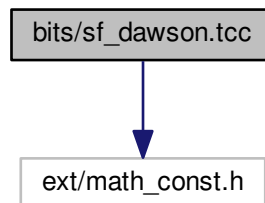
11.5.2.1 `#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1`

Definition at line 31 of file `sf_chebyshev.tcc`.

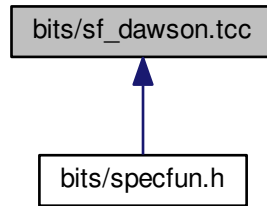
11.6 `bits/sf_dawson.tcc` File Reference

```
#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>`.

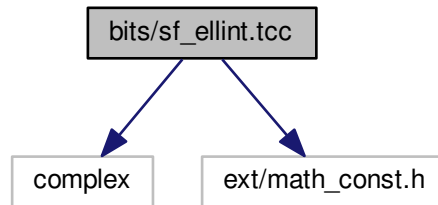
11.6.2 Macro Definition Documentation

11.6.2.1 [#define _GLIBCXX_BITS_SF_DAWSON_TCC 1](#)

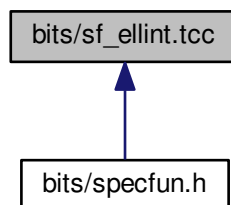
Definition at line 31 of file `sf_dawson.tcc`.

11.7 bits/sf_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)`

11.7.1 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

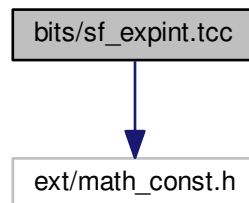
11.7.2.1 `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

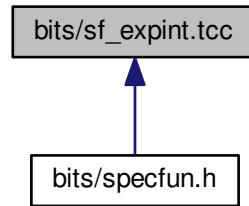
11.8 bits/sf_expint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_expint.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_EXPINT_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__coshint \(const _Tp __x\)](#)
Return the hyperbolic cosine integral $li(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint \(unsigned int __n, _Tp __x\)](#)
Return the exponential integral $E_n(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint \(_Tp __x\)](#)
Return the exponential integral $Ei(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint_asymp \(unsigned int __n, _Tp __x\)](#)
Return the exponential integral $E_n(x)$ for large argument.
- [template<typename _Tp > _Tp std::__detail::__expint_E1 \(_Tp __x\)](#)
Return the exponential integral $E_1(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint_E1_asymp \(_Tp __x\)](#)
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- [template<typename _Tp > _Tp std::__detail::__expint_E1_series \(_Tp __x\)](#)

Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei (_Tp __x)`

Return the exponential integral $Ei(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_asymp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_series (_Tp __x)`

Return the exponential integral $Ei(x)$ by series summation.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by series summation.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_large_n (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large order.

- `template<typename _Tp >`
`_Tp std::__detail::__logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__sinhint (const _Tp __x)`

Return the hyperbolic sine integral $li(x)$.

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>`.

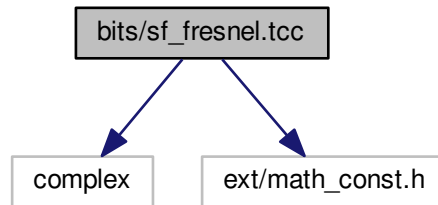
11.8.2 Macro Definition Documentation

11.8.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

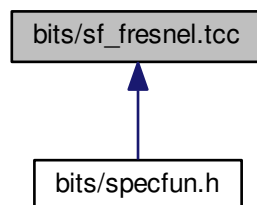
Definition at line 47 of file `sf_expint.tcc`.

11.9 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 $Sf[C(x) + iS(x)]$.
- `template<typename _Tp >`
`void std::__detail::__fresnel_cont_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`
This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- `template<typename _Tp >`
`void std::__detail::__fresnel_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`
This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

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

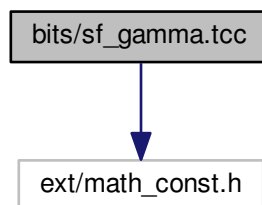
11.9.2.1 `#define _GLIBCXX_BITS_SF_FRESNEL_TCC 1`

Definition at line 31 of file `sf_fresnel.tcc`.

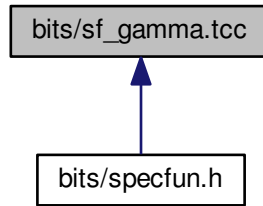
11.10 `bits/sf_gamma.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_gamma.tcc`:



This graph shows which files directly or indirectly include this file:



Classes

- struct [std::__detail::_Factorial_table< _Tp >](#)

Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Functions

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`
This returns Bernoulli numbers from a table or by summation for larger values.
- `template<typename _Tp >`
`_Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)`
Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`
Return the double factorial of the integer n.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`
Return the factorial of the integer n.
- `template<typename _Tp >`
`_Tp std::__detail::__gamma (_Tp __x)`
Return $\Gamma(x)$.
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)`
Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__gamma_series (_Tp __a, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)`
Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)`
Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`
Return the logarithm of the double factorial of the integer n.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`
Return the logarithm of the factorial of the integer n.
- `template<typename _Tp >`
`_Tp std::__detail::__log_gamma (_Tp __x)`
Return $\log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use `__log_gamma_sign`.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`
Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)`
Return $\log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

- `template<typename _Tp >`
`_Tp std::__detail::__log_gamma_sign (_Tp __x)`
Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)`
Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

- `template<typename _Tp >`
`_Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)`
Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $(n)_n = n!$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)`
Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(a), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__pgamma (_Tp __a, _Tp __x)`
Return the regularized lower incomplete gamma function. The regularized lower incomplete gamma function is defined by

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

is the lower incomplete gamma function.

- `template<typename _Tp >`

`_Tp std::__detail::__pochhammer_l(_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $(n)_n = n!$.

- `template<typename _Tp >`

`_Tp std::__detail::__pochhammer_u(_Tp __a, _Tp __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`

`_Tp std::__detail::__psi(_Tp __x)`

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1 - x) - \pi \cot(\pi x)$$

.

- `template<typename _Tp >`

`_Tp std::__detail::__psi(unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

- `template<typename _Tp >`

`_Tp std::__detail::__psi_asymp(_Tp __x)`

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

.

- `template<typename _Tp >`

`_Tp std::__detail::__psi_series(_Tp __x)`

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

.

- `template<typename _Tp >`
`_Tp std::__detail::__qgamma (_Tp __a, _Tp __x)`

Return the regularized upper incomplete gamma function. The regularized upper incomplete gamma function is defined by

$$Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

is the upper incomplete gamma function.

Variables

- `constexpr _Factorial_table< long double > std::__detail::_S_double_factorial_table` [301]
- `constexpr _Factorial_table< long double > std::__detail::_S_factorial_table` [171]
- `constexpr _Factorial_table< long double > std::__detail::_S_neg_double_factorial_table` [999]
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::_S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::_S_num_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

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

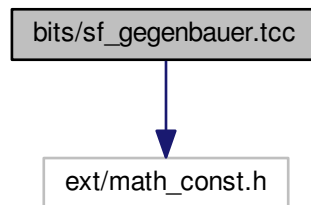
11.10.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

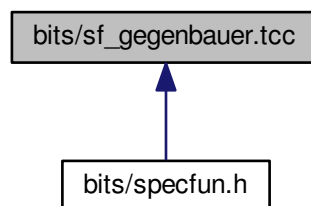
11.11 `bits/sf_gegenbauer.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_gegenbauer.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`

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

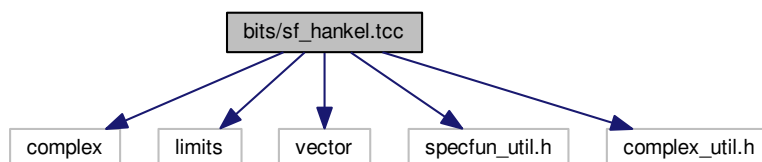
11.11.2.1 `#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1`

Definition at line 31 of file `sf_gegenbauer.tcc`.

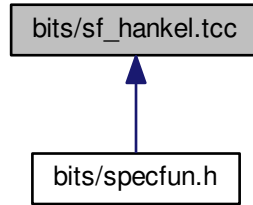
11.12 bits/sf_hankel.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include "specfun_util.h"
#include "complex_util.h"
```

Include dependency graph for `sf_hankel.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_HANKEL_TCC 1](#)

Functions

- `template<typename _Tp >`
`void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`

- `template<typename _Tp >`
`void std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`
`void std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`
`void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`
Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`
Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &_H1sum, std::complex< _Tp > &_H1psum, std::complex< _Tp > &_H2sum, std::complex< _Tp > &_H2psum)`
Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Bessel function.
- `template<typename _Tp >`
`void std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)`
Helper to compute complex spherical Hankel functions and their derivatives.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Neumann function.

11.12.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.12.2 Macro Definition Documentation

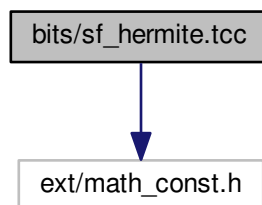
11.12.2.1 `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

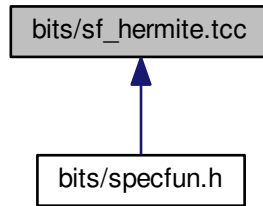
11.13 `bits/sf_hermite.tcc` File Reference

```
#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 .

11.13.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.13.2 Macro Definition Documentation

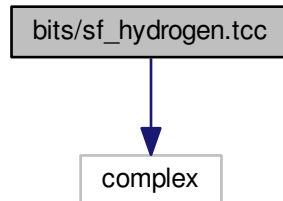
11.13.2.1 [#define _GLIBCXX_BITS_SF_HERMITE_TCC 1](#)

Definition at line 42 of file `sf_hermite.tcc`.

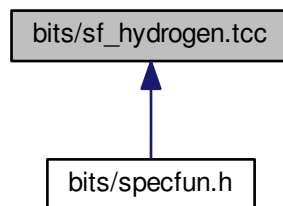
11.14 bits/sf_hydrogen.tcc File Reference

```
#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](#) (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)

11.14.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.14.2 Macro Definition Documentation

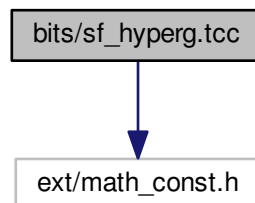
11.14.2.1 `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Definition at line 31 of file sf_hydrogen.tcc.

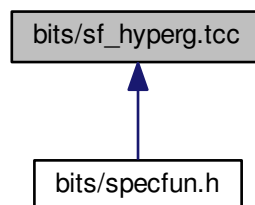
11.15 bits/sf_hyperg.tcc File Reference

```
#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.

11.15.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.15.2 Macro Definition Documentation

11.15.2.1 `#define _GLIBCXX_BITS_SF_HYPERG_TCC 1`

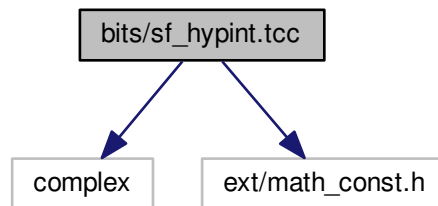
Definition at line 44 of file sf_hyperg.tcc.

11.16 bits/sf_hypint.tcc File Reference

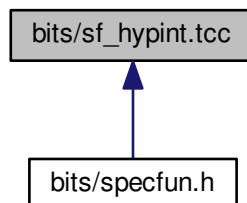
```
#include <complex>
```

```
#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.

11.16.1 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

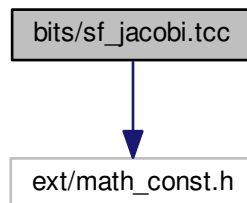
11.16.2.1 `#define _GLIBCXX_BITS_SF_HYPINT_TCC 1`

Definition at line 31 of file `sf_hypint.tcc`.

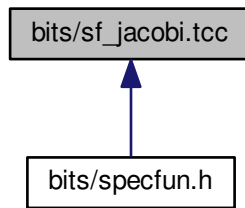
11.17 bits/sf_jacobi.tcc File Reference

`#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_num_t< _Tp > std::__detail::__zernike \(unsigned int __n, int __m, _Tp __rho, _Tp __phi\)](#)

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

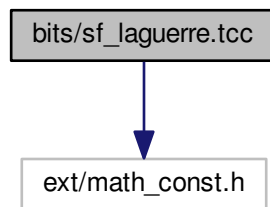
11.17.2.1 [#define _GLIBCXX_BITS_SF_JACOBI_TCC 1](#)

Definition at line 31 of file `sf_jacobi.tcc`.

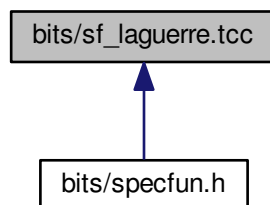
11.18 bits/sf_laguerre.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_laguerre.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_LAGUERRE_TCC](#) 1

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__assoc_laguerre` (unsigned int __n, unsigned int __m, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree m : $L_n^m(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__laguerre` (unsigned int __n, _Tp __x)
This routine returns the Laguerre polynomial of order n : $L_n(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__poly_laguerre` (unsigned int __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__poly_laguerre_hyperg` (unsigned int __n, _Tpa __alpha1, _Tp __x)
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__poly_laguerre_large_n` (unsigned __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree $\alpha > -1$ for large n . Abramowitz & Stegun, 13.5.21.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__poly_laguerre_recursion` (unsigned int __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$ by recursion.

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

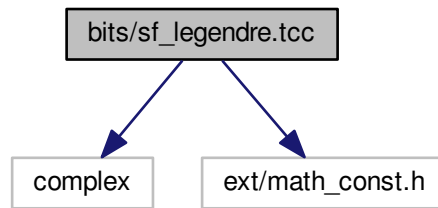
11.18.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

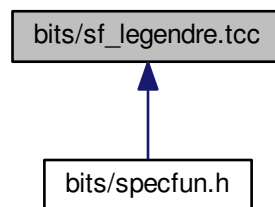
11.19 bits/sf_legendre.tcc File Reference

```
#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.

11.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

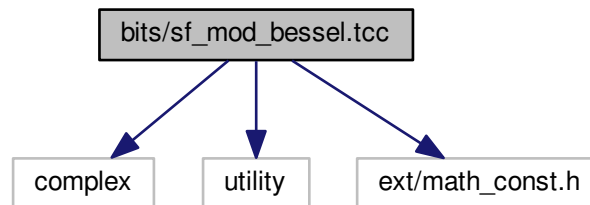
11.19.2 Macro Definition Documentation

11.19.2.1 `#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1`

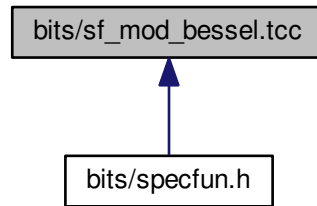
Definition at line 47 of file `sf_legendre.tcc`.

11.20 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 ν : $I_\nu(x)$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. 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 ν .

- `template<typename _Tp >`
`void std::__detail::__fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w_1'(x)$ and $w_2'(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

- `template<typename _Tp >`
`void std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i_n'(x)$ and $k_n'(x)$ respectively.

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>`.

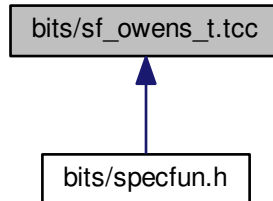
11.20.2 Macro Definition Documentation

11.20.2.1 `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

11.21 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 \(_Tp __x\)](#)

11.21.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

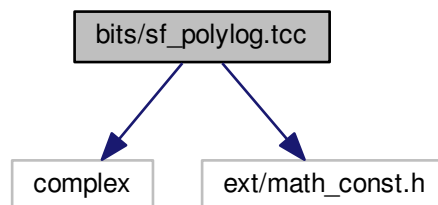
11.21.2 Macro Definition Documentation

11.21.2.1 [#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1](#)

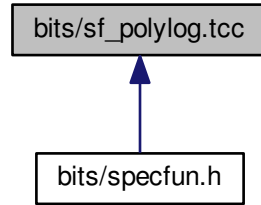
Definition at line 31 of file `sf_owens_t.tcc`.

11.22 bits/sf_polylog.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
Include dependency graph for sf_polylog.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__bose_einstein \(_Tp __s, _Tp __x\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__clamp_0_m2pi \(std::complex< _Tp > __w\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__clamp_pi \(std::complex< _Tp > __w\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__clausen \(unsigned int __m, std::complex< _Tp > __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__clausen \(unsigned int __m, _Tp __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__clausen_c \(unsigned int __m, std::complex< _Tp > __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__clausen_c \(unsigned int __m, _Tp __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__clausen_s \(unsigned int __m, std::complex< _Tp > __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__clausen_s \(unsigned int __m, _Tp __w\)](#)
- [template<typename _Tp > _Tp std::__detail::__dirichlet_beta \(std::complex< _Tp > __w\)](#)

- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`bool std::__detail::__fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`
`bool std::__detail::__fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool std::__detail::__fpimag (const _Tp)`
- `template<typename _Tp >`
`bool std::__detail::__fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool std::__detail::__fpreal (const _Tp)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`
`_Tp std::__detail::__polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > std::__detail::__polylog_exp (_Tp __s, ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp = double>`
`_Tp std::__detail::evenzeta (unsigned int __k)`

11.22.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

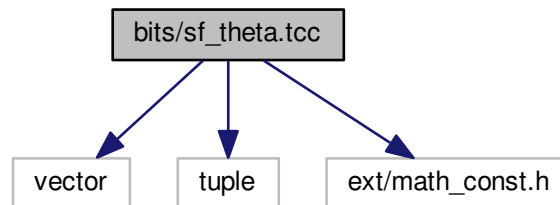
11.22.2 Macro Definition Documentation

11.22.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

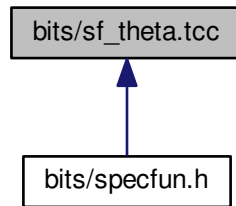
Definition at line 41 of file `sf_polylog.tcc`.

11.23 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)`

11.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

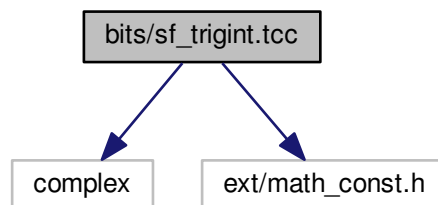
11.23.2 Macro Definition Documentation

11.23.2.1 `#define _GLIBCXX_BITS_SF_THETA_TCC 1`

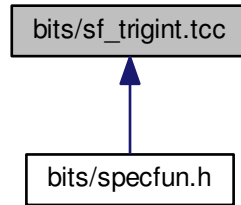
Definition at line 31 of file `sf_theta.tcc`.

11.24 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.

11.24.1 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

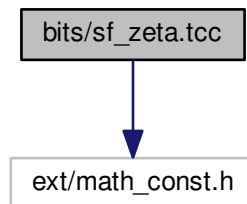
11.24.2.1 `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

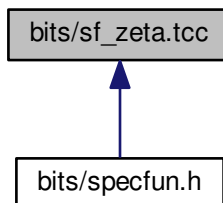
11.25 bits/sf_zeta.tcc File Reference

```
#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`]

11.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.25.2 Macro Definition Documentation

11.25.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

11.26 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_gamma.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_dawson.tcc>
#include <bits/sf_ellint.tcc>
#include <bits/sf_expint.tcc>
#include <bits/sf_fresnel.tcc>
#include <bits/sf_gegenbauer.tcc>
#include <bits/sf_hyperg.tcc>
#include <bits/sf_hypint.tcc>
#include <bits/sf_jacobi.tcc>
#include <bits/sf_laguerre.tcc>
#include <bits/sf_legendre.tcc>
#include <bits/sf_hydrogen.tcc>
#include <bits/sf_mod_bessel.tcc>
#include <bits/sf_hermite.tcc>
#include <bits/sf_theta.tcc>
#include <bits/sf_trigint.tcc>
#include <bits/sf_zeta.tcc>
#include <bits/sf_owens_t.tcc>
#include <bits/sf_polylog.tcc>
#include <bits/sf_airy.tcc>
#include <bits/sf_hankel.tcc>
#include <bits/sf_distributions.tcc>
```

Include dependency graph for specfun.h:



Namespaces

- [__gnu_cxx](#)
- [std](#)

Macros

- `#define __cpp_lib_math_special_functions 201603L`
- `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Enumerations

- enum { [__gnu_cxx::GLIBCXX_JACOBI_SN](#), [__gnu_cxx::GLIBCXX_JACOBI_CN](#), [__gnu_cxx::GLIBCXX_JACOBI_DN](#) }

Functions

- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::airy_ai](#) (_Tp __x)
- float [__gnu_cxx::airy_aif](#) (float __x)
- long double [__gnu_cxx::airy_ail](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::airy_bi](#) (_Tp __x)
- float [__gnu_cxx::airy_bif](#) (float __x)
- long double [__gnu_cxx::airy_bil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type std::assoc_laguerre](#) (unsigned int __n, unsigned int __m, _Tp __x)
- float [std::assoc_laguerref](#) (unsigned int __n, unsigned int __m, float __x)
- long double [std::assoc_laguerrel](#) (unsigned int __n, unsigned int __m, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type std::assoc_legendre](#) (unsigned int __l, unsigned int __m, _Tp __x)
- float [std::assoc_legendref](#) (unsigned int __l, unsigned int __m, float __x)
- long double [std::assoc_legendrel](#) (unsigned int __l, unsigned int __m, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::bernoulli](#) (unsigned int __n)
- float [__gnu_cxx::bernoullif](#) (unsigned int __n)
- long double [__gnu_cxx::bernoullil](#) (unsigned int __n)
- template<typename _Tpa, typename _Tpb >
[__gnu_cxx::__promote_2<_Tpa, _Tpb>::__type std::beta](#) (_Tpa __a, _Tpb __b)
- float [std::betaf](#) (float __a, float __b)
- long double [std::betal](#) (long double __a, long double __b)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::bincoef](#) (unsigned int __n, unsigned int __k)
- float [__gnu_cxx::bincoeff](#) (unsigned int __n, unsigned int __k)
- long double [__gnu_cxx::bincoefl](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_t](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_tf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_tl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_u](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_uf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_ul](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_v](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_vf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_vl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_w](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_wf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_wl](#) (unsigned int __n, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen` (unsigned int __m, _Tp __w)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen` (unsigned int __m, std::complex< _Tp > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_cf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_cl` (unsigned int __m, long double __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_sf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_sl` (unsigned int __m, long double __w)
- `float __gnu_cxx::clausenf` (unsigned int __m, float __w)
- `std::complex< float > __gnu_cxx::clausenf` (unsigned int __m, std::complex< float > __w)
- `long double __gnu_cxx::clausenl` (unsigned int __m, long double __w)
- `std::complex< long double > __gnu_cxx::clausenl` (unsigned int __m, std::complex< long double > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (_Tp __k)
- `float std::comp_ellint_1f` (float __k)
- `long double std::comp_ellint_1l` (long double __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (_Tp __k)
- `float std::comp_ellint_2f` (float __k)
- `long double std::comp_ellint_2l` (long double __k)
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (_Tp __k, _Tpn __nu)
- `float std::comp_ellint_3f` (float __k, float __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.*
- `long double std::comp_ellint_3l` (long double __k, long double __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.*
- `template<typename _Tk >`
`__gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d` (_Tk __k)
- `float __gnu_cxx::comp_ellint_df` (float __k)
- `long double __gnu_cxx::comp_ellint_dl` (long double __k)
- `float __gnu_cxx::comp_ellint_rf` (float __x, float __y)
- `long double __gnu_cxx::comp_ellint_rl` (long double __x, long double __y)
- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf` (_Tx __x, _Ty __y)
- `float __gnu_cxx::comp_ellint_rg` (float __x, float __y)
- `long double __gnu_cxx::comp_ellint_rl` (long double __x, long double __y)
- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg` (_Tx __x, _Ty __y)
- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg` (_Tpa __a, _Tpc __c, _Tp __x)
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg_lim` (_Tpc __c, _Tp __x)
- `float __gnu_cxx::conf_hyperg_limf` (float __c, float __x)
- `long double __gnu_cxx::conf_hyperg_liml` (long double __c, long double __x)
- `float __gnu_cxx::conf_hypergf` (float __a, float __c, float __x)
- `long double __gnu_cxx::conf_hypergl` (long double __a, long double __c, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::coshint (_Tp __x)`
- `float __gnu_cxx::coshintf (float __x)`
- `long double __gnu_cxx::coshintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::cosint (_Tp __x)`
- `float __gnu_cxx::cosintf (float __x)`
- `long double __gnu_cxx::cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_if (float __nu, float __x)`
- `long double std::cyl_bessel_il (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_jf (float __nu, float __x)`
- `long double std::cyl_bessel_jl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_k (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_kf (float __nu, float __x)`
- `long double std::cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __nu, std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __nu, std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_neumann (_Tpnu __nu, _Tp __x)`
- `float std::cyl_neumannf (float __nu, float __x)`
- `long double std::cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)`

- float [__gnu_cxx::digammaf](#) (float __z)
- long double [__gnu_cxx::digammal](#) (long double __z)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::dilog](#) (_Tp __x)
- float [__gnu_cxx::dilogf](#) (float __x)
- long double [__gnu_cxx::dilogl](#) (long double __x)
- template<typename _Tp >
 _Tp [__gnu_cxx::dirichlet_beta](#) (_Tp __s)
- float [__gnu_cxx::dirichlet_betaf](#) (float __s)
- long double [__gnu_cxx::dirichlet_betall](#) (long double __s)
- template<typename _Tp >
 _Tp [__gnu_cxx::dirichlet_eta](#) (_Tp __s)
- float [__gnu_cxx::dirichlet_etaf](#) (float __s)
- long double [__gnu_cxx::dirichlet_etaall](#) (long double __s)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::double_factorial](#) (int __n)
- float [__gnu_cxx::double_factorialf](#) (int __n)
- long double [__gnu_cxx::double_factoriall](#) (int __n)
- template<typename _Tp, typename _Tpp >
 [__gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_1](#) (_Tp __k, _Tpp __phi)
- float [std::ellint_1f](#) (float __k, float __phi)
- long double [std::ellint_1l](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tpp >
 [__gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_2](#) (_Tp __k, _Tpp __phi)
- float [std::ellint_2f](#) (float __k, float __phi)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for float argument.*
- long double [std::ellint_2l](#) (long double __k, long double __phi)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$.*
- template<typename _Tp, typename _Tpn, typename _Tpp >
 [__gnu_cxx::__promote_3<_Tp, _Tpn, _Tpp>::__type std::ellint_3](#) (_Tp __k, _Tpn __nu, _Tpp __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- float [std::ellint_3f](#) (float __k, float __nu, float __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for float argument.*
- long double [std::ellint_3l](#) (long double __k, long double __nu, long double __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- template<typename _Tk, typename _Tp, typename _Ta, typename _Tb >
 [__gnu_cxx::__promote_num_t<_Tk, _Tp, _Ta, _Tb> __gnu_cxx::ellint_cel](#) (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float [__gnu_cxx::ellint_celf](#) (float __k_c, float __p, float __a, float __b)
- long double [__gnu_cxx::ellint_cell](#) (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk, typename _Tphi >
 [__gnu_cxx::__promote_num_t<_Tk, _Tphi> __gnu_cxx::ellint_d](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::ellint_df](#) (float __k, float __phi)
- long double [__gnu_cxx::ellint_dall](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tk >
 [__gnu_cxx::__promote_num_t<_Tp, _Tk> __gnu_cxx::ellint_el1](#) (_Tp __x, _Tk __k_c)
- float [__gnu_cxx::ellint_el1f](#) (float __x, float __k_c)
- long double [__gnu_cxx::ellint_el1l](#) (long double __x, long double __k_c)
- template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >
 [__gnu_cxx::__promote_num_t<_Tp, _Tk, _Ta, _Tb> __gnu_cxx::ellint_el2](#) (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)

- float [__gnu_cxx::ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [__gnu_cxx::ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
[__gnu_cxx::__promote_num_t<_Tx, _Tk, _Tp>](#) [__gnu_cxx::ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [__gnu_cxx::ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [__gnu_cxx::ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
[__gnu_cxx::__promote_num_t<_Tp, _Up>](#) [__gnu_cxx::ellint_rc](#) (_Tp __x, _Up __y)
- float [__gnu_cxx::ellint_rcf](#) (float __x, float __y)
- long double [__gnu_cxx::ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rdf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rff](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rgf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp, _Wp>](#) [__gnu_cxx::ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [__gnu_cxx::ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [__gnu_cxx::ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
[_Tp](#) [__gnu_cxx::ellnome](#) (_Tp __k)
- float [__gnu_cxx::ellnomef](#) (float __k)
- long double [__gnu_cxx::ellnomel](#) (long double __k)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::type](#) [std::expint](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::expint](#) (unsigned int __n, _Tp __x)
- float [std::expintf](#) (float __x)
- float [__gnu_cxx::expintf](#) (unsigned int __n, float __x)
- long double [std::expintl](#) (long double __x)
- long double [__gnu_cxx::expintl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::factorial](#) (unsigned int __n)
- float [__gnu_cxx::factorialf](#) (unsigned int __n)
- long double [__gnu_cxx::factoriall](#) (unsigned int __n)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::fresnel_c](#) (_Tp __x)
- float [__gnu_cxx::fresnel_cf](#) (float __x)
- long double [__gnu_cxx::fresnel_cl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::fresnel_s](#) (_Tp __x)
- float [__gnu_cxx::fresnel_sf](#) (float __x)
- long double [__gnu_cxx::fresnel_sl](#) (long double __x)

- `template<typename _Tn, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_l (_Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_lf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ll (long double __n, long double __x)`
- `template<typename _Tn, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_u (_Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_uf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ul (long double __n, long double __x)`
- `template<typename _Talpha, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)`
- `float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)`
- `long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)`
- `float std::hermitef (unsigned int __n, float __x)`
- `long double std::hermitel (unsigned int __n, long double __x)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::heuman_lambdaf (float __k, float __phi)`
- `long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)`
- `template<typename _Tp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta (_Tp __s, _Up __a)`
- `template<typename _Tp, typename _Up >`
`std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)`
- `long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetacf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetacdl (long double __a, long double __b, long double __x)`
- `float __gnu_cxx::ibetaf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_cnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_dnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)`

- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_snf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::jacobi_zetaf (float __k, float __phi)`
- `long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)`
- `float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::laguerre (unsigned int __n, _Tp __x)`
- `float std::laguerref (unsigned int __n, float __x)`
- `long double std::laguerrel (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)`
- `float __gnu_cxx::lbincoeff (unsigned int __n, unsigned int __k)`
- `long double __gnu_cxx::lbincoeffl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)`
- `float __gnu_cxx::ldouble_factorialf (int __n)`
- `long double __gnu_cxx::ldouble_factoriall (int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::legendre (unsigned int __l, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::legendre_qf (unsigned int __n, float __x)`
- `long double __gnu_cxx::legendre_ql (unsigned int __n, long double __x)`
- `float std::legendref (unsigned int __l, float __x)`
- `long double std::legendrel (unsigned int __l, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)`
- `float __gnu_cxx::lfactorialf (unsigned int __n)`
- `long double __gnu_cxx::lfactoriall (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint (_Tp __x)`
- `float __gnu_cxx::logintf (float __x)`
- `long double __gnu_cxx::logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l (_Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_lf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_u (_Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_uf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)`
- `float __gnu_cxx::owens_tf (float __h, float __a)`
- `long double __gnu_cxx::owens_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::pgamma (_Ta __a, _Tp __x)`

- float [__gnu_cxx::pgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::pgammal](#) (long double __a, long double __x)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::pochhammer_l](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_lf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn> __gnu_cxx::pochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tp, typename _Wp >
[__gnu_cxx::__promote_num_t<_Tp, _Wp> __gnu_cxx::polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
std::complex< [__gnu_cxx::__promote_num_t<_Tp, _Wp> > __gnu_cxx::polylog](#) (_Tp __s, std::complex< _Tp > __w)
- float [__gnu_cxx::polylogf](#) (float __s, float __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __w)
- long double [__gnu_cxx::polylogl](#) (long double __s, long double __w)
- std::complex< long double > [__gnu_cxx::polylogl](#) (long double __s, std::complex< long double > __w)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::psi](#) (_Tp __x)
- float [__gnu_cxx::psif](#) (float __x)
- long double [__gnu_cxx::psil](#) (long double __x)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::__promote_num_t<_Ta, _Tp> __gnu_cxx::qgamma](#) (_Ta __a, _Tp __x)
- float [__gnu_cxx::qgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::qgammal](#) (long double __a, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [__gnu_cxx::radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [__gnu_cxx::radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type std::riemann_zeta](#) (_Tp __s)
- float [std::riemann_zetaf](#) (float __s)
- long double [std::riemann_zetal](#) (long double __s)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinc_pif](#) (float __x)
- long double [__gnu_cxx::sinc_pil](#) (long double __x)
- float [__gnu_cxx::sincf](#) (float __x)
- long double [__gnu_cxx::sincl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::sinhc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinhc_pif](#) (float __x)
- long double [__gnu_cxx::sinhc_pil](#) (long double __x)
- float [__gnu_cxx::sinhcf](#) (float __x)
- long double [__gnu_cxx::sinhcl](#) (long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinhint (_Tp __x)`
- `float __gnu_cxx::sinhintf (float __x)`
- `long double __gnu_cxx::sinhintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sinint (_Tp __x)`
- `float __gnu_cxx::sinintf (float __x)`
- `long double __gnu_cxx::sinintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::sph_bessel (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x)`
- `float std::sph_besself (unsigned int __n, float __x)`
- `long double std::sph_bessell (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Ttheta, typename _Tphi >`
`std::complex< __gnu_cxx::__promote_num_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)`
- `std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)`
- `std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`
- `float std::sph_legendref (unsigned int __l, unsigned int __m, float __theta)`
- `long double std::sph_legendrel (unsigned int __l, unsigned int __m, long double __theta)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::sph_neumann (unsigned int __n, _Tp __x)`
- `float std::sph_neumannf (unsigned int __n, float __x)`
- `long double std::sph_neumannl (unsigned int __n, long double __x)`

- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_1f (float __nu, float __x)`
- `long double __gnu_cxx::theta_1l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_2f (float __nu, float __x)`
- `long double __gnu_cxx::theta_2l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_3f (float __nu, float __x)`
- `long double __gnu_cxx::theta_3l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu, _Tp > __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_4f (float __nu, float __x)`
- `long double __gnu_cxx::theta_4l (long double __nu, long double __x)`
- `template<typename _Tp_k, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_c (_Tp_k __k, _Tp __x)`
- `float __gnu_cxx::theta_cf (float __k, float __x)`
- `long double __gnu_cxx::theta_cl (long double __k, long double __x)`
- `template<typename _Tp_k, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_d (_Tp_k __k, _Tp __x)`
- `float __gnu_cxx::theta_df (float __k, float __x)`
- `long double __gnu_cxx::theta_dl (long double __k, long double __x)`
- `template<typename _Tp_k, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_n (_Tp_k __k, _Tp __x)`
- `float __gnu_cxx::theta_nf (float __k, float __x)`
- `long double __gnu_cxx::theta_nl (long double __k, long double __x)`
- `template<typename _Tp_k, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k, _Tp > __gnu_cxx::theta_s (_Tp_k __k, _Tp __x)`
- `float __gnu_cxx::theta_sf (float __k, float __x)`
- `long double __gnu_cxx::theta_sl (long double __k, long double __x)`
- `template<typename _Trho, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Trho, _Tphi > __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)`
- `float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)`
- `long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)`

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 __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

11.26.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file `specfun.h`.

Index

- `_Airy`
 - `std::__detail::_Airy`, [260](#)
- `_Airy_asymp`
 - `std::__detail::_Airy_asymp`, [264](#)
- `_Airy_asymp_series`
 - `std::__detail::_Airy_asymp_series`, [271](#)
- `_GLIBCXX_BITS_SF_AIRY_TCC`
 - `sf_airy.tcc`, [287](#)
- `_GLIBCXX_BITS_SF_BESSEL_TCC`
 - `sf_bessel.tcc`, [289](#)
- `_GLIBCXX_BITS_SF_BETA_TCC`
 - `sf_beta.tcc`, [291](#)
- `_GLIBCXX_BITS_SF_CARDINAL_TCC`
 - `sf_cardinal.tcc`, [292](#)
- `_GLIBCXX_BITS_SF_CHEBYSHEV_TCC`
 - `sf_chebyshev.tcc`, [294](#)
- `_GLIBCXX_BITS_SF_DAWSON_TCC`
 - `sf_dawson.tcc`, [295](#)
- `_GLIBCXX_BITS_SF_ELLINT_TCC`
 - `sf_ellint.tcc`, [298](#)
- `_GLIBCXX_BITS_SF_EXPINT_TCC`
 - `sf_expint.tcc`, [300](#)
- `_GLIBCXX_BITS_SF_FRESNEL_TCC`
 - `sf_fresnel.tcc`, [302](#)
- `_GLIBCXX_BITS_SF_GAMMA_TCC`
 - `sf_gamma.tcc`, [308](#)
- `_GLIBCXX_BITS_SF_GEGENBAUER_TCC`
 - `sf_gegenbauer.tcc`, [309](#)
- `_GLIBCXX_BITS_SF_HANKEL_TCC`
 - `sf_hankel.tcc`, [312](#)
- `_GLIBCXX_BITS_SF_HERMITE_TCC`
 - `sf_hermite.tcc`, [313](#)
- `_GLIBCXX_BITS_SF_HYDROGEN_TCC`
 - `sf_hydrogen.tcc`, [315](#)
- `_GLIBCXX_BITS_SF_HYPERG_TCC`
 - `sf_hyperg.tcc`, [317](#)
- `_GLIBCXX_BITS_SF_HYPINT_TCC`
 - `sf_hypint.tcc`, [318](#)
- `_GLIBCXX_BITS_SF_JACOBI_TCC`
 - `sf_jacobi.tcc`, [319](#)
- `_GLIBCXX_BITS_SF_LAGUERRE_TCC`
 - `sf_laguerre.tcc`, [321](#)
- `_GLIBCXX_BITS_SF_LEGENDRE_TCC`
 - `sf_legendre.tcc`, [323](#)
- `_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`
 - `sf_mod_bessel.tcc`, [325](#)
- `_GLIBCXX_BITS_SF_OWENS_T_TCC`
 - `sf_owens_t.tcc`, [326](#)
- `_GLIBCXX_BITS_SF_POLYLOG_TCC`
 - `sf_polylog.tcc`, [329](#)
- `_GLIBCXX_BITS_SF_THETA_TCC`
 - `sf_theta.tcc`, [331](#)
- `_GLIBCXX_BITS_SF_TRIGINT_TCC`
 - `sf_trigint.tcc`, [333](#)
- `_GLIBCXX_BITS_SF_ZETA_TCC`
 - `sf_zeta.tcc`, [335](#)
- `_GLIBCXX_JACOBI_CN`
 - GNU Extended Mathematical Special Functions, [52](#)
- `_GLIBCXX_JACOBI_DN`
 - GNU Extended Mathematical Special Functions, [52](#)
- `_GLIBCXX_JACOBI_SN`
 - GNU Extended Mathematical Special Functions, [52](#)
- `_N_FGH`
 - `std::__detail::_Airy_series`, [278](#)
- `_Num_Euler_Maclaurin_zeta`
 - `std::__detail`, [256](#)
- `_S_2pi_3`
 - `std::__detail::_Airy`, [261](#)
- `_S_5pi_6`
 - `std::__detail::_Airy`, [261](#)
- `_S_Ai`
 - `std::__detail::_Airy_series`, [275](#)
- `_S_Ai0`
 - `std::__detail::_Airy_series`, [278](#)
- `_S_Aip0`
 - `std::__detail::_Airy_series`, [279](#)
- `_S_Airy`
 - `std::__detail::_Airy_series`, [275](#)
- `_S_Bi`
 - `std::__detail::_Airy_series`, [276](#)
- `_S_Bi0`
 - `std::__detail::_Airy_series`, [279](#)
- `_S_Bip0`
 - `std::__detail::_Airy_series`, [279](#)
- `_S_Euler_Maclaurin_zeta`
 - `std::__detail`, [256](#)
- `_S_FGH`
 - `std::__detail::_Airy_series`, [276](#)
- `_S_Fock`
 - `std::__detail::_Airy_series`, [277](#)

- `_S_Gi0`
 - `std::__detail::__Airy_series`, 279
- `_S_Gip0`
 - `std::__detail::__Airy_series`, 279
- `_S_Hi0`
 - `std::__detail::__Airy_series`, 279
- `_S_Hip0`
 - `std::__detail::__Airy_series`, 279
- `_S_NaN`
 - `std::__detail::__Airy`, 261
- `_S_Scorer`
 - `std::__detail::__Airy_series`, 277
- `_S_Scorer2`
 - `std::__detail::__Airy_series`, 278
- `_S_absarg_ge_pio3`
 - `std::__detail::__Airy_asymp`, 264
- `_S_absarg_lt_pio3`
 - `std::__detail::__Airy_asymp`, 264
- `_S_c`
 - `std::__detail::__Airy_asymp_data< __float128 >`, 267
 - `std::__detail::__Airy_asymp_data< double >`, 268
 - `std::__detail::__Airy_asymp_data< float >`, 268
 - `std::__detail::__Airy_asymp_data< long double >`, 269
- `_S_cNaN`
 - `std::__detail::__Airy`, 261
- `_S_d`
 - `std::__detail::__Airy_asymp_data< __float128 >`, 267
 - `std::__detail::__Airy_asymp_data< double >`, 268
 - `std::__detail::__Airy_asymp_data< float >`, 268
 - `std::__detail::__Airy_asymp_data< long double >`, 269
- `_S_double_factorial_table`
 - `std::__detail`, 256
- `_S_eps`
 - `std::__detail::__Airy_series`, 279
- `_S_factorial_table`
 - `std::__detail`, 256
- `_S_i`
 - `std::__detail::__Airy`, 261
 - `std::__detail::__Airy_series`, 280
- `_S_log10min`
 - `std::__detail::__Airy_series`, 280
- `_S_max_cd`
 - `std::__detail::__Airy_asymp_data< __float128 >`, 267
 - `std::__detail::__Airy_asymp_data< double >`, 268
 - `std::__detail::__Airy_asymp_data< float >`, 269
 - `std::__detail::__Airy_asymp_data< long double >`, 269
- `_S_neg_double_factorial_table`
 - `std::__detail`, 256
- `_S_num_double_factorials`
 - `std::__detail`, 256
- `std::__detail`, 257
- `_S_num_double_factorials< float >`
 - `std::__detail`, 257
- `_S_num_double_factorials< long double >`
 - `std::__detail`, 257
- `_S_num_factorials`
 - `std::__detail`, 257
- `_S_num_factorials< double >`
 - `std::__detail`, 257
- `_S_num_factorials< float >`
 - `std::__detail`, 257
- `_S_num_factorials< long double >`
 - `std::__detail`, 257
- `_S_num_neg_double_factorials`
 - `std::__detail`, 257
- `_S_num_neg_double_factorials< double >`
 - `std::__detail`, 257
- `_S_num_neg_double_factorials< float >`
 - `std::__detail`, 257
- `_S_num_neg_double_factorials< long double >`
 - `std::__detail`, 258
- `_S_num_zetam1`
 - `std::__detail`, 258
- `_S_pi`
 - `std::__detail::__Airy`, 261
 - `std::__detail::__Airy_series`, 280
- `_S_pi_3`
 - `std::__detail::__Airy`, 261
- `_S_pi_6`
 - `std::__detail::__Airy`, 261
- `_S_sqrt_pi`
 - `std::__detail::__Airy`, 261
 - `std::__detail::__Airy_asymp_series`, 271
 - `std::__detail::__Airy_series`, 280
- `_S_zetam1`
 - `std::__detail`, 258
- `_Val`
 - `std::__detail::__AiryAuxilliaryState`, 281
 - `std::__detail::__AiryState`, 283
- `__STDCPP_MATH_SPEC_FUNCS__`
 - `specfun.h`, 346
- `__airy`
 - `std::__detail`, 156
- `__airy_ai`
 - `std::__detail`, 157
- `__airy_arg`
 - `std::__detail`, 157
- `__airy_bi`
 - `std::__detail`, 157
- `__assoc_laguerre`
 - `std::__detail`, 157
- `__assoc_legendre_p`
 - `std::__detail`, 158
- `__bernoulli`

- `std::__detail`, 159
- `__bernoulli_2n`
 - `std::__detail`, 159
- `__bernoulli_series`
 - `std::__detail`, 159
- `__beta`
 - `std::__detail`, 160
- `__beta_gamma`
 - `std::__detail`, 160
- `__beta_inc`
 - `std::__detail`, 161
- `__beta_inc_cont_frac`
 - `std::__detail`, 161
- `__beta_lgamma`
 - `std::__detail`, 162
- `__beta_product`
 - `std::__detail`, 162
- `__bincoef`
 - `std::__detail`, 163
- `__bose_einstein`
 - `std::__detail`, 163
- `__chebyshev_recur`
 - `std::__detail`, 164
- `__chebyshev_t`
 - `std::__detail`, 164
- `__chebyshev_u`
 - `std::__detail`, 165
- `__chebyshev_v`
 - `std::__detail`, 165
- `__chebyshev_w`
 - `std::__detail`, 166
- `__chshint`
 - `std::__detail`, 166
- `__chshint_cont_frac`
 - `std::__detail`, 167
- `__chshint_series`
 - `std::__detail`, 167
- `__clamp_0_m2pi`
 - `std::__detail`, 167
- `__clamp_pi`
 - `std::__detail`, 167
- `__clausen`
 - `std::__detail`, 167, 168
- `__clausen_c`
 - `std::__detail`, 168, 169
- `__clausen_s`
 - `std::__detail`, 169, 170
- `__cmplx`
 - `std::__detail::Airy_asymp`, 264
- `__comp_ellint_1`
 - `std::__detail`, 170
- `__comp_ellint_2`
 - `std::__detail`, 171
- `__comp_ellint_3`
 - `std::__detail`, 171
- `__comp_ellint_d`
 - `std::__detail`, 172
- `__comp_ellint_rf`
 - `std::__detail`, 172
- `__comp_ellint_rg`
 - `std::__detail`, 172
- `__conf_hyperg`
 - `std::__detail`, 172
- `__conf_hyperg_lim`
 - `std::__detail`, 173
- `__conf_hyperg_lim_series`
 - `std::__detail`, 173
- `__conf_hyperg_luke`
 - `std::__detail`, 174
- `__conf_hyperg_series`
 - `std::__detail`, 174
- `__coshint`
 - `std::__detail`, 174
- `__cpp_lib_math_special_functions`
 - `specfun.h`, 346
- `__cyl_bessel`
 - `std::__detail`, 175
- `__cyl_bessel_i`
 - `std::__detail`, 175
- `__cyl_bessel_ij_series`
 - `std::__detail`, 176
- `__cyl_bessel_ik`
 - `std::__detail`, 176
- `__cyl_bessel_ik_asymp`
 - `std::__detail`, 178
- `__cyl_bessel_ik_steel`
 - `std::__detail`, 178
- `__cyl_bessel_j`
 - `std::__detail`, 179
- `__cyl_bessel_jn`
 - `std::__detail`, 179
- `__cyl_bessel_jn_asymp`
 - `std::__detail`, 179
- `__cyl_bessel_jn_steel`
 - `std::__detail`, 180
- `__cyl_bessel_k`
 - `std::__detail`, 180
- `__cyl_hankel_1`
 - `std::__detail`, 181
- `__cyl_hankel_2`
 - `std::__detail`, 182
- `__cyl_neumann`
 - `std::__detail`, 183
- `__cyl_neumann_n`
 - `std::__detail`, 183
- `__dawson`
 - `std::__detail`, 183
- `__dawson_cont_frac`
 - `std::__detail`, 183

- std::__detail, 184
- __dawson_series
 - std::__detail, 184
- __debye_region
 - std::__detail, 184
- __dilog
 - std::__detail, 184
- __dirichlet_beta
 - std::__detail, 185
- __dirichlet_eta
 - std::__detail, 186
- __double_factorial
 - std::__detail, 187
- __ellint_1
 - std::__detail, 187
- __ellint_2
 - std::__detail, 187
- __ellint_3
 - std::__detail, 188
- __ellint_cel
 - std::__detail, 188
- __ellint_d
 - std::__detail, 189
- __ellint_el1
 - std::__detail, 189
- __ellint_el2
 - std::__detail, 189
- __ellint_el3
 - std::__detail, 189
- __ellint_rc
 - std::__detail, 189
- __ellint_rd
 - std::__detail, 190
- __ellint_rf
 - std::__detail, 191
- __ellint_rg
 - std::__detail, 191
- __ellint_rj
 - std::__detail, 192
- __ellnome
 - std::__detail, 193
- __ellnome_k
 - std::__detail, 193
- __ellnome_series
 - std::__detail, 193
- __expint
 - std::__detail, 193, 194
- __expint_E1
 - std::__detail, 195
- __expint_E1_asymp
 - std::__detail, 195
- __expint_E1_series
 - std::__detail, 196
- __expint_Ei
 - std::__detail, 196
- __expint_Ei_asymp
 - std::__detail, 197
- __expint_Ei_series
 - std::__detail, 197
- __expint_En_cont_frac
 - std::__detail, 198
- __expint_En_recursion
 - std::__detail, 198
- __expint_En_series
 - std::__detail, 199
- __expint_asymp
 - std::__detail, 194
- __expint_large_n
 - std::__detail, 199
- __factorial
 - std::__detail, 200
 - std::__detail::Factorial_table, 284
- __fermi_dirac
 - std::__detail, 200
- __fock_airy
 - std::__detail, 201
- __fpequal
 - std::__detail, 201
- __fpimag
 - std::__detail, 201, 202
- __fpreal
 - std::__detail, 202
- __fresnel
 - std::__detail, 202
- __fresnel_cont_frac
 - std::__detail, 203
- __fresnel_series
 - std::__detail, 203
- __gamma
 - std::__detail, 203
- __gamma_cont_frac
 - std::__detail, 204
- __gamma_l
 - std::__detail, 204
- __gamma_series
 - std::__detail, 204
- __gamma_temme
 - std::__detail, 204
- __gamma_u
 - std::__detail, 205
- __gauss
 - std::__detail, 205
- __gegenbauer_poly
 - std::__detail, 205
- __gnu_cxx, 129
- __hankel
 - std::__detail, 206
- __hankel_debye

- `std::__detail`, 206
- `__hankel_params`
 - `std::__detail`, 207
- `__hankel_uniform`
 - `std::__detail`, 207
- `__hankel_uniform_olver`
 - `std::__detail`, 207
- `__hankel_uniform_outer`
 - `std::__detail`, 208
- `__hankel_uniform_sum`
 - `std::__detail`, 208
- `__heuman_lambda`
 - `std::__detail`, 209
- `__hurwitz_zeta`
 - `std::__detail`, 209, 210
- `__hurwitz_zeta_euler_maclaurin`
 - `std::__detail`, 210
- `__hydrogen`
 - `std::__detail`, 210
- `__hyperg`
 - `std::__detail`, 211
- `__hyperg_luke`
 - `std::__detail`, 211
- `__hyperg_reflect`
 - `std::__detail`, 211
- `__hyperg_series`
 - `std::__detail`, 212
- `__jacobi_sncndn`
 - `std::__detail`, 213
- `__jacobi_zeta`
 - `std::__detail`, 213
- `__laguerre`
 - `std::__detail`, 213
- `__legendre_q`
 - `std::__detail`, 213
- `__log_bincoef`
 - `std::__detail`, 214
- `__log_double_factorial`
 - `std::__detail`, 214
- `__log_factorial`
 - `std::__detail`, 215
 - `std::__detail::Factorial_table`, 284
- `__log_gamma`
 - `std::__detail`, 215
- `__log_gamma_bernoulli`
 - `std::__detail`, 216
- `__log_gamma_lanczos`
 - `std::__detail`, 216
- `__log_gamma_sign`
 - `std::__detail`, 216
- `__log_gamma_spouge`
 - `std::__detail`, 217
- `__log_pochhammer_l`
 - `std::__detail`, 217
- `__log_pochhammer_u`
 - `std::__detail`, 218
- `__logint`
 - `std::__detail`, 218
- `__max_FGH`
 - `std::__detail`, 255
- `__max_FGH< double >`
 - `std::__detail`, 255
- `__max_FGH< float >`
 - `std::__detail`, 256
- `__n`
 - `std::__detail::Factorial_table`, 284
- `__owens_t`
 - `std::__detail`, 219
- `__pgamma`
 - `std::__detail`, 219
- `__pochhammer_l`
 - `std::__detail`, 219
- `__pochhammer_u`
 - `std::__detail`, 220
- `__poly_hermite`
 - `std::__detail`, 220
- `__poly_hermite_asymp`
 - `std::__detail`, 221
- `__poly_hermite_recursion`
 - `std::__detail`, 221
- `__poly_jacobi`
 - `std::__detail`, 222
- `__poly_laguerre`
 - `std::__detail`, 222
- `__poly_laguerre_hyperg`
 - `std::__detail`, 223
- `__poly_laguerre_large_n`
 - `std::__detail`, 224
- `__poly_laguerre_recursion`
 - `std::__detail`, 224
- `__poly_legendre_p`
 - `std::__detail`, 225
- `__poly_radial_jacobi`
 - `std::__detail`, 225
- `__polylog`
 - `std::__detail`, 226, 227
- `__polylog_exp`
 - `std::__detail`, 227
- `__polylog_exp_asymp`
 - `std::__detail`, 228
- `__polylog_exp_int_neg`
 - `std::__detail`, 228, 229
- `__polylog_exp_int_pos`
 - `std::__detail`, 229, 230
- `__polylog_exp_neg`
 - `std::__detail`, 230, 231
- `__polylog_exp_neg_even`
 - `std::__detail`, 231

[__polylog_exp_neg_odd](#)
 std::__detail, [232](#)
[__polylog_exp_negative_real_part](#)
 std::__detail, [233](#)
[__polylog_exp_pos](#)
 std::__detail, [233](#), [234](#)
[__polylog_exp_real_neg](#)
 std::__detail, [235](#), [236](#)
[__polylog_exp_real_pos](#)
 std::__detail, [236](#)
[__psi](#)
 std::__detail, [237](#)
[__psi_asymp](#)
 std::__detail, [237](#)
[__psi_series](#)
 std::__detail, [238](#)
[__qgamma](#)
 std::__detail, [238](#)
[__riemann_zeta](#)
 std::__detail, [238](#)
[__riemann_zeta_alt](#)
 std::__detail, [239](#)
[__riemann_zeta_euler_maclaurin](#)
 std::__detail, [239](#)
[__riemann_zeta_glob](#)
 std::__detail, [239](#)
[__riemann_zeta_m_1](#)
 std::__detail, [239](#)
[__riemann_zeta_m_1_sum](#)
 std::__detail, [240](#)
[__riemann_zeta_product](#)
 std::__detail, [240](#)
[__riemann_zeta_sum](#)
 std::__detail, [241](#)
[__sinc](#)
 std::__detail, [241](#)
[__sinc_pi](#)
 std::__detail, [241](#)
[__sincosint](#)
 std::__detail, [242](#)
[__sincosint_asymp](#)
 std::__detail, [242](#)
[__sincosint_cont_frac](#)
 std::__detail, [242](#)
[__sincosint_series](#)
 std::__detail, [242](#)
[__sinhc](#)
 std::__detail, [243](#)
[__sinhc_pi](#)
 std::__detail, [243](#)
[__sinhint](#)
 std::__detail, [243](#)
[__sph_bessel](#)
 std::__detail, [244](#)

[__sph_bessel_ik](#)
 std::__detail, [245](#)
[__sph_bessel_jn](#)
 std::__detail, [245](#)
[__sph_hankel](#)
 std::__detail, [246](#)
[__sph_hankel_1](#)
 std::__detail, [246](#), [247](#)
[__sph_hankel_2](#)
 std::__detail, [247](#)
[__sph_harmonic](#)
 std::__detail, [248](#)
[__sph_legendre](#)
 std::__detail, [248](#)
[__sph_neumann](#)
 std::__detail, [249](#)
[__theta_1](#)
 std::__detail, [251](#)
[__theta_2](#)
 std::__detail, [251](#)
[__theta_2_asymp](#)
 std::__detail, [252](#)
[__theta_2_sum](#)
 std::__detail, [252](#)
[__theta_3](#)
 std::__detail, [252](#)
[__theta_3_asymp](#)
 std::__detail, [253](#)
[__theta_3_sum](#)
 std::__detail, [253](#)
[__theta_4](#)
 std::__detail, [253](#)
[__theta_c](#)
 std::__detail, [253](#)
[__theta_d](#)
 std::__detail, [253](#)
[__theta_n](#)
 std::__detail, [254](#)
[__theta_s](#)
 std::__detail, [254](#)
[__zernike](#)
 std::__detail, [254](#)
[__znorm1](#)
 std::__detail, [255](#)
[__znorm2](#)
 std::__detail, [255](#)

[Ai](#)
 std::__detail::__AiryState, [283](#)

[Aip](#)
 std::__detail::__AiryState, [283](#)

[airy_ai](#)
 GNU Extended Mathematical Special Functions, [52](#)
[airy_aif](#)

- GNU Extended Mathematical Special Functions, [53](#)
- airy_ail
 - GNU Extended Mathematical Special Functions, [53](#)
- airy_bi
 - GNU Extended Mathematical Special Functions, [53](#)
- airy_bif
 - GNU Extended Mathematical Special Functions, [54](#)
- airy_bil
 - GNU Extended Mathematical Special Functions, [54](#)
- 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, [54](#)
- bernoullif
 - GNU Extended Mathematical Special Functions, [54](#)
- bernoullil
 - GNU Extended Mathematical Special Functions, [55](#)
- 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, [283](#)
- bincoef
 - GNU Extended Mathematical Special Functions, [55](#)
- bincoeff
 - GNU Extended Mathematical Special Functions, [55](#)
- bincoefl
 - GNU Extended Mathematical Special Functions, [55](#)
- Bip
 - std::__detail::_AiryState, [283](#)
- bits/sf_airy.tcc, [285](#)
- bits/sf_bessel.tcc, [287](#)
- bits/sf_beta.tcc, [289](#)
- bits/sf_cardinal.tcc, [291](#)
- bits/sf_chebyshev.tcc, [293](#)
- bits/sf_dawson.tcc, [294](#)
- bits/sf_ellint.tcc, [296](#)
- bits/sf_expint.tcc, [298](#)
- bits/sf_fresnel.tcc, [301](#)
- bits/sf_gamma.tcc, [302](#)
- bits/sf_gegenbauer.tcc, [308](#)
- bits/sf_hankel.tcc, [309](#)
- bits/sf_hermite.tcc, [312](#)
- bits/sf_hydrogen.tcc, [314](#)
- bits/sf_hyperg.tcc, [315](#)
- bits/sf_hypint.tcc, [317](#)
- bits/sf_jacobi.tcc, [318](#)
- bits/sf_laguerre.tcc, [320](#)
- bits/sf_legendre.tcc, [321](#)
- bits/sf_mod_bessel.tcc, [323](#)
- bits/sf_owens_t.tcc, [325](#)
- bits/sf_polylog.tcc, [326](#)
- bits/sf_theta.tcc, [329](#)
- bits/sf_trigint.tcc, [331](#)
- bits/sf_zeta.tcc, [333](#)
- bits/specfun.h, [336](#)
- 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, [156](#)
- chebyshev_t
 - GNU Extended Mathematical Special Functions, [55](#)
- chebyshev_tf
 - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev_tl
 - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev_u
 - GNU Extended Mathematical Special Functions, [56](#)
- chebyshev_uf
 - GNU Extended Mathematical Special Functions, [57](#)
- chebyshev_ul
 - GNU Extended Mathematical Special Functions, [57](#)
- chebyshev_v
 - GNU Extended Mathematical Special Functions, [57](#)
- chebyshev_vf
 - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev_vl
 - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev_w
 - GNU Extended Mathematical Special Functions, [58](#)
- chebyshev_wf
 - GNU Extended Mathematical Special Functions, [59](#)
- chebyshev_wl
 - GNU Extended Mathematical Special Functions, [59](#)
- clausen
 - GNU Extended Mathematical Special Functions, [59](#), [60](#)
- clausen_c
 - GNU Extended Mathematical Special Functions, [60](#)
- clausen_cf
 - GNU Extended Mathematical Special Functions, [61](#)
- clausen_cl
 - GNU Extended Mathematical Special Functions, [61](#)
- clausen_s
 - GNU Extended Mathematical Special Functions, [61](#)
- clausen_sf
 - GNU Extended Mathematical Special Functions, [62](#)
- clausen_sl
 - GNU Extended Mathematical Special Functions, [62](#)
- clausenf
 - GNU Extended Mathematical Special Functions, [62](#)
- clausenl
 - GNU Extended Mathematical Special Functions, [63](#)
- 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, [63](#)
- comp_ellint_df
 - GNU Extended Mathematical Special Functions, [64](#)
- comp_ellint_dl
 - GNU Extended Mathematical Special Functions, [64](#)
- comp_ellint_rf
 - GNU Extended Mathematical Special Functions, [64](#)
- comp_ellint_rg
 - GNU Extended Mathematical Special Functions, [65](#)
- conf_hyperg
 - GNU Extended Mathematical Special Functions, [66](#)
- conf_hyperg_lim
 - GNU Extended Mathematical Special Functions, [66](#)
- conf_hyperg_limf
 - GNU Extended Mathematical Special Functions, [66](#)
- conf_hyperg_liml
 - GNU Extended Mathematical Special Functions, [66](#)

- GNU Extended Mathematical Special Functions, [67](#)
- conf_hypergf
 - GNU Extended Mathematical Special Functions, [67](#)
- conf_hypergl
 - GNU Extended Mathematical Special Functions, [67](#)
- coshint
 - GNU Extended Mathematical Special Functions, [67](#)
- coshintf
 - GNU Extended Mathematical Special Functions, [68](#)
- coshintl
 - GNU Extended Mathematical Special Functions, [68](#)
- cosint
 - GNU Extended Mathematical Special Functions, [68](#)
- cosintf
 - GNU Extended Mathematical Special Functions, [69](#)
- cosintl
 - GNU Extended Mathematical Special Functions, [69](#)
- 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, [69](#), [70](#)
- cyl_hankel_1f
 - GNU Extended Mathematical Special Functions, [70](#)
- cyl_hankel_1l
 - GNU Extended Mathematical Special Functions, [71](#)
- cyl_hankel_2
 - GNU Extended Mathematical Special Functions, [71](#), [72](#)
- cyl_hankel_2f
 - GNU Extended Mathematical Special Functions, [72](#)
- cyl_hankel_2l
 - GNU Extended Mathematical Special Functions, [73](#)
- 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, [73](#)
- dawsonf
 - GNU Extended Mathematical Special Functions, [74](#)
- dawsonl
 - GNU Extended Mathematical Special Functions, [74](#)
- digamma
 - GNU Extended Mathematical Special Functions, [74](#)
- digammaf
 - GNU Extended Mathematical Special Functions, [74](#)
- digammal
 - GNU Extended Mathematical Special Functions, [74](#)
- dilog
 - GNU Extended Mathematical Special Functions, [74](#)
- dilogf
 - GNU Extended Mathematical Special Functions, [75](#)
- dilogl
 - GNU Extended Mathematical Special Functions, [75](#)
- dirichlet_beta
 - GNU Extended Mathematical Special Functions, [75](#)
- dirichlet_betaf
 - GNU Extended Mathematical Special Functions, [76](#)
- dirichlet_betal
 - GNU Extended Mathematical Special Functions, [76](#)
- dirichlet_eta
 - GNU Extended Mathematical Special Functions, [76](#)
- dirichlet_etaf
 - GNU Extended Mathematical Special Functions, [76](#)
- dirichlet_etall
 - GNU Extended Mathematical Special Functions, [77](#)
- double_factorial
 - GNU Extended Mathematical Special Functions, [77](#)
- double_factorialf
 - GNU Extended Mathematical Special Functions, [77](#)
- double_factoriall
 - GNU Extended Mathematical Special Functions, [77](#)
- 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, [77](#)
- ellint_celf
 - GNU Extended Mathematical Special Functions, [78](#)
- ellint_cell
 - GNU Extended Mathematical Special Functions, [78](#)
- ellint_d
 - GNU Extended Mathematical Special Functions, [78](#)
- ellint_df
 - GNU Extended Mathematical Special Functions, [79](#)
- ellint_dl
 - GNU Extended Mathematical Special Functions, [79](#)
- ellint_el1
 - GNU Extended Mathematical Special Functions, [79](#)
- ellint_el1f
 - GNU Extended Mathematical Special Functions, [80](#)
- ellint_el1l
 - GNU Extended Mathematical Special Functions, [80](#)
- ellint_el2
 - GNU Extended Mathematical Special Functions, [80](#)
- ellint_el2f
 - GNU Extended Mathematical Special Functions, [80](#)
- ellint_el2l
 - GNU Extended Mathematical Special Functions, [81](#)
- ellint_el3
 - GNU Extended Mathematical Special Functions, [81](#)
- ellint_el3f
 - GNU Extended Mathematical Special Functions, [81](#)
- ellint_el3l
 - GNU Extended Mathematical Special Functions, [82](#)
- ellint_rc
 - GNU Extended Mathematical Special Functions, [82](#)
- ellint_rcf
 - GNU Extended Mathematical Special Functions, [83](#)
- ellint_rcl
 - GNU Extended Mathematical Special Functions, [83](#)
- ellint_rd
 - GNU Extended Mathematical Special Functions, [83](#)
- ellint_rdf
 - GNU Extended Mathematical Special Functions, [84](#)
- ellint_rdl
 - GNU Extended Mathematical Special Functions, [84](#)
- ellint_rf
 - GNU Extended Mathematical Special Functions, [84](#)
- ellint_rff
 - GNU Extended Mathematical Special Functions, [84](#)
- ellint_rfl
 - GNU Extended Mathematical Special Functions, [85](#)
- ellint_rg
 - GNU Extended Mathematical Special Functions, [85](#)
- ellint_rgf
 - GNU Extended Mathematical Special Functions, [86](#)
- ellint_rgl
 - GNU Extended Mathematical Special Functions, [86](#)
- ellint_rj
 - GNU Extended Mathematical Special Functions, [86](#)
- ellint_rjf
 - GNU Extended Mathematical Special Functions, [87](#)
- ellint_rjl
 - GNU Extended Mathematical Special Functions, [87](#)
- ellnome
 - GNU Extended Mathematical Special Functions, [87](#)
- ellnomef
 - GNU Extended Mathematical Special Functions, [87](#)
- ellnomel
 - GNU Extended Mathematical Special Functions, [88](#)
- evenzeta
 - std::__detail, [255](#)
- expint
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [88](#)
- expintf
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [88](#)
- expintl
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [89](#)
- factorial
 - GNU Extended Mathematical Special Functions, [89](#)
- factorialf
 - GNU Extended Mathematical Special Functions, [89](#)
- factoriall
 - GNU Extended Mathematical Special Functions, [89](#)
- fai
 - std::__detail::_AiryAuxilliaryState, [281](#)
- faip
 - std::__detail::_AiryAuxilliaryState, [281](#)
- fresnel_c
 - GNU Extended Mathematical Special Functions, [89](#)
- fresnel_cf
 - GNU Extended Mathematical Special Functions, [90](#)
- fresnel_cl
 - GNU Extended Mathematical Special Functions, [90](#)
- fresnel_s
 - GNU Extended Mathematical Special Functions, [90](#)
- fresnel_sf
 - GNU Extended Mathematical Special Functions, [90](#)
- fresnel_sl
 - GNU Extended Mathematical Special Functions, [90](#)
- GNU Extended Mathematical Special Functions, [44](#)
 - _GLIBCXX_JACOBI_CN, [52](#)
 - _GLIBCXX_JACOBI_DN, [52](#)
 - _GLIBCXX_JACOBI_SN, [52](#)
 - airy_ai, [52](#)
 - airy_aif, [53](#)

airy_ail, [53](#)
airy_bi, [53](#)
airy_bif, [54](#)
airy_bil, [54](#)
bernoulli, [54](#)
bernoullif, [54](#)
bernoullil, [55](#)
bincoef, [55](#)
bincoeff, [55](#)
bincoefl, [55](#)
chebyshev_t, [55](#)
chebyshev_tf, [56](#)
chebyshev_tl, [56](#)
chebyshev_u, [56](#)
chebyshev_uf, [57](#)
chebyshev_ul, [57](#)
chebyshev_v, [57](#)
chebyshev_vf, [58](#)
chebyshev_vl, [58](#)
chebyshev_w, [58](#)
chebyshev_wf, [59](#)
chebyshev_wl, [59](#)
clausen, [59](#), [60](#)
clausen_c, [60](#)
clausen_cf, [61](#)
clausen_cl, [61](#)
clausen_s, [61](#)
clausen_sf, [62](#)
clausen_sl, [62](#)
clausenf, [62](#)
clausenl, [63](#)
comp_ellint_d, [63](#)
comp_ellint_df, [64](#)
comp_ellint_dl, [64](#)
comp_ellint_rf, [64](#)
comp_ellint_rg, [65](#)
conf_hyperg, [66](#)
conf_hyperg_lim, [66](#)
conf_hyperg_limf, [66](#)
conf_hyperg_liml, [67](#)
conf_hypergf, [67](#)
conf_hypergl, [67](#)
coshint, [67](#)
coshintf, [68](#)
coshintl, [68](#)
cosint, [68](#)
cosintf, [69](#)
cosintl, [69](#)
cyl_hankel_1, [69](#), [70](#)
cyl_hankel_1f, [70](#)
cyl_hankel_1l, [71](#)
cyl_hankel_2, [71](#), [72](#)
cyl_hankel_2f, [72](#)
cyl_hankel_2l, [73](#)
dawson, [73](#)
dawsonf, [74](#)
dawsonl, [74](#)
digamma, [74](#)
digammaf, [74](#)
digammal, [74](#)
dilog, [74](#)
dilogf, [75](#)
dilogl, [75](#)
dirichlet_beta, [75](#)
dirichlet_betaf, [76](#)
dirichlet_betel, [76](#)
dirichlet_eta, [76](#)
dirichlet_etaf, [76](#)
dirichlet_etel, [77](#)
double_factorial, [77](#)
double_factorialf, [77](#)
double_factoriall, [77](#)
ellint_cel, [77](#)
ellint_celf, [78](#)
ellint_cell, [78](#)
ellint_d, [78](#)
ellint_df, [79](#)
ellint_dl, [79](#)
ellint_el1, [79](#)
ellint_el1f, [80](#)
ellint_el1l, [80](#)
ellint_el2, [80](#)
ellint_el2f, [80](#)
ellint_el2l, [81](#)
ellint_el3, [81](#)
ellint_el3f, [81](#)
ellint_el3l, [82](#)
ellint_rc, [82](#)
ellint_rcf, [83](#)
ellint_rcl, [83](#)
ellint_rd, [83](#)
ellint_rdf, [84](#)
ellint_rdl, [84](#)
ellint_rf, [84](#)
ellint_rff, [84](#)
ellint_rfl, [85](#)
ellint_rg, [85](#)
ellint_rgf, [86](#)
ellint_rgl, [86](#)
ellint_rj, [86](#)
ellint_rjf, [87](#)
ellint_rjl, [87](#)
ellnome, [87](#)
ellnomef, [87](#)
ellnomel, [88](#)
expint, [88](#)
expintf, [88](#)
expintl, [89](#)

factorial, [89](#)
 factorialf, [89](#)
 factoriall, [89](#)
 fresnel_c, [89](#)
 fresnel_cf, [90](#)
 fresnel_cl, [90](#)
 fresnel_s, [90](#)
 fresnel_sf, [90](#)
 fresnel_sl, [90](#)
 gamma_l, [90](#)
 gamma_lf, [90](#)
 gamma_ll, [91](#)
 gamma_u, [91](#)
 gamma_uf, [91](#)
 gamma_ul, [91](#)
 gegenbauer, [91](#)
 gegenbauerf, [92](#)
 gegenbauerl, [92](#)
 heuman_lambda, [92](#)
 heuman_lambdaf, [93](#)
 heuman_lambdal, [93](#)
 hurwitz_zeta, [93](#)
 hurwitz_zetaf, [93](#)
 hurwitz_zetal, [93](#)
 hyperg, [94](#)
 hypergf, [94](#)
 hypergl, [94](#)
 ibeta, [95](#)
 ibetac, [95](#)
 ibetacf, [96](#)
 ibetacl, [96](#)
 ibetaf, [96](#)
 ibetal, [96](#)
 jacobi, [96](#)
 jacobi_cn, [97](#)
 jacobi_cnf, [97](#)
 jacobi_cnl, [98](#)
 jacobi_dn, [98](#)
 jacobi_dnf, [98](#)
 jacobi_dnl, [99](#)
 jacobi_sn, [99](#)
 jacobi_snf, [99](#)
 jacobi_snl, [100](#)
 jacobi_zeta, [100](#)
 jacobi_zetaf, [100](#)
 jacobi_zetal, [100](#)
 jacobif, [101](#)
 jacobil, [101](#)
 lbincoef, [101](#)
 lbincoeff, [101](#)
 lbincoefl, [101](#)
 ldouble_factorial, [101](#)
 ldouble_factorialf, [102](#)
 ldouble_factoriall, [102](#)
 legendre_q, [102](#)
 legendre_qf, [102](#)
 legendre_ql, [102](#)
 lfactorial, [102](#)
 lfactorialf, [102](#)
 lfactoriall, [103](#)
 logint, [103](#)
 logintf, [103](#)
 logintl, [103](#)
 lpochhammer_l, [103](#)
 lpochhammer_lf, [104](#)
 lpochhammer_ll, [104](#)
 lpochhammer_u, [104](#)
 lpochhammer_uf, [104](#)
 lpochhammer_ul, [104](#)
 owens_t, [104](#)
 owens_tf, [105](#)
 owens_tl, [105](#)
 pgamma, [105](#)
 pgammaf, [105](#)
 pgammal, [105](#)
 pochhammer_l, [105](#)
 pochhammer_lf, [105](#)
 pochhammer_ll, [106](#)
 pochhammer_u, [106](#)
 pochhammer_uf, [106](#)
 pochhammer_ul, [106](#)
 polylog, [106](#)
 polylogf, [107](#)
 polylogl, [107](#)
 psi, [108](#)
 psif, [108](#)
 psil, [108](#)
 qgamma, [108](#)
 qgammaf, [108](#)
 qgammal, [108](#)
 radpoly, [109](#)
 radpolyf, [109](#)
 radpolyl, [109](#)
 sinc, [110](#)
 sinc_pi, [110](#)
 sinc_pif, [111](#)
 sinc_pil, [111](#)
 sincf, [111](#)
 sincl, [111](#)
 sinhc, [111](#)
 sinhc_pi, [112](#)
 sinhc_pif, [112](#)
 sinhc_pil, [112](#)
 sinhcf, [112](#)
 sinhcl, [112](#)
 sinhint, [112](#)
 sinhintf, [113](#)
 sinhintl, [113](#)

- sinint, [113](#)
- sinintf, [113](#)
- sinintl, [114](#)
- sph_bessel_i, [114](#)
- sph_bessel_if, [114](#)
- sph_bessel_il, [115](#)
- sph_bessel_k, [115](#)
- sph_bessel_kf, [116](#)
- sph_bessel_kl, [116](#)
- sph_hankel_1, [116](#), [117](#)
- sph_hankel_1f, [117](#)
- sph_hankel_1l, [117](#), [118](#)
- sph_hankel_2, [118](#)
- sph_hankel_2f, [119](#)
- sph_hankel_2l, [119](#), [120](#)
- sph_harmonic, [120](#)
- sph_harmonicf, [120](#)
- sph_harmonicl, [121](#)
- theta_1, [121](#)
- theta_1f, [121](#)
- theta_1l, [121](#)
- theta_2, [122](#)
- theta_2f, [122](#)
- theta_2l, [122](#)
- theta_3, [122](#)
- theta_3f, [123](#)
- theta_3l, [123](#)
- theta_4, [123](#)
- theta_4f, [124](#)
- theta_4l, [124](#)
- theta_c, [124](#)
- theta_cf, [124](#)
- theta_cl, [125](#)
- theta_d, [125](#)
- theta_df, [125](#)
- theta_dl, [125](#)
- theta_n, [126](#)
- theta_nf, [126](#)
- theta_nl, [126](#)
- theta_s, [126](#)
- theta_sf, [127](#)
- theta_sl, [127](#)
- zernike, [127](#)
- zernikef, [128](#)
- zernikel, [128](#)
- gai
 - std::__detail::__AiryAuxilliaryState, [281](#)
- gaip
 - std::__detail::__AiryAuxilliaryState, [281](#)
- gamma_l
 - GNU Extended Mathematical Special Functions, [90](#)
- gamma_if
 - GNU Extended Mathematical Special Functions, [90](#)
- gamma_II
 - GNU Extended Mathematical Special Functions, [91](#)
- gamma_u
 - GNU Extended Mathematical Special Functions, [91](#)
- gamma_uf
 - GNU Extended Mathematical Special Functions, [91](#)
- gamma_ul
 - GNU Extended Mathematical Special Functions, [91](#)
- gegenbauer
 - GNU Extended Mathematical Special Functions, [91](#)
- gegenbauerf
 - GNU Extended Mathematical Special Functions, [92](#)
- gegenbauerl
 - GNU Extended Mathematical Special Functions, [92](#)
- hai
 - std::__detail::__AiryAuxilliaryState, [281](#)
- haip
 - std::__detail::__AiryAuxilliaryState, [281](#)
- 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, [92](#)
- heuman_lambdaf
 - GNU Extended Mathematical Special Functions, [93](#)
- heuman_lambdal
 - GNU Extended Mathematical Special Functions, [93](#)
- hurwitz_zeta
 - GNU Extended Mathematical Special Functions, [93](#)
- hurwitz_zetaf
 - GNU Extended Mathematical Special Functions, [93](#)
- hurwitz_zetal
 - GNU Extended Mathematical Special Functions, [93](#)
- hyperg
 - GNU Extended Mathematical Special Functions, [94](#)
- hypergf
 - GNU Extended Mathematical Special Functions, [94](#)
- hypergl
 - GNU Extended Mathematical Special Functions, [94](#)
- ibeta
 - GNU Extended Mathematical Special Functions, [95](#)
- ibetac
 - GNU Extended Mathematical Special Functions, [95](#)
- ibetacf
 - GNU Extended Mathematical Special Functions, [96](#)
- ibetacl
 - GNU Extended Mathematical Special Functions, [96](#)
- ibetaf
 - GNU Extended Mathematical Special Functions, [96](#)
- ibetal
 - GNU Extended Mathematical Special Functions, [96](#)

- inner_radius
 - std::__detail::__Airy, [262](#)
 - std::__detail::__Airy_default_radii< double >, [272](#)
 - std::__detail::__Airy_default_radii< float >, [273](#)
 - std::__detail::__Airy_default_radii< long double >, [274](#)
- jacobi
 - GNU Extended Mathematical Special Functions, [96](#)
- jacobi_cn
 - GNU Extended Mathematical Special Functions, [97](#)
- jacobi_cnf
 - GNU Extended Mathematical Special Functions, [97](#)
- jacobi_cnl
 - GNU Extended Mathematical Special Functions, [98](#)
- jacobi_dn
 - GNU Extended Mathematical Special Functions, [98](#)
- jacobi_dnf
 - GNU Extended Mathematical Special Functions, [98](#)
- jacobi_dnl
 - GNU Extended Mathematical Special Functions, [99](#)
- jacobi_sn
 - GNU Extended Mathematical Special Functions, [99](#)
- jacobi_snf
 - GNU Extended Mathematical Special Functions, [99](#)
- jacobi_snl
 - GNU Extended Mathematical Special Functions, [100](#)
- jacobi_zeta
 - GNU Extended Mathematical Special Functions, [100](#)
- jacobi_zetaf
 - GNU Extended Mathematical Special Functions, [100](#)
- jacobi_zetal
 - GNU Extended Mathematical Special Functions, [100](#)
- jacobif
 - GNU Extended Mathematical Special Functions, [101](#)
- jacobil
 - GNU Extended Mathematical Special Functions, [101](#)
- 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, [101](#)
- lbincoeff
 - GNU Extended Mathematical Special Functions, [101](#)
- lbincoefl
 - GNU Extended Mathematical Special Functions, [101](#)
- ldouble_factorial
 - GNU Extended Mathematical Special Functions, [101](#)
- ldouble_factorialf
 - GNU Extended Mathematical Special Functions, [102](#)
- ldouble_factoriall
 - GNU Extended Mathematical Special Functions, [102](#)
- legendre
 - C++17/IS29124 Mathematical Special Functions, [38](#)
- legendre_q
 - GNU Extended Mathematical Special Functions, [102](#)
- legendre_qf
 - GNU Extended Mathematical Special Functions, [102](#)
- legendre_ql
 - GNU Extended Mathematical Special Functions, [102](#)
- legendref
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- legendrel
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- lfactorial
 - GNU Extended Mathematical Special Functions, [102](#)
- lfactorialf
 - GNU Extended Mathematical Special Functions, [102](#)
- lfactoriall
 - GNU Extended Mathematical Special Functions, [103](#)
- logint
 - GNU Extended Mathematical Special Functions, [103](#)
- logintf
 - GNU Extended Mathematical Special Functions, [103](#)
- logintl
 - GNU Extended Mathematical Special Functions, [103](#)
- lpochhammer_l
 - GNU Extended Mathematical Special Functions, [103](#)
- lpochhammer_lf
 - GNU Extended Mathematical Special Functions, [104](#)
- lpochhammer_ll
 - GNU Extended Mathematical Special Functions, [104](#)
- lpochhammer_u
 - GNU Extended Mathematical Special Functions, [104](#)
- lpochhammer_uf
 - GNU Extended Mathematical Special Functions, [104](#)
- lpochhammer_ul
 - GNU Extended Mathematical Special Functions, [104](#)
- operator()
 - std::__detail::__Airy, [260](#)
 - std::__detail::__Airy_asymp, [265](#)
 - std::__detail::__Airy_asymp_series, [271](#)
- outer_radius
 - std::__detail::__Airy, [262](#)
 - std::__detail::__Airy_default_radii< double >, [272](#)
 - std::__detail::__Airy_default_radii< float >, [273](#)
 - std::__detail::__Airy_default_radii< long double >, [274](#)
- owens_t
 - GNU Extended Mathematical Special Functions, [104](#)
- owens_tf
 - GNU Extended Mathematical Special Functions, [105](#)
- owens_tl
 - GNU Extended Mathematical Special Functions, [105](#)

- pgamma
 - GNU Extended Mathematical Special Functions, [105](#)
- pgammaf
 - GNU Extended Mathematical Special Functions, [105](#)
- pgammal
 - GNU Extended Mathematical Special Functions, [105](#)
- pochhammer_l
 - GNU Extended Mathematical Special Functions, [105](#)
- pochhammer_lf
 - GNU Extended Mathematical Special Functions, [105](#)
- pochhammer_ll
 - GNU Extended Mathematical Special Functions, [106](#)
- pochhammer_u
 - GNU Extended Mathematical Special Functions, [106](#)
- pochhammer_uf
 - GNU Extended Mathematical Special Functions, [106](#)
- pochhammer_ul
 - GNU Extended Mathematical Special Functions, [106](#)
- polylog
 - GNU Extended Mathematical Special Functions, [106](#)
- polylogf
 - GNU Extended Mathematical Special Functions, [107](#)
- polylogl
 - GNU Extended Mathematical Special Functions, [107](#)
- psi
 - GNU Extended Mathematical Special Functions, [108](#)
- psif
 - GNU Extended Mathematical Special Functions, [108](#)
- psil
 - GNU Extended Mathematical Special Functions, [108](#)
- qgamma
 - GNU Extended Mathematical Special Functions, [108](#)
- qgammaf
 - GNU Extended Mathematical Special Functions, [108](#)
- qgammal
 - GNU Extended Mathematical Special Functions, [108](#)
- radpoly
 - GNU Extended Mathematical Special Functions, [109](#)
- radpolyf
 - GNU Extended Mathematical Special Functions, [109](#)
- radpolyl
 - GNU Extended Mathematical Special Functions, [109](#)
- 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, [156](#)
- scalar_type
 - std::__detail::_Airy, [260](#)
- std::__detail::_Airy_asymp_series, [270](#)
- sf_airy.tcc
 - _GLIBCXX_BITS_SF_AIRY_TCC, [287](#)
- sf_bessel.tcc
 - _GLIBCXX_BITS_SF_BESSEL_TCC, [289](#)
- sf_beta.tcc
 - _GLIBCXX_BITS_SF_BETA_TCC, [291](#)
- sf_cardinal.tcc
 - _GLIBCXX_BITS_SF_CARDINAL_TCC, [292](#)
- sf_chebyshev.tcc
 - _GLIBCXX_BITS_SF_CHEBYSHEV_TCC, [294](#)
- sf_dawson.tcc
 - _GLIBCXX_BITS_SF_DAWSON_TCC, [295](#)
- sf_ellint.tcc
 - _GLIBCXX_BITS_SF_ELLINT_TCC, [298](#)
- sf_expint.tcc
 - _GLIBCXX_BITS_SF_EXPINT_TCC, [300](#)
- sf_fresnel.tcc
 - _GLIBCXX_BITS_SF_FRESNEL_TCC, [302](#)
- sf_gamma.tcc
 - _GLIBCXX_BITS_SF_GAMMA_TCC, [308](#)
- sf_gegenbauer.tcc
 - _GLIBCXX_BITS_SF_GEGENBAUER_TCC, [309](#)
- sf_hankel.tcc
 - _GLIBCXX_BITS_SF_HANKEL_TCC, [312](#)
- sf_hermite.tcc
 - _GLIBCXX_BITS_SF_HERMITE_TCC, [313](#)
- sf_hydrogen.tcc
 - _GLIBCXX_BITS_SF_HYDROGEN_TCC, [315](#)
- sf_hyperg.tcc
 - _GLIBCXX_BITS_SF_HYPERG_TCC, [317](#)
- sf_hypint.tcc
 - _GLIBCXX_BITS_SF_HYPINT_TCC, [318](#)
- sf_jacobi.tcc
 - _GLIBCXX_BITS_SF_JACOBI_TCC, [319](#)
- sf_laguerre.tcc
 - _GLIBCXX_BITS_SF_LAGUERRE_TCC, [321](#)
- sf_legendre.tcc
 - _GLIBCXX_BITS_SF_LEGENDRE_TCC, [323](#)
- sf_mod_bessel.tcc
 - _GLIBCXX_BITS_SF_MOD_BESSEL_TCC, [325](#)
- sf_owens_t.tcc
 - _GLIBCXX_BITS_SF_OWENS_T_TCC, [326](#)
- sf_polylog.tcc
 - _GLIBCXX_BITS_SF_POLYLOG_TCC, [329](#)
- sf_theta.tcc
 - _GLIBCXX_BITS_SF_THETA_TCC, [331](#)
- sf_trigint.tcc
 - _GLIBCXX_BITS_SF_TRIGINT_TCC, [333](#)
- sf_zeta.tcc
 - _GLIBCXX_BITS_SF_ZETA_TCC, [335](#)
- sinc
 - GNU Extended Mathematical Special Functions, [110](#)
- sinc_pi

- GNU Extended Mathematical Special Functions, [110](#)
- `sinc_pif`
 - GNU Extended Mathematical Special Functions, [111](#)
- `sinc_pil`
 - GNU Extended Mathematical Special Functions, [111](#)
- `sincf`
 - GNU Extended Mathematical Special Functions, [111](#)
- `sincl`
 - GNU Extended Mathematical Special Functions, [111](#)
- `sinhc`
 - GNU Extended Mathematical Special Functions, [111](#)
- `sinhc_pi`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhc_pif`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhc_pil`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhcf`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhcl`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhint`
 - GNU Extended Mathematical Special Functions, [112](#)
- `sinhintf`
 - GNU Extended Mathematical Special Functions, [113](#)
- `sinhintl`
 - GNU Extended Mathematical Special Functions, [113](#)
- `sinint`
 - GNU Extended Mathematical Special Functions, [113](#)
- `sinintf`
 - GNU Extended Mathematical Special Functions, [113](#)
- `sinintl`
 - GNU Extended Mathematical Special Functions, [114](#)
- `specfun.h`
 - `__STDCPP_MATH_SPEC_FUNCS__`, [346](#)
 - `__cpp_lib_math_special_functions`, [346](#)
- `sph_bessel`
 - C++17/IS29124 Mathematical Special Functions, [40](#)
- `sph_bessel_i`
 - GNU Extended Mathematical Special Functions, [114](#)
- `sph_bessel_if`
 - GNU Extended Mathematical Special Functions, [114](#)
- `sph_bessel_il`
 - GNU Extended Mathematical Special Functions, [115](#)
- `sph_bessel_k`
 - GNU Extended Mathematical Special Functions, [115](#)
- `sph_bessel_kf`
 - GNU Extended Mathematical Special Functions, [116](#)
- `sph_bessel_kl`
 - GNU Extended Mathematical Special Functions, [116](#)
- `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, [116](#), [117](#)
- `sph_hankel_1f`
 - GNU Extended Mathematical Special Functions, [117](#)
- `sph_hankel_1l`
 - GNU Extended Mathematical Special Functions, [117](#), [118](#)
- `sph_hankel_2`
 - GNU Extended Mathematical Special Functions, [118](#)
- `sph_hankel_2f`
 - GNU Extended Mathematical Special Functions, [119](#)
- `sph_hankel_2l`
 - GNU Extended Mathematical Special Functions, [119](#), [120](#)
- `sph_harmonic`
 - GNU Extended Mathematical Special Functions, [120](#)
- `sph_harmonicf`
 - GNU Extended Mathematical Special Functions, [120](#)
- `sph_harmonicl`
 - GNU Extended Mathematical Special Functions, [121](#)
- `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`, [137](#)
- `std::`
 - `__detail`, [139](#)
 - `_Num_Euler_Maclaurin_zeta`, [256](#)
 - `_S_Euler_Maclaurin_zeta`, [256](#)
 - `_S_double_factorial_table`, [256](#)
 - `_S_factorial_table`, [256](#)
 - `_S_neg_double_factorial_table`, [256](#)
 - `_S_num_double_factorials`, [256](#)
 - `_S_num_double_factorials< double >`, [257](#)
 - `_S_num_double_factorials< float >`, [257](#)
 - `_S_num_double_factorials< long double >`, [257](#)
 - `_S_num_factorials`, [257](#)
 - `_S_num_factorials< double >`, [257](#)
 - `_S_num_factorials< float >`, [257](#)
 - `_S_num_factorials< long double >`, [257](#)
 - `_S_num_neg_double_factorials`, [257](#)
 - `_S_num_neg_double_factorials< double >`, [257](#)
 - `_S_num_neg_double_factorials< float >`, [257](#)
 - `_S_num_neg_double_factorials< long double >`, [258](#)
 - `_S_num_zetam1`, [258](#)
 - `_S_zetam1`, [258](#)

- [__airy, 156](#)
- [__airy_ai, 157](#)
- [__airy_arg, 157](#)
- [__airy_bi, 157](#)
- [__assoc_laguerre, 157](#)
- [__assoc_legendre_p, 158](#)
- [__bernoulli, 159](#)
- [__bernoulli_2n, 159](#)
- [__bernoulli_series, 159](#)
- [__beta, 160](#)
- [__beta_gamma, 160](#)
- [__beta_inc, 161](#)
- [__beta_inc_cont_frac, 161](#)
- [__beta_lgamma, 162](#)
- [__beta_product, 162](#)
- [__bincoef, 163](#)
- [__bose_einstein, 163](#)
- [__chebyshev_recur, 164](#)
- [__chebyshev_t, 164](#)
- [__chebyshev_u, 165](#)
- [__chebyshev_v, 165](#)
- [__chebyshev_w, 166](#)
- [__chshint, 166](#)
- [__chshint_cont_frac, 167](#)
- [__chshint_series, 167](#)
- [__clamp_0_m2pi, 167](#)
- [__clamp_pi, 167](#)
- [__clausen, 167, 168](#)
- [__clausen_c, 168, 169](#)
- [__clausen_s, 169, 170](#)
- [__comp_ellint_1, 170](#)
- [__comp_ellint_2, 171](#)
- [__comp_ellint_3, 171](#)
- [__comp_ellint_d, 172](#)
- [__comp_ellint_rf, 172](#)
- [__comp_ellint_rg, 172](#)
- [__conf_hyperg, 172](#)
- [__conf_hyperg_lim, 173](#)
- [__conf_hyperg_lim_series, 173](#)
- [__conf_hyperg_luke, 174](#)
- [__conf_hyperg_series, 174](#)
- [__coshint, 174](#)
- [__cyl_bessel, 175](#)
- [__cyl_bessel_i, 175](#)
- [__cyl_bessel_ij_series, 176](#)
- [__cyl_bessel_ik, 176](#)
- [__cyl_bessel_ik_asymp, 178](#)
- [__cyl_bessel_ik_steel, 178](#)
- [__cyl_bessel_j, 179](#)
- [__cyl_bessel_jn, 179](#)
- [__cyl_bessel_jn_asymp, 179](#)
- [__cyl_bessel_jn_steel, 180](#)
- [__cyl_bessel_k, 180](#)
- [__cyl_hankel_1, 181](#)
- [__cyl_hankel_2, 182](#)
- [__cyl_neumann, 183](#)
- [__cyl_neumann_n, 183](#)
- [__dawson, 183](#)
- [__dawson_cont_frac, 184](#)
- [__dawson_series, 184](#)
- [__debye_region, 184](#)
- [__dilog, 184](#)
- [__dirichlet_beta, 185](#)
- [__dirichlet_eta, 186](#)
- [__double_factorial, 187](#)
- [__ellint_1, 187](#)
- [__ellint_2, 187](#)
- [__ellint_3, 188](#)
- [__ellint_cel, 188](#)
- [__ellint_d, 189](#)
- [__ellint_el1, 189](#)
- [__ellint_el2, 189](#)
- [__ellint_el3, 189](#)
- [__ellint_rc, 189](#)
- [__ellint_rd, 190](#)
- [__ellint_rf, 191](#)
- [__ellint_rg, 191](#)
- [__ellint_rj, 192](#)
- [__ellnome, 193](#)
- [__ellnome_k, 193](#)
- [__ellnome_series, 193](#)
- [__expint, 193, 194](#)
- [__expint_E1, 195](#)
- [__expint_E1_asymp, 195](#)
- [__expint_E1_series, 196](#)
- [__expint_Ei, 196](#)
- [__expint_Ei_asymp, 197](#)
- [__expint_Ei_series, 197](#)
- [__expint_En_cont_frac, 198](#)
- [__expint_En_recursion, 198](#)
- [__expint_En_series, 199](#)
- [__expint_asymp, 194](#)
- [__expint_large_n, 199](#)
- [__factorial, 200](#)
- [__fermi_dirac, 200](#)
- [__fock_airy, 201](#)
- [__fpequal, 201](#)
- [__fpimag, 201, 202](#)
- [__fpreal, 202](#)
- [__fresnel, 202](#)
- [__fresnel_cont_frac, 203](#)
- [__fresnel_series, 203](#)
- [__gamma, 203](#)
- [__gamma_cont_frac, 204](#)
- [__gamma_l, 204](#)
- [__gamma_series, 204](#)
- [__gamma_temme, 204](#)
- [__gamma_u, 205](#)

- __gauss, 205
- __gegenbauer_poly, 205
- __hankel, 206
- __hankel_debye, 206
- __hankel_params, 207
- __hankel_uniform, 207
- __hankel_uniform_olver, 207
- __hankel_uniform_outer, 208
- __hankel_uniform_sum, 208
- __heuman_lambda, 209
- __hurwitz_zeta, 209, 210
- __hurwitz_zeta_euler_maclaurin, 210
- __hydrogen, 210
- __hyperg, 211
- __hyperg_luke, 211
- __hyperg_reflect, 211
- __hyperg_series, 212
- __jacobi_sncndn, 213
- __jacobi_zeta, 213
- __laguerre, 213
- __legendre_q, 213
- __log_bincoef, 214
- __log_double_factorial, 214
- __log_factorial, 215
- __log_gamma, 215
- __log_gamma_bernoulli, 216
- __log_gamma_lanczos, 216
- __log_gamma_sign, 216
- __log_gamma_spouge, 217
- __log_pochhammer_l, 217
- __log_pochhammer_u, 218
- __logint, 218
- __max_FGH, 255
- __max_FGH< double >, 255
- __max_FGH< float >, 256
- __owens_t, 219
- __pgamma, 219
- __pochhammer_l, 219
- __pochhammer_u, 220
- __poly_hermite, 220
- __poly_hermite_asymp, 221
- __poly_hermite_recursion, 221
- __poly_jacobi, 222
- __poly_laguerre, 222
- __poly_laguerre_hyperg, 223
- __poly_laguerre_large_n, 224
- __poly_laguerre_recursion, 224
- __poly_legendre_p, 225
- __poly_radial_jacobi, 225
- __polylog, 226, 227
- __polylog_exp, 227
- __polylog_exp_asymp, 228
- __polylog_exp_int_neg, 228, 229
- __polylog_exp_int_pos, 229, 230
- __polylog_exp_neg, 230, 231
- __polylog_exp_neg_even, 231
- __polylog_exp_neg_odd, 232
- __polylog_exp_negative_real_part, 233
- __polylog_exp_pos, 233, 234
- __polylog_exp_real_neg, 235, 236
- __polylog_exp_real_pos, 236
- __psi, 237
- __psi_asymp, 237
- __psi_series, 238
- __qgamma, 238
- __riemann_zeta, 238
- __riemann_zeta_alt, 239
- __riemann_zeta_euler_maclaurin, 239
- __riemann_zeta_glob, 239
- __riemann_zeta_m_1, 239
- __riemann_zeta_m_1_sum, 240
- __riemann_zeta_product, 240
- __riemann_zeta_sum, 241
- __sinc, 241
- __sinc_pi, 241
- __sincosint, 242
- __sincosint_asymp, 242
- __sincosint_cont_frac, 242
- __sincosint_series, 242
- __sinhc, 243
- __sinhc_pi, 243
- __sinhint, 243
- __sph_bessel, 244
- __sph_bessel_ik, 245
- __sph_bessel_jn, 245
- __sph_hankel, 246
- __sph_hankel_1, 246, 247
- __sph_hankel_2, 247
- __sph_harmonic, 248
- __sph_legendre, 248
- __sph_neumann, 249
- __theta_1, 251
- __theta_2, 251
- __theta_2_asymp, 252
- __theta_2_sum, 252
- __theta_3, 252
- __theta_3_asymp, 253
- __theta_3_sum, 253
- __theta_4, 253
- __theta_c, 253
- __theta_d, 253
- __theta_n, 254
- __theta_s, 254
- __zernike, 254
- __znorm1, 255
- __znorm2, 255
- COSINT, 156
- evenzeta, 255

SININT, 156
 std::__detail::_Airy
 _Airy, 260
 _S_2pi_3, 261
 _S_5pi_6, 261
 _S_NaN, 261
 _S_cNaN, 261
 _S_i, 261
 _S_pi, 261
 _S_pi_3, 261
 _S_pi_6, 261
 _S_sqrt_pi, 261
 inner_radius, 262
 operator(), 260
 outer_radius, 262
 scalar_type, 260
 value_type, 260
 std::__detail::_Airy<_Tp>, 259
 std::__detail::_Airy_asymp
 _Airy_asymp, 264
 _S_absarg_ge_pio3, 264
 _S_absarg_lt_pio3, 264
 __cmplx, 264
 operator(), 265
 std::__detail::_Airy_asymp<_Tp>, 262
 std::__detail::_Airy_asymp_data<__float128>, 266
 _S_c, 267
 _S_d, 267
 _S_max_cd, 267
 std::__detail::_Airy_asymp_data<_Tp>, 266
 std::__detail::_Airy_asymp_data<double>, 267
 _S_c, 268
 _S_d, 268
 _S_max_cd, 268
 std::__detail::_Airy_asymp_data<float>, 268
 _S_c, 268
 _S_d, 268
 _S_max_cd, 269
 std::__detail::_Airy_asymp_data<long double>, 269
 _S_c, 269
 _S_d, 269
 _S_max_cd, 269
 std::__detail::_Airy_asymp_series
 _Airy_asymp_series, 271
 _S_sqrt_pi, 271
 operator(), 271
 scalar_type, 270
 value_type, 271
 std::__detail::_Airy_asymp_series<_Sum>, 270
 std::__detail::_Airy_default_radii<_Tp>, 272
 std::__detail::_Airy_default_radii<double>, 272
 inner_radius, 272
 outer_radius, 272
 std::__detail::_Airy_default_radii<float>, 273
 inner_radius, 273
 outer_radius, 273
 std::__detail::_Airy_default_radii<long double>, 273
 inner_radius, 274
 outer_radius, 274
 std::__detail::_Airy_series
 _N_FGH, 278
 _S_Ai, 275
 _S_Ai0, 278
 _S_Aip0, 279
 _S_Airy, 275
 _S_Bi, 276
 _S_Bi0, 279
 _S_Bip0, 279
 _S_FGH, 276
 _S_Fock, 277
 _S_Gi0, 279
 _S_Gip0, 279
 _S_Hi0, 279
 _S_Hip0, 279
 _S_Scorer, 277
 _S_Scorer2, 278
 _S_eps, 279
 _S_i, 280
 _S_log10min, 280
 _S_pi, 280
 _S_sqrt_pi, 280
 std::__detail::_Airy_series<_Tp>, 274
 std::__detail::_AiryAuxilliaryState
 _Val, 281
 fai, 281
 faip, 281
 gai, 281
 gaip, 281
 hai, 281
 haip, 281
 z, 282
 std::__detail::_AiryAuxilliaryState<_Tp>, 280
 std::__detail::_AiryState
 _Val, 283
 Ai, 283
 Aip, 283
 Bi, 283
 Bip, 283
 true_Wronskian, 283
 Wronskian, 283
 z, 283
 std::__detail::_AiryState<_Tp>, 282
 std::__detail::_Factorial_table
 __factorial, 284
 __log_factorial, 284
 __n, 284
 std::__detail::_Factorial_table<_Tp>, 284

- theta_1
 - GNU Extended Mathematical Special Functions, [121](#)
- theta_1f
 - GNU Extended Mathematical Special Functions, [121](#)
- theta_1l
 - GNU Extended Mathematical Special Functions, [121](#)
- theta_2
 - GNU Extended Mathematical Special Functions, [122](#)
- theta_2f
 - GNU Extended Mathematical Special Functions, [122](#)
- theta_2l
 - GNU Extended Mathematical Special Functions, [122](#)
- theta_3
 - GNU Extended Mathematical Special Functions, [122](#)
- theta_3f
 - GNU Extended Mathematical Special Functions, [123](#)
- theta_3l
 - GNU Extended Mathematical Special Functions, [123](#)
- theta_4
 - GNU Extended Mathematical Special Functions, [123](#)
- theta_4f
 - GNU Extended Mathematical Special Functions, [124](#)
- theta_4l
 - GNU Extended Mathematical Special Functions, [124](#)
- theta_c
 - GNU Extended Mathematical Special Functions, [124](#)
- theta_cf
 - GNU Extended Mathematical Special Functions, [124](#)
- theta_cl
 - GNU Extended Mathematical Special Functions, [125](#)
- theta_d
 - GNU Extended Mathematical Special Functions, [125](#)
- theta_df
 - GNU Extended Mathematical Special Functions, [125](#)
- theta_dl
 - GNU Extended Mathematical Special Functions, [125](#)
- theta_n
 - GNU Extended Mathematical Special Functions, [126](#)
- theta_nf
 - GNU Extended Mathematical Special Functions, [126](#)
- theta_nl
 - GNU Extended Mathematical Special Functions, [126](#)
- theta_s
 - GNU Extended Mathematical Special Functions, [126](#)
- theta_sf
 - GNU Extended Mathematical Special Functions, [127](#)
- theta_sl
 - GNU Extended Mathematical Special Functions, [127](#)
- true_Wronskian
 - std::__detail::_AiryState, [283](#)
- value_type
 - std::__detail::_Airy, [260](#)
 - std::__detail::_Airy_asymp_series, [271](#)
- Wronskian
 - std::__detail::_AiryState, [283](#)
- z
 - std::__detail::_AiryAuxilliaryState, [282](#)
 - std::__detail::_AiryState, [283](#)
- zernike
 - GNU Extended Mathematical Special Functions, [127](#)
- zernikef
 - GNU Extended Mathematical Special Functions, [128](#)
- zernikel
 - GNU Extended Mathematical Special Functions, [128](#)